软件工程师开发大系

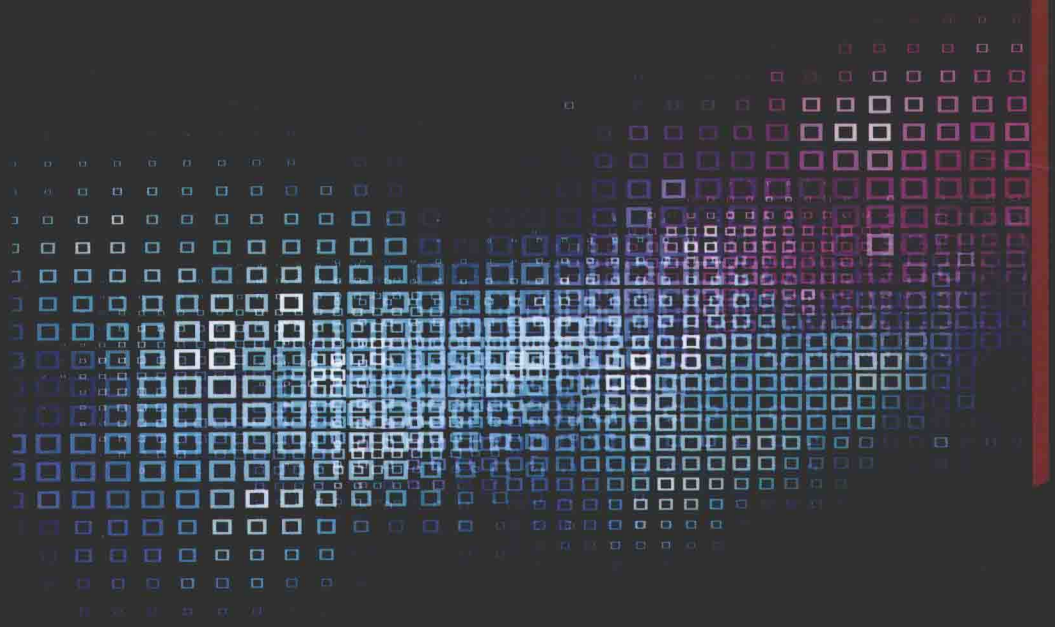
C#

开发实例大全（基础卷）

600个典型实例及源码分析，涵盖25个应用方向

✔工作应用速查 ✔项目开发参考 ✔学习实战练习

软件开发技术联盟◎编著



- 工作应用速查：本书实例全面、系统，涉及程序开发的各个方面，适合各级程序开发人员速查速用。
- 项目开发参考：程序员借助本书提供的实例源代码，可以快速搭建工程项目，提高开发效率。
- 学习实战练习：入门者的实战训练大全，不但可以激发学习兴趣，更可提高编程实战能力和编程思维水平。

DVD

清华大学出版社

软件工程师开发大系

C#开发实例大全

（基础卷）

软件开发技术联盟　编著

清华大学出版社

内 容 简 介

《C#开发实例大全（基础卷）》筛选、汇集了C#开发从基础知识到高级应用各个层面约600个实例及源代码，每个实例都按实例说明、关键技术、设计过程、详尽注释、秘笈心法的顺序进行了分析解读。全书分6篇共25章，主要内容有C#开发环境的使用、C#语言基础应用、字符串处理技术、数组和集合的使用、面向对象编程技术、数据结构与算法、Windows窗体基础、特色窗体界面、窗体控制技术、MDI窗体和继承窗体、Windows常用控件的使用、Windows高级控件的使用、DataGridView数据控件、自定义用户控件、文件基本操作、文件夹基本操作、文件流操作、加密、解密及解压缩文件、C#与Word互操作、高效应用Excel、基本图形绘制、图像处理技术、常用图表应用、动画处理技术、音频与视频控制。配书光盘附带了实例的源程序和部分视频。

《C#开发实例大全（基础卷）》既适合C#程序员参考和查阅，也适合C#初学者，如高校学生、软件开发培训学员及相关求职人员学习、练习、速查使用。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

C#开发实例大全．基础卷/软件开发技术联盟编著．—北京：清华大学出版社，2016
（软件工程师开发大系）
ISBN 978-7-302-38438-0

I．①C…　II．①软…　III．①C语言-程序设计　IV．①TP312

中国版本图书馆CIP数据核字（2014）第260769号

责任编辑：赵洛育
封面设计：李志伟
版式设计：魏　远
责任校对：王　云
责任印制：沈　露

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
　　社 总 机：010-62770175　　**邮　　购**：010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张**：56.75　　**字　　数**：1870千字
　　（附光盘1张）
版　　次：2016年1月第1版　　**印　　次**：2016年1月第1次印刷
印　　数：1～3500
定　　价：128.00元

产品编号：052252-01

前　言

Preface

特别说明：

《C#开发实例大全》分为基础卷（即本书）和提高卷两册。本书的前身是《C#开发实战1200例（第I卷）》。

编写目的

1．方便程序员查阅

程序开发是一项艰辛的工作，挑灯夜战、加班加点是常有的事。在开发过程中，一个技术问题可能会占用几天甚至更长时间。如果有一本开发实例大全可供翻阅，从中找到相似的实例作参考，也许几分钟就可以解决问题。本书编写的主要目的就是方便程序员查阅、提高开发效率。

2．通过分析大量源代码，达到快速学习之目的

本书提供了约600个开发实例及源代码，附有相应的注释、实例说明、关键技术、设计过程和秘笈心法，对实例中的源代码进行了比较透彻的解析。相信这种办法对激发学习兴趣、提高学习效率极有帮助。

3．通过阅读大量源代码，达到提高熟练度之目的

俗话说“熟能生巧”，读者只有通过阅读、分析大量源代码，并亲自动手去做，才能够深刻理解、运用自如，进而提高编程熟练度，适应工作之需要。

4．实例源程序可以“拿来”就用，提高了效率

本书的很多实例，可以根据实际应用需求稍加改动，拿来就用，不必再去从头编写，从而节约了时间，提高了工作效率。

本书内容

本书分为6篇25章，共约600个实例，主要内容有C#开发环境的使用、C#语言基础应用、字符串处理技术、数组和集合的使用、面向对象编程技术、数据结构与算法、Windows窗体基础、特色窗体界面、窗体控制技术、MDI窗体和继承窗体、Windows常用控件的使用、Windows高级控件的使用、DataGridView数据控件、自定义用户控件、文件基本操作、文件夹基本操作、文件流操作、加密、解密及解压缩文件、C#与Word互操作、高效应用Excel、基本图形绘制、图像处理技术、常用图表应用、动画处理技术、音频与视频控制。

书中所选实例均来源于一线开发人员的项目开发实践，囊括了开发中经常碰到和需要解决的热点、难点问题，使读者可以快速解决开发中的难题，提高编程效率。本书知识结构如下图所示。

本书在讲解实例时采用统一的编排样式，多数实例由“实例说明”“关键技术”“设计过程”“秘笈心法”4部分构成。其中，“实例说明”部分采用图文结合的方式介绍实例的功能和运行效果；“关键技术”部分介绍了实例使用的重点、难点技术；“设计过程”部分讲解了实例的详细开发过程；“秘笈心法”部分给出了与实例相关的技巧和经验总结。

本书特点

1．实例极为丰富

本书精选了约 600 个实例，另外一册《C#开发实例大全（提高卷）》也精选了提高部分约 600 个实例，这样，两册图书总计约 1200 个实例，可以说是目前市场上实例最多、知识点最全面、内容最丰富的软件开发类图书，涵盖了编程中各个方面的应用。

2．程序解释详尽

本书提供的实例及源代码，附有相应的注释、实例说明、关键技术、设计过程和秘笈心法。分析解释详尽，便于快速学习。

3．实践实战性强

本书的实例及源代码很多来自现实开发中，光盘中给出了绝大多数实例的全部源代码，读者可以直接调用、研读、练习。

关于光盘

1．实例学习注意事项

读者在按照本书学习、练习的过程中，可以从光盘中复制源代码，修改时注意去掉源码文件的只读属性。**有些实例需要使用相应的数据库或第三方资源，在使用前需要进行相应配置，具体步骤请参考书中或者光盘中**

的配置说明。

2．实例源代码及视频位置

本书光盘提供了实例的源代码，位置在光盘中的“MR\章号\实例序号”文件夹下，例如，“MR\04\106”表示实例106，位于第4章。部分实例提供的视频讲解，也可根据以上方式查找。**由于有些实例源代码较长，限于篇幅，图书中只给出了关键代码，完整代码放置在光盘中。**

3．视频使用说明

本书提供了部分实例的视频讲解，在目录中标题前边有视频图标的实例，即表示在光盘中有视频讲解。视频采用EXE文件格式，无须使用播放器，双击就可以直接播放。

读者对象

C#程序员，C#初学者，如高校大学生、求职人员、培训机构学员等。

本书服务

如果您使用本书的过程中遇到问题，可以通过如下方式与我们联系。

☑ 服务QQ：4006751066

☑ 服务网站：http://www.mingribook.com

本书作者

本书由软件开发技术联盟组织编写，参与编写的程序员有赛奎春、王小科、王国辉、王占龙、高春艳、张鑫、杨丽、辛洪郁、周佳星、申小琦、张宝华、葛忠月、王雪、李贺、吕艳妃、王喜平、张领、杨贵发、李根福、刘志铭、宋禹蒙、刘丽艳、刘莉莉、王雨竹、刘红艳、隋光宇、郭鑫、崔佳音、张金辉、王敬洁、宋晶、刘佳、陈英、张磊、张世辉、高茹、陈威、张彦国、高飞、李严。

编　者

目 录

Contents

第 1 篇 C#编程基础篇

第 1 章 C#开发环境的使用 2

1.1 Visual Studio 开发环境的安装与配置 3

实例 001 配置合适的 Visual Studio 2012 开发环境 3

实例 002 设置程序代码行号 4

实例 003 使开发环境全屏显示 5

实例 004 根据需要创建所需解决方案 6

1.2 Visual Studio 开发环境的使用 8

实例 005 为程序设置版本和帮助信息 8

实例 006 设置 Windows 应用程序启动窗体 9

实例 007 统一窗体中控件的字体设置 10

实例 008 通过“格式”菜单布局窗体 10

1.3 快速开发项目必备 11

实例 009 为项目添加 DLL 文件引用 11

实例 010 为项目添加已有类 12

实例 011 为项目添加第三方控件 13

实例 012 为项目添加已有窗体 14

第 2 章 C#语言基础应用 15

2.1 代码的注释 16

实例 013 对单行代码进行注释 16

实例 014 快速对多行代码进行注释 17

实例 015 使用“///”标记给代码段添加说明 18

2.2 运算符的使用 20

实例 016 使用引号运算符进行赋值 20

实例 017 使用算术运算符开发简单计算器 21

实例 018 使用“^”运算符对数字进行加密 22

实例 019 巧用位移运算符获取汉字编码值 24

实例 020 使用条件运算符判断指定年份是不是闰年 25

实例 021 使用流程控制语句报销业务花销 26

2.3 关键字的使用 27

实例 022 使用 checked 关键字处理溢出错误 27

实例 023 使用 typeof 关键字获取类的内部结构 28

实例 024 使用 using 关键字有效回收资源 29

实例 025 使用 is 关键字检查对象是否与给定类型兼容 30

实例 026 使用 as 关键字将对象转换为指定类型 32

2.4 流程控制语句的使用 34

实例 027 小明去学校和医院分别要走哪条路 34

实例 028 利用条件语句判断用户登录身份 35

实例 029 判断指定月份属于哪个季节 36

实例 030 使用 switch 语句更改窗体颜色 37

实例 031 循环向控制台中输入内容 38

实例 032 递归算法的经典面试题 39

实例 033 制作一个数字猜猜看小游戏 40

实例 034 使用 goto 语句在数组中搜索指定图书 42

第 3 章 字符串处理技术 44

3.1 字符及字符串转换 45

实例 035 将字母全部转换为大写或小写 45

实例 036 字母与 ASCII 码的转换 46

实例 037 汉字与区位码的转换 48

实例 038 将汉字转换为拼音 49

3.2 常用字符及字符串操作 50

实例 039 根据标点符号对字符串进行分行 50

实例 040 判断用户输入的用户名是否正确 51

实例 041 将字符串的每个字符进行颠倒输出 52

实例 042 去掉字符串中的所有空格 54

实例 043 从字符串中分离文件路径、文件名及扩展名 55

实例 044 获取字符串中汉字的个数 57

实例 045 批量替换某一类字符串 58

实例 046 对字符串进行加密与解密 59

3.3 常用数字处理技术 61

实例 047　判断输入的货币值是否为数字 61
实例 048　对计算结果进行四舍五入 62
实例 049　商品金额的大小写转换 63
实例 050　设置货币值中使用的小数位数 64
实例 051　格式化输入数据为货币格式 65
实例 052　开发一个进制转换器 66
3.4　日期时间格式的数据处理 67
实例 053　动态获得系统当前日期和时间 67
实例 054　手动设置系统日期时间 69
实例 055　根据生日自动计算员工年龄 70
实例 056　根据年份判断十二生肖 71
实例 057　获取当前日期是星期几 72
实例 058　获取当前年的天数 73
实例 059　获取当前月的天数 74
实例 060　取得当前日期的前一天 75
实例 061　将日期格式化为指定格式 76
实例 062　使用 DateDiff 方法获取日期时间的间隔数 77
实例 063　使用 DateAdd 方法向指定日期添加一段时间间隔 78
实例 064　使用 TimeSpan 对象获取时间间隔 80
实例 065　使用 Sleep 方法延迟时间 81
实例 066　如何确定程序的运行时间 82
实例 067　使用 ParseExact 方法将字符串转化为日期格式 84
实例 068　使用 ToString 方法格式化日期 85
实例 069　使用 ToDateTime 方法转换日期显示格式 86
3.5　常用数字验证技巧 87
实例 070　使用正则表达式验证电话号码 87
实例 071　使用正则表达式验证输入密码条件 88
实例 072　使用正则表达式验证邮政编号 89
实例 073　使用正则表达式验证手机号 89
实例 074　使用正则表达式验证身份证号 90
实例 075　使用正则表达式验证两位小数 91
实例 076　使用正则表达式验证一年的 12 个月份 92
实例 077　使用正则表达式验证一个月的 31 天 93
实例 078　使用正则表达式验证数字输入 94
实例 079　使用正则表达式验证密码长度 95
实例 080　使用正则表达式验证非零的正整数 96
实例 081　使用正则表达式验证非零的负整数 97
3.6　常用字符验证技巧 98
实例 082　使用正则表达式验证大写字母 98
实例 083　使用正则表达式验证小写字母 99
实例 084　使用正则表达式检查字符串中重复出现的词 99
实例 085　使用正则表达式替换字符串 101
实例 086　使用正则表达式拆分字符串 102
实例 087　使用正则表达式验证输入字母 102
实例 088　使用正则表达式验证中文汉字输入 103
实例 089　使用正则表达式验证输入字符串 104
3.7　网络验证应用技巧 105
实例 090　使用正则表达式验证 E-mail 格式 105
实例 091　使用正则表达式验证 IP 地址 106
实例 092　使用正则表达式验证 URL 107
第 4 章　数组和集合的使用 109
4.1　数组的基本操作 110
实例 093　获取多维数组的行数与列数 110
实例 094　按指定条件在数组中检索元素 111
实例 095　在数组中添加一个元素 112
实例 096　在数组中添加一个数组 113
实例 097　不改变长度删除数组中的元素 115
实例 098　删除数组元素后改变其长度 116
4.2　常用数组排序算法 117
实例 099　使用选择排序法对一维数组进行排序 117
实例 100　使用冒泡排序法对一维数组进行排序 118
实例 101　使用快速排序法对一维数组进行排序 119
实例 102　使用直接插入法对一维数组进行排序 121
实例 103　使用希尔排序法对一维数组进行排序 122
实例 104　使用 Sort 方法对数组进行快速排序 124
实例 105　反转数组中元素的顺序 125
4.3　常用集合的使用 126
实例 106　向班级集合中添加学生信息 126
实例 107　使用哈希表对 XML 文件进行查询 127
实例 108　计算两个矩形矩阵的乘积 129
第 5 章　面向对象编程技术 131
5.1　属性和方法 132
实例 109　使用属性存储用户编号和姓名 132
实例 110　通过定义方法求一个数的平方 133
实例 111　使用重载方法实现不同类型数据的计算 135

5.2 结构与类 136
实例 112 通过结构计算矩形的面积 136
实例 113 通过类继承计算梯形面积 137
实例 114 封装类实现一个简单的计算器 139
实例 115 使用面向对象思想查找字符串中的所有数字 140
5.3 面向对象技术高级应用 141
实例 116 自定义抽象类计算圆形的面积 141
实例 117 利用接口实现选择不同的语言 143
实例 118 使用密封类密封用户信息 145
实例 119 通过重写虚方法实现加法运算 146
实例 120 通过类的多态性确定人类的说话行为 147
5.4 迭代器和分部类的使用 149
实例 121 使用迭代器显示公交车站点 149
实例 122 使用迭代器实现倒序遍历 150
实例 123 使用迭代器实现文字的动态效果 151
实例 124 使用分部类实现多种计算方法 153
实例 125 使用分部类记录学生信息 155
5.5 泛型的使用 156
实例 126 使用泛型存储不同类型的数据列表 156
实例 127 通过泛型查找不同数组中的值 158
实例 128 通过继承泛型类实现输出学生信息 159
实例 129 通过泛型实现子窗体的不同操作 160
实例 130 使用泛型去掉数组中的重复数字 161
第 6 章 数据结构与算法 163
6.1 数据结构的实现 164
实例 131 单向链表的实现 164
实例 132 双向链表的实现 168
实例 133 堆栈的实现 173
实例 134 队列的实现 175
实例 135 树的实现 177
6.2 常见算法的实际应用 180
实例 136 计算 $1+2^2+3^3+4^4+\cdots+n^n$ 的值 180
实例 137 计算 10！的值 181
实例 138 求最大公约数 181
实例 139 求最小公倍数 182
实例 140 判断素数的算法 183
实例 141 按要求生成指定位数的编号 184
实例 142 身份证号从 15 位升到 18 位的算法 186
实例 143 歌德巴赫猜想的算法实现 187
实例 144 八皇后问题的算法实现 188
实例 145 百钱百鸡的算法实现 190
实例 146 韩信点兵的算法实现 191
实例 147 实现斐波纳契数列求和 192
实例 148 求水仙花数的算法实现 193
实例 149 约瑟夫环问题的算法实现 194
实例 150 C#实现杨辉三角算法 195
实例 151 如何将 B 转换成 GB、MB 和 KB 196
实例 152 0~N 位数的任意组合 197
实例 153 在数组中快速查找近似值 199
实例 154 猴子选大王算法的实现 200
实例 155 使用 MD5 算法对密码进行加密 201
实例 156 凯撒密码算法的实现 202

第 2 篇　Windows 窗体篇

第 7 章 Windows 窗体基础 206
7.1 设置窗体位置 207
实例 157 控制窗体加载时的位置 207
实例 158 设置窗体在屏幕中的位置 208
实例 159 从上次关闭位置启动窗体 209
实例 160 始终在桌面最顶层显示的窗体 210
7.2 设置窗体大小 211
实例 161 限制窗体大小 211
实例 162 根据桌面大小调整窗体大小 212
实例 163 自定义最大化、最小化和关闭按钮 213
实例 164 手动改变自制窗体的大小 215
实例 165 禁止改变窗体的大小 218
7.3 设置窗体的标题栏 218
实例 166 使窗体标题栏文字右对齐 218
实例 167 没有标题栏也可以更改窗体的大小 219
实例 168 设置闪烁的标题栏 220
7.4 设置窗体的背景 221
实例 169 设置窗体背景颜色为淡蓝色 221
实例 170 设置窗体背景为指定图片 222
实例 171 使背景图片自动适应窗体的大小 223

实例 172　使背景图片在窗体中居中显示 224
实例 173　使背景图片在窗体中平铺显示 225
实例 174　清空窗体的背景 226

第 8 章　特色窗体界面 227
8.1　导航菜单界面 228
实例 175　创建类似 OutLook 的窗体 228
实例 176　带导航菜单的主界面 229
实例 177　图形化的导航界面 230
8.2　特殊形状的窗体 232
实例 178　创建非矩形窗体 232
实例 179　建立字体形状窗体 233
实例 180　使控件大小随窗体自动调整 234
实例 181　带分隔栏的窗体 234
8.3　窗体效果 235
实例 182　制作半透明渐显窗体 235
实例 183　使窗体背景色渐变 236
实例 184　随机更换主界面背景 237
实例 185　磁性窗体的设计 238
实例 186　制作鼠标穿透窗体 246
实例 187　窗体换肤程序 247
8.4　窗体动画 250
实例 188　窗体中的滚动字幕 250
实例 189　动画显示窗体 251
实例 190　制作类似 Office 的提示精灵 252
实例 191　动画形式的程序界面 254
8.5　特色程序界面 255
实例 192　仿 QQ 抽屉式窗体 255
实例 193　设计软件启动界面 257
实例 194　以树形显示的程序界面 259
实例 195　动态按钮的窗体界面 260

第 9 章　窗体控制技术 262
9.1　窗体常用操作 263
实例 196　以全屏方式显示程序窗体 263
实例 197　通过子窗体刷新父窗体 264
实例 198　拖动无边框窗体 265
实例 199　禁用窗口上的关闭按钮 267
实例 200　向窗体中拖放图片并显示 268
实例 201　使应用程序在进程中消失 269
9.2　其他窗体技术 270
实例 202　在关闭窗体时弹出确认对话框 270
实例 203　隐藏应用程序在任务栏中的图标 272
实例 204　实现动态系统托盘图标 272
实例 205　实现气泡提示窗口 275
实例 206　从桌面右下角显示的 Popup 窗口提醒 276
实例 207　设置可执行文件的生成图标 278

第 10 章　MDI 窗体和继承窗体 279
10.1　MDI 窗体的使用 280
实例 208　设置窗体为父窗体 280
实例 209　使子窗体最大化显示 281
实例 210　对子窗体进行水平排列 282
实例 211　对子窗体进行垂直排列 283
实例 212　对子窗体进行层叠排列 284
10.2　继承窗体的使用 285
实例 213　使用继承选择器创建继承窗体 285
实例 214　以编程方式创建继承窗体 286
实例 215　修改继承窗体中的控件属性 287

第 3 篇　控件应用篇

第 11 章　Windows 常用控件的使用 290
11.1　按钮控件的使用 291
实例 216　轻松打造绚丽按钮 291
实例 217　给 Button 控件创建快捷键 292
实例 218　使 Button 控件大小自动适应文本长度 293
实例 219　得到鼠标焦点后自动放大的 Button 按钮 294
实例 220　动态创建多个 Button 控件 295
11.2　TextBox 控件应用 296
实例 221　制作多行文本框 296
实例 222　制作密码文本框 297
实例 223　在 TextBox 控件中显示回车符 297
实例 224　只允许输入数字的 TextBox 控件 298
实例 225　在 TextBox 控件底端显示下划线 299
实例 226　屏蔽 TextBox 控件上的粘贴功能 300
实例 227　屏蔽 TextBox 控件上默认的右键菜单 301
11.3　ComboBox 控件应用 302

实例 228　设置 ComboBox 控件的默认选项为第一项 302
实例 229　将数据表中的字段添加到 ComboBox 控件中 303
实例 230　在 ComboBox 下拉列表中显示图片 304
实例 231　用 ComboBox 控件制作浏览器网址输入框 305
实例 232　实现带查询功能的 ComboBox 控件 306
11.4　RichTextBox 控件应用 308
实例 233　在 RichTextBox 控件中添加超链接文字 308
实例 234　在 RichTextBox 控件中插入图片 309
实例 235　在 RichTextBox 控件中显示 RTF 格式的文件 310
实例 236　使用 RichTextBox 控件保存文件 312
实例 237　为 RichTextBox 控件添加自定义滚动条 313
实例 238　在 RichTextBox 控件中实现关键字描红 314
实例 239　在 RichTextBox 控件中替换文本文字 315
实例 240　在 RichTextBox 控件中实现项目符号功能 316
实例 241　设置 RichTextBox 控件中的文本对齐方式 318
11.5　ListBox 控件应用 319
实例 242　在 ListBox 控件中实现选择项功能 319
实例 243　选中 ListBox 控件中全部条目 320
实例 244　对 ListBox 控件中的数据进行排序 321
实例 245　在 ListBox 控件中查找指定项 322
实例 246　将数据库数据添加到组合框中 323
实例 247　在 ListBox 控件间交换数据 324
实例 248　借助绑定控件实现数据选择录入 325
11.6　ListView 控件应用 326
实例 249　ListView 控件间的数据移动 326
实例 250　将数据库数据添加到 ListView 控件 327
实例 251　在 ListView 控件中实现修改功能 328
实例 252　在 ListView 控件中对数据排序 330
实例 253　在 ListView 控件中绘制底纹 331
实例 254　在列表视图中拖动视图项 332
实例 255　使 ListView 控件中的选择项高亮显示 333
实例 256　制作带复选框的 ListView 控件 335
11.7　选择类控件应用 336
实例 257　利用选择控件实现权限设置 336
实例 258　利用选择控件实现复杂查询 337
11.8　TreeView 控件应用 339
实例 259　在 TreeView 控件节点中显示图片 339
实例 260　实现带复选框的 TreeView 控件 340
实例 261　将数据库数据显示到树视图中 341
实例 262　用树形列表动态显示菜单 342
实例 263　用 TreeView 控件遍历磁盘目录 344
实例 264　修改 TreeView 控件的节点文本 345
实例 265　将 XML 文件节点绑定到 TreeView 控件中 346
11.9　选项卡控件的使用 347
实例 266　更改选项卡的显示样式 347
实例 267　添加多个选项卡 348
实例 268　获取选中的选项卡名称 349
实例 269　删除指定的选项卡 350
11.10　菜单应用实例 351
实例 270　为菜单添加快捷键 351
实例 271　设置菜单项是否可用 352
实例 272　将菜单项的字体设置为粗体 353
实例 273　创建级联菜单 354
实例 274　级联菜单的动态合并 355
实例 275　带历史信息的菜单 356
实例 276　像开始菜单一样漂亮的菜单 357
实例 277　制作任务栏托盘菜单 357
实例 278　可以拉伸的菜单 358
实例 279　仿 XP 系统的任务栏菜单 360
11.11　工具栏设计 361
实例 280　带图标的工具栏 361
实例 281　带背景图片的工具栏 362
实例 282　设计浮动工具栏 363
实例 283　带下拉菜单的工具栏 364
实例 284　具有提示功能的工具栏 364
11.12　状态栏设计 365
实例 285　在状态栏中显示复选框 365
实例 286　在状态栏中显示当前系统操作员 366
实例 287　在状态栏中实时显示当前系统时间 367
实例 288　带进度条的状态栏 368

实例 289 制作动画效果的状态栏 369

第 12 章 Windows 高级控件的使用 371

12.1 对话框组件的使用 372

实例 290 获得弹出对话框的相关返回值 372
实例 291 使用 OpenFileDialog 组件打开文件 373
实例 292 设置 OpenFileDialog 组件中只能选择图片文件 374
实例 293 使用 OpenFileDialog 组件打开多个文件 375
实例 294 使用 SaveFileDialog 组件保存文件 376
实例 295 使用 FolderBrowserDialog 组件选择文件夹 377
实例 296 设置 FolderBrowserDialog 组件的默认路径 378
实例 297 使用 FontDialog 组件设置字体 379
实例 298 使用 ColorDialog 组件设置字颜色 380

12.2 Timer 组件的使用 381

实例 299 使用 Timer 组件制作计时器 381
实例 300 使用 Timer 组件显示当前系统时间 382
实例 301 使用 Timer 组件制作左右飘动的窗体 383
实例 302 使用 Timer 组件实现世界杯倒计时 384
实例 303 使用 Timer 组件实现人物动画效果 385

12.3 EventLog 组件 386

实例 304 使用 EventLog 组件读写事件日志 386
实例 305 使用 EventLog 组件保存 Windows 系统日志 388
实例 306 使用 EventLog 组件向本机现有日志中添加条目 389

12.4 其他常用控件的典型应用 391

实例 307 使用 MaskedTextBox 控件实现输入验证 391
实例 308 制作日历计划任务 392
实例 309 在 ProgressBar 控件中显示进度百分比 394
实例 310 使用 BackgroundWorker 组件执行异步操作 395
实例 311 使用 ErrorProvider 组件验证文本框输入 398
实例 312 使用 FileSystemWatcher 组件监视日志文件 399
实例 313 使用 HelpProvider 组件调用帮助文件 401
实例 314 使用 Process 组件访问本地进程 402
实例 315 使用 ServiceController 组件控制计算机服务 403
实例 316 使用 ImageList 组件制作动画图片 406
实例 317 使用 DirectoryEntry 组件建立虚拟目录 407

12.5 常用控件控制技术 409

实例 318 程序运行时智能增减控件 409
实例 319 多控件焦点循环移动 410
实例 320 使用控件的 Tag 属性传递信息 412
实例 321 为控件设置快捷键 413

12.6 控件焦点变换 414

实例 322 按回车键变换控件焦点 414
实例 323 程序运行时拖动控件 415
实例 324 控件得到焦点时变色 417

第 13 章 DataGridView 数据控件 418

13.1 DataGridView 控件基本设置 419

实例 325 设置 DataGridView 控件中网格线的样式 419
实例 326 设置 DataGridView 控件中字体的样式 419
实例 327 在 DataGridView 控件中设置数据显示格式 420
实例 328 设置 DataGridView 单元格的文本对齐方式 421
实例 329 在 DataGridView 控件中验证数据输入 422
实例 330 在 DataGridView 的单元格中启用换行 423
实例 331 禁止 DataGridView 控件中添加和删除行 424
实例 332 禁用 DataGridView 控件列表头自动排序功能 425

13.2 DataGridView 控件实用开发 426

实例 333 对 DataGridView 控件进行数据绑定 426
实例 334 DataGridView 选中单元格时整个行背景变色 427
实例 335 在 DataGridView 控件中隔行换色 429
实例 336 在 DataGridView 控件中显示下拉列表 430
实例 337 在 DataGridView 控件中显示图片 431
实例 338 为 DataGridView 控件实现复选功能 431

实例 339　在 DataGridView 控件中添加“合计”和“平均值” 433
实例 340　使用主从表查看生产单详细信息 434
实例 341　将 DataGridView 中数据导出到 Word 435
实例 342　将 DataGridView 中数据导出到 Excel 437
实例 343　像 Excel 一样复制 DataGridView 中数据 439
实例 344　从 DataGridView 中拖放数据到 TreeView 443

第 14 章　自定义用户控件 447
14.1　重绘 Windows 标准控件 448
实例 345　美化 ComboBox 控件下拉列表 448
实例 346　美化单选按钮控件 449
实例 347　美化复选框控件 453
实例 348　重绘 ListBox 控件 455
14.2　自定义 Windows 控件 457
实例 349　自定义水晶按钮控件 457
实例 350　自制数值文本框组件 461
实例 351　设计带行数和标尺的 RichTextBox 控件 466
实例 352　自制平滑进度条控件 471
实例 353　制作 Vista 风格的日历 473
实例 354　DataGridView 和 BindingSource 复合控件 477

第 4 篇　文件操作篇

第 15 章　文件基本操作 482
15.1　获取文件基本信息 483
实例 355　获取文件的大小 483
实例 356　获取文件扩展名 484
实例 357　获取文件创建时间 485
实例 358　获取文件最后一次修改时间 486
实例 359　获取文件名中禁止使用的字符 487
15.2　文件的创建和删除 488
实例 360　创建和删除文件 488
实例 361　生成随机文件名或文件夹名 489
实例 362　建立临时文件 490
实例 363　根据日期动态建立文件 491
实例 364　清空回收站中的所有文件 492
15.3　查找文件 493
实例 365　检查文件是否存在 493
实例 366　获取文件夹下的所有子文件夹及文件的名称 494
实例 367　搜索文件 496
15.4　修改文件 497
实例 368　修改文件属性 497
实例 369　修改文件及目录的名字 498
实例 370　将长文件名转换成短文件名 499
实例 371　文件批量更名 500
15.5　复制文件 502
实例 372　使用 FileStream 复制大文件 502
实例 373　复制文件时显示复制进度 504
实例 374　批量复制文件 506
15.6　指定类型的文件操作 507
实例 375　使用 C#操作 INI 文件 507
实例 376　使用 C#操作 XML 文件 509
实例 377　创建 PDF 文档 513
15.7　其他 514
实例 378　获取窗口文本 514
实例 379　判断文件是否正在被使用 516
实例 380　C#中实现文件拖放 517
实例 381　根据内容对文件进行比较 518
实例 382　获取文件夹中的图标资源 519
实例 383　将一个文件分割为多个小文件 521
实例 384　将多个小文件合并为一个文件 523

第 16 章　文件夹基本操作 525
16.1　获取文件夹信息 526
实例 385　获得文件夹创建时间 526
实例 386　获取临时文件目录 527
实例 387　获取应用程序所在目录 527
实例 388　得到系统当前目录 528
实例 389　获取路径名禁止使用的字符 529
16.2　文件夹操作 530
实例 390　判断指定文件夹是否存在 530
实例 391　创建文件夹 531

实例 392 以当前日期时间为根据创建文件夹 532
实例 393 移动文件夹 533
实例 394 删除文件夹 533
实例 395 获取所有逻辑磁盘目录 534
实例 396 获取指定文件夹的上级目录 537
实例 397 使用递归法删除文件夹中的所有文件 538
实例 398 对指定文件夹中的文件进行分类存储 539
实例 399 伪装文件夹 540

第 17 章 文件流操作 544
17.1 文件的读取和写入 545
实例 400 创建一个文件用于写入 UTF-8 编码的文本 545
实例 401 OpenRead 方法打开现有文件并读取 546
实例 402 OpenWrite 方法打开现有文件并进行写入 547
实例 403 打开现有 UTF-8 编码文本文件并进行读取 549
实例 404 读取文件中的第一行数据 550
实例 405 按行读取文本文件中的数据 550
实例 406 读取文件中的所有数据 551
实例 407 向文件中写入追加数据 552
17.2 常用的文件流操作 553
实例 408 将文本文件转换成网页文件 553
实例 409 读写内存流数据 555
实例 410 创建并写入二进制文件数据 556
实例 411 读取二进制文件中的内容 558
实例 412 使用缓冲流复制文件 559
实例 413 解析只有一种格式的文本文件 560
实例 414 解析含有多种格式的文本文件 562

第 18 章 加密、解密及解压缩文件 565
18.1 加密与解密文件 566
实例 415 对文本文件进行加密与解密 566
实例 416 利用图片对文件进行加密与解密 569
实例 417 使用 ROT13 算法加密解密文件 571
实例 418 使用对称算法加密、解密文件 573
实例 419 使用口令加密可执行文件 575
18.2 压缩和解压缩文件 579
实例 420 使用 GZip 压缩文件 579
实例 421 使用 GZip 解压文件 580
实例 422 使用 WinRAR 压缩文件 582
实例 423 使用 WinRAR 解压文件 583
实例 424 批量压缩和解压缩文件 585

第 5 篇 C#与 Office 高效开发篇

第 19 章 C#与 Word 互操作 590
19.1 Word 文档基本操作 591
实例 425 将 Word 文档嵌入到 WinForm 窗体中 591
实例 426 创建 Word 文档 592
实例 427 向 Word 文档中写入文字 594
实例 428 在 Word 文档中插入图片 595
实例 429 在 Word 文档中绘制表格 597
实例 430 向 Word 文档中插入 Windows 控件 599
实例 431 设置 Word 文档中的字体样式 600
实例 432 统计 Word 文档中的字符数 602
实例 433 将一个 Word 文档分割为多个小的 Word 文档 604
实例 434 将多个 Word 文档合并为一个 Word 文档 606
实例 435 批量替换 Word 文档中指定的字符串 609
19.2 读取外部数据到 Word 中 612
实例 436 读取文本文件到 Word 中 612
实例 437 读取多个文本文件到同一 Word 文档中 613
实例 438 读取 Access 数据到 Word 并对数据列进行计算 615
实例 439 将 SQL Server 数据查询结果输出到 Word 618
实例 440 将 XML 中的数据读取到 Word 文档中 620
19.3 将 Word 中的数据进行输出处理 622
实例 441 读取 Word 数据到文本文件 622
实例 442 将 Word 中的数据分解到多个文本文件中 624
实例 443 读取 Word 文档中表格数据到 Access 数据库 626

实例 444　读取 Word 文档中表格数据到 SQL Server 数据库 629
实例 445　实时读取 Word 文档中表格数据到 SQL Server 数据库 630
实例 446　每天定时读取 Word 文档中表格数据给指定数据库 632
实例 447　将 Word 文档转换为 HTML 网页 634
实例 448　在打印预览中显示 Word 文档 637
实例 449　提取 Word 文档中的目录 638
实例 450　在 Word 文档中建立小型应用程序 640

第 20 章　高效应用 Excel 643

20.1　Excel 基本操作 644
实例 451　将 Excel 文件嵌入到 WinForm 窗体中 644
实例 452　创建 Excel 文件并保存 645
实例 453　为 Excel 工作簿设置和清除密码 646
实例 454　为一个 Excel 文件创建多个工作表 648
实例 455　删除 Excel 文件中指定的工作表 650

20.2　读取外部数据到 Excel 651
实例 456　读取文本文件到 Excel 并对数据列进行格式化 651
实例 457　将文本文件数据分解到 Excel 中的不同数据表 653
实例 458　读取多个 Word 文档文件到 Excel 同一数据表 654
实例 459　读取 Access 数据到 Excel 658
实例 460　将 SQL Server 数据查询结果输出到 Excel 659
实例 461　实时读取不同数据到 Excel 进行汇总处理 661
实例 462　每天定时将各地上报结果处理到 Excel 664

20.3　将 Excel 数据进行输出处理 667
实例 463　读取 Excel 指定表数据到文本文件 667
实例 464　将 Excel 数据分解到多个文本文件 668
实例 465　读取多个 Excel 文件数据到 Word 文档 669
实例 466　读取 Excel 查询结果到 Access 数据库 672
实例 467　将多个 Excel 文件输出到同一个 SQL Server 数据库 674
实例 468　实时读取 Excel 数据到 SQL Server 数据库 677
实例 469　批量读取 Excel 文件给指定数据库 678
实例 470　每天定时读取 Excel 文件给指定数据库 680

20.4　利用 Excel 进行数据挖掘 683
实例 471　在 Excel 的指定范围内搜索文本 683
实例 472　利用 Excel 对数据进行多列排序 685
实例 473　利用 Excel 将数据生成图表 688
实例 474　利用 Excel 对数据进行多表计算 690
实例 475　利用 Excel 对数据进行格式转换 693
实例 476　将多个 Excel 文件汇总到一个 Excel 文件 694
实例 477　将多个 Excel 文件进行自动汇总 696
实例 478　自动用递增变化的数据填充 Excel 697

第 6 篇　图形图像与多媒体篇

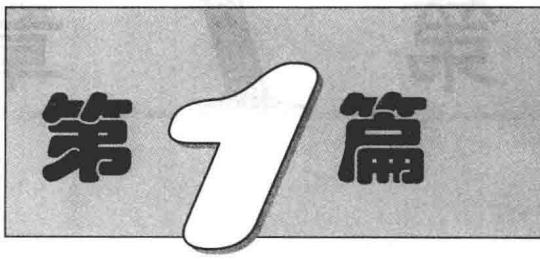
第 21 章　基本图形绘制 702

21.1　绘制基本图形 703
实例 479　绘制直线 703
实例 480　绘制矩形 704
实例 481　绘制正方形 705
实例 482　绘制椭圆 706
实例 483　绘制圆弧 707
实例 484　绘制指定角度的扇形 708
实例 485　绘制贝塞尔曲线 709
实例 486　绘制多边形 710
实例 487　绘制文本 711

21.2　绘制组合图形 712
实例 488　简单画图程序 712
实例 489　绘制公章 716
实例 490　波形图的绘制 717
实例 491　绘制图形验证码 718
实例 492　绘制中文验证码 719
实例 493　使用双缓冲技术绘图 721

第 22 章　图像处理技术 723
22.1　图像格式转换 724
实例 494　BMP 转换为 JPG 格式 724
实例 495　JPG 转换成 BMP 格式 725
实例 496　位图转化为 WMF 726
实例 497　ICO 文件转化为位图 727
实例 498　GIF 动画与 JPG 图片的转换 728
实例 499　批量图像格式转换 729
22.2　图像预览 732
实例 500　局部图像放大 732
实例 501　浏览大图片 733
实例 502　剪切图片 734
实例 503　鼠标拖拽图像 736
22.3　图像的缩放与变换 737
实例 504　放大和缩小图像 737
实例 505　生成图片缩略图 738
实例 506　旋转 JPG 图像 739
实例 507　图像的翻转 741
实例 508　以任意角度旋转图像 741
实例 509　以椭圆形显示图像 743
22.4　图像的特殊效果 744
实例 510　百叶窗效果显示图像 744
实例 511　推拉效果显示图像 745
实例 512　水平交错效果显示图像 746
实例 513　垂直交错效果显示图像 748
实例 514　纹理效果显示图像 749
实例 515　浮雕效果显示图像 750
实例 516　底片效果显示图像 752
实例 517　积木效果显示图像 753
实例 518　马赛克效果显示图像 754
实例 519　柔化效果显示图像 755
实例 520　雾化效果显示图像 757
实例 521　锐化效果显示图像 758
实例 522　黑白效果显示图像 759
实例 523　光晕效果显示图像 760
实例 524　分块效果显示图像 761
实例 525　油画效果显示图像 762
实例 526　四周扩散形式显示图像 764
22.5　特殊效果的文字 765
实例 527　倒影效果的文字 765
实例 528　投影效果的文字 766
实例 529　印版效果的文字 767
实例 530　阴影效果的文字 768
实例 531　倾斜效果的文字 769
实例 532　渐变效果的文字 770
实例 533　缩放效果的文字 772
实例 534　辉光效果的文字 773
22.6　图像识别 775
实例 535　查看图片的像素 775
实例 536　获取图片类型 776
实例 537　获取指定点的 RGB 值 777
实例 538　设置图像中指定位置的像素值 778
实例 539　在图像文件中实现自定义标记 780
22.7　常用的图像工具 782
实例 540　屏幕抓图 782
实例 541　抓取网站整页面 784
实例 542　屏幕颜色拾取器 787
实例 543　不失真压缩图片 789
22.8　图像应用 792
实例 544　随机更换壁纸程序 792
实例 545　制作屏幕保护程序 793
实例 546　为数码照片添加日期 796
实例 547　批量添加图片水印 797
实例 548　仿 QQ 截图 801
实例 549　模拟石英钟 803
实例 550　制作画桃花小游戏 805
实例 551　打造自己的开心农场 806
第 23 章　常用图表应用 810
23.1　简单图表的绘制 811
实例 552　绘制柱形图 811
实例 553　绘制面形图 812
23.2　柱形图表 813
实例 554　利用柱形图分析汇总数据 813
实例 555　利用柱形图表分析商品走势 815
实例 556　对排序数据进行分析 817
实例 557　利用控件实现柱形图分析 818
实例 558　在柱形图的指定位置显示说明文字 819
23.3　折线图表 821
实例 559　利用折线图分析产品销售走势 821
实例 560　利用折线图分析彩票中奖情况 823

实例 561　多曲线数据分析 825
实例 562　网站人气指数曲线分析 828
23.4　饼型图表 830
实例 563　利用饼型图分析公司男女比率 830
实例 564　利用饼型图分析产品市场占有率 831
实例 565　利用多饼型图分析企业人力资源情况 832
实例 566　在饼型图的外围显示说明文字 834
实例 567　制作一个可以旋转的饼型图 838
第 24 章　动画处理技术 840
24.1　图像动画 841
实例 568　随鼠标移动的图像 841
实例 569　十字光标定位 842
实例 570　抓取鼠标形状 843
实例 571　图片自动播放 844
24.2　多媒体动画的制作 846
实例 572　利用 Image 制作小动画 846
实例 573　制作家庭影集 847
实例 574　制作 AVI 播放器 848
实例 575　制作电影特效效果 850
24.3　多媒体动画的播放 851
实例 576　播放 GIF 动画 851
实例 577　播放 Flash 动画 853
实例 578　播放 FLV 文件 855
实例 579　产品电子报价 857
实例 580　产品滚动展示程序 859
第 25 章　音频与视频控制 861
25.1　CD、VCD 播放 862
实例 581　播放指定的 avi-mid-wav 文件 862
实例 582　获取多媒体详细信息列表 863
25.2　MP3、WAV 播放 864
实例 583　获取 MP3 文件的播放时间 864
实例 584　多功能 MP3 播放器 866
实例 585　获取 MP3 文件的歌词 868
实例 586　学校体操定时音乐播放 870
实例 587　M3U 歌词文件的创建及删除 871
实例 588　异步加载并播放声音文件 874
25.3　媒体控制 875
实例 589　获取声音设备的名称及 PNPDeviceID 875
实例 590　检测是否安装声卡 876
实例 591　打开和关闭 CDROM 877
实例 592　控制 PC 喇叭发声 878
实例 593　获取显示设备的名称及 PNPDeviceID 880
实例 594　获取显示设备的最大、最小及当前刷新率 880
实例 595　获取显示设备的当前显示模式 881
实例 596　收听网络电台 882
25.4　多媒体应用 883
实例 597　制作开机祝福程序 883
实例 598　电子相册屏幕保护程序 884
实例 599　歌曲播放屏幕保护程序 886
实例 600　开发一个语音计算器 887



C#编程基础篇

- 第 1 章　C#开发环境的使用
- 第 2 章　C#语言基础应用
- 第 3 章　字符串处理技术
- 第 4 章　数组和集合的使用
- 第 5 章　面向对象编程技术
- 第 6 章　数据结构与算法

第 1 章

C#开发环境的使用

- Visual Studio 开发环境的安装与配置
- Visual Studio 开发环境的使用
- 快速开发项目必备

1.1　Visual Studio 开发环境的安装与配置

实例 001	配置合适的 Visual Studio 2012 开发环境	初级
	光盘位置：光盘\MR\01\001	趣味指数：★★★★☆

实例说明

作为程序开发人员，每个人都有自己的专长，如针对 Windows 应用程序开发、针对 Web 网站开发等，Visual Studio 2012 开发环境安装完成之后，开发人员需要配置合适的开发环境。例如，Visual Studio 2012 开发环境就针对 C#应用程序和 Web 网站开发出两种不同的菜单表现形式，如图 1.1 和图 1.2 所示。

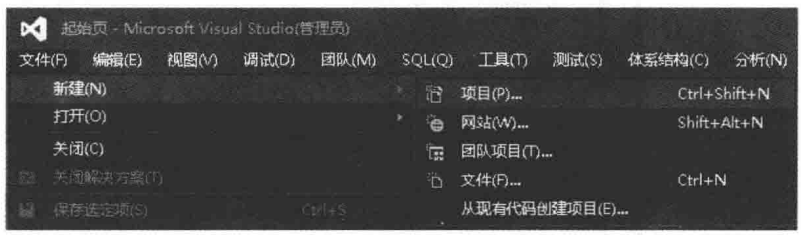


图 1.1　C#应用程序开发的菜单

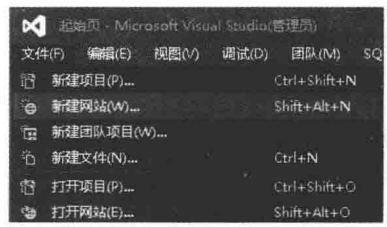


图 1.2　Web 网站开发的菜单

设计过程

（1）安装完 Visual Studio 2012 开发环境后，第一次运行，弹出“选择默认环境设置”对话框，如图 1.3 所示。

（2）在此对话框中，用户可以根据实际情况选择适合自己的开发环境设置，然后单击“退出 Visual Studio”按钮即可。

（3）如果 Visual Studio 2012 开发环境不是第一次运行，用户可以在 Visual Studio 2012 开发环境的菜单栏中选择“工具”/“导入和导出设置”命令，弹出如图 1.4 所示的“导入和导出设置向导”对话框。

图 1.3　“选择默认环境设置”对话框

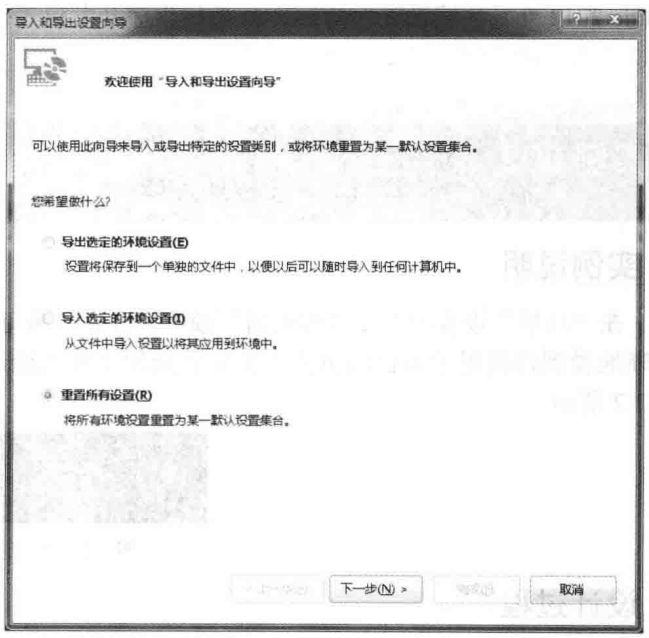


图 1.4　“导入和导出设置向导”对话框

（4）选中“重置所有设置”单选按钮后，单击“下一步”按钮，进入如图 1.5 所示的“保存当前设置”界面。

（5）选中“否，仅重置设置，从而覆盖我的当前设置”单选按钮后，单击“下一步”按钮，进入如图 1.6 所示的“选择一个默认设置集合”界面。

图 1.5 “保存当前设置”界面

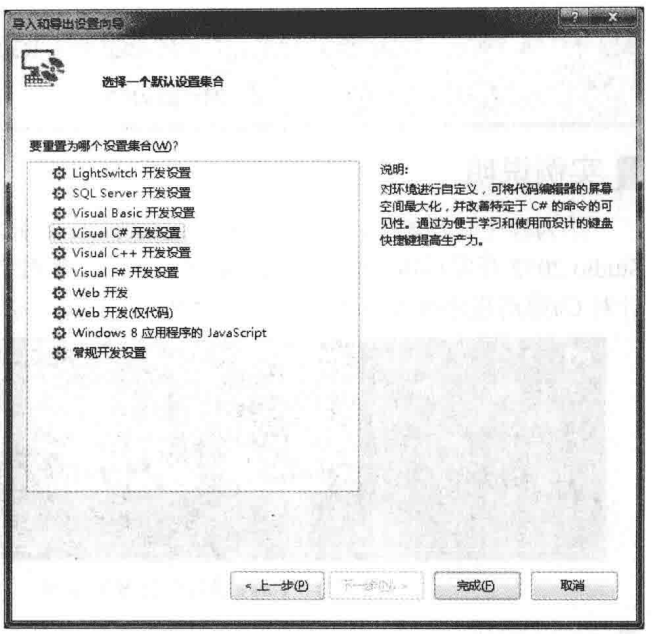


图 1.6 “选择一个默认设置集合”界面

（6）在此界面中用户可以根据实际情况选择适合自己的开发环境设置，最后单击“完成”按钮即可。

秘笈心法

心法领悟 001：使用最适合自己的开发环境。

开发人员在开发程序时，设置开发环境是最基本的一个环节。例如，针对 C/S 模式开发的人员，需要将 Visual Studio 2012 开发环境设置为 Windows 应用程序开发模式；而针对 B/S 模式开发的人员，则需要将 Visual Studio 2012 开发环境设置为 Web 网站开发模式。

实例 002 设置程序代码行号

光盘位置：光盘\MR\01\002　　初级　　趣味指数：★★★★

实例说明

在“环境”设置的“文本编辑器”项中，开发人员可以设置后台代码的行号显示功能，应用此功能，可以清晰地看到代码置于后台编辑器中的位置及程序发生错误时错误代码的位置。设置完行号之后的程序代码如图 1.7 所示。

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Windows.Forms;
```

图 1.7 显示代码行号

设计过程

（1）打开 Visual Studio 2012 开发环境，在菜单栏中选择“工具”/“选项”命令，弹出如图 1.8 所示的“选

项”对话框。

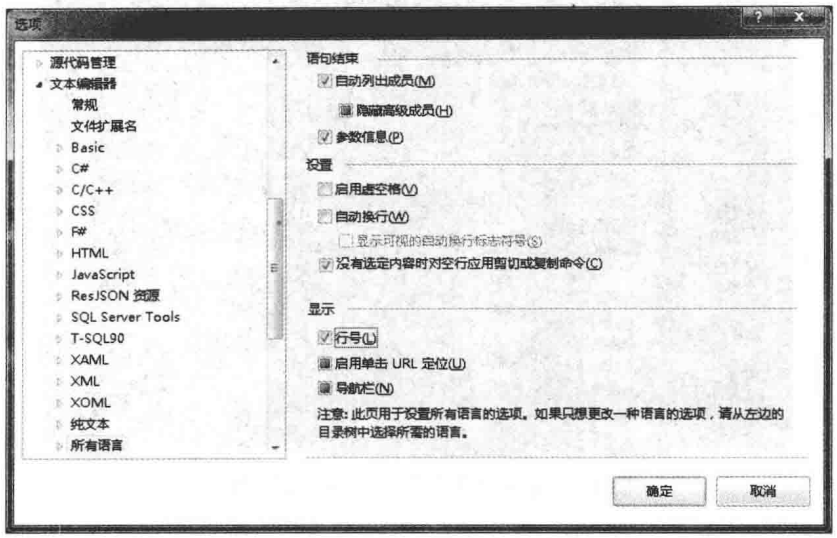


图 1.8　“选项”对话框

（2）选择“文本编辑器”节点下的“所有语言”选项，在右侧选中“行号”复选框，再单击“确定”按钮即可。

说明：图 1.8 中选择了“所有语言”选项，如果读者只想针对某一种语言的代码显示行号，也可以选择 C#、Basic 或者 XML 等选项。

秘笈心法

心法领悟 002：快速找到程序的出错位置。

在 Visual Studio 2012 开发环境中设置完显示代码行号后，如果在调试程序过程中出现错误，可以通过代码行号快速地找到程序出错的位置。

实例 003　使开发环境全屏显示

光盘位置：光盘\MR\01\003　　初级　　趣味指数：★★★☆

实例说明

为了在编码时能更清楚地显示代码，同时能够一次浏览更多的代码，可以将 Visual Studio 2012 开发环境全屏显示。

设计过程

（1）打开 Visual Studio 2012 开发环境。

（2）在菜单栏中选择“视图”/“全屏显示”命令（如图 1.9 所示），或者直接按 Shift+Alt+Enter 组合键即可。

秘笈心法

心法领悟 003：使用快捷键切换 Visual Studio 2012 开发环境全屏状态。

Visual Studio 2012 开发环境全屏显示之后，开发人员可以通过按 Shift+Alt+Enter 组合键退出全屏。另外，还可以在正常状态下，通过按 Shift+Alt+Enter 组合键使 Visual Studio 2012 开发环境全屏显示。

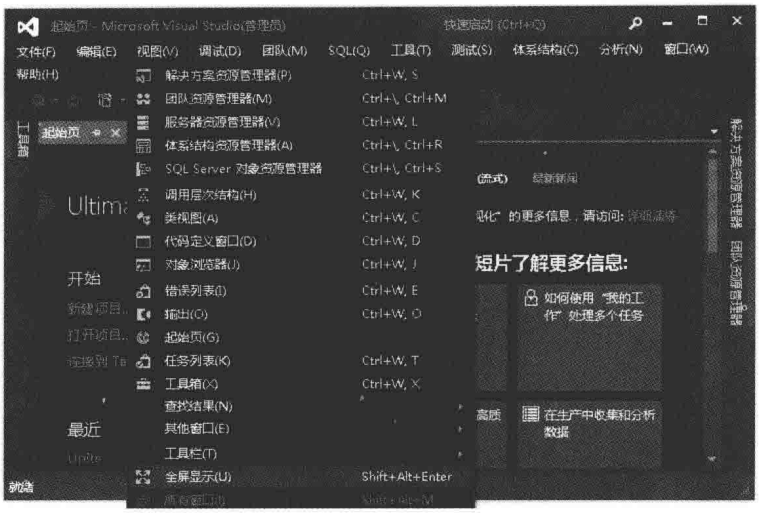


图 1.9　选择“视图”/“全屏显示”命令

实例 004　根据需要创建所需解决方案

光盘位置：光盘\MR\01\004

中级

趣味指数：★★★★☆

实例说明

Visual Studio 2012 是一套完整的开发工具集，用于生成 Windows 窗体应用程序、ASP.NET Web 应用程序、XML Web Services 和移动应用程序，它提供了在设计、开发、调试和部署 Windows 应用程序、Web 应用程序、XML Web Services 和传统的客户端应用程序时所需的工具。本实例将分别演示如何使用 Visual Studio 2012 开发环境创建控制台应用程序、Windows 窗体应用程序和 Web 网站应用程序。

关键技术

本实例主要演示如何使用 Visual Studio 2012 开发环境创建控制台应用程序、Windows 窗体应用程序和 Web 网站应用程序，那么首先需要知道控制台应用程序、Windows 窗体应用程序和 Web 网站应用程序分别是什么。

（1）控制台应用程序

控制台应用程序是 Windows 系统组件的一部分，它通常以 DOS 窗口形式进行输出。

（2）Windows 窗体应用程序

Windows 窗体应用程序是指可以在 Windows 平台上运行的所有程序。例如，开发人员经常用的 C#编程词典软件就是 Windows 窗体应用程序。

（3）Web 网站应用程序

Web 网站应用程序是指可以在 Internet 上运行的所有程序，它以网站形式输出。例如，开发人员经常访问的明日科技图书网（http://www.mingribook.com）就是 Web 网站应用程序。

设计过程

（1）打开 Visual Studio 2012 开发环境，在菜单栏中选择“文件”/“新建”命令，可以看到允许新建的项，这里有项目、网站和文件 3 个选项，如图 1.10 所示。

（2）开发人员可以根据自己的需要选择要新建的项，如果要创建控制台应用程序或者 Windows 窗体应用程序，则选择“项目”命令，弹出如图 1.11 所示的“新建项目”对话框。在该对话框中选择要创建的项目模板，如控制台应用程序或者 Windows 窗体应用程序，然后单击“确定”按钮即可。

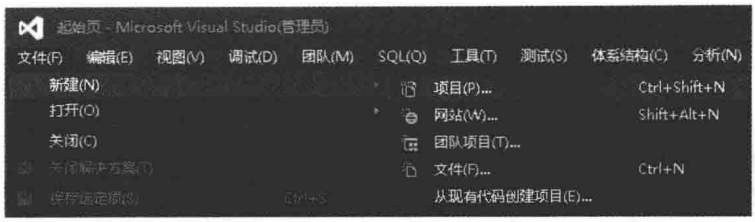


图 1.10　选择“新建”命令

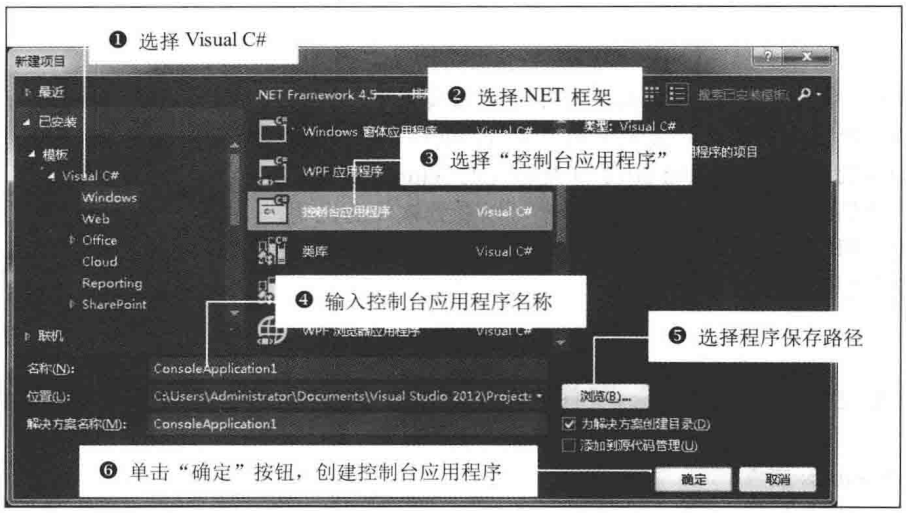


图 1.11　“新建项目”对话框

（3）如果要创建 Web 网站应用程序，则选择“网站”命令，弹出如图 1.12 所示的“新建网站”对话框。在该对话框中选择“ASP.NET 网站”或者“ASP.NET 空网站”选项后，单击“确定”按钮，即可创建一个 Web 网站应用程序。

图 1.12　“新建网站”对话框

秘笈心法

心法领悟 004：C/S 模式和 B/S 模式的区别。

C/S 模式，即 Windows 窗体应用程序，它主要运行在客户端上，如 C#编程词典软件；而 B/S 模式是指网站应用程序，它需要在服务器上进行架设，然后在客户端通过浏览器来进行访问，如明日编程词典网站（http://www.mrbccd.com）。

1.2 Visual Studio 开发环境的使用

实例 005	为程序设置版本和帮助信息	中级
	光盘位置：光盘\MR\01\005	趣味指数：★★★★

实例说明

在计算机中使用一款软件时，经常会在软件中看到其版本、所属公司等信息。例如，C#程序开发人员经常使用的 Visual Studio 2012 开发环境，当用户将鼠标移动到 Visual Studio 2012 开发环境的 setup.exe 安装文件上时，会弹出一个信息提示框，在该提示框中可以看到软件的说明、公司、版本、创建日期以及大小等信息，如图 1.13 所示。本实例将详细讲解如何为程序设置版本及其帮助等信息。

图 1.13 在信息提示框中显示软件版本等信息

关键技术

本实例主要是通过在 AssemblyInfo.cs 文件中设置程序集的相关属性来实现的，下面对本实例用到的关键技术进行详细讲解。

（1）AssemblyInfo.cs 文件

该文件是创建项目时自动生成的一个文件，它包含一些常用的信息，主要用来给当前程序集添加一些属性标记，如版本号等。

技巧：不建议开发人员手动编写 AssemblyInfo.cs 文件，可以在其自动生成之后修改其中的代码。

（2）AssemblyTitle 属性

该属性定义程序集清单的程序集标题自定义属性，其构造函数的语法格式如下：

```
public AssemblyTitleAttribute(string title)
```

参数说明

title：程序集标题。

（3）AssemblyCompany 属性

该属性定义程序集清单的公司名称自定义属性，其构造函数的语法格式如下：

```
public AssemblyCompanyAttribute(string company)
```

参数说明

company：公司名称信息。

（4）AssemblyVersion 属性

该属性指定正在属性化的程序集的版本，其构造函数的语法格式如下：

```
public AssemblyVersionAttribute(string version)
```

参数说明

version：属性化程序集的版本号，该版本号由 4 个值组成，分别为主版本、次版本、内部版本号和修订号。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetSoftInfo。

（2）在“解决方案资源管理器”面板中找到 AssemblyInfo.cs 文件，双击打开该文件，这时即可为程序设置版本和帮助等信息，具体实现代码如下：

```
//有关程序集的常规信息通过以下
//特性集控制。更改这些特性值可修改
```

```
//与程序集关联的信息
[assembly: AssemblyTitle("为程序设置版本和帮助信息")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("明日科技")]
[assembly: AssemblyProduct("为程序设置版本和帮助信息")]
[assembly: AssemblyCopyright("Copyright ©  2013")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
//将 ComVisible 设置为 false，使此程序集中的类型
//对 COM 组件不可见。如果需要从 COM 访问此程序集中的类型，
//则将该类型上的 ComVisible 特性设置为 true
[assembly: ComVisible(false)]
//如果此项目向 COM 公开，则下列 GUID 用于类型库的 ID
[assembly: Guid("70716844-b86e-4d8a-893d-342d34206e48")]
//程序集的版本信息由下面 4 个值组成:
//主版本
//次版本
//生成号
//修订号
//可以指定所有这些值，也可以使用“生成号”和“修订号”的默认值，
//方法是按如下所示使用“*”:
//[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

秘笈心法

心法领悟 005：通过设置 AssemblyInfo 文件标识自己的软件著作。

在 AssemblyInfo 文件中有一个 AssemblyCompany 属性，通过该属性可以设置开发该软件的作者名称，以便作者更好地维护自己的软件著作权。

实例 006	设置 Windows 应用程序启动窗体	中级
	光盘位置：光盘\MR\01\006	趣味指数：★★★★

实例说明

一个完整的 Windows 应用程序中一般都有多个窗体，如果要调试程序，必须要设置首先运行的窗体，这时就需要设置项目的启动窗体。本实例将详细讲解如何设置 Windows 应用程序的启动窗体。

关键技术

本实例主要用到了 Application 类的 Run 方法，下面进行详细讲解。

Application 类提供了 static 方法和属性，以管理应用程序，如启动和停止应用程序、处理 Windows 消息的方法和获取应用程序信息的属性等，其 Run 方法用来在当前线程上开始运行标准应用程序消息循环，该方法为可重载方法，它有 3 种重载形式，分别如下：

```
public static void Run()
public static void Run(ApplicationContext context)
public static void Run(Form mainForm)
```

参数说明

❶ context：一个 ApplicationContext，应用程序将在其中运行。

❷ mainForm：一个 Form，它代表要使之可见的窗体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetStartForm。

（2）在该项目中添加一个 Form2 窗体，以便将其设置为启动窗体。

（3）在“解决方案资源管理器”面板中找到Program.cs文件，双击打开该文件，这时即可通过修改Application.Run方法中的参数来随意设置启动窗体。例如，本实例设置Windows应用程序的启动窗体为Form2，格式如下：

```
Application.Run(new Form2());
```

说明：Program.cs文件是创建Windows窗体应用程序时自动生成的一个文件，它是程序的主程序文件。

秘笈心法

心法领悟006：Main方法的使用。

Main方法是C#程序的主入口点，每个C#程序都包含一个Main方法，Windows窗体应用程序中的Main方法存在于Program.cs文件中，在该方法中，可以编写程序启动时需要用到的C#代码。

实例007 统一窗体中控件的字体设置

光盘位置：光盘\MR\01\007　　中级　　趣味指数：★★★☆

实例说明

当窗体上有很多控件时，如果逐个设置字体属性，会非常繁琐，这时可以将字体属性设置一致的控件选中进行统一设置，这样可以大大节省开发程序的时间。实例运行效果如图1.14所示。

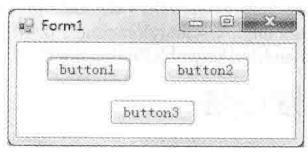


图1.14　统一窗体中控件的字体设置

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为SetFont。

（2）在默认窗体Form1中添加3个Button控件。

（3）在Form1窗体中选中添加的3个Button控件，然后单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，再在弹出的“属性”窗口中对其字体进行统一设置。这里将字体设置为“宋体”，字体大小设置为10，字体颜色设置为Red，如图1.15所示。

图1.15　“属性”窗口

秘笈心法

心法领悟007：更快地提高程序开发效率。

在开发程序时，开发效率是非常重要的一个指标，使用Visual Studio 2012开发环境中的“属性”窗口可以对窗体中的控件属性进行统一设置，从而更快地提高程序的开发效率。

实例008 通过“格式”菜单布局窗体

光盘位置：光盘\MR\01\008　　中级　　趣味指数：★★★☆

实例说明

开发程序时，为了使窗体美观大方，可以通过“格式”菜单来布局窗体上的控件，如可以设置3个Button控件之间的水平间距相同。实例运行效果如图1.16所示。

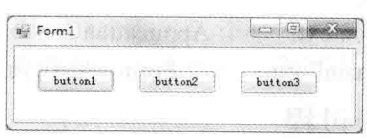


图1.16　设置3个Button控件之间的水平间距相同

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为LayoutForm。

（2）在默认窗体 Form1 中添加 3 个 Button 控件。

（3）在 Form1 窗体中选中添加的 3 个 Button 控件，再在菜单栏中依次选择“格式”/“水平间距”/“相同间隔”命令（如图 1.17 所示），使 3 个 Button 控件之间的水平间距相同。

（4）开发人员还可以通过“格式”菜单在窗体中设置控件的对齐方式、大小以及垂直间距等，其对应的子菜单分别如图 1.18～图 1.20 所示。

图 1.17　选择“格式”/“水平间距”/“相同间隔”命令

图 1.18　设置对齐方式

图 1.19　设置控件大小

图 1.20　设置垂直间距

秘笈心法

心法领悟 008：界面设计在程序开发中的重要性。

软件界面设计不只是懂得高超的艺术设计就可以实现的，还必须有丰富的计算机知识、媒体表示知识和人体工程学知识等。当今的程序开发领域中，.NET 无疑是未来的发展主流，然而在开发软件时，对于开发人员来说，界面排版和代码编写同等重要，界面的好坏直接影响到软件的外观和易用性。

1.3　快速开发项目必备

实例 009	为项目添加 DLL 文件引用	中级
	光盘位置：光盘\MR\01\009	趣味指数：★★★★

实例说明

DLL 文件引用就是在当前项目中引用其他 DLL 文件，可以是用 C#编写的类库，也可以是其他语言编写的类库。这样主要是为了提高程序的开发效率，将 DLL 文件中的一些已有功能直接进行使用，而不用再去编写实现相同功能的代码。本实例在创建完的项目中添加 Word 9.0 引用，添加完该引用之后的效果如图 1.21 所示。

说明：在图 1.21 中可以看到有 Office 和 VBIDE 两个引用，这两个引用是在添加 Word 9.0 引用时自动添加的。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddDLL。

（2）在“解决方案资源管理器”面板中选中项目并单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令，或者在菜单栏中选择“项目”/“添加引用”命令，弹出如图 1.22 所示的“引用管理器”对话框。

（3）在图 1.22 所示的对话框中可以找到要添加的 DLL 文件。本实例中首先选择 COM 节点下的“类型库”

选项，然后选中 Microsoft Word 9.0 Object Library 复选框，单击“确定”按钮，即可将 Word 9.0 引用添加到当前项目中。

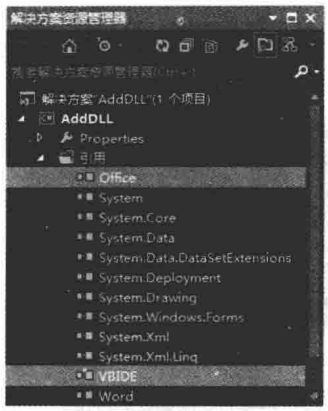


图 1.21 添加完成的 Word 9.0 引用

图 1.22 “引用管理器”对话框

秘笈心法

心法领悟 009：DLL 文件的使用。

DLL 文件是一种最常用的第三方组件表示形式，在 C#中引用 DLL 文件后，可以直接使用其中编写好的方法来实现相应的功能，从而提高程序的开发效率。

实例 010　为项目添加已有类　　中级

光盘位置：光盘\MR\01\010　　趣味指数：★★★★

实例说明

已有类就是用户已经编写好的类，或者他人编写好的类，通过添加已有类可以节省编写相同功能的代码所需的时间，提高程序的开发效率。本实例在创建完的项目中添加已有类 Class1.cs，添加完该类之后的效果如图 1.23 所示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddClass。

（2）在“解决方案资源管理器”面板中选中项目名称，再单击鼠标右键，在弹出的快捷菜单中选择“添加”/“现有项”命令，弹出“添加现有项”对话框，如图 1.24 所示。

图 1.23 添加完成的类文件

图 1.24 “添加现有项”对话框

（3）在图 1.24 所示的对话框中找到要添加的类文件（.cs 类文件）并选中，这里找到 Class1.cs 类文件，再单击“添加”按钮，即可将选中的类文件添加到当前项目中。

秘笈心法

心法领悟 010：什么是已有类？

已有类就是已经编写好的类，类是面向对象编程的核心，它可以封装数据成员、函数成员和其他类等信息，C#中使用 class 关键字来声明类。

实例 011	为项目添加第三方控件 光盘位置：光盘\MR\01\011	中级 趣味指数：★★★★

实例说明

第三方控件可以是开发人员自己开发的一些自定义控件，也可以是第三方厂商开发的一个控件。使用第三方控件主要是为了简化程序步骤，减少程序代码的编写量，从而提高程序的开发效率。本实例在创建完的项目中添加第三方控件 Windows Media Player，添加完该控件之后的工具箱效果如图 1.25 所示。

图 1.25　添加完 Windows Media Player 控件之后的工具箱

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Add3Control。

（2）在“工具箱”窗口中单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令，弹出“选择工具箱项”对话框，如图 1.26 所示。

（3）在图 1.26 所示的对话框中通过切换各个选项卡可以找到要添加的第三方控件。本实例中首先选择“COM 组件”选项卡，然后选中 Windows Media Player 复选框，单击“确定”按钮，即可将 Windows Media Player 控件添加到当前项目的工具箱中，这时从工具箱中拖放该控件到窗体上即可使用。

说明：如果在图 1.26 所示的各个选项卡中没有找到要添加的第三方控件，可以通过单击“浏览”按钮，在打开的“浏览”对话框（如图 1.27 所示）中查找要添加的表示第三方控件的 DLL 文件，然后将其选中，并依次单击“打开”“确定”按钮将选中的第三方控件添加到工具箱中。

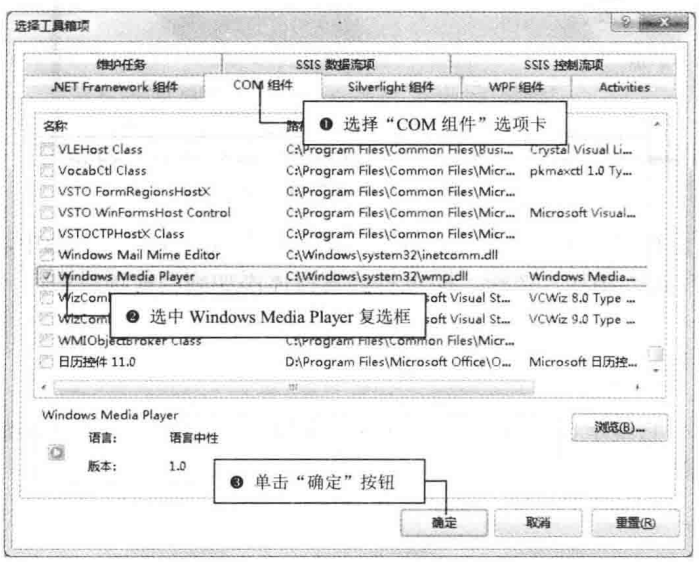


图 1.26　“选择工具箱项”对话框

图 1.27　“浏览”对话框

秘笈心法

心法领悟 011：什么是第三方控件？

C#中的第三方控件是指自定义的控件或者用户控件，它们继承自.NET 类库中的某些基类，并重写或者扩展了一些属性和方法等，从而实现新的功能。使用第三方控件有很大的可定制性，开发人员可以根据需要设置不同的特性，从而使其适应特定项目的需求。

实例 012	为项目添加已有窗体	中级
	光盘位置：光盘\MR\01\012	趣味指数：★★★★

实例说明

已有窗体就是用户或者他人已经编写好的窗体，通过添加已有窗体可以节省编写相同功能的窗体所需的时间，提高程序的开发效率。本实例在创建完的项目中添加一个进销存管理系统的登录窗体，添加该窗体效果如图 1.28 所示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddForm。

（2）在“解决方案资源管理器”面板中选中项目名称，再单击鼠标右键，在弹出的快捷菜单中选择“添加”/“现有项”命令，弹出“添加现有项”对话框，如图 1.29 所示。

图 1.28 添加的进销存管理系统登录窗体

图 1.29 “添加现有项”对话框

（3）在图 1.29 所示的对话框中找到要添加的已有窗体文件（包括窗体的代码文件和设计文件）并选中，这里找到进销存管理系统的登录窗体文件并选中，单击“添加”按钮，即可将已有窗体添加到当前项目中。

秘笈心法

心法领悟 012：认识 WinForm 窗体。

WinForm 窗体是 Windows 窗体应用程序的核心，它继承自 Form 类，开发人员可以在 WinForm 窗体上拖放各种控件，并编写各种事件处理代码，从而实现所需的功能。

第 2 章

C#语言基础应用

- 代码的注释
- 运算符的使用
- 关键字的使用
- 流程控制语句的使用

2.1 代码的注释

实例 013	对单行代码进行注释 光盘位置：光盘\MR\02\013	初级 趣味指数：★★★☆

实例说明

在程序开发过程中，为了方便日后的维护和增强代码的可读性，有必要养成在代码中适当地加入注释内容的良好习惯。在代码的关键位置加入注释，可以帮助理解代码的实现目的与实现方式，使程序工作流程更加清晰明了。单行代码的注释格式如图 2.1 所示。

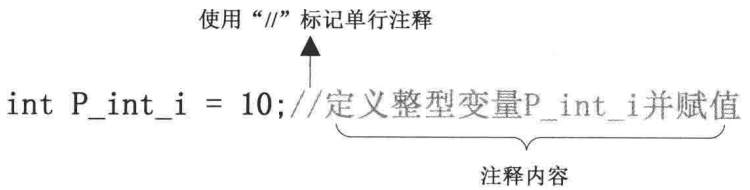


图 2.1 使用“//”注释单行代码

关键技术

本实例重点在于向读者介绍单行代码注释的重要性，以及怎样使用“//”标记注释单行代码。下面对本实例用到的关键技术进行详细讲解。

从图 2.1 中可以看到，使用了“//”标记注释单行代码，如果在某一行中出现“//”标记，那么“//”标记所在行后面的内容均为注释内容。

在程序的编写和调试过程中，经常使用注释暂时屏蔽一部分代码的执行，被注释后的内容将不会被编译器处理和执行。可以使用“//”标记注释多行内容，首先选中多行内容，然后单击 Visual Studio 2012 工具栏中的“注释选中行”按钮，如图 2.2 所示，这样可以注释所有选中的行。

图 2.2 “注释选中行”按钮

同样也可以单击 Visual Studio 2012 工具栏中的“取消对选中行的注释”按钮（如图 2.3 所示），以取消选中行的注释。

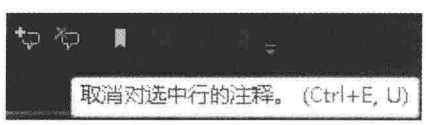


图 2.3 “取消对选中行的注释”按钮

技巧： 前面已经介绍了注释使用的几种方法，现在介绍一个小技巧，通过键盘快捷键对单行或多行代码标记“//”注释，然而根据“键盘映射方案”的不同，快捷键也会有所不同。设置“键盘映射方案”的步骤如下：在 Visual Studio 2012 中选择“工具”/“选项”/“环境”/“键盘”选项（如图 2.4 所示），设置“键盘映射方案”为 Visual C# 2005，单击“确定”按钮保存设置。现在可以按 Ctrl+E+C 组合键注释选中的行，按 Ctrl+E+U 组合键取消选中行的注释。

图 2.4　设置“键盘映射方案”

设计过程

（1）单行代码注释的使用非常简单，打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 Single Commentate。

（2）程序的主要代码如下：

```
static void Main()                                          //入口方法
{
    int P_int_i = 10;                                       //定义整型变量 P_int_i 并赋值
    int P_int_i2 = 20;                                      //定义整型变量 P_int_i2 并赋值
    System.Console.WriteLine(P_int_i + P_int_i2);           //输出计算结果
    System.Console.ReadLine();                              //等待回车继续
}
```

秘笈心法

心法领悟 013：方便地使用注释。

本书中程序代码注释的覆盖率是很高的，这样有助于读者理解程序代码的意图。由此可以想到，一个健康、优秀的程序中，清晰明了的注释是必不可少的，而且在代码编写过程中，应当养成添加注释的好习惯。

实例 014　快速对多行代码进行注释

光盘位置：光盘\MR\02\014　　初级　　趣味指数：★★★☆

实例说明

可以使用“/**/”对多行代码进行注释，“/**/”的使用与“//”有所不同。多行代码注释使用“/*”标记注释开始，使用“*/”标记注释结束，代码中“/*”与“*/”中的内容被标记为注释的内容，如图 2.5 所示。

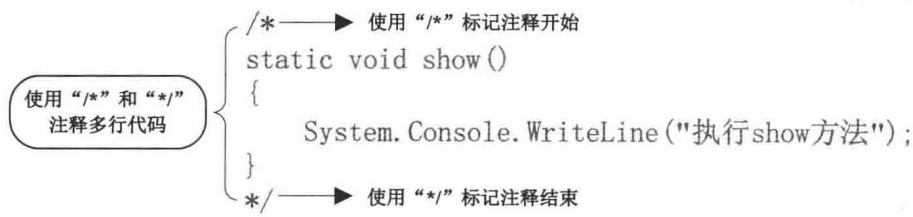


图 2.5　使用“/*”和“*/”注释多行代码

关键技术

本实例重点在于向读者介绍怎样使用“/**/”标记注释多行代码。下面对本实例用到的关键技术进行详细

讲解。

从图 2.5 中可以看到，show 方法已经被“/**/”标记注释。“/**/”多行注释可以用作代码说明，但是它经常被用于暂时屏蔽一部分代码的执行，“/**/”多行注释可以方便地屏蔽多行代码。

注意： “/**/”多行注释不能嵌套在另一个“/**/”中，而单行注释“//”与“///”则可以嵌套在“/**/”多行注释中。

设计过程

（1）多行代码注释的使用非常简单，打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 Many Commentate。

（2）程序的主要代码如下：

```
/*
 多行代码注释
 * 一般用于暂时屏蔽代码
 */
class program
{
    static void Main()
    {
    }
    /*
    static void show()
    {
        System.Console.WriteLine("执行 show 方法");
    }
    */
}
```

秘笈心法

心法领悟 014：使用“/**/”暂时屏蔽一部分代码的执行。

在调试程序时，可以方便地使用“//”标记暂时屏蔽单行代码，如果需要暂时屏蔽多行代码，那么使用“/**/”标记是非常合适的。

实例 015　使用“///”标记给代码段添加说明

光盘位置：光盘\MR\02\015　　初级　　趣味指数：★★★★

实例说明

使用“//”标记可以注释单行代码，使用“///”标记也可以注释单行代码，而且“///”标记单行注释的使用方法几乎与“//”标记相同。如果在某一行中出现“///”标记，那么“///”标记所在行后面的内容均为注释内容。但是“///”标记很少用于单行代码的注释，“///”标记一般用于为代码段添加说明，如图 2.6 所示。

```
/// <summary>
/// 程序入口方法 ——————————————————▶方法摘要信息
/// </summary>
/// <param name="args">用于向方法传递多个参数</param>——▶方法参数说明
/// <returns>方法返回整型数值</returns> ——————————▶方法返回值说明
static int Main(string[] args)
{
    return 0;
}
```

图 2.6　使用“///”为代码段添加说明

关键技术

本实例重点在于向读者介绍怎样使用“///”标记为代码段添加说明，以及怎样使用“///”标记生成自动文档。下面对本实例用到的关键技术进行详细讲解。

从图 2.6 中可以看到，使用了“///”标记为代码段添加说明，<summary></summary>中的信息为方法的摘要信息，<param></param>中的信息为方法的参数说明，<returns></returns>中的信息为方法返回值说明。

“///”标记不仅可以为代码段添加说明，它还有一项更重要的工作，就是用于生成自动文档。自动文档一般用于描述项目，使项目更加清晰直观。在 Visual Studio 2012 中可以通过设置项目属性来生成自动文档。具体步骤如下：在“解决方案资源管理器”面板中右击项目，在弹出的快捷菜单中选择“属性”命令，再在弹出的对话框中选择“生成*”选项卡，然后在“输出”栏中选中“XML 文档文件”复选框，如图 2.7 所示。

执行了上面的步骤后，编译程序集时会自动在输出路径中生成 XML 文档。生成的 XML 文档格式如图 2.8 所示。

图 2.7　设置项目属性生成自动文档

```
<?xml version="1.0" ?>
- <doc>
  - <assembly>
      <name>Code Commentate</name>
    </assembly>
  - <members>
    - <member name="T:program">
        <summary>定义program类</summary>
      </member>
    - <member name="M:program.Main(System.String[])">
        <summary>程序入口方法</summary>
        <param name="args">用于向方法传递多个参数</param>
        <returns>方法返回整型数值</returns>
      </member>
    </members>
</doc>
```

图 2.8　自动生成 XML 文档

说明：在代码的编写过程中应当养成良好的编码习惯，适当地在代码中加入注释，使用“///”标记为代码段添加说明。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 Code Commentate。

（2）程序的主要代码如下：

```
/// <summary>
///定义 program 类
/// </summary>
class program
{
    /// <summary>
    ///程序入口方法
    /// </summary>
    /// <param name="args">用于向方法传递多个参数</param>
    /// <returns>方法返回整型数值</returns>
    static int Main(string[] args)
```

```
    {
        return 0;
    }
}
```

秘笈心法

心法领悟 015：“///”注释的妙用。

前面已经介绍过“///”注释不仅可以为代码段添加注释，而且还具有生成自动文档的强大功能。自动文档一般用于描述项目，使项目更加清晰直观。

2.2 运算符的使用

实例 016 使用引号运算符进行赋值　　初级

光盘位置：光盘\MR\02\016　　趣味指数：★★★★☆

实例说明

“字符串”是如此的常用，以至于我们每天都要用到它，但是在使用字符串的过程中也许会发现，一些字符如双引号、反斜杠等，这些字符使用正常方法是无法显示出来的，那么，遇到这种情况就需要使用一些特殊的方法来正常显示字符串。本实例中将使用多种方法演示怎样在字符串中正常显示特殊字符。实例运行效果如图 2.9 所示。

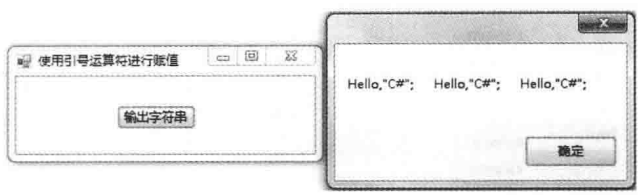


图 2.9　使用引号运算符进行赋值

关键技术

本实例重点考查的是如何在字符串中正常显示特殊字符，下面对本实例用到的关键技术进行详细讲解。

字符串是一组不可变的字符集合，它是由一系列 Unicode 字符组成的。如果定义一个字符内含有转义序列元素，则会发生转义，其中，转义序列包括标准的“\”字符，后跟一个“u”和一个 4 位十六进制值。可以通过下面的代码进一步了解。

"hellow\"C#\""与"hellow\u0022C#\u0022"表示的字符串相同。

说明：从上面的代码中可以看到“\u0022”表示一个双引号字符。

为了避免转义序列元素转义，可以通过以下两种方案避免转义。

（1）使用“@”标识符避免转义序列元素转义。

```
@"C:\Boot.int"
```

（2）可以指定字符串字面值（同时使用两个反斜杠）实现。

```
C:\\Boot.int
```

常用的转义序列列表如表 2.1 所示。

表 2.1　转义序列列表

转义序列	产生的字符	字符的 Unicode 值	转义序列	产生的字符	字符的 Unicode 值
\'	单引号	0x0027	\0	空	0x0000
\"	双引号	0x0022	\r	回车	0x000D
\\	反斜杠	0x005C			

说明：仔细阅读表 2.1 中的内容，了解转义序列的使用方法，以达到轻松使用转义序列表示特殊字符的目的。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，更改 Button 控件的 Name 属性为 Btn_Get，更改 Text 属性为“输出字符串”，此控件用来显示字符串信息。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    string str = "\"";                                          //定义字符串变量 str 并赋值为引号
    string P_strOne, P_strTwo, P_strThree;                      //定义 3 个字符串变量
    P_strOne = "Hello,\"C#\";";                                 //为第一个字符串变量赋值
    P_strTwo = "Hello," + '\u0022' + "C#" + '\u0022' + ";";     //为第二个字符串变量赋值
    P_strThree = "Hello," + str + "C#" + str + ";";             //为第三个字符串变量赋值
    MessageBox.Show(                                            //在消息提示框中输出 3 个字符串
        P_strOne + "        " + P_strTwo + "        " + P_strThree);
}
```

秘笈心法

心法领悟 016：应当怎样理解转义字符？

在理解转义字符前，有一个概念一定要清楚，字符串是不可变的一系列 Unicode 字符组成的集合，而前面讲到的转义序列如“\u0022”，都是字符串中的一个元素，也就是说“\u0022”只是表示一个双引号字符。

实例 017　使用算术运算符开发简单计算器

光盘位置：光盘\MR\02\017　　初级　　趣味指数：★★★★

实例说明

在代码的编写过程中经常会使用算术运算符，算术运算符常用于算术运算。本实例使用算术运算符开发一个简单计算器，计算器只是进行累加运算，使用泛型集合记录每一个将要进行累加计算的数值，使用 Foreach 遍历集合累加所有数值，同时显示计算表达式，最后通过算术运算符累加得到计算结果。实例运行效果如图 2.10 所示。

关键技术

本实例实现时主要使用了算术运算符，下面对其进行详细讲解。

算术运算符用来对数值进行加、减、乘、除及取模等算术运算。实例中主要用到了加运算符，加运算符通过两个或多个数值相加来执行标准的加法运算，以得到计算器累加数值的结果。加运算符如图 2.11 所示。

图 2.10　使用算术运算符开发简单计算器

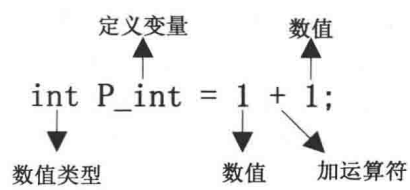


图 2.11　加运算符

说明：在使用加运算符时要注意，一些数值如两个 byte 数值相加后会将数值的类型提升为 int 类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Calculator。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于显示计算结果。再向窗体中添加两个容器控件 GroupBox，在第一个 GroupBox 中放入多个 Button 控件，用于输入数值；在第二个 GroupBox 中放入 3 个 Button 控件，分别用于加运算、显示和清空信息。

（3）程序的主要代码如下：

```
private List<double> G_list_value = new List<double>();                    //记录累加数值
private bool G_bl_add = false;                                              //判断是否刚刚按下+号
private bool G_bl_value = false;                                            //判断是否刚刚按下=号
private bool G_bl_key = false;                                              //防止连续按+号
private void btn_add_Click(object sender, EventArgs e)
{
    if (G_bl_value)                                                         //判断是否刚刚按下=号
    {
        G_bl_value = false;                                                 //设置刚刚按下的不是=号
        G_bl_key = true;                                                    //设置刚刚按下的是+号
    }
    else
    {
        if (!G_bl_key)                                                      //判断是否连续按+号
        {
            G_list_value.Add(  double.Parse(txt_value.Text));               //向集合中添加累加的数值
            GetValue();                                                     //计算累加数值并输出
            lb_express.Text = GetString();                                  //得到数值的字符串表示
            G_bl_add = true;                                                //设置已经按下+号
            G_bl_key = true;                                                //防止多次按下+号
        }
    }
}
```

秘笈心法

心法领悟 017：怎样理解自增运算符。

程序设计过程中会经常用到算术运算符，对数据进行加、减、乘、除、取模等运算，也可以方便地使用自增运算符对自身的数值加 1。自增运算符 i++与++i 都可以使 i 变量自加 1，但是它们运算的方式有些不同，i++是先参与其他运算后本身数值再自加 1，而++i 是先自身加 1 后再参与其他运算。

实例 018　使用“^”运算符对数字进行加密

光盘位置：光盘\MR\02\018　　中级　　趣味指数：★★★★

实例说明

在介绍实例之前先来简短了解一下加密的概念。加密是指通过某种特殊的方法，更改已有信息的内容，使得未授权的用户即使得到了加密信息，如果没有正确解密的方法，那么也无法得到信息的内容。谈到加密的话题，一些读者朋友一定非常感兴趣，而且会联想到复杂的加密算法，本实例只是使用异或“^”运算符简单地实现对数字加密的功能。实例运行效果如图 2.12 所示。

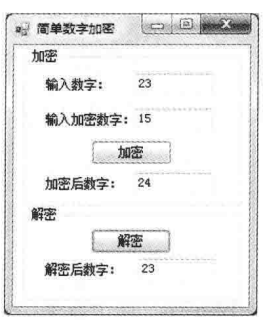


图 2.12　简单数字加密运行结果

关键技术

本实例实现时主要使用了异或运算符对数字进行异或运算，达

到简单加密数字的目的，下面对其进行详细讲解。

异或运算符“^”用于比较两个二进制数的相应位。在执行按位异或运算时，如果两个二进制数的相应位都为 1 或两个二进制数的相应位都为 0，则返回 0；如果两个二进制数的相应位其中一个为 1，另一个为 0，则返回 1。

现在来了解一下使用异或加密或解密的执行过程，数值 23 转换为二进制为 10111，加密数字的数值 15 转换为二进制为 1111。对比两个二进制数的值，从右向左按位对比，如果两个二进制数的相应位都为 1 或两个二进制数的相应位都为 0，则返回 0；如果两个二进制数的相应位中一个为 1，另一个为 0，则返回 1。最后得到的结果为二进制值 11000，转换到十进制为 24，所以得到的加密结果为 24。而解密过程也是很简单的，只是将加密结果 24 与加密数字 15 进行异或运算，将 24 转换二进制值为 11000，将 15 转换二进制值为 1111，进行异或运算后，得到结果为 23，这样又还原了加密的数据。

说明： 实例中只是简单地使用了异或运算符计算两个整型数值以达到加密目的，所以本实例只可以对整型数值进行加密运算，并不适合其他数据的加密。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Encrypt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个容器控件 GroupBox。在第一个 GroupBox 中放入 3 个 TextBox 控件和一个 Button 控件，分别用于输入数值、输入加密数值、显示加密后的数值和计算加密信息；在第二个 GroupBox 中放入一个 TextBox 控件和一个 Button 控件，分别用于显示解密后的信息和计算解密信息。

（3）程序的主要代码如下：

```
private void btn_Encrypt_Click(object sender, EventArgs e)
{
    int P_int_Num, P_int_Key;                                                    //定义两个值类型变量
    if (int.TryParse(txt_Num.Text, out P_int_Num) && int.TryParse(txt_Key.Text, out P_int_Key))   //判断输入是否是数值
    {
        txt_Encrypt.Text = (P_int_Num ^ P_int_Key).ToString();                   //加密数值
    }
    else
    {
        MessageBox.Show("请输入数值", "出现错误！ ");                              //提示输入信息不正确
    }
}
private void btn_Revert_Click(object sender, EventArgs e)
{
    int P_int_Key, P_int_Encrypt;                                                //定义两个值类型变量
    if (int.TryParse(txt_Encrypt.Text, out P_int_Key)                            //判断输入是否是数值
        && int.TryParse(txt_Key.Text, out P_int_Encrypt))
    {
        txt_Revert.Text = (P_int_Encrypt ^ P_int_Key).ToString();                //解密数值
    }
    else
    {
        MessageBox.Show("请输入数值", "出现错误！ ");                              //提示输入信息不正确
    }
}
```

秘笈心法

心法领悟 018：GroupBox 控件的使用。

GroupBox 控件是一个分组框控件，一般在窗体中划分多个区域时使用该控件。

实例 019 巧用位移运算符获取汉字编码值

光盘位置：光盘\MR\02\019　　中级　　趣味指数：★★★★☆

实例说明

我们伟大的祖国拥有五千年的悠久历史和灿烂的文化，而汉字是这几千年来文化的一种沉积。汉字在计算机中以字符的形式出现，还可以使用位移运算符获取汉字的编码值。实例运行效果如图 2.13 所示。

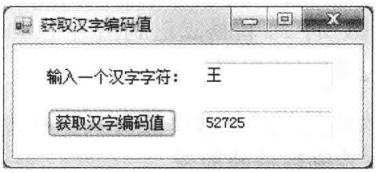


图 2.13　使用位移运算符获取汉字编码值

关键技术

本实例实现时主要用到位移运算符，下面对其进行详细讲解。

位移运算符分为左位移运算符“<<”和右位移运算符“>>”，分别用于向左和向右执行移位运算。对于 X<<N 或 X>>N 形式的运算，含义是将 X 向左或向右移动 N 位，X 的类型可以是 int、uint、long、ulong、byte、sbyte、short 和 ushort。需要注意的是，byte、sbyte、short 和 ushort 类型的值在进行位移操作后值的类型将自动转换成 int 类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetCode。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件和一个 Button 控件。两个 TextBox 控件分别用于添加汉字和显示汉字编码值，Button 控件用于计算汉字编码值。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    try
    {
        char chr = txt_chr.Text[0];                                    //获得一个汉字字符
        byte[] gb2312_bt =                                             //使用 gb2312 编码方式获得字节序列
            Encoding.GetEncoding("gb2312").GetBytes(new Char[] { chr });
        int n = (int)gb2312_bt[0] << 8;                                //将字节序列的第一个字节向左移 8 位
        n += (int)gb2312_bt[1];                                        //第一个字节移 8 位后与第二个字节相加得到汉字编码
        txt_Num.Text = n.ToString();                                   //显示汉字编码
    }
    catch (Exception)
    {
        MessageBox.Show(                                               //异常提示信息
            "请输入汉字字符！","出现错误！");
    }
}
```

秘笈心法

心法领悟 019：位移运算符使用技巧。

在进行位移运算时，当数值的二进制数每次向左移 1 位就相当于乘以 2，当数值每次向右移 1 位就相当于除以 2。

实例 020　使用条件运算符判断指定年份是不是闰年

光盘位置：光盘\MR\02\020　　初级　　趣味指数：★★★★

实例说明

闰年是为了弥补因人为历法规定造成的每一年的天数与地球实际公转周期的时间差而设定的，而补上时间差的年份被称作闰年，闰年共有 366 天。现在可以使用条件运算符判断用户输入的年份是否为闰年。人们常说“四年一闰，百年不闰，四百年再闰”，那么，怎样计算闰年呢？计算闰年的方法很简单，指定年份如果能被 400 整除就为闰年，或者指定年份可以整除 4 但不能整除 100 也为闰年，有了算法后程序的设计就变得简单多了。实例运行效果如图 2.14 所示。

关键技术

本实例实现时主要用到条件运算符，下面对其进行详细讲解。

条件运算符（?:）又叫三元运算符，它会根据布尔类型值或者布尔类型表达式返回两个值中的一个，如图 2.15 所示。

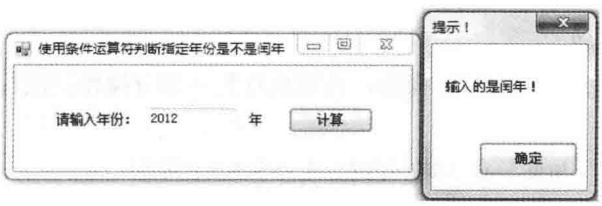


图 2.14　使用条件运算符判断指定年份是不是闰年

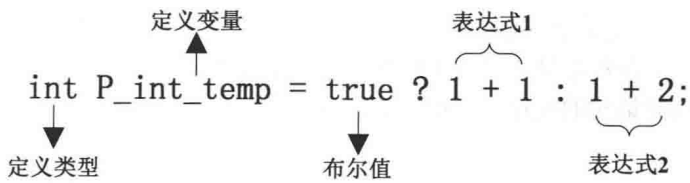


图 2.15　条件运算符的使用方法

从图 2.15 中可以看到，当条件运算符的布尔值为 true 时计算表达式 1，并将结果交给变量 P_int_temp，此时变量 P_int_temp 的值应当为 2。

说明：图 2.15 中条件运算符的布尔值可以替换为布尔表达式，通过计算布尔表达式的值来判断是返回表达式 1 的结果还是返回表达式 2 的结果。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetYear。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件和一个 Button 控件。TextBox 控件用于用户输入年份，Button 控件用于计算输入年份是否为闰年。

（3）程序的主要代码如下：

```
private void btn_GetMessage_Click(object sender, EventArgs e)
{
    ushort P_usint_temp;                                              //定义局部变量
    if (ushort.TryParse(txt_year.Text, out P_usint_temp))             //将输入字符串转换为数值
    {
        MessageBox.Show(                                              //输出计算结果
            (P_usint_temp % 4 == 0 && P_usint_temp % 100 != 0)        //判断是否为闰年
            || P_usint_temp % 400 == 0 ? "输入的是闰年！" : "输入的不是闰年！",
            "提示！");
    }
    else
    {
        MessageBox.Show("请输入正确数值！", "提示！");                  //提示输入数值不正确
    }
}
```

秘笈心法

心法领悟 020：适当使用条件运算符。

从本实例中可以看到，在 MessageBox.Show 方法中直接嵌套使用了条件运算符，这种内联的方法使程序更加简洁，如果使用 if 语句也可以完成上面条件运算符所做的工作，但是使用 if 语句完成此功能的代码要比使用条件运算符多很多，适当地使用条件运算符会使代码更加清晰明了。

实例 021	使用流程控制语句报销业务花销	初级
	光盘位置：光盘\MR\02\021	趣味指数：★★★★

实例说明

使用流程控制语句报销业务花销，流程控制语句一般用于进行逻辑判断后执行相应操作，在下面的实例中主要用来判断支出是否为业务花销，如果是业务花销则“正常报销！”，如果不是业务花销则是“不符合规定报销”。实例运行效果如图 2.16 所示。

关键技术

本实例实现时主要用到了 if 语句，下面对其进行详细讲解。

在程序开发过程中经常会用到 if 语句，如果条件满足那么将执行相应操作，否则执行另一部分操作。流程控制语句的使用方法如图 2.17 所示。

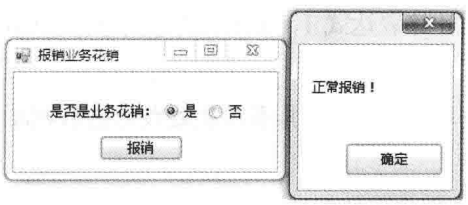


图 2.16　使用流程控制语句报销业务花销

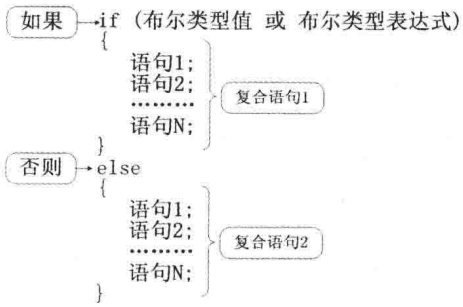


图 2.17　流程控制语句使用方法图

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IfThenElse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 RadioButton 单选按钮，两个单选按钮的 Name 属性分别为 rbt_true 和 rtb_false，这两个单选按钮用于判断报销是否为业务花销；添加一个 Button 控件，用于计算业务花销。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (rbtn_true.Checked)                                    //判断报销是否为业务花销
    {
        MessageBox.Show("正常报销！");                          //正常报销
    }
    else
    {
        MessageBox.Show("不符合规定报销！");                    //不符合规定报销
    }
}
```

秘笈心法

心法领悟 021：熟练地使用 if 语句。

流程控制语句一般用来控制程序执行逻辑，所以熟练地使用流程控制语句是程序开发的基础，if 语句后面小括号内的布尔类型值或布尔类型表达式用于逻辑判断，根据判断的结果选择执行不同的语句块。

2.3　关键字的使用

实例 022　使用 checked 关键字处理溢出错误　　初级

光盘位置：光盘\MR\02\022　　趣味指数：★★★★☆

实例说明

在进行数学运算时，由于变量类型不同，数值的值域也有所不同。如果变量中的数值超出了变量的值域，则会出现溢出情况，出现溢出时变量中的数值将不准确。怎样有效地防止溢出呢？下面结合实例演示怎样使用 checked 关键字检查是否出现溢出。实例运行效果如图 2.18 所示。

图 2.18　使用 checked 关键字处理溢出错误

说明：本实例中演示溢出所使用的数值类型为 byte，byte 类型的值域为 0～255，所以如果 byte 类型变量的值高于 255 或者小于 0 都会出现溢出。

关键技术

本实例实现时主要用到了 checked 关键字，下面对其进行详细讲解。

在进行算术运算时，可以使用 checked 关键字有效地处理溢出错误。当然，使用 checked 关键字可能对程序的性能会有一点点的影响，可以在代码的关键位置合理地使用 checked 关键字检查溢出，使用一点点的性能换来更加健康的程序，这样还是很好的。

注意：如果不使用 checked 关键字，数值出现溢出时将不会出现任何异常信息，这时将会出现一件可怕的事情，程序中的数据已经不正确，但是还没有发现。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CalcRAreaByAbstractClass。

（2）更改默认窗体 Form1 的 Name 属性为 Checked，向窗体中添加 3 个 TextBox 控件，分别用于输入计算溢出的数值；添加一个 Button 控件，用于数值计算。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    byte bt_One, bt_Two;                                          //定义两个 byte 类型变量
    if (byte.TryParse(                                            //为两个 byte 类型变量赋值
        txt_Add_One.Text, out bt_One)
        && byte.TryParse(txt_Add_Two.Text, out bt_Two))
    {
        try
        {
            checked { bt_One += bt_Two; }                         //使用 checked 关键字判断是否有溢出
```

```
            txt_Result.Text = bt_One.ToString();                    //输出相加后的结果
        }
        catch (OverflowException ex)
        {
            MessageBox.Show(ex.Message,"出错！ ");                  //输出异常信息
        }
    }
    else
    {
        MessageBox.Show("请输入 255 以内的数字!");                  //输出错误信息
    }
}
```

秘笈心法

心法领悟 022：使用 try…catch 捕捉异常。

try…catch 语句允许在 try 后面的大括号{}中放置可能发生异常情况的程序代码，对这些程序代码进行监控。在 catch 后面的大括号{}中则放置处理错误的程序代码，以处理程序发生的异常。

实例 023　使用 typeof 关键字获取类的内部结构

初级

光盘位置：光盘\MR\02\023　　趣味指数：★★★★☆

实例说明

typeof 关键字用于获取类型的 System.Type 对象，此时调用 Type 对象的 GetMethods 方法会得到类型中定义的方法对象的集合，调用方法对象集合中的每一个方法对象的 GetParameters 方法，会得到每一个方法的参数集合，最后通过 richTextBox 控件显示类型中定义的方法及方法参数的信息。实例运行效果如图 2.19 所示。

图 2.19　使用 typeof 关键字获取类的内部结构

关键技术

本实例实现时主要用到了 typeof 关键字，下面对其进行详细讲解。

typeof 关键字用于获取类型的 System.Type 对象。通过 System.Type 对象调用相应的方法，可以得到类型中定义的成员。我们所熟知的反射机制也使用了类似方式，使用反射机制首先是向应用域中加载程序集，然后得到 System.Type 对象，最后通过 System.Type 对象中的成员信息来执行相应操作。typeof 关键字的使用方法如图 2.20 所示。

另外，也可以调用对象的 GetType 方法来得到该类型的 System.Type 对象，如图 2.21 所示。

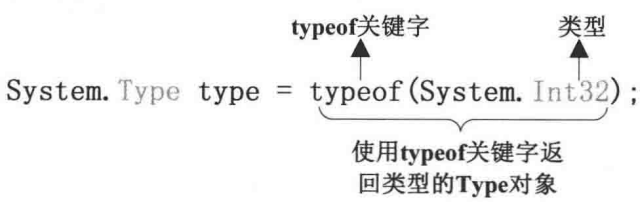


图 2.20　typeof 关键字的使用方法

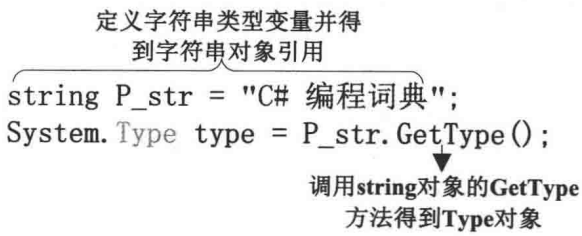


图 2.21　GetType 方法的使用

从图 2.21 中可以看到，首先定义了一个字符串类型的变量 P_str，此变量得到了一个字符串对象的引用，然后调用此字符串对象的 GetType 方法，得到字符串类型的 System.Type 实例。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TypeOf。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 richTextBox 控件，用于显示类的内部结构；添加一个 Button 控件，用于执行显示操作。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    Type type = typeof(System.Int32);                                    //获得 string 类型的 Type 对象
    foreach (MethodInfo method in type.GetMethods())                     //遍历 string 类中定义的所有公共方法
    {
        rtbox_text.AppendText(
            "方法名称：" + method.Name + Environment.NewLine);           //输出方法名称
        foreach (ParameterInfo parameter in method.GetParameters())      //遍历公共方法中的所有参数
        {
            rtbox_text.AppendText(
                "  参数：" + parameter.Name + Environment.NewLine);      //输出参数名称
        }
    }
}
```

秘笈心法

心法领悟 023：在哪里还可以用到 typeof 关键字？

本实例中使用 typeof 关键字得到 System.Type 对象，通过调用 System.Type 对象的方法得到类型的成员。但是除了上述用法外，在哪里还可以用到 typeof 关键字呢？可以使用 typeof 关键字判断窗体控件类型，并清空窗体中所有 TextBox 控件的文本信息。主要代码如下：

```
foreach (Control c in Controls)                                          //遍历窗体控件集合
{
    if (c.GetType()==typeof(TextBox))                                    //判断是否为 TextBox 控件
    {
        ((TextBox)c).Clear();                                            //清空控件内容
    }
}
```

实例 024　使用 using 关键字有效回收资源

光盘位置：光盘\MR\02\024　　初级　趣味指数：★★★★☆

实例说明

在进行文件操作后要显式调用文件流的 Close 方法释放文件资源，在使用数据库连接时也要调用连接对象的 Close 方法释放数据库资源。如果忘记调用 Close 方法，有可能会导致程序执行异常，而且还会导制垃圾收集器多次回收对象，造成内存的浪费。本实例将会演示使用 using 关键字有效回收资源，实例运行效果如图 2.22 所示。

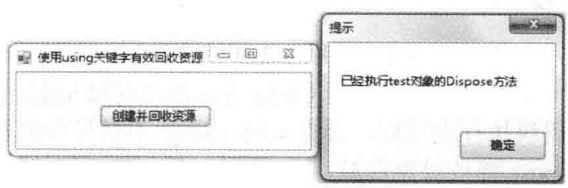


图 2.22　使用 using 关键字有效回收资源

关键技术

本实例实现时主要用到了 using 关键字，下面对其进行详细讲解。

using 关键字可以用来引用命名空间，如 using System。using 关键字还有一种方便的功能，它可以帮助有效地回收对象资源。回收的资源包括文件资源、网络资源及数据库资源等。使用 using 关键字回收资源的工作原理非常简单，使用方法如图 2.23 所示。

从图 2.23 中可以看到，代码中使用了 using 关键字，在其后的小括号中创建了一个 test 类型的对象，using

关键字要求小括号内产生的对象必须实现 IDisposable 接口或者其基类实现了 IDisposable 接口，如果没有实现此接口，那么在编译时会出现错误。在 using 关键字下面的代码块执行完毕后，会自动将 test 对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

也可以使用 using 关键字同时回收多个对象资源，使用方法如图 2.24 所示。

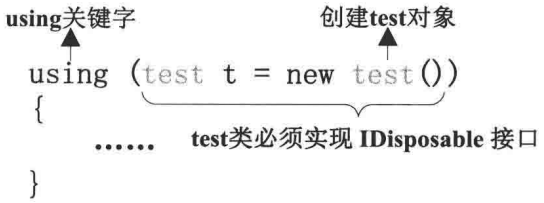


图 2.23　using 关键字的使用方法

图 2.24　多个 using 关键字的使用方法

从图 2.24 中可以看到，使用 using 关键字可以回收多个对象的资源，其使用方法与前面介绍的方法基本相同。

注意：前面已经提到过，using 关键字后面小括号内创建的对象必须要实现 IDisposable 接口，或者此对象的基类已经实现了 IDisposable 接口。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Using。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于演示使用 using 关键字回收资源。

（3）程序的主要代码如下：

```
private void btn_true_Click(object sender, EventArgs e)
{
    using (new test())                          //在 using 关键字中创建 test 对象
    {
    }                                           //using 关键字块执行完成后会自动调用 test 对象的 Dispose 方法
}
class test : IDisposable                        //定义 test 类实现 IDisposable 接口
{
    public void Dispose()                       //实现接口方法
    {
        MessageBox.Show(                        //弹出消息对话框
            "已经执行 test 对象的 Dispose 方法","提示");
    }
}
```

秘笈心法

心法领悟 024：使用 using 关键字回收资源更轻松。

虽然可以手动调用 Close 方法回收对象资源，但是使用 using 关键字更为简洁友好。在 using 关键字下面的代码块执行完毕后，会自动将 using 小括号中创建的对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 025　使用 is 关键字检查对象是否与给定类型兼容

初级

光盘位置：光盘\MR\02\025

趣味指数：★★★☆

实例说明

在程序的开发过程中经常会使用类型转换，如果类型转换不成功则会出现异常，从抛出异常到捕获并处理异常，无形中增加了系统的开销，而且太过频繁地处理异常还会严重地影响系统的稳定性。is 关键字可以有效

地解决上面出现的问题。is 关键字用于检查对象是否与给定类型兼容，如果兼容则返回 true，如果不兼容则返回 false。在进行类型转换前，可以先使用 is 关键字判断对象是否与指定类型兼容，如果兼容才进行转换，这样的转换是安全的。实例运行效果如图 2.25 所示。

图 2.25　使用 is 关键字检查对象是否与给定类型兼容

关键技术

本实例实现时主要用到了 is 关键字，下面对其进行详细讲解。

is 关键字主要用于检查对象是否与指定类型兼容，如果兼容则返回 true，如果不兼容则返回 false。is 关键字的使用方法如图 2.26 所示。

定义字符串类型变量并得到字符串对象引用

```
string P_str = "C#编程宝典";

object P_obj = P_str;
```

将string对象隐式转换为object类型的对象

```
if (P_obj is string)  string P_str2 = (string)P_obj;
```

使用is关键字判断P_obj对象是否与string类型兼容

如果P_obj对象与string类型兼容则进行显式转换

图 2.26　is 关键字的使用方法（一）

说明：从图 2.26 中可以看到，首先创建一个字符串对象，然后将字符串对象隐式转换为 object 类型的对象，最后使用 is 关键字判断 object 类型的对象是否与 string 类型兼容，结果为 true 则执行后面的显式转换。由此可以想到，由于使用了 is 关键字在前面做判断，显式转换是安全的，不会抛出异常。在图 2.27 中演示了 is 关键字检测到对象与指定类型不兼容的情况。

定义字符串类型变量并得到字符串对象引用

```
string P_str = "C#编程宝典";

object P_obj = P_str;
```

将string对象隐式转换为object类型的对象

```
if (P_obj is Stream)  string P_str2 = (Stream)P_obj;
```

使用is关键字判断P_obj对象是否与Stream类型兼容

如果P_obj对象与Stream类型兼容则进行显式转换

图 2.27　is 关键字的使用方法（二）

说明：从图 2.27 中可以看到，首先创建一个字符串对象，然后将字符串对象隐式转换为 object 类型的对象，最后使用 is 关键字判断 object 类型的对象是否与 Stream 类型兼容。由于 object 类型的对象是从 string 对象隐式转换而来，所以 object 类型的对象不可能与 Stream 类型兼容，结果为 false，不会执行后面的显式转换。使用 is 关键字的好处是在进行显式转换前可以首先判断转换是否成功，从而杜绝了类型转换中出现的异常。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Equal。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于判断选定对象与类型是否兼容；添加两个 GroupBox 控件，分别向其中添加两个 RadioButton 单选按钮，用于定义选择的对象与类型。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    object P_obj = rbtn_target1.Checked ?                                    //正确地为变量添加引用
        (object)"C# 编程词典" : new System.IO.FileInfo(@"d:\");
    if (rbtn_class1.Checked)                                                 //判断选择了哪一个类型
    {
        if (P_obj is System.String)                                          //判断对象是否为字符串类型
            MessageBox.Show("对象与指定类型兼容", "提示！ ");                  //提示兼容信息
        else
            MessageBox.Show("对象与指定类型不兼容","提示！ ");                 //提示不兼容信息
    }
    else
    {
        if (P_obj is System.IO.FileInfo)                                     //判断对象是否为文件类型
            MessageBox.Show("对象与指定类型兼容", "提示！ ");                  //提示兼容信息
        else
            MessageBox.Show("对象与指定类型不兼容", "提示！ ");                //提示不兼容信息
    }
}
```

秘笈心法

心法领悟 025：使用 is 关键字可以有效地防止类型转换中出现的异常。

is 关键字用于检查对象是否与给定类型兼容，如果兼容则返回 true，如果不兼容则返回 false。在进行类型转换前，可以先使用 is 关键字判断对象是否与指定类型兼容，如果兼容才进行转换，这样的转换是安全的。

实例 026　使用 as 关键字将对象转换为指定类型

光盘位置：光盘\MR\02\026　　初级　　趣味指数：★★★★☆

实例说明

使用 as 关键字可以将对象转换为指定类型，与 is 关键字不同，is 关键字用于检查对象是否与给定类型兼容，如果兼容则返回 true，如果不兼容则返回 false。而 as 关键字会直接进行类型转换，如果转换成功将会返回转换后的对象，如果转换不成功则不会抛出异常而是返回 NULL。下面的实例中创建了一个 FileStream 对象，并使用 as 关键字试图将 FileStream 对象转换为 object 类型、Stream 类型和 String 类型。实例运行效果如图 2.28 所示。

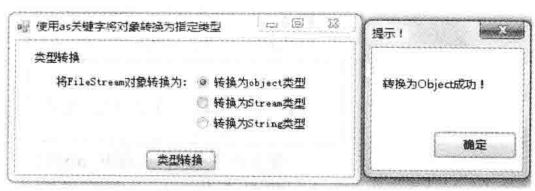


图 2.28　使用 as 关键字将对象转换为指定类型

关键技术

本实例实现时主要用到了 as 关键字，下面对其进行详细讲解。

as 关键字用于引用类型之间执行转换。as 关键字的使用方法如图 2.29 所示。

从图 2.29 中可以看到，首先创建一个字符串对象，然后将字符串对象隐式转换为 object 类型的对象 P_obj，

使用 as 关键字将 P_obj 转换为字符串对象，最后检查字符串对象是否为空。如果为空，则转换不成功；如果不为空，则转换成功并执行后面的语句块。

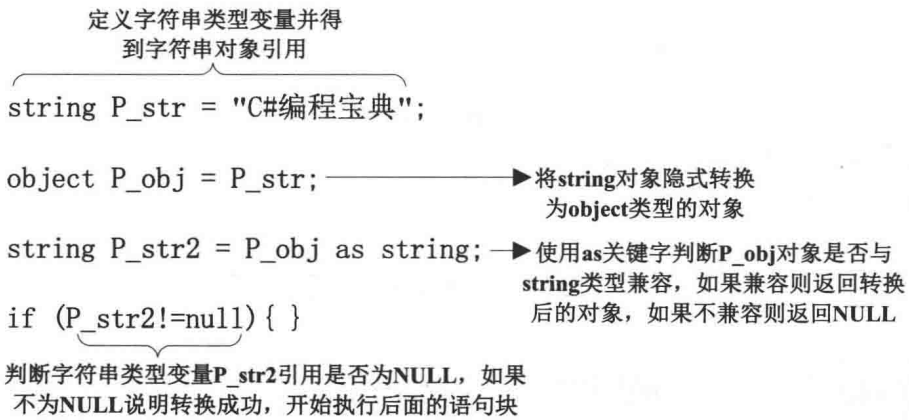


图 2.29　as 关键字的使用方法

说明：as 关键字与 is 关键字一样都不会抛出异常，但是相比之下 as 关键字要更方便一些。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Transform。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于类型转换；添加一个 GroupBox 控件，向此控件中添加 3 个 RadioButton 单选按钮，分别用于选择将对象转换为哪种类型。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    if (rbtn_object.Checked)                                    //如果选择转换为 object 类型
    {
        using (FileStream P_filestream =                        //创建文件流对象
            new FileStream(@"d:\log.txt", System.IO.FileMode.Create))
        {
            object P_object =                                   //使用 as 关键字转换类型
                P_filestream as object;
            if (P_object != null)                               //判断转换是否成功
            {
                MessageBox.Show("转换为 Object 成功！", "提示！");
            }
            else
            {
                MessageBox.Show("转换为 Object 不成功！", "提示！");
            }
        }
    }
    if (rbtn_stream.Checked)                                    //如果选择转换为 Stream 类型
    {
        using (FileStream P_filestream =                        //创建文件流对象
            new FileStream(@"d:\log.txt", System.IO.FileMode.Create))
        {
            object P_obj = P_filestream;
            Stream P_stream =                                   //使用 as 关键字转换类型
                P_obj as Stream;
            if (P_stream != null)                               //判断转换是否成功
            {
                MessageBox.Show("转换为 Stream 成功！", "提示！");
            }
            else
            {
```

```
                MessageBox.Show("转换为 Stream 不成功！", "提示！");
            }
        }
    }
}
```

秘笈心法

心法领悟 026：使用 as 关键字在转换过程中不会出现异常。

as 关键字与 is 关键字一样都不会抛出异常，使用 as 关键字对对象进行类型转换，如果转换成功将会返回转换后的对象，如果转换不成功则不会抛出异常而是返回 NULL。

2.4 流程控制语句的使用

实例 027	小明去学校和医院分别要走哪条路 光盘位置：光盘\MR\02\027	初级 趣味指数：★★★★

实例说明

人类的大脑就好比是一部逻辑机器，每天都要处理大量的数据。那么，怎样使编写的程序也同样具有逻辑处理能力呢？使用 if 语句可以根据逻辑判断执行相应的语句块，根据不同的条件做出不同的行为，如用来判断小明去学校和医院分别要走哪条路。实例运行效果如图 2.30 所示。

图 2.30 小明去学校和医院分别要走哪条路

关键技术

本实例实现时主要用到了 if 语句，下面对本实例用到的关键技术进行详细讲解。

if 语句的小括号中的布尔值或布尔表达式的值为 true，则执行复合语句 1；如果值为 false，则执行 else 后面的复合语句 2，格式如下：

```
if(布尔值或布尔表达式)
{
复合语句 1
}
else
{
复合语句 2
}
```

注意： else 关键字和后面的复合语句 2 是可选的，如果没有 else 关键字和后面的复合语句 2 时，当 if 小括号中的布尔值为 false 时，将不会执行复合语句 1 中的语句，而是跳转到下面的语句继续执行。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WhichWay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于计算“小明”应当走哪条路；添加两个 RadioButton 单选按钮，分别用来指示小明要去学校还是医院。

（3）程序的主要代码如下：

```
private void btn_go_Click(object sender, EventArgs e)
{
    if (rbtn_school.Checked)                                    //判断小明去学校还是去医院
    {
        MessageBox.Show("向左走","提示！");                      //如果去学校则向左走
    }
```

```
    else
    {
        MessageBox.Show("向右走", "提示！ ");                    //如果去医院则向右走
    }
}
```

秘笈心法

心法领悟 027：适当地使用三元运算符代替 if 语句代码会更简洁。

在使用 if 语句进行判断并赋值时，可以适当地使用三元运算符代替，适当地使用条件运算符会使代码更加清晰明了，使用三元运算符又叫条件运算符，它首先会判断布尔值或布尔表达式，并根据布尔结果返回表达式 1 的值，或者返回表达式 2 的值。

实例 028　利用条件语句判断用户登录身份

光盘位置：光盘\MR\02\028　　初级　　趣味指数：★★★★☆

实例说明

在应用程序运行时，经常需要判断用户登录身份，根据用户登录身份的不同，给予相应的操作权限，下面将使用条件语句判断用户登录身份。实例运行效果如图 2.31 所示。

关键技术

本实例实现时主要用到了 if 语句及 ComboBox 控件，下面对其进行详细讲解。

ComboBox 是窗体中的下拉列表控件，在使用 ComboBox 控件前，可以先向 ComboBox 控件的 Items 集合中添加下拉列表中将要显示的对象（一般为字符串对象），由于 Items 集合的 Add 方法接受 Object 对象，所以任何对象都可以放入 Items 集合中，ComboBox 控件在显示下拉列表时会自动调用 Items 集合中每一个 Object 对象的 ToString 方法，得到字符串对象，并在 ComboBox 控件中显示。

例如，可以在 ComboBox 控件中添加一个 test 对象。实现代码如下：

```
private void button1_Click_1(object sender, EventArgs e)
{
    comboBox1.Items.Add(new test());                    //向 Items 集合中添加对象
}
class test                                              //定义 test 类
{
    public override string ToString()                   //重写 ToString 方法
    {
        return "C#编程词典";                            //返回字符串对象
    }
}
```

上面的代码运行结果如图 2.32 所示。

图 2.31　利用条件语句判断用户登录身份

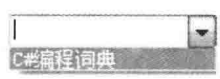


图 2.32　在 ComboBox 控件中显示信息

由于 test 类重写了 ToString 方法，所以 ComboBox 控件中会显示“C#编程词典”字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Login。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 下拉列表控件，用于

选择登录的用户；添加一个 Button 控件，用于判断用户登录信息并使用消息框显示。

（3）程序的主要代码如下：

```
private void btn_login_Click(object sender, EventArgs e)
{
    if (cbox_select.SelectedItem. ToString() == "admin")                    //判断用户登录信息
    {
        MessageBox.Show("管理员登录","提示！ ");                              //如果是 admin 登录，则提示管理员登录
    }
    else
    {
        MessageBox.Show("普通用户登录", "提示！ ");                           //如果是 user 登录，则提示普通用户登录
    }
}
```

秘笈心法

心法领悟 028：设置下拉列表中选中的内容为只读。

在程序运行中，如果不希望用户更改 ComboBox 下拉列表中选中的内容，可以设置 DropDownStyle 属性为 DropDownList，这样可以使用户选择的项为只读。

实例 029　判断指定月份属于哪个季节

光盘位置：光盘\MR\02\029　　初级　　趣味指数：★★★★☆

实例说明

每一年都有春、夏、秋、冬 4 个季节，而每个季节都有大约 3 个月时间，春季从 3 月到 5 月，夏季从 6 月到 8 月，秋季从 9 月到 11 月，而冬季从 12 月到次年的 2 月。了解了上面的信息后，可以使用 switch 多路选择语句判断用户选定的月份属于哪一个季节。实例运行效果如图 2.33 所示。

关键技术

本实例主要使用了 switch 多路选择语句，下面对其进行详细讲解。

switch 语句是多路选择语句，它通过一个表达式的值来使程序从多个分支中选取一个用于执行的分支。switch 语句的使用方法如图 2.34 所示。

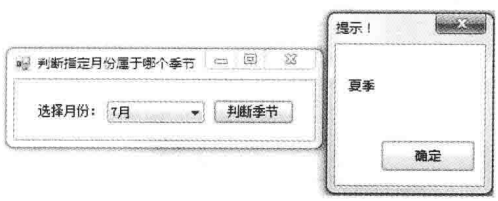


图 2.33　判断指定月份属于哪个季节

```
switch (表达式)
{
    case 常量表达式1 :
          语句块1
          break;
    case 常量表达式2 :
          语句块2
          break;
    case 常量表达式3 :
          语句块3
          break;
}
```

图 2.34　switch 语句的使用方法

说明：switch 表达式的值只可以是整型、字符串、枚举和布尔类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetSeason。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 下拉列表控件，用于用户选择月份；添加一个 Button 控件，用于判断选择的月份属于哪个季节。

（3）程序的主要代码如下：

```
private void btn_true_Click(object sender, EventArgs e)
{
    switch (                                                    //根据所选月份判断季节
        cbox_select.SelectedIndex + 1)
    {
        case 3:
        case 4:
        case 5:
            MessageBox.Show("春季", "提示！");                    //提示选择春季
            break;
        case 6:
        case 7:
        case 8:
            MessageBox.Show("夏季", "提示！");                    //提示选择夏季
            break;
        case 9:
        case 10:
        case 11:
            MessageBox.Show("秋季", "提示！");                    //提示选择秋季
            break;
        case 12:
        case 1:
        case 2:
            MessageBox.Show("冬季", "提示！");                    //提示选择冬季
            break;
        default:                                                //如果没有选择月份弹出提示信息
            MessageBox.Show("请选择月份","提示！");
            break;
    }
}
```

秘笈心法

心法领悟 029：switch 语句中多个 case 可以使用一个 break。

在 switch 多路选择语句中，多个 case 标签可以使用一个 break 关键字，但是在这种情况下，只有最后一个 case 标签中可以带有语句块，而前面的 case 标签中不可以带有语句块内容。

实例 030　使用 switch 语句更改窗体颜色

光盘位置：光盘\MR\02\030　　中级　　趣味指数：★★★★

实例说明

在窗体编程中，可以通过 Text 属性设置窗体的标题，通过 Width 属性设置窗体的宽度，通过 Height 属性设置窗体的高度，也可以通过 BackColor 属性设置窗体的背景颜色。这一切操作起来都非常的简单，对于知识的活学活用才是掌握知识的根本。本实例中使用 switch 多路选择语句，根据用户的选择更改窗体颜色。实例运行效果如图 2.35 所示。

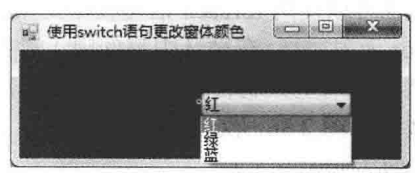


图 2.35　使用 switch 语句更改窗体颜色

关键技术

本实例实现时主要用到 switch 多路选择语句及窗体的 BackColor 属性，由于 switch 多路选择语句在实例 029 中已经做过介绍，所以这里主要讲解窗体的 BackColor 属性。

由于 C#窗体中所有控件都继承于 Control 类，所以所有控件都具有 Control 类定义的属性和方法，其中就包括了 BackColor 属性，该属性用于获取或设置控件的背景颜色。

说明：可以使用 Color 结构的静态属性获取 Color 对象，如 Color.Red；也可以使用 Color 结构的静态方法 Color.FromArgb，并根据 RGB 3 种颜色的组合找到需要的颜色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 下拉列表控件，用于选择窗体的颜色。

（3）程序的主要代码如下：

```
private void cbox_select_SelectedIndexChanged(object sender, EventArgs e)
{
    switch (cbox_select.SelectedIndex)                    //使用 switch 语句判断窗体使用哪种颜色
    {
        case 0:
            this.BackColor = Color.Red;                   //窗体设置为红色
            break;
        case 1:
            this.BackColor = Color.Green;                 //窗体设置为绿色
            break;
        case 2:
            this.BackColor = Color.Blue;                  //窗体设置为蓝色
            break;
    }
}
```

秘笈心法

心法领悟 030：可以使用 Timer 组件实现定时更改窗体颜色。

可以在本实例的基础上做一个延伸，使用 Timer 组件计时，首先向窗体中拖入 Timer 组件，然后设置其 Enable 属性为 true，设置 Interval 属性为 30000，在 Tick 事件中写入随机更改窗体颜色的代码，在窗体执行时会自动更换背景颜色。

实例 031 循环向控制台中输入内容

光盘位置：光盘\MR\02\031　　中级　　趣味指数：★★★☆

实例说明

在程序的开发过程中经常会用到循环语句，可以使用 for 循环遍历数组，也可以使用 foreach 语句枚举出集合中的每一个元素。下面将会使用 for 无限循环，在循环中每间隔 1 秒钟都会向控制台中输入系统时间，然后再清空控制台信息，以实现动态显示系统时间的效果。实例运行效果如图 2.36 所示。

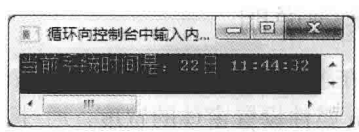


图 2.36　循环向控制台中输入内容

关键技术

本实例实现时主要用到了 for 循环以及 Console 类，下面对其进行详细讲解。

for 语句在循环语句中有着重要的位置，首先它计算一个初始化序列，然后当某个条件为真时，重复执行嵌套语句并计算一个迭代表达式序列；如果为 false，则退出 for 循环。for 语句常用于遍历数组集合，基本格式如下：

```
for(初始化语句 1; 每次循环前验证的终止条件; 每次循环结束时执行的语句)
{
```

```
【语句块】
}
```

Console 类是一个静态类，它不可以被继承。Console 类提供了多个静态方法，如 Console.WriteLine 方法用于向控制台中输出一行字符串，Console.Clear 方法用于清空控制台中的信息，Console.ReadLine 方法用于从标准输入流读取下一行字符。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 InputMessage。

（2）程序的主要代码如下：

```
class program
{
    static void Main()
    {
        System.Console.Title = "循环向控制台中输入内容";                    //定义控制台标题
        System.Console.WindowWidth = 30;                                    //设置控制台窗体宽度
        System.Console.WindowHeight = 2;                                    //设置控制台窗体高度
        for (; ; )                                                          //开始无限循环
        {
            System.Console.WriteLine("当前系统时间是：{0}",                  //输出系统当前时间
                System.DateTime.Now.ToString("dd 日 hh:mm:sss"));
            System.Threading.Thread.Sleep(1000);                            //线程挂起 1 秒钟
            System.Console.Clear();                                         //清空控制台信息
        }
    }
}
```

秘笈心法

心法领悟 031：for 循环是遍历数组的好帮手。

使用 while 语句可以遍历数组，使用 goto 语句也可以遍历数组，但是遍历数组最好还是使用 for 语句。for 语句好像是为遍历数组量身定做的一样，for 语句中计数器的值可以当作数组的索引，方便遍历数组中的元素。

实例 032　递归算法的经典面试题

光盘位置：光盘\MR\02\032　　中级　　趣味指数：★★★★☆

实例说明

在编程世界中，充斥着大量的数据，怎样方便有效地处理数据呢？这时算法就显得尤为重要了。下面的实例中介绍了一个经典的面试题，有一组数 1、1、2、3、5、8、13、21、34…，要求用递归算法算出这组数的第 30 个数是多少？实例运行效果如图 2.37 所示。

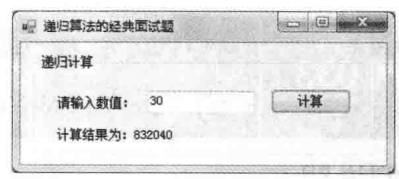


图 2.37　使用递归算法的经典面试题

关键技术

本实例实现时主要用到了递归算法，下面对其进行详细讲解。

递归算法在程序设计中被广泛应用，递归是方法在运行中直接或间接调用自身而产生的重入现象，它是计算机科学的一个重要概念，而且在算法的世界中占有着重要的位置，使用递归方法可以使代码变得更加简洁和清晰。在生活中，可以从视觉的角度了解递归的概念，准备两面平行的返光镜，人站在两面返光镜的中间，向其中一面返光镜观看，会看到自己和返光镜，而返光镜中还有自己和返光镜，就是这样一直递归下去。

说明：由于线程栈有固定的大小，如果进行递归的层次太深，有可能会出现溢出。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Arithmetic。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于用户输入信息；添加一个 Button 控件，用于计算结果；添加一个 Label 控件，用于显示计算结果。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    int P_int_temp;                                          //定义整型变量
    if (int.TryParse(txt_value.Text, out P_int_temp))        //为变量赋值
    {
        lb_result.Text =                                     //输出计算结果
            "计算结果为：" + Get(P_int_temp).ToString();
    }
    else
    {
        MessageBox.Show(                                     //提示输入正确数值
            "请输入正确的数值！","提示！");
    }
}
/// <summary>
///递归算法
/// </summary>
/// <param name="i">参与计算的数值</param>
/// <returns>计算结果</returns>
int Get(int i)
{
    if (i <= 0)                                              //判断数值是否小于 0
        return 0;                                            //返回数值 0
    else if (i >= 0 && i <= 2)                               //判断位数是否大于等于 0 并且小于等于 2
        return 1;                                            //返回数值 1
    else                                                     //如果不满足上述条件执行下面的语句
        return Get(i - 1) + Get(i - 2);                      //返回指定位数前两位数的和
}
```

秘笈心法

心法领悟 032：应当注意线程栈的溢出。

在 C#中，每建立一条线程都会被分配大约 1MB 大小的地址空间，由于线程栈有固定的大小，如果进行递归的层次太深，有可能会出现溢出。

实例 033　制作一个数字猜猜看小游戏　中级

光盘位置：光盘\MR\02\033　　趣味指数：★★★★☆

实例说明

程序的世界中充斥着大量的数据与复杂的逻辑运算，怎样才能处理好数据间的逻辑关系？怎样才能锻炼我们的逻辑思维能力？开发一个简单的小游戏是不错的选择。游戏中首先使用随机对象产生一个 1～100 之间的整数，当用户单击“开始”按钮时，动态生成 100 个按钮，并开始计时，如果用户单击的数字小于随机数，那么被单击的按钮变为红色，并显示字符串“小”；如果用户单击的数字大于随机数，那么被单击的按钮变为红色，并显示字符串“大”；如果单击的数字等于随机数就会弹出消息框，提示已经猜对了数字，并显示用时及猜测次数。实例运行效果如图 2.38 所示。

关键技术

本实例实现时主要用到了多线程技术，下面对其进行详细讲解。

在使用多线程前要先引用命名空间 System.Threading，引用命名空间后就可以在需要的地方方便地创建并使用线程。定义线程如图 2.39 所示。

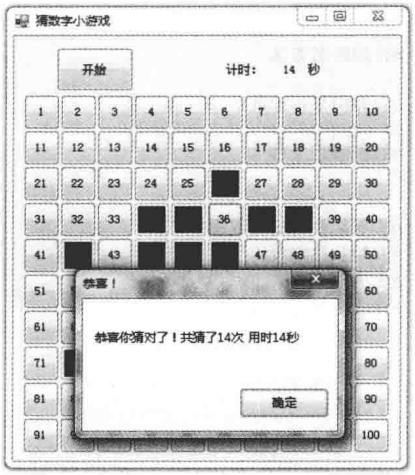


图 2.38　数字猜猜看小游戏

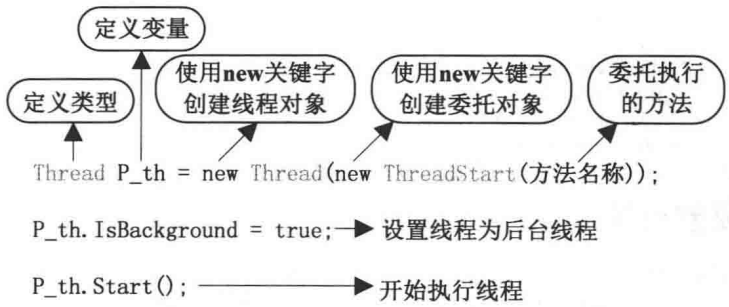


图 2.39　定义线程

从图 2.39 中可以看到，创建线程对象的构造方法中使用了 ThreadStart()委托，当线程开始执行时，将会执行委托中定义的方法。通过 IsBackground 属性设置 P_th 线程为后台线程，调用 P_th 对象的 Start 方法启动线程，开始执行。在线程开始后可以调用线程对象的 Abort 方法终止线程。现在会发现定义和使用线程就是这样简单。

说明： 创建的线程分为前台线程和后台线程，通过线程对象的 IsBackground 属性可以设置线程为前台线程或后台线程。前台线程与后台线程的区别是，后台线程不会使托管程序处于运行状态。也就是说，如果进程停止所有前台线程，那么系统会认为此进程处于非运行状态，将会停止所有后台线程并关闭。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NumGame。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于控制游戏开始。

（3）程序的主要代码如下：

```
private void btn_begin_Click(object sender, EventArgs e)
{
    RemoveControl();                                          //清空所有无用对象
    int p_int_x = 10;                                         //X 坐标初始值为 10
    int p_int_y = 60;                                         //Y 坐标初始值为 60
    for (int i = 0; i < 100; i++)                             //添加 100 个按钮
    {
        Button bt = new Button();                             //创建 Button 控件
        bt.Text = (i + 1).ToString();                         //设置 Button 控件的文本值
        bt.Name = (i + 1).ToString();                         //设置 Button 控件的 Name 属性
        bt.Width = 35;                                        //设置 Button 控件的宽
        bt.Height = 35;                                       //设置 Button 控件的高
        bt.Location = new Point(p_int_x, p_int_y);            //设置 Button 控件的位置
        bt.Click += new EventHandler(bt_Click);               //定义 Button 控件的事件
        p_int_x += 36;                                        //设置下一个控件的位置
        if ((i + 1) % 10 == 0)                                //设置换行
        {
            p_int_x = 10;                                     //换行后重新设置 X 坐标
            p_int_y += 36;                                    //换行后重新设置 Y 坐标
        }
        Controls.Add(bt);                                     //将 Button 控件放入窗体控件集合中
    }
    G_th = new System.Threading.Thread(delegate()             //新建一条线程
        {
```

```
            int P_int_count = 0;                                          //初始化计数器
            while (true)                                                  //开始无限循环
            {
                P_int_count = ++P_int_count > 100000000 ? 0 : P_int_count;  //计数器累加
                this.Invoke(                                              //将代码交给主线程执行
                    (MethodInvoker)delegate                               //使用匿名方法
                    {
                        lb_time.Text = P_int_count.ToString();            //窗体中显示计数
                    });
                System.Threading.Thread.Sleep(1000);                      //线程睡眠 1 秒
            }
        });
    G_th.IsBackground = true;                                             //设置线程为后台线程
    G_th.Start();                                                         //开始执行线程
    G_int_num = G_random.Next(1, 100);                                    //生成随机数
    btn_begin.Enabled = false;                                            //停用“开始”按钮
}
```

秘笈心法

心法领悟 033：使用线程时要注意线程同步的问题。

适当地使用线程，可以增加程序的运行效率，使程序运行更加流畅，但是线程间的同步问题也显得尤为重要，可以使用线程锁或监视器有效地处理线程同步问题。

实例 034　使用 goto 语句在数组中搜索指定图书

光盘位置：光盘\MR\02\034　　中级　　趣味指数：★★★★

实例说明

将图书信息放入字符串数据中，可以使用 System.Array 的 IndexOf 静态方法搜索指定的图书，也可以使用 for 循环遍历数组查找指定图书，而在下面的实例中使用了 goto 语句和一个计数器实现简单的循环，从 string 数组中搜索图书。实例运行效果如图 2.40 所示。

图 2.40　使用 goto 语句在数组中搜索指定图书

关键技术

本实例实现时主要用到了 goto 语句，下面对其进行详细讲解。

goto 语句是一个非常灵活的语句，可以使用 goto 语句跳转到定义标签的位置，这里所说的标签包括 switch 语句中的 case 标签、default 标签和语句声明的标签，也可以使用 goto 语句跳出多层嵌套循环。

标签语句的格式如下：

```
【标签】:【语句】
```

说明：标签的作用域为声明它的整个语句块，包括其中的嵌套块。goto 语句可以在标签的作用域中将程序执行的位置移动到当前标签的位置。

goto 语句的 3 种格式如下：

```
goto【标签】
goto case【参数表达式】
goto default;
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Goto。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用于显示数组中的图书信息；添加一个 GroupBox 控件，向 GroupBox 控件中添加一个 TextBox 控件，用于输入信息；向 GroupBox 控件中添加一个 Button 控件，用于开始搜索图书信息。

（3）程序的主要代码如下：

```
private void btn_query_Click(object sender, EventArgs e)
{
    int i = 0;                                                          //定义计数器
label1:                                                                 //定义标签
    if (G_str_array[i].Contains(txt_query.Text))                        //判断是否找到图书
    {
        lbox_str.SelectedIndex = i;                                     //选中查找到的结果
        MessageBox.Show(txt_query.Text + " 已经找到！", "提示！");        //提示找到信息
        return;
    }
    i++;
    if (i < G_str_array.Length) goto label1;                            //条件满足则跳转到标签
    MessageBox.Show(txt_query.Text + " 没有找到！", "提示！");           //提示未找到信息
}
```

秘笈心法

心法领悟 034：不要轻易使用 goto 语句。

从上面的实例中可以看到，使用 goto 语句可以很方便地跳转到定义标签所在的位置，但是在这里建议大家，不要过多使用 goto 语句，由于 goto 语句过于灵活，或多或少地会干扰程序的执行逻辑。

第 3 章

字符串处理技术

- 字符及字符串转换
- 常用字符及字符串操作
- 常用数字处理技术
- 日期时间格式的数据处理
- 常用数字验证技巧
- 常用字符验证技巧
- 网络验证应用技巧

3.1　字符及字符串转换

实例 035	将字母全部转换为大写或小写 光盘位置：光盘\MR\03\035	初级 趣味指数：★★★★☆

实例说明

在程序设计过程中，经常会遇到一种情况，在验证用户登录时，如果用户名不区分大小写，那么在代码中应当使用一种方法排除字母大小写的因素，然后再对比数据库中的用户名与用户输入的用户名是否相等。可以先将数据库中的用户名全部转为大写，再将用户输入的用户名转为大写，最后对比是否相等。本实例中所介绍的技术可以方便地将字符串中的字母全部转换为大写或小写。实例运行效果如图 3.1 所示。

图 3.1　将字母全部转换为大写或小写

关键技术

本实例实现时主要用到了字符串对象的 ToUpper 方法和 ToLower 方法，下面对其进行详细讲解。

使用字符串对象的 ToUpper 方法可以将字符串中的字母全部转换为大写，格式如图 3.2 所示。

图 3.2　调用字符串对象的 ToUpper 方法将字母全部转换为大写

从图 3.2 中可以看到，字符串对象调用 ToUpper 方法后，会返回一个将原字符串转换为大写的新字符串，并将新字符串的引用交给 P_str_book 变量。

使用字符串对象的 ToLower 方法可以将字符串中的字母全部转换为小写，格式如图 3.3 所示。

图 3.3　调用字符串对象的 ToLower 方法将字母全部转换为小写

从图 3.3 中可以看到，字符串对象调用 ToLower 方法后，会返回一个将原字符串转换为小写的新字符串，并将新字符串的引用交给 P_str_book 变量。

技巧：字符串在创建后就成为不可变的对象，当调用字符串对象的方法操作字符串时，会产生新的字符串对象，而不是更改原来的字符串对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ToUpperOrLower。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入和

输出字符串信息；添加两个 RadioButton 单选按钮，分别用于选择字符串转换为大写或小写；添加一个 Button 控件，用于执行字符串转换。

（3）程序的主要代码如下：

```
private void btn_change_Click(object sender, EventArgs e)
{
    if (rbtn_upper.Checked)                                          //判断字符串转换为大写或小写
    {
        txt_changed.Text =                                           //将字符串转换为大写
            txt_string.Text.ToUpper();
    }
    else
    {
        txt_changed.Text = txt_string.Text.ToLower();                //将字符串转换为小写

    }
}
```

秘笈心法

心法领悟 035：字符串是不可变的对象。

在深入使用字符串之前，有一个概念一定要理解，字符串是不可变的对象。理解了这一概念，对后面熟练使用字符串有着很大的帮助。字符串的不可变性，意味着每当对字符串进行操作时，都将产生一个新的字符串对象，如果频繁地操作字符串对象，会在托管堆中产生大量的无用字符串，增加垃圾收集器的压力，从而造成系统资源的浪费。

实例 036　字母与 ASCII 码的转换

光盘位置：光盘\MR\03\036　　初级　　趣味指数：★★★★☆

实例说明

ASCII（American Standard Code for Information Interchange，美国信息互换标准代码）是基于拉丁字母的编码系统，也是现今最通用的单字节编码系统。在程序设计中，可以方便地将字母转换为 ASCII 码，也可以将 ASCII 码方便地转换为字母。实例运行效果如图 3.4 所示。

关键技术

本实例实现时主要用到了 Encoding 对象的 GetBytes 方法，下面对其进行详细讲解。

Encoding 对象的 GetBytes 方法接收一个字符串或字符数组作为参数，最后返回字节数组，可以根据字节数组得到字母的 ASCII 码，如图 3.5 所示。

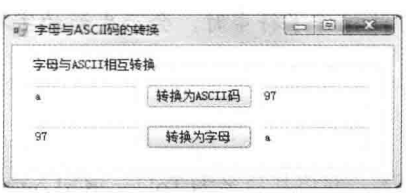


图 3.4　字母与 ASCII 码的转换

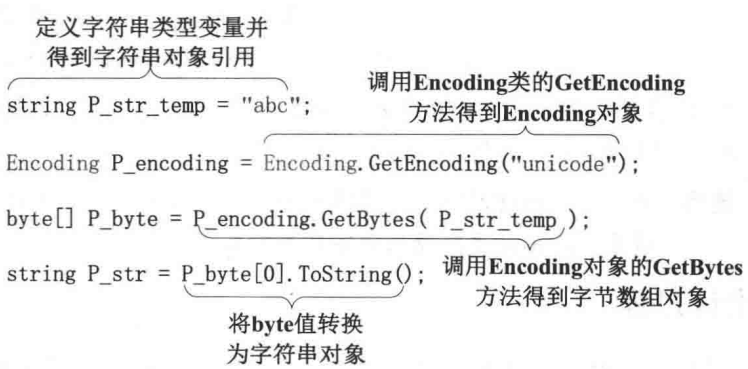


图 3.5　根据字节数组得到字母的 ASCII 码

从图 3.5 中可以看到，代码中首先使用 Encoding 类的 GetEncoding 静态方法得到了 Encoding 对象，然后调用 Encoding 对象的 GetBytes 方法，该方法接收一个字符串或字符数组作为参数，最后 GetBytes 方法会返回字节数组对象，可以根据字节数组的第 0 个索引来得到字符串中第一个字母的 ASCII 码。

注意：字符 Char 是值类型，它总是表示成 16 位 Unicode 代码值。

现在已经了解到 Char 是值类型，如果将 Char 显式转换为数值类型，可以方便地得到 ASCII 码值；相反，如果将 ASCII 码数值强制转换为 Char，将会得到一个 Char 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ASCII。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“字母与 ASCII 码的转换”，向窗体中添加一个 GroupBox 控件，向 GroupBox 控件中添加 4 个 TextBox 控件，分别用于输入和输出字符及 ASCII 码信息；向 GroupBox 控件中添加两个 Button 控件，分别用于将字母转换为 ASCII 码，或者将 ASCII 码转换为字母。

（3）程序的主要代码如下：

```
private void btn_ToASCII_Click(object sender, EventArgs e)
{
    if (txt_char.Text != string.Empty)                                    //判断输入是否为空
    {
        if (Encoding.GetEncoding("unicode").                              //判断输入是否为字母
            GetBytes(new char[] { txt_char.Text[0] })[1] == 0)
        {
            txt_ASCII.Text = Encoding.GetEncoding(                        //得到字符的 ASCII 码值
                "unicode").GetBytes( txt_char.Text ).ToString();
        }
        else
        {
            txt_ASCII.Text = string.Empty;                                //输出空字符串
            MessageBox.Show("请输入字母！ ","提示！ ");                    //提示用户信息
        }
    }
}
private void btn_ToChar_Click(object sender, EventArgs e)
{
    if (txt_ASCII2.Text != string.Empty)                                  //判断输入是否为空
    {
        int P_int_Num;                                                    //定义整型局部变量
        if (int.TryParse(                                                 //将输入的字符转换为数字
            txt_ASCII2.Text, out P_int_Num))
        {
            txt_Char2.Text =
                ((char)P_int_Num).ToString();                             //将 ASCII 码转换为字符
        }
        else
        {
            MessageBox.Show(                                              //如果输入不符合要求则弹出提示框
                "请输入正确的 ASCII 码值。", "错误！ ");
        }
    }
}
```

秘笈心法

心法领悟 036：将字母显式转换为数值会得到字符的 ASCII 码值。

已经知道 Char 是值类型，现在可以将字母显式转换为整数数值，从而方便地得到字母的 ASCII 码。同样地，可以将整数数值显式转换为 Char，从而得到字母。

实例 037 汉字与区位码的转换

光盘位置：光盘\MR\03\037　　中级　　趣味指数：★★★★☆

实例说明

区位码是一个 4 位的十进制数，每个区位码都对应着一个唯一的汉字，区位码的前两位叫做区码，后两位叫做位码。考生在填写高考信息表时会用到汉字区位码，如在报考志愿表中也需要填写汉字区位码，在程序设计中可以方便地将汉字转换为区位码。实例运行效果如图 3.6 所示。

关键技术

本实例重点介绍怎样从汉字字符中得到区位码，下面对其进行详细讲解。

得到汉字区位码的过程十分简单，首先通过 Encoding 对象的 GetBytes 方法得到汉字的字节数组，将字节数组的第一位和第二位分别转换为整型数值，然后将得到的两个整型数值分别减 160 后转换为字符串，连接两个字符串就组成了汉字区位码。转换区位码的过程如图 3.7 所示。

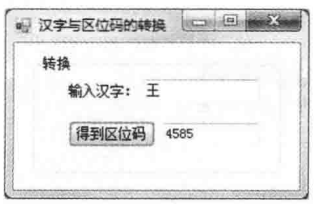


图 3.6　汉字与区位码的转换

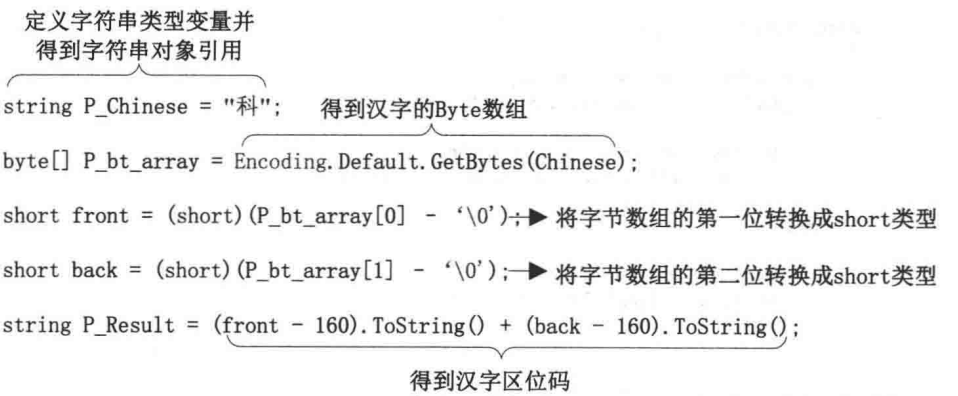


图 3.7　转换区位码的过程

注意：Encoding 对象的 GetBytes 方法提供了多个重载，可以接收字符串、字符数组等对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChineseCode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“汉字与区位码的转换”，向窗体中添加一个 GroupBox 控件，向 GroupBox 控件中添加两个 TextBox 控件，分别用于输入汉字信息和输出区位码信息；向 GroupBox 控件中添加一个 Button 控件，用于将汉字转换为区位码。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    if (txt_Chinese.Text != string.Empty)                          //判断输入是否为空
    {
        try
        {
            txt_Num.Text =                                         //得到汉字区位码信息
                getCode(txt_Chinese.Text);
        }
        catch (IndexOutOfRangeException ex)
        {
            MessageBox.Show(                                       //使用消息对话框提示异常信息
                ex.Message + "请输入正确的汉字", "出错！ ");
```

```
        }
    }
}
/// <summary>
///得到汉字区位码方法
/// </summary>
/// <param name="strChinese">汉字字符</param>
/// <returns>返回汉字区位码</returns>
public string getCode(string Chinese)
{
    string P_str_Code = "";
    byte[] P_bt_array = new byte[2];                                    //定义一个字节数组，用户存储汉字
    P_bt_array = Encoding.Default.GetBytes(Chinese);                    //为字节数组赋值
    int front = (short)(P_bt_array[0] - '\0');                          //将字节数组的第一位转换成 int 类型
    int back = (short)(P_bt_array[1] - '\0');                           //将字节数组的第二位转换成 int 类型
    P_str_Code = (front - 160).ToString() + (back - 160).ToString();    //计算区位码
    return P_str_Code;                                                  //返回区位码
}
```

秘笈心法

心法领悟 037：可以使用 FileStream 对象将字节数组写入文件。

使用 Encoding 对象的 GetBytes 方法可以得到字符串对象的字节数组，现在可以创建一个 FileStream 对象，方便地将字节数组写入文件中。同样地，可以从文件中读取字节数组，然后调用 Encoding 对象的 GetString 方法将字符数组转换为字符串。

实例 038　将汉字转换为拼音

光盘位置：光盘\MR\03\038　　中级　　趣味指数：★★★★☆

实例说明

我们经常使用拼音或五笔等输入法向文档中输入汉字。使用拼音输入法得到汉字的过程是，通过用户输入的拼音，输入法会智能地匹配到相应的汉字或汉字中的词并输出。本实例将会介绍一个很有趣的功能，即将汉字转换为拼音。实例运行效果如图 3.8 所示。

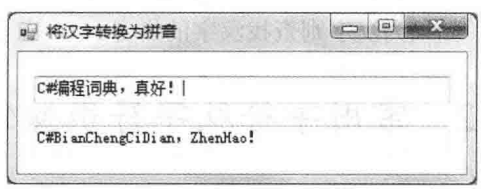


图 3.8　将汉字转换为拼音

关键技术

本实例重点介绍将汉字转换为拼音的技术，下面对其进行详细讲解。

在实例中使用了 PinYin 类，此类用于将汉字转换为拼音。下面对 PinYin 类使用的正则表达式和 ToCharArray 方法进行详细介绍。

在 C#中使用正则表达式，首先要创建正则表达式对象，正则表达式对象的构造方法中接收一个正则表达式字符串，对象创建后，可以调用对象的 IsMatch 方法，对该方法中的字符串进行验证，如果验证成功将返回 true，如果不成功则返回 false，如图 3.9 所示。

调用字符串对象的 ToCharArray 方法，可以根据当前的字符串对象中的字符内容，产生一个新的字符数组。字符串对象的 ToCharArray 方法如图 3.10 所示。

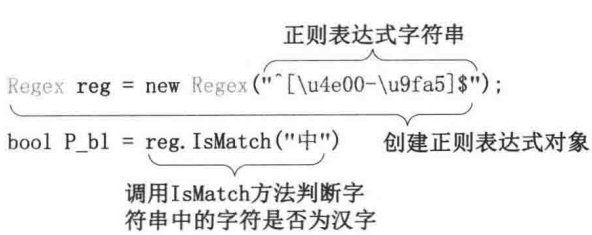


图 3.9　使用正则表达式

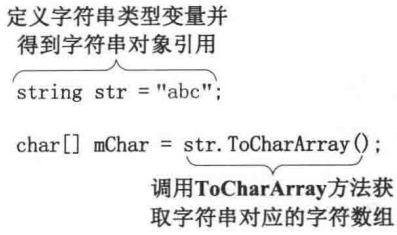


图 3.10　字符串对象的 ToCharArray 方法

注意：正则表达式经常用于数字或字符串等信息的验证及截取。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChineseToABC。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入汉字信息和输出拼音信息。

（3）程序的主要代码如下：

```
/// <summary>
///将汉字转换为拼音的方法
/// </summary>
/// <param name="str">汉字字符串</param>
/// <returns>拼音字符串</returns>
public string GetABC(string str)
{
    Regex reg = new Regex("^[\u4e00-\u9fa5]$");                    //验证输入是否为汉字
    byte[] arr = new byte[2];                                      //定义字节数组
    string pystr = "";                                             //定义字符串变量并添加引用
    char[] mChar = str.ToCharArray();                              //获取汉字对应的字符数组
    return GetStr(mChar,pystr,reg,arr);                            //返回获取到的汉字拼音
}
```

秘笈心法

心法领悟 038：适当地使用正则表达式，可以让字符串的处理变得更轻松。

使用正则表达式，可以方便地操作字符串，对字符串进行验证或提取。在本实例中，使用正则表达式验证字符串中的每一个字符是否为汉字，如果为汉字则查找汉字的拼音。

3.2　常用字符及字符串操作

实例 039	根据标点符号对字符串进行分行	初级
	光盘位置：光盘\MR\03\039	趣味指数：★★★☆

实例说明

在了解本实例前，首先需要理解一个概念——字符串是不可改变的对象，也就是说，字符串在创建以后，就不会被改变，当使用字符串对象的 Replace、split 或 Remove 等方法操作字符串时，实际上是产生了一个新的字符串对象，原有的字符串如果没有被引用，将会被垃圾收集器回收。如果频繁地使用字符串中的方法对字符串进行操作，会产生大量的没有被引用的字符串对象，这会增加垃圾收集的压力，造成系统资源的浪费。如果需要大量地操作字符串，怎样操作才合理呢？可以使用 StringBuilder 有效地解决上面出现的问题，使用 StringBuilder 操作字符串不会产生新的字符串对象，这样处理才更加方便有效。本实例的操作流程是，首先创建 StringBuilder 对象，使用 StringBuilder 对象对字符串进行分行操作。实例运行效果如图 3.11 所示。

关键技术

本实例重点介绍怎样使用 StringBuilder 便捷、高效地操作字符串，下面对其进行详细讲解。

由于字符串的不可变性，使用 StringBuilder 操作字符串无疑是非常方便和有效的方法。StringBuilder 对象的使用方法如图 3.12 所示。

图 3.11　根据标点符号对字符串进行分行

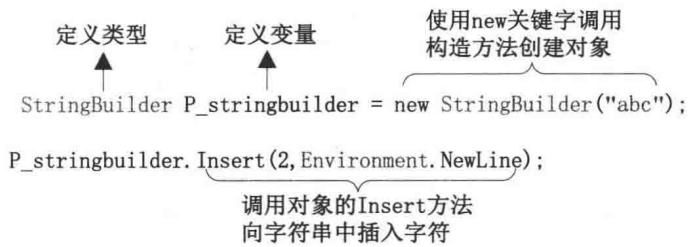


图 3.12　StringBuilder 对象的使用方法

从图 3.12 中可以看到，建立 StringBuilder 对象后，可以调用操作字符串的方法，从而方便操作字符串对象。

技巧：适当地使用 StringBuilder 操作字符串，会使程序运行更加高效。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Lines。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入字符串信息和输出分行后的字符串信息；添加一个 Button 控件，用于处理字符串。

（3）程序的主要代码如下：

```
private void btn_true_Click(object sender, EventArgs e)
{
    StringBuilder P_stringbuilder =
        new StringBuilder(txt_string.Text);                    //创建字符串处理对象
    for (int i = 0; i < P_stringbuilder.Length; i++)           //开始循环
        if (P_stringbuilder[i] == ',')                         //判断是否出现（,）号
            P_stringbuilder.Insert(++i,                        //向字符串内添加换行符
                Environment.NewLine);
    txt_Lines.Text =                                           //得到分行后的字符串
        P_stringbuilder.ToString();
}
```

秘笈心法

心法领悟 039：适当地使用 StringBuilder 操作字符串。

字符串不可变的概念在实例 035 的秘笈心法中已经进行了详细介绍，如果调用字符串对象的方法频繁地操作字符串会产生大量的无用对象，从而浪费系统资源。使用 StringBuilder 对象操作字符串会有效地解决上述问题，因为使用 StringBuilder 操作字符串不会产生新的字符串对象，在使用 StringBuilder 对象前首先要引用命名空间 System.Text。

实例 040	判断用户输入的用户名是否正确	初级
	光盘位置：光盘\MR\03\040	趣味指数：★★★☆

实例说明

在程序的开发过程中，经常需要判断用户输入的用户名是否正确，可以通过对比用户输入的用户名字符串是否

与数据库中或者已经存在集合中的字符串相等，来决定用户输入的用户名是否正确。实例运行效果如图 3.13 所示。

关键技术

本实例重点介绍使用“==”运算符对比字符串是否相等，下面对其进行详细讲解。

可以使用“==”运算符判断两个数值是否相等（如图 3.14 所示），也可以使用对象的 Equals 方法判断两个引用类型是否引用同一个对象，在字符串操作中可以方便地使用运算符重载计算两个字符串是否相等。

图 3.13　判断用户输入的用户名是否正确

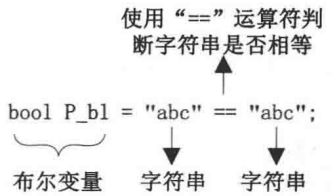


图 3.14　判断两个字符串是否相等

从图 3.14 中可以看到，使用了“==”运算符判断两个字符串是否相等。结果会返回一个布尔值，如果相等返回 true，如果不相等则返回 false。

说明：实例中介绍了怎样使用“==”运算符判断两个字符串是否相等，还可以使用“+”运算符方便地连接两个字符串的内容形成一个新的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UserName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入用户名；添加一个 Button 控件，用于验证用户信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (txt_username.Text == "admin")                          //判断是否输入正确用户名
    {
        MessageBox.Show("登录成功！", "提示！");                 //提示登录成功
    }
    else
    {
        MessageBox.Show("用户名错误","错误！");                  //提示登录不成功
    }
}
```

秘笈心法

心法领悟 040：怎样理解字符串留用？

因为字符串是不可变的对象，如果多个字符串变量引用多个内容相同的字符串，这样可能会浪费一些内存资源；如果在托管堆中只保留一个字符串对象，这样会更加有效地利用内存。字符串留用的原理就是这样，所有内容相同的字符串只有一个字符串实例，如 string P_str="abc"与 string P_str2="abc"两个字符串变量引用的是同一个字符串对象。

实例 041	将字符串的每个字符进行颠倒输出	中级
	光盘位置：光盘\MR\03\041	趣味指数：★★★★★

实例说明

可以使用很多方法对字符串进行操作，可以使用字符串对象的 Insert 方法向字符串中插入新的字符串，使

用 Remove 方法删除字符串对象中的一部分字符，使用 Replace 方法替换字符串对象中的字符串，也可以使用 IndexOf 方法查找字符串中指定字符或字符串的索引，本实例将会使用一种方法对字符串进行颠倒输出。实例运行效果如图 3.15 所示。

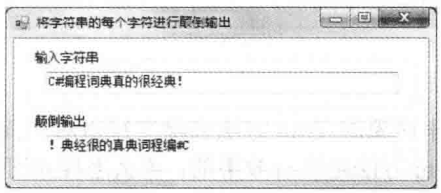


图 3.15　将字符串的每个字符进行颠倒输出

关键技术

本实例重点介绍怎样使用 Array 类的 Reverse 静态方法反转数组元素，下面对其进行详细讲解。Reverse 静态方法的使用如图 3.16 所示。

从图 3.16 中可以看到，Array 类的 Reverse 方法中接收一个数组对象作为参数，使用 Reverse 方法可以方便地反转数组元素。Reverse 方法还提供了一个重载，可以反转数组内指定的一部分元素，如图 3.17 所示。

图 3.16　Reverse 静态方法的使用

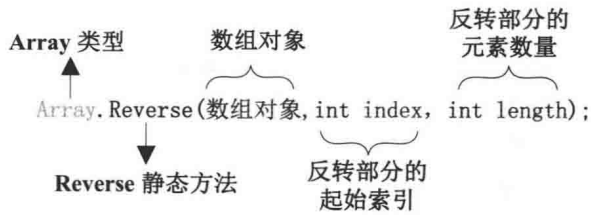


图 3.17　Reverse 静态方法的重载

从图 3.17 中可以看到，使用了 Array 类的 Reverse 方法重载，该方法重载接收一个数组对象和两个整型数值，两个整型数值分别表示反转数组元素的起始索引和反转数组的元素数量。

技巧：使用 Array 类的 Reverse 静态方法只可以反转数组元素，如果要反转集合中的元素，可以使用集合对象的 Reverse 方法，方便地反转集合中的元素。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReverseStr。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入字符串和输出颠倒字符串。

（3）程序的主要代码如下：

```
private void txt_input_TextChanged(object sender, EventArgs e)
{
    char[] P_chr=txt_input.Text.ToCharArray();                    //从字符串中得到字节数组
    Array.Reverse(P_chr, 0, txt_input.Text.Length);               //反转字节数组
    txt_output.Text =
        new StringBuilder().Append(P_chr).ToString();             //将字节数组转换为字符串并输出
}
```

秘笈心法

心法领悟 041：字符串是一组不可变的字符数组。

实例中一直在强调字符串是不可变的对象，而字符串本身则是一个字符数组，可以使用字符串对象的 Length 属性获取字符数组的长度，而且字符串对象也可以使用索引的方式访问其中每一个字符。

实例 042　去掉字符串中的所有空格

光盘位置：光盘\MR\03\042　　初级　　趣味指数：★★★☆

实例说明

在字符串操作中，可以使用字符串对象的 Trim 方法去除字符串对象前端和后端的所有空格，但是，如果空格在字符串的中间位置出现，使用 Trim 方法是没有效果的，那么怎样才可以有效地去除空格呢？这正是本实例的实现目的。实例运行效果如图 3.18 所示。

关键技术

本实例重点介绍怎样使用字符串对象的 ToCharArray 方法将字符串中的字符复制到 Unicode 字符数组，以及方便地使用迭代器遍历数组中的元素，下面对其进行详细讲解。

在字符串的操作中，可以方便地使用字符串对象的 ToCharArray 方法将字符串中的字符复制到 Unicode 字符数组，然后对 Unicode 字符数组的元素进行操作。字符串对象的 ToCharArray 方法如图 3.19 所示。

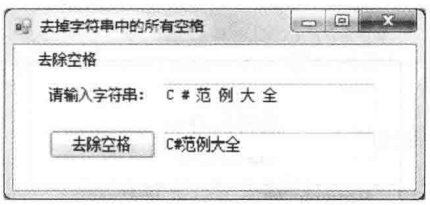


图 3.18　去掉字符串中的所有空格

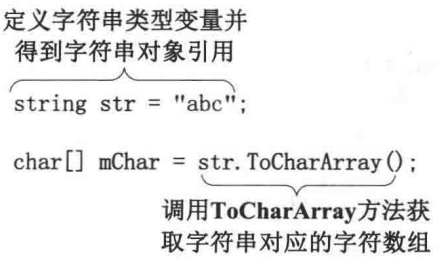


图 3.19　字符串对象的 ToCharArray 方法

从图 3.19 中可以看到，调用字符串对象的 ToCharArray 方法可以将字符串中的字符复制到 Unicode 字符数组，ToCharArray 方法已经在实例 038 中进行了详细介绍。

Array 类是所有数组的基类，可以使用 for 循环和 foreach 迭代器遍历数组元素。for 循环的工作原理是根据数组中元素的索引遍历数组中所有元素。由于所有数组都隐式实现 IEnumerable 接口，所以可以方便地使用 foreach 迭代器遍历数组元素。迭代器的内部工作原理如图 3.20 所示。

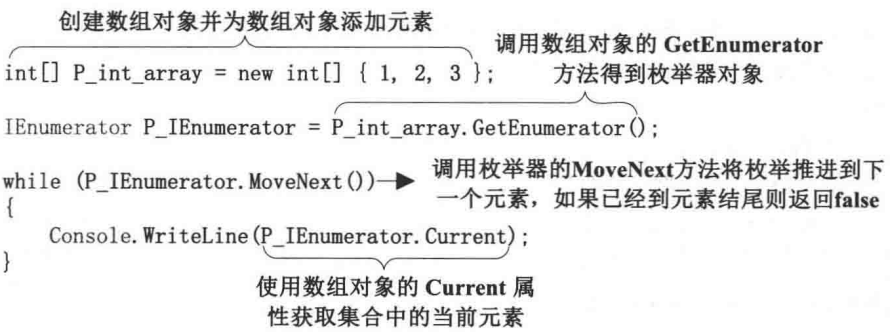


图 3.20　迭代器的内部工作原理

从图 3.20 中可以看到，首先使用了数组对象的 GetEnumerator 方法返回枚举器对象，然后使用枚举器对象的 MoveNext 方法枚举每一个数组元素，最后通过 Current 属性获取数组中的每一个元素。

说明：有一个概念一定要理解，由于数组元素隐式地实现了 IEnumerable 接口，所以可以使用 foreach 迭代器遍历数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RemoveBlank。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入字符串和输出去除空格的字符串；添加一个 Button 控件，用于处理字符串信息。

（3）程序的主要代码如下：

```
private void btn_RemoveBlank_Click(object sender, EventArgs e)
{
    char[] P_chr = txt_str.Text.ToCharArray();                          //得到字符数组
    IEnumerator P_ienumerator_chr =                                     //得到枚举器
        P_chr.GetEnumerator();
    StringBuilder P_stringbuilder =                                     //创建 stringbuilder 对象
        new StringBuilder();
    while (P_ienumerator_chr.MoveNext())                                //开始枚举
    {
        P_stringbuilder.Append(                                         //向 stringbuilder 对象中添加非空格字符
            (char)P_ienumerator_chr.Current != ' ' ?
            P_ienumerator_chr.Current.ToString() : string.Empty);
    }
    txt_removeblank.Text =                                              //得到没有空格的字符串
        P_stringbuilder.ToString();
}
```

秘笈心法

心法领悟 042：使用迭代器遍历集合更轻松。

可以使用 for 循环根据数组或集合的索引方便地遍历数组或集合的每一个元素。由于数组或集合都实现了 IEnumerable 接口，所以可以使用 foreach 迭代器遍历数组的每一个元素。与 for 循环根据索引访问元素有所不同，foreach 会枚举出集合中的每一个元素。

实例 043　从字符串中分离文件路径、文件名及扩展名　中级

光盘位置：光盘\MR\03\043　　趣味指数：★★★★☆

实例说明

对文件进行操作时，首先要得到文件路径信息，然后创建文件对象，通过 IO 流将数据读取到内存中并进行处理。在操作文件过程中可能还需要提取文件的一些信息，如文件的路径、文件名及文件扩展名。通过实例中介绍的方法，可以方便地得到上述信息。实例运行效果如图 3.21 所示。

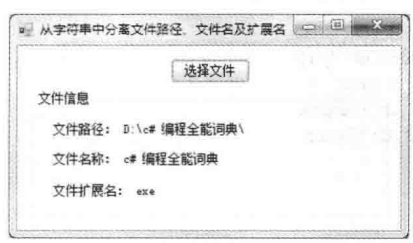


图 3.21　从字符串中分离文件路径、文件名及扩展名

关键技术

本实例重点介绍怎样使用字符串对象的 Substring 方法截取字符串，以及使用 LastIndexOf 方法查找字符或字符串在指定字符串中的索引，下面对其进行详细讲解。

在字符串操作中，可以调用字符串对象的 Substring 方法截取字符串中的一部分字符串。字符串对象的

Substring 方法如图 3.22 所示。

从图 3.22 中可以看到，字符串对象调用 Substring 方法截取字符串对象的一部分内容产生一个新的字符串。0 代表开始截取的初始索引位置，2 代表所截取的字符串长度。

使用字符串对象的 LastIndexOf 方法可以搜索指定内容在字符串对象中最后一个匹配项的索引位置。字符串对象的 LastIndexOf 方法如图 3.23 所示。

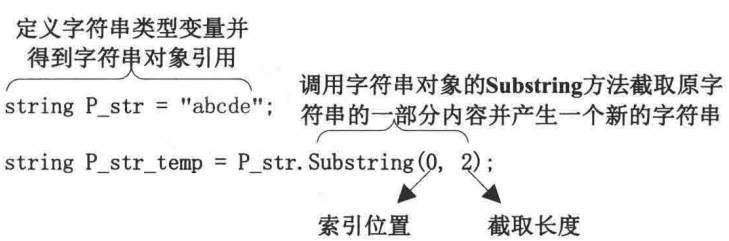


图 3.22　字符串对象的 Substring 方法

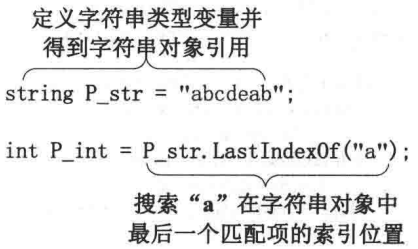


图 3.23　字符串对象的 LastIndexOf 方法

从图 3.23 中可以看到，字符串对象的 LastIndexOf 方法返回一个整数数值，该数值是搜索内容在字符串对象中的最后一个匹配项的索引，如果没有搜索到指定内容，那么会返回-1。

技巧：在字符串操作中，字符串对象可以调用 IndexOf 与 LastIndexOf 方法方便地查找指定字符及字符串在字符串对象中的索引。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FilePathString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 Label 控件，分别用于输出文件的路径、名称和扩展名；添加一个 Button 控件，用于处理字符串中的路径信息。

（3）程序的主要代码如下：

```
private void btn_Openfile_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)                    //判断是否选择了文件
    {
        string P_str_all = openFileDialog1.FileName;                        //记录选择的文件全路径
        string P_str_path =                                                 //获取文件路径
            P_str_all.Substring(0, P_str_all.LastIndexOf("\\") + 1);
        string P_str_filename =                                             //获取文件名
            P_str_all.Substring(P_str_all.LastIndexOf("\\") + 1,
            P_str_all.LastIndexOf(".") -
            (P_str_all.LastIndexOf("\\")+1));
        string P_str_fileexc =                                              //获取文件扩展名
            P_str_all.Substring(P_str_all.LastIndexOf(".")+1,
            P_str_all.Length - P_str_all.LastIndexOf(".")-1);
        lb_filepath.Text = "文件路径：  " + P_str_path;                      //显示文件路径
        lb_filename.Text = "文件名称：  " + P_str_filename;                  //显示文件名
        lb_fileexc.Text = "文件扩展名：  " + P_str_fileexc;                  //显示扩展名
    }
}
```

秘笈心法

心法领悟 043：IndexOf 与 LastIndexOf 方法有什么不同？

使用 IndexOf 方法与 LastIndexOf 方法都可以用来查找字符或字符串在指定字符串对象中的索引，如果未找到匹配的字符或字符串则会返回-1。IndexOf 与 LastIndexOf 不同之处在于，IndexOf 方法从字符串对象的前端向后端查找第一个匹配项的索引，而 LastIndexOf 方法从字符串对象的后端向前端查找第一个匹配项的索引。

实例 044　获取字符串中汉字的个数

光盘位置：光盘\MR\03\044　　中级　　趣味指数：★★★★☆

实例说明

字符串中可以包括数字、字母、汉字或者其他字符。使用 Char 类型的 IsDigit 静态方法可以判断字符串中的某个字符是否为数字，使用 Char 类型的 IsLetter 静态方法可以判断字符串中的某个字符是否为字母。实例中将介绍一种方法用来判断字符串中的某个字符是否为汉字，通过此方法可以计算字符串中汉字的数量。实例运行效果如图 3.24 所示。

关键技术

本实例重点介绍怎样使用正则表达式判断字符是否为汉字，以及了解字符串是一组不可变的字符集的概念，下面对其进行详细讲解。

可以使用正则表达式判断指定的字符是否为汉字，正则表达式已经在实例 038 中详细介绍过，这里不再过多叙述。

字符串是一组不可变的字符集，可以使用索引访问字符串中的每一个字符，如图 3.25 所示。

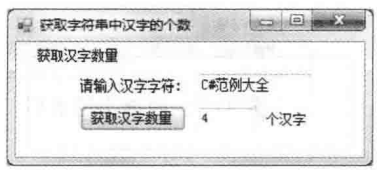


图 3.24　获取字符串中汉字的个数

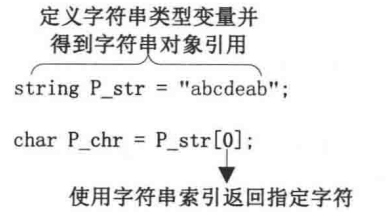


图 3.25　使用字符串的索引返回指定字符

从图 3.25 中可以看到，使用索引返回字符串中指定字符‘a’。

注意：字符串对象的索引是只读的，只可以读取字符串对象中的字符，不可以根据索引更改字符串中的字符。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetCount。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入字符串信息和输出字符串中包含汉字的数量；添加一个 Button 控件，用于计算字符串中的汉字数量。

（3）程序的主要代码如下：

```
private void btn_GetCount_Click(object sender, EventArgs e)
{
    int P_scalar = 0;                                          //定义值类型变量并赋值为 0
    Regex P_regex =                                            //创建正则表达式对象，用于判断字符是否为汉字
        new Regex("^[\u4E00-\u9FA5]{0,}$");
    for (int i = 0; i < txt_str.Text.Length; i++)              //遍历字符串中的每一个字符
    {
        P_scalar =                                             //如果检查的字符是汉字则计数器加 1
            P_regex.IsMatch(txt_str.Text[i].
            ToString()) ? ++P_scalar : P_scalar;
    }
    txt_count.Text = P_scalar.ToString();                      //显示汉字数量
}
```

秘笈心法

心法领悟 044：使用正则表达式判断字符是否为汉字非常方便。

使用正则表达式可以非常方便地操作字符串，经常用来验证用户输入的信息。例如，可以判断用户输入的手机号码格式是否正确，验证用户输入的身份证号码格式是否正确。本实例中使用正则表达式来判断字符串中的字符是否为汉字，如果是汉字则计数器加 1，最后得到字符串中所有汉字的数量。

实例 045 批量替换某一类字符串

光盘位置：光盘\MR\03\045　　高级　　趣味指数：★★★★★

实例说明

在字符串操作中，可以使用字符串对象的 Split 方法拆分字符串，还可以使用字符串对象的 Substring 方法截取一部分字符串。字符串对象提供了很多方便实用的方法，本实例中将介绍使用字符串对象的 Replace 方法替换某一类字符串。实例运行效果如图 3.26 和图 3.27 所示。

图 3.26　替换字符串前

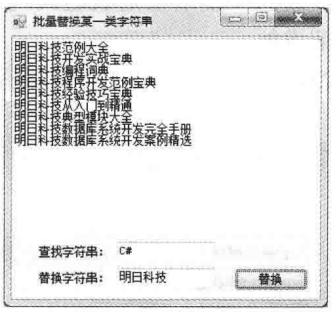


图 3.27　替换字符串后

关键技术

本实例重点介绍字符串对象的 Replace 方法的使用，下面对其进行详细讲解。

使用字符串对象的 Replace 方法可以方便地替换字符串中指定的内容。由于字符串是不可变的，Replace 方法会返回一个新的字符串对象。字符串对象的 Replace 方法如图 3.28 所示。

从图 3.28 中可以看到，字符串对象调用了 Replace 方法，该方法会返回一个被替换内容的新字符串。新字符串对象被 P_str_temp 变量引用。

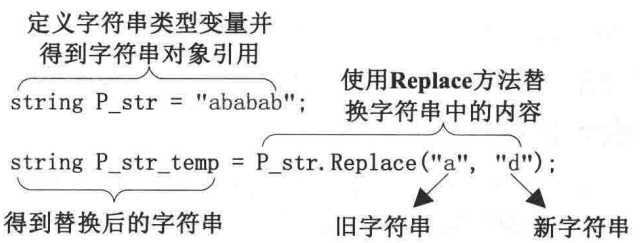


图 3.28　字符串对象的 Replace 方法

注意：由于字符串是不可变的，所以图 3.28 中的 P_str 对象调用了 Replace 方法后，P_str 中的字符串应当为“ababab”没有变化，而 P_str_temp 中的字符串为“dbdbdb”。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReplaceString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 TextBox 控件，分别用于输入字符串信息、输入查找信息和输入替换信息；添加一个 Button 控件，用于字符串替换。

（3）程序的主要代码如下：

```
private void btn_replace_Click(object sender, EventArgs e)
{
    txt_str.Text =                                          //使用字符串对象的 Replace 方法替换字符串
        txt_str.Text.Replace(txt_find.Text, txt_replace.Text);
}
```

秘笈心法

心法领悟 045：Replace 方法可以批量替换字符串。

调用字符串对象的 Replace 方法可以方便地替换字符串中指定的内容。有一点要注意，Replace 方法并不是只替换掉一个匹配的字符串，而是一次性替换掉所有匹配的字符串，由于字符串是不可变的，Replace 方法会返回一个新的字符串对象。

实例 046　对字符串进行加密与解密　高级

光盘位置：光盘\MR\03\046　　趣味指数：★★★★★

实例说明

在程序开发过程中，字符串是较为常见的数据类型，经常被用于存放文本信息，如果在字符串中保存一些敏感信息，不希望其他人看到，要怎样才能做到最安全呢？本实例中将介绍一种方法，可以方便地对字符串进行加密和解密，使字符串中的信息更加安全。实例运行效果如图 3.29 所示。

图 3.29　对字符串进行加密与解密

关键技术

本实例重点介绍使用 CryptoStream 类加密和解密字符串，下面对其进行详细讲解。

在使用 CryptoStream 前要先引用命名空间 using System.Security.Cryptography，CrytoStream 的加密方法如图 3.30 所示。

```
                    创建内存流对象
MemoryStream P_Stream_MS = new MemoryStream();
CryptoStream P_CryptStream_Stream = new CryptoStream(  ──▶ 创建加密流对象
                P_Stream_MS,  ──▶ 内存流对象
加密流构造方法中的3个参数
                                                  对称算法密钥
                new DESCryptoServiceProvider().CreateEncryptor(P_byte_key, P_byte_key),──▶ 加密器对象
                                                                  对称算法初始化向量
                CryptoStreamMode.Write);──▶ 对加密流写访问

P_CryptStream_Stream.Write(P_byte_data, 0, P_byte_data.Length);──▶ 向加密流中写入数据

P_CryptStream_Stream.FlushFinalBlock();──▶ 将数据压入基础流

byte[] P_bt_temp = P_Stream_MS.ToArray();──▶ 得到加密信息的字节数组

string P_str = Convert.ToBase64String(P_bt_temp);──▶ 得到加密字符串
```

图 3.30　使用 CryptoStream 对字符串加密

从图 3.30 中可以看到，字符串加密的过程是首先创建内存流对象，然后将内存流作为参数创建加密流对象，

通过 Write 方法向加密流写入数据以实现加密目的，最后从内存流中取出加密后的数据。现在已经大体了解了字符串加密的过程，下面再来了解一下数据解密的过程。CryptoStream 的解密方法如图 3.31 所示。

```
byte[] P_byte_data = Convert.FromBase64String(str);  ——▶ 将加密后的字符串转换为字节序列

MemoryStream P_Stream_MS = new MemoryStream(P_byte_data);——▶创建内存流对象并写入数据

CryptoStream P_CryptStream_Stream = new CryptoStream(  ——▶创建加密流对象
    P_Stream_MS,  ——▶内存流对象
    new DESCryptoServiceProvider().CreateDecryptor(P_byte_key, P_byte_key),——▶解密器对象
    CryptoStreamMode.Read);  ——▶对加密流读访问

byte[] P_bt_temp = new byte[200];——▶创建字节序列对象

MemoryStream P_MemoryStream_temp = new MemoryStream();  ——▶ 创建内存流对象

int i = 0;——▶创建计数器

while ((i = P_CryptStream_Stream.Read(P_bt_temp, 0, P_bt_temp.Length)) > 0)——▶得到解密数据
{
    P_MemoryStream_temp.Write(P_bt_temp, 0, i);——▶ 将解密后的数据放入内存流
}
string P_str = Encoding.Unicode.GetString(P_MemoryStream_temp.ToArray());——▶ 得到解密后的字符串
```

加密流构造方法中的3个参数

对称算法密钥

对称算法初始化向量

图 3.31　使用 CryptoStream 对字符串解密

从图 3.31 中可以看到，字符串解密过程首先将加密信息写入内存流，然后根据内存流作为参数创建加密流对象，最后通过加密流读取解密数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StringEncrypt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 6 个 TextBox 控件，分别用于显示和输入加密解密信息；添加两个 Button 控件，分别用于加密和解密字符串。

（3）程序的主要代码如下：

```
private void btn_Encrypt_Click(object sender, EventArgs e)
{
    if (txt_password.Text.Length == 4)                          //判断加密密钥长度是否正确
    {
        try
        {
            txt_EncryptStr.Text =                               //调用实例 ToEncrypt 方法得到加密后的字符串
                new Encrypt().ToEncrypt(
                txt_password.Text, txt_str.Text);
        }
        catch (Exception ex)                                    //捕获异常
        {
            MessageBox.Show(ex.Message);                        //输出异常信息
        }
    }
    else
    {
        MessageBox.Show("密钥长度不符！", "提示");                  //提示用户输入密钥长度不正确
    }
}
```

秘笈心法

心法领悟 046：内存流操作很重要。

MemoryStream 内存流在程序设计中占有着很重要的位置。例如，进行克隆操作时，可以先将对象序列化到

内存流中，然后再将对象反序列化出来，得到一个与原来对象完全相同的新对象。

3.3　常用数字处理技术

实例 047	判断输入的货币值是否为数字 光盘位置：光盘\MR\03\047	初级 趣味指数：★★★☆

实例说明

在软件运行过程中，经常需要用户输入数值、货币值等信息，然后进行处理。如果用户输入了非法的信息，如在货币值中输入了字母“a”，那么在运行时会抛出异常，可以通过捕获异常来判断输入信息是否合法。但是，这样并不是好的处理方法，本实例中将介绍一个方便快捷的方法处理此问题。实例运行效果如图 3.32 所示。

图 3.32　判断输入的货币值是否为数字

关键技术

本实例重点介绍将字符串类型转换为数值类型所使用的 TryParse 方法，下面对其进行详细讲解。

TryParse 方法首先会尝试将字符串转换为指定数值，如果转换成功，TryParse 方法会返回 true，同时会向参数中的变量赋值；如果转换失败，TryParse 会返回 false。语法格式如下：

```
double.TryParse(string s,out double result)
```

TryParse 方法中的参数说明如表 3.1 所示。

表 3.1　TryParse 方法中的参数说明

参　数	描　述
double	双精度数值
s	用于转换为数值的字符串
out	使用 out 关键字，会将参数通过引用来传递。要注意，如果使用 out 关键字传递参数，那么此参数在传递前不可以被初始化
result	如果转换成功，result 将得到转换后的数值

说明：由于 TryParse 方法使用时避免了异常的出现，所以适当地使用 TryParse 方法是一个很好的选择。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TryNum。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入货币信息；添加一个 Button 控件，用于判断输入信息是否为货币信息。

（3）程序的主要代码如下：

```
private void btn_true_Click(object sender, EventArgs e)
{
    double P_dbl_value;                                          //定义变量
    if (double.TryParse(                                         //判断输入信息是否正确
```

```
        txt_value.Text, out P_dbl_value))
    {
        MessageBox.Show("输入的数值正确！");                                    //提示正确信息
    }
    else
    {
        MessageBox.Show(                                                        //提示错误信息
            "输入的数值有误，请重新输入！","提示！");
    }
}
```

秘笈心法

心法领悟 047：使用 TryParse 方法可以有效防止数值转换中的异常。

所有数值类型都有 Parse 方法，此方法可以方便地将字符串对象转换为数值，但是，如果字符串中的信息不是标准的数值信息，则在转换过程中将会出现异常。从出现异常到捕获异常最后处理异常，这些都是要耗费系统资源的。而 TryParse 方法的出现就是为了防止转换过程中出现的异常，该方法将会返回一个布尔值，如果转换成功返回 true，如果转换不成功则返回 false，并不会抛出异常。

实例 048　对计算结果进行四舍五入

光盘位置：光盘\MR\03\048　　初级　　趣味指数：★★★★

实例说明

在了解了算术运算符后，可以方便地对数值进行加、减、乘、除等操作。如果对运算结果进行四舍五入，可能会想到 Math 类中的 Round 方法，但是此方法并不适合标准的四舍五入，因为数值 3.4445 如果使用 Round 方法保留两位小数，那么得到的结果为 3.44，显然，这不是我们想要的，本实例中将使用一种特殊的方法对数值进行四舍五入。实例运行效果如图 3.33 所示。

图 3.33　对计算结果进行四舍五入

关键技术

本实例重点介绍对小数数值进行四舍五入的方法，下面对其进行详细讲解。

对小数数值进行四舍五入，首先应当确认保留小数位，如果数值的小数精度大于保留小数位，那么开始四舍五入计算。四舍五入的方法非常简单，在所有要丢失精度的小数位中加 5，如果大于 10 则向前进位，最后计算出四舍五入的结果。

技巧：在程序设计过程中，适当地使用 TryParse 方法可以有效地避免数值转换过程中出现的异常。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Round。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 TextBox 控件，分别用于输入和输出计算结果；添加一个 Combobox 控件，用于设置计算结果保留的小数位数；添加一个 Button 控件，用于计算两个数值相加的结果。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    double P_dbl_d1, P_dbl_d2;                                          //定义两个 double 类型的变量
    if (double.TryParse(txt_add1.Text, out P_dbl_d1) &&                 //判断输入信息是否正确
```

```
        double.TryParse(txt_add2.Text, out P_dbl_d2))
    {
        txt_add3.Text =                                          //得到四舍五入后的值
            Round(P_dbl_d1 + P_dbl_d2,
            cbox_select.SelectedIndex + 1).ToString();
    }
    else
    {
        MessageBox.Show(                                         //提示错误信息
            "输入数值不正确！", "提示！");
    }
}
```

秘笈心法

心法领悟 048：了解 Math 类的 Round 静态方法。

Math 类中的 Round 方法并不适合标准的四舍五入，它只是简单的将保留小数位数后面的数去掉。

实例 049　商品金额的大小写转换　　中级

光盘位置：光盘\MR\03\049　　趣味指数：★★★★☆

实例说明

数字“123”对应着大写金额的“壹贰叁”，在程序设计过程中，经常需要对金额进行运算，最后输出结果。在输出金额的过程中，可以使用本实例所介绍的知识，将小写金额转换为大写金额并输出。实例运行效果如图 3.34 所示。

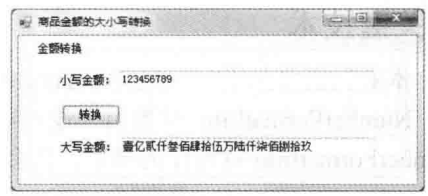


图 3.34　商品金额的大小写转换

关键技术

本实例重点介绍商品金额的大小写转换的方法，下面对其进行详细讲解。

实例中将商品金额转换为大写的方法定义在 Upper 类中，首先将小写金额分 4 位截取，分别将 4 位数替换为大写并在大写金额中添加位修饰，如拾、佰、仟、万、十万等。

说明：理解本实例前，需要熟练掌握字符串处理，以及数值的类型转换，这样有助于读者理解程序代码中大写金额的转换过程。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Transform。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入小写金额和输出大写金额；添加一个 Button 控件，用于转换金额信息。

（3）程序的主要代码如下：

```
private void btn_transform_Click(object sender, EventArgs e)
{
    int P_int_temp;                                              //定义整型变量
    if (int.TryParse(txt_lower.Text,out P_int_temp))
    {
        txt_upper.Text =                                         //获取转换为大写金额的字符串
            new Upper().NumToChinese(txt_lower.Text);
    }
    else
    {
        MessageBox.Show(                                         //错误提示信息
```

```
            "请输入正确整数数值","提示！");
        }
    }
```

秘笈心法

心法领悟 049：object 类的 ToString 方法。

C#中的所有类型都隐式继承自 object 基类，由于 object 类定义了 ToString 的虚方法，所以所有对象都可以调用 ToString 方法，而且还可以在类中使用 override 关键字重写基类的 ToString 方法。

实例 050	设置货币值中使用的小数位数	高级
	光盘位置：光盘\MR\03\050	趣味指数：★★★★★

实例说明

在货币值运算中，经常需要截取小数位数。本实例将介绍一种简捷的方法，合理地截取货币值中的小数位数。需要注意的是，本实例中截取小数位数使用了类似于四舍五入的方法，但是与四舍五入又有些不同，截取小数位时，只计算保留小数位的后一位，如果小数值大于 4 则向上进一位；如果小于等于 4，则丢弃。实例运行效果如图 3.35 所示。

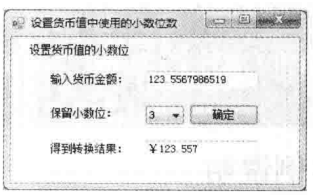


图 3.35　设置货币值中使用的小数位数

关键技术

本实例重点介绍 NumberFormatInfo 类的使用方法，下面对其进行详细讲解。

NumberFormatInfo 对象用来设置数值格式以及如何显示数值。当调用数值的 ToString 方法时，如果使用 NumberFormatInfo 对象作为参数，将会返回数值的货币形式的字符串。

注意：NumberFormatInfo 对象的 CurrencyDecimalDigits 属性可以设置货币值中使用的小数位数，在舍去小数位时，它只计算保留小数位的后一位是否大于 4，如果大于 4 则向前进一位，而不是对货币值的小数位进行标准的四舍五入。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetValue。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入货币金额和显示转换结果；添加一个 Combobox 控件，用于设置显示结果保留的小数位数；添加一个 Button 控件，用于计算转换结果。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    double P_dbl_value;                                              //定义 double 类型的变量
    if (double.TryParse(txt_input.Text, out P_dbl_value))            //判断输入是否正确
    {
        System.Globalization.NumberFormatInfo GN =                   //实例化 NumberFormatInfo 对象
            new System.Globalization.
                CultureInfo("zh-CN", false).NumberFormat;
        GN.CurrencyDecimalDigits = Convert.ToInt32(cbox_select.Text); //设置保留位数
        txt_output.Text = P_dbl_value.ToString("C", GN);             //将输入的小数转换为货币形式
    }
    else
    {
        MessageBox.Show("请输入正确的数值！","提示！");               //用户提示信息
    }
}
```

秘笈心法

心法领悟 050：数值类型都实现了 IFormattable 接口。

所有数值类型都实现了 IFormattable 接口，该接口中定义了 ToString 方法，所以所有数值类型都实现了这个 ToString 方法，该方法会使用指定的格式格式化当前数值。

实例 051　格式化输入数据为货币格式

光盘位置：光盘\MR\03\051　　初级　　趣味指数：★★★★

实例说明

如果开发商品管理系统，那么经常需要计算并显示商品金额，怎样将数值转换为货币类型呢？这正是本实例所要介绍的内容。通过调用数值对象的 ToString 方法，可以输出数值的货币格式。实例运行效果如图 3.36 所示。

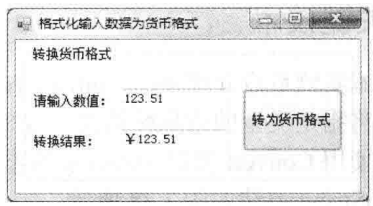


图 3.36　格式化输入数据为货币格式

关键技术

本实例重点介绍 NumberFormatInfo 对象的 CurrencyGroupSeparator 属性，下面对其进行详细讲解。

NumberFormatInfo 对象的 CurrencyGroupSeparator 属性用于设置货币值中对小数点左边的数字分组的字符串。

创建 NumberFormatInfo 对象的过程在实例 050 中已经详细介绍过，本实例不再详细讲解。

说明：使用 CurrencyGroupSeparator 属性对货币值分组，使货币值显示更为直观。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Format。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入数值和显示转换结果；添加一个 Button 控件，用于将数值格式化为货币格式。

（3）程序的主要代码如下：

```
private void btn_Set_Click(object sender, EventArgs e)
{
    double P_dbl_temp;                                                    //定义 double 类型的变量
    if (double.TryParse(txt_str.Text, out P_dbl_temp))                    //验证输入是否正确并赋值
    {
        System.Globalization.NumberFormatInfo GN =                        //实例化 NumberFormatInfo 对象
            new System.Globalization.
                CultureInfo("zh-CN", false).NumberFormat;
        GN.CurrencyGroupSeparator = ",";                                  //设置货币值中用来分组的字符串
        txt_result.Text = P_dbl_temp.ToString("C", GN);                   //格式化为货币格式并显示
    }
    else
    {
        MessageBox.Show("请输入正确的货币值！","提示！");                  //输出错误信息
    }
}
```

秘笈心法

心法领悟 051：重写 object 类的 ToString 方法。

由于所有类型都隐式继承自 object 基类，所以所有对象都可以调用 ToString 方法，由于 ToString 是虚方法，所以在子类中可以使用 override 关键字重写 ToString 虚方法。

实例 052 开发一个进制转换器

光盘位置：光盘\MR\03\052

高级

趣味指数：★★★★★

实例说明

电脑中的 CPU（中央处理器）有着强大的处理能力，它处理着大量的二进制信息。在生活中使用的是十进制信息，而电脑中处理的是二进制信息，可以使用相应的方法将十进制信息转换为二进制信息，也可以使用相应的方法在二进制、八进制、十进制和十六进制之间方便地进行转换。本实例介绍了一种方法，可以便捷地实现进制转换。实例运行效果如图 3.37 所示。

关键技术

本实例重点介绍使用 Convert.ToString 方法将十进制信息转换为指定类型的信息，以及使用 Convert.ToInt64 方法将指定类型的信息转换为十进制信息，下面对其进行详细讲解。

使用 Convert 类的 ToString 静态方法可以将十进制数值分别转换为二进制、八进制、十六进制。Convert 类的 ToString 方法如图 3.38 所示。

图 3.37 开发一个进制转换器

```
Convert.ToString(100, 2);  ——▶ 将十进制数值100转换为二进制

Convert.ToString(100, 8);  ——▶ 将十进制数值100转换为八进制

Convert.ToString(100, 16); ——▶ 将十进制数值100转换为十六进制
```

图 3.38 Convert 类的 ToString 方法

从图 3.38 中可以看到，ToString 静态方法接收两个参数，第一个参数是十进制数值，第二个参数指明将十进制数值转换为二进制、八进制或十六进制。

注意： Convert 类的 ToString 方法返回的是字符串对象，由此可以想到，ToString 方法的转换过程是将十进制数值转换为二进制、八进制或十六进制数值的字符串形式。

使用 Convert 类的 ToInt64 静态方法可以将二进制、八进制、十六进制数值的字符串表示转换为十进制数值。Convert 类的 ToInt64 方法如图 3.39 所示。

```
Convert.ToInt64("10110", 2);  ——▶ 将字符串表示的二进制数值10110转换为十进制数值

Convert.ToInt64("123", 8);    ——▶ 将字符串表示的八进制数值123转换为十进制数值

Convert.ToInt64("1F", 16);    ——▶ 将字符串表示的十六进制数值1F转换为十进制数值
```

图 3.39 Convert 类的 ToInt64 方法

从图 3.39 中可以看到，ToInt64 静态方法接收两个参数，第一个参数是一个字符串表示的数值，第二个参数指明字符串表示的数值是二进制、八进制或十六进制数值，并根据第二个参数将字符串表示的数值转换为十进制数值。

注意： Convert 类的 ToInt64 方法返回的是十进制数值，由此可以想到，ToInt64 方法的转换过程是将字符串表示的二进制、八进制或十六进制数值转换为十进制数值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Conversion。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入数值和显示转换后的数值；添加两个 Combobox 控件，分别用于设置输入数值的进制类型和转换为数值的进制类型；添加一个 Button 控件，用于进制转换。

（3）程序的主要代码如下：

```
private void Action()
{
    if (cbox_select.SelectedIndex != 3)                          //判断用户输入的是否为十六进制数
    {
        long P_lint_value;                                       //定义长整型变量
        if (cbox_select.SelectedIndex == 0)                      //判断用户输入的是否为十进制数
        {
            switch (cbox_select2.SelectedIndex)
            {
                case 0:
                    txt_result.Text = txt_value.Text;            //默认十进制
                    break;
                case 1:
                    txt_result.Text =                            //将十进制转为二进制
                        new Transform().TenToBinary(
                        long.Parse(txt_value.Text));
                    break;
                case 2:
                    txt_result.Text =                            //将十进制转为八进制
                        new Transform().TenToEight(
                        long.Parse(txt_value.Text));
                    break;
                case 3:
                    txt_result.Text =                            //将十进制转为十六进制
                        new Transform().TenToSixteen(
                        long.Parse(txt_value.Text));
                    break;
            }
        }
    }
}
```

秘笈心法

心法领悟 052：Convert.ToString 与 Convert.ToInt64 可以配合使用。

使用 Convert.ToString 方法可以将十进制数值转换为二进制、八进制或十六进制数值的字符串表示，使用 Convert.ToInt64 可以将二进制、八进制或十六进制数值的字符串表示转换为十进制值。那么，将二进制数值转换为十六进制数值应当怎样转换呢？可以配合使用上面的两个方法，首先使用 Convert.ToInt64 方法将二进制数值转换为十进制数值，然后调用 Convert.ToString 方法将十进制数值转换为十六进制数值的字符串表示。

3.4　日期时间格式的数据处理

实例 053	动态获得系统当前日期和时间	高级
	光盘位置：光盘\MR\03\053	趣味指数：★★★★★

实例说明

信息技术的成功，使人类的发展进入了快车道，在飞速的发展中，时间的概念对于我们来说越来越重要。本实

例将介绍一种方法，该方法会间隔地获取系统时间，并在窗体中动态显示时间信息。实例运行效果如图 3.40 所示。

关键技术

本实例重点介绍怎样方便地获取系统时间，下面对其进行详细讲解。

使用 DateTime 结构的 Now 静态属性可以轻松地获取当前系统时间，如图 3.41 所示。

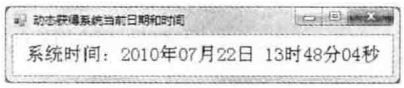


图 3.40 动态获得系统当前日期和时间

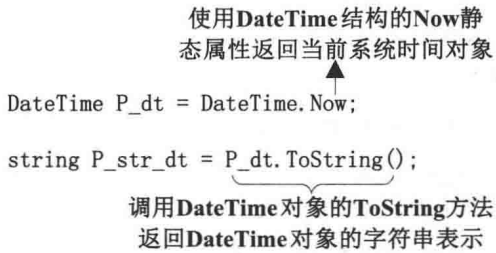


图 3.41 DateTime 结构的 Now 静态属性

从图 3.41 中可以看到，DateTime 结构使用 Now 属性获取当前系统时间对象，然后再调用时间对象的 ToString 方法，返回当前系统时间的字符串表示。

注意： DateTime 结构的 Now 静态属性只是得到一个系统时间对象，该时间对象不会随着系统时间的变化而变化，如果要动态显示系统时间，可以使用计时器间隔地获取系统时间对象并显示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体的 Load 事件中写入操作信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    System.Threading.Thread P_thread =                                    //创建线程
        new System.Threading.Thread(
        () =>                                                             //使用 Lambda 表达式
        {
            while (true)                                                  //无限循环
            {
                this.Invoke(                                              //操作窗体线程
                    (MethodInvoker)delegate()                             //使用匿名方法
                        {
                            this.Refresh();                               //刷新窗体
                            Graphics P_Graphics =                         //创建绘图对象
                                CreateGraphics();
                            P_Graphics.DrawString("系统时间：" +           //在窗体中绘出系统时间
                                DateTime.Now.ToString(
                                "yyyy 年 MM 月 dd 日 HH 时 mm 分 ss 秒"),
                                new Font("宋体", 15),
                                Brushes.Blue,
                                new Point(10, 10));
                        });
                System.Threading.Thread.Sleep(1000);                      //线程挂起 1 秒钟
            }
        });
    P_thread.IsBackground = true;                                         //将线程设置为后台线程
    P_thread.Start();                                                     //线程开始执行
}
```

秘笈心法

心法领悟 053：可以适当地使用线程替代 Timer 控件。

在窗体应用程序开发过程中，应当尽量避免使用窗体线程做高强度的运算或 IO 操作，如果窗体线程参与了

过多的运算，会导致用户的操作不能及时分配到资源，用户界面会出现卡或无响应情况。

实例 054　手动设置系统日期时间

光盘位置：光盘\MR\03\054　　高级　　趣味指数：★★★★★

实例说明

在程序设计过程中，经常会使用 DateTime 结构的 Now 属性获取系统时间，那么，怎样设置系统时间呢？本实例将介绍一种方法，轻松地实现此功能。实例运行效果如图 3.42 所示。

图 3.42　手动设置系统日期时间

关键技术

本实例重点介绍怎样使用 DateTime 结构的 Now 属性获取系统时间，以及使用 GetLocalTime 和 SetLocalTime 方法获取和设置系统时间，下面对其进行详细讲解。

（1）Now 属性

DateTime 结构的 Now 属性用于获取计算机上的当前日期和时间。语法格式如下：

```
public static DateTime Now { get; }
```

属性值：DateTime 结构，当前系统日期和时间的 DateTime 对象。

（2）GetLocalTime 方法

SetSystemDateTime 类的 GetLocalTime 方法用于获取系统时间信息。语法格式如下：

```
public static extern void GetLocalTime(SystemTime st)
```

参数说明

st：SystemTime 对象，获取系统时间信息，并赋值到 SystemTime 对象中。

（3）SetLocalTime 方法

SetSystemDateTime 类的 SetLocalTime 方法用于设置系统时间信息。语法格式如下：

```
public static extern void SetLocalTime(SystemTime st)
```

参数说明

st：SystemTime 对象，使用 SystemTime 对象设置系统时间信息。

注意： 调用 GetLocalTime 和 SetLocalTime 方法需要引用非托管程序集，所以需要引用 System.Runtime.InteropServices 命名空间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetDate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 DateTimePicker 控件，分别用于选择日期和时间；添加两个 Button 控件，分别用于获取和设置系统时间。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("您真的确定更改系统当前日期吗？",                    //设置系统当前日期时间
        "信息提示", MessageBoxButtons.OK) == DialogResult.OK)
    {
        DateTime Year = this.dateTimePicker1.Value;                          //得到时间信息
        SystemTime MySystemTime = new SystemTime();                          //创建系统时间类的对象
```

```
        SetSystemDateTime.GetLocalTime(MySystemTime);                                    //得到系统时间
        MySystemTime.vYear = (ushort)this.dateTimePicker1.Value.Year;                    //设置年
        MySystemTime.vMonth = (ushort)this.dateTimePicker1.Value.Month;                  //设置月
        MySystemTime.vDay = (ushort)this.dateTimePicker1.Value.Day;                      //设置日
        MySystemTime.vHour = (ushort)this.dateTimePicker2.Value.Hour;                    //设置小时
        MySystemTime.vMinute = (ushort)this.dateTimePicker2.Value.Minute;                //设置分
        MySystemTime.vSecond = (ushort)this.dateTimePicker2.Value.Second;                //设置秒
        SetSystemDateTime.SetLocalTime(MySystemTime);                                    //设置系统时间
        button1_Click(null, null);                                                       //执行按钮单击事件
    }
}
```

秘笈心法

心法领悟 054：获取系统日期信息。

使用 DateTime 结构的 Now 静态属性，可以方便地获取系统日期对象，通过调用日期对象的 ToString 方法，并在该方法参数中添加格式化字符串，将会返回指定日期格式的字符串。

实例 055　根据生日自动计算员工年龄

光盘位置：光盘\MR\03\055　　中级　　趣味指数：★★★★☆

实例说明

计算员工年龄，可以使用当前系统日期减去员工生日日期，得到的结果就是员工的年龄。本实例中使用了 Visual Basic 程序集中 DateAndTime 类的 DateDiff 静态方法来计算两个日期的差，从而得到员工年龄。实例运行效果如图 3.43 所示。

图 3.43　根据生日自动计算员工年龄

关键技术

本实例重点介绍怎样使用 DateDiff 方法计算员工年龄，下面对其进行详细讲解。

使用 DateAndTime 类的 DateDiff 静态方法可以方便地获取日期时间的间隔数。语法格式如下：

```
DateDiff(DateInterval Internal , DateTime Date1 , DateTime Date2 , FirstDayOfWeek DayOfWeek , FirstWeekOfYear WeekOfYear);
```

DateDiff 方法中的参数说明如表 3.2 所示。

表 3.2　DateDiff 方法中的参数说明

参　数	描　述
Internal	DateInterval 枚举值，指定 Date1 与 Date2 时间间隔的单位为“天”
Date1	计算中使用的第一个时间值
Date2	计算中使用的第二个时间值
DayOfWeek	用于指定一周的第一天，如果未指定则使用 FirstDayOfWeek.Sunday
WeekOfYear	用于指定一年的第一周，如果未指定则使用 FirstWeekOfYear.Jan1

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetAge。

（2）由于 DateDiff 是 Visual Basic 中的方法，所以要添加 Visual Basic 程序集的引用，首先，在“解决方案资源管理器”面板中右击“引用”，在弹出的快捷菜单中选择“添加引用”命令，如图 3.44 所示。

在弹出的“添加引用”对话框中选择 Microsoft.VisualBasic 程序集的引用，单击“确定”按钮，如图 3.45 所示。

图 3.44　选择“添加引用”命令

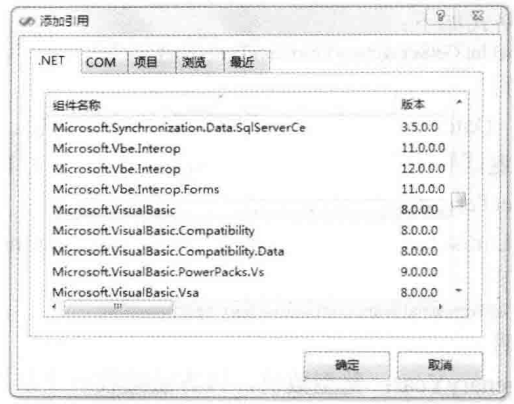


图 3.45　添加 Microsoft.VisualBasic 程序集的引用

最后在代码中添加命名空间的引用“using Microsoft.VisualBasic;”。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 DateTimePicker 控件，用于选择生日信息；添加一个 Button 控件，用于显示员工年龄。

（4）程序的主要代码如下：

```
private void btn_GetAge_Click(object sender, EventArgs e)
{
    long P_BirthDay = DateAndTime.DateDiff(DateInterval.Year,                    //计算年龄
        dtpicker_BirthDay.Value,DateTime.Now,
        FirstDayOfWeek.Sunday, FirstWeekOfYear.Jan1);
    MessageBox.Show(string.Format("年龄为：  {0}岁。",                            //输出年龄信息
        P_BirthDay.ToString()),"提示！");
}
```

秘笈心法

心法领悟 055：使用 TimeSpan 代替 DateDiff 方法。

实例中使用了 DateDiff 方法，通过该方法计算系统时间与员工生日之间相隔的年数来判断员工的年龄。同样地，也可以直接使用系统时间减去员工生日的时间，结果得到一个 TimeSpan 对象，通过 TimeSpan 对象的 Days 属性得到相隔的天数，使用相隔的天数除以 365 即可得到员工的年龄。

实例 056　根据年份判断十二生肖

光盘位置：光盘\MR\03\056　　初级　　趣味指数：★★★★

实例说明

我们伟大的祖国拥有着五千年的悠久历史和灿烂的文化，其中，十二生肖的故事广为流传，十二生肖也是记年的一种方法，本实例将向读者演示怎样计算本年是十二生肖的哪一个。实例运行效果如图 3.46 所示。

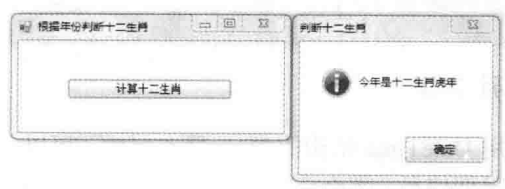


图 3.46　根据年份判断十二生肖

关键技术

本实例重点介绍怎样使用 ChineseLunisolarCalendar 对象的 GetSexagenaryYear 和 GetTerrestrialBranch 方法判断本年的生肖信息，下面对其进行详细讲解。

（1）GetSexagenaryYear 方法

ChineseLunisolarCalendar 对象的 GetSexagenaryYear 方法用于计算与指定日期对应的甲子（60 年）循环中

的年。语法格式如下：

```
public virtual int GetSexagenaryYear(DateTime time)
```

参数说明

❶ time：DateTime 对象，表示要读取的时间信息。

❷ int：返回整型数值，表示甲子循环中的一个从 1～60 的数字，它与 date 参数对应。

（2）GetTerrestrialBranch 方法

ChineseLunisolarCalendar 对象的 GetTerrestrialBranch 方法用于计算甲子（60 年）循环中指定年份的地支。语法格式如下：

```
public int GetTerrestrialBranch(int sexagenaryYear)
```

参数说明

❶ sexagenaryYear：整型数值，该方法接收一个从 1～60 的整数参数，表示甲子循环中的一年。

❷ int：返回整型数值，表示一个从 1～12 的整数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetShengXiao。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于显示本年的生肖信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    System.Globalization.ChineseLunisolarCalendar chinseCaleander =
        new System.Globalization.ChineseLunisolarCalendar();                    //创建日历对象
    string TreeYear = "鼠牛虎兔龙蛇马羊猴鸡狗猪";                                  //创建字符串对象
    int intYear = chinseCaleander.GetSexagenaryYear(DateTime.Now);              //计算年信息
    string Tree = TreeYear.Substring(chinseCaleander.                           //得到生肖信息
        GetTerrestrialBranch(intYear) - 1, 1);
    MessageBox.Show("今年是十二生肖" + Tree + "年",                              //输出生肖信息
        "判断十二生肖", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 056：使用求余的方式判断生肖信息。

实例中使用了 GetTerrestrialBranch 方法根据年份判断生肖，现在可以使用更加简单的方法判断年份的十二生肖，即使用指定的年份除以 12 求余，如果余数为 0 则为猴年，余数为 1 则为鸡年，依此类推。

实例 057　获取当前日期是星期几

光盘位置：光盘\MR\03\057　　中级　　趣味指数：★★★★★

实例说明

使用 DateTime 结构的 Now 静态属性，可以方便地获取系统日期信息，那么，怎样才能知道今天是星期几呢？实例中调用了时间对象的 ToString 方法，在该方法的参数中添加适当的格式化字符串，将返回日期的星期信息。实例运行效果如图 3.47 所示。

图 3.47　获取当前日期是星期几

关键技术

本实例重点介绍怎样使用 DateTime 结构的 Now 属性获取时间信息，以及使用 DateTime 对象的 ToString 方法将时间转换为字符串形式，下面对其进行详细讲解。

（1）Now 属性

DateTime 结构的 Now 属性用于获取计算机上的当前日期和时间。语法格式如下：

```
public static DateTime Now { get; }
```

参数说明

属性值：DateTime 结构，当前系统日期和时间的 DateTime 对象。

（2）ToString 方法

DateTime 对象的 ToString 方法用于根据指定的区域性特定格式信息将 DateTime 对象的值转换为它的等效字符串表示形式。语法格式如下：

```
public string ToString(IFormatProvider provider)
```

参数说明

❶ provider：一个 IFormatProvider，它提供区域性特定的格式设置信息。

❷ string：返回字符串对象，表示由 provider 指定的当前 DateTime 对象的值的字符串表示形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetWeek。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于显示星期信息。

（3）程序的主要代码如下：

```
private void btn_GetWeek_Click(object sender, EventArgs e)
{
    MessageBox.Show("今天是： "                                          //显示星期信息
        + DateTime.Now.ToString("dddd"), "提示！ ");
}
```

秘笈心法

心法领悟 057：判断今天是本月中的第几天。

使用 DateTime 结构的 Now 静态属性，可以方便地获取系统日期对象，通过系统日期对象的 Day 属性可以判断出今天是本月中的第几天。

实例 058　获取当前年的天数

光盘位置：光盘\MR\03\058　　初级　　趣味指数：★★★★

实例说明

一年有 365 天，但是也有例外，闰年会比普通年份多一天，闰年是为了弥补因人为历法规定造成的每一年的天数与地球实际公转周期的时间差而设定的，而补上时间差的年份被称作闰年，闰年共有 366 天。实例中会通过判断所属年份是否为闰年来计算一年中的天数。实例运行效果如图 3.48 所示。

图 3.48　获取当前年的天数

关键技术

本实例重点介绍怎样使用 IsLeapYear 方法判断指定年份是否为闰年，下面对其进行详细讲解。

实例中使用了 IsLeapYear 方法判断指定年份是否为闰年，可以通过判断指定的年份是否为闰年的方式来计算天数，如果指定的年份是闰年则有 366 天，如果指定的年份不是闰年则有 365 天。IsLeapYear 方法的使用如

图 3.49 所示。

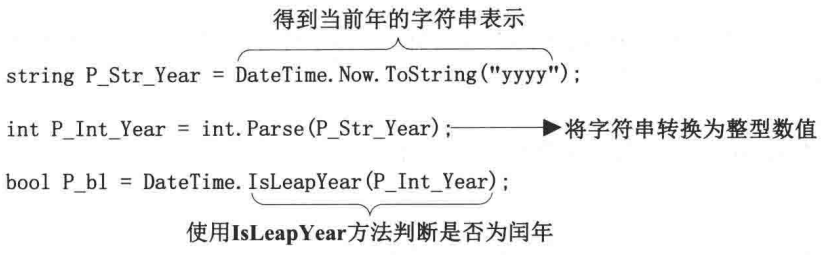


图 3.49 IsLeapYear 方法的使用

从图 3.49 中可以看到，IsLeapYear 方法接收一个整型数值作为参数，整型数值是年份信息，方法将返回一个布尔值，如果指定年份为闰年，则返回 true，否则返回 false。

技巧：也可以使用自定义的方法判断指定年是否为闰年，指定年份如果能被 400 整除就为闰年，或者指定年份可以整除 4 但不能整除 100 也为闰年。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetDays。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于显示本年所包含的天数。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    if (DateTime.IsLeapYear(int.Parse(                                    //判断是否为闰年
        DateTime.Now.ToString("yyyy"))))
    {
        MessageBox.Show("本年有 366 天","提示！");                          //显示天数信息
    }
    else
    {
        MessageBox.Show("本年有 365 天", "提示！");                         //显示天数信息
    }
}
```

秘笈心法

心法领悟 058：怎样判断闰年？

闰年是为了弥补因人为历法规定造成的每一年的天数与地球实际公转周期的时间差而设定的，而补上时间差的年份被称作闰年，闰年共有 366 天。现在可以使用条件运算符判断用户输入的年份是否为闰年。人们常说“四年一闰，百年不闰，四百年再闰”，那么，怎样计算闰年呢？计算闰年的方法很简单，指定年份如果能被 400 整除就为闰年，或者指定年份可以整除 4 但不能整除 100 也为闰年。

实例 059 获取当前月的天数

光盘位置：光盘\MR\03\059　　初级　　趣味指数：★★★★

实例说明

程序设计过程中，经常需要计算指定月份所包含的天数，实例 058 中已经介绍过，通过判断指定的年份是否为闰年来计算一年的天数，那么，怎样计算指定的月份包含的天数呢？本实例将介绍怎样使用 DateTime 结构的 DaysInMonth 方法得到指定月份所包含的天数。实例运行效果如图 3.50 所示。

关键技术

本实例重点介绍怎样使用 DaysInMonth 方法计算指定月份的天数，下面对其进行详细讲解。

DaysInMonth 方法的具体应用如图 3.51 所示。

图 3.50　获取当前月的天数

图 3.51　DaysInMonth 方法的具体应用

从图 3.51 中可以看到，DaysInMonth 方法接收两个整型数值的参数，两个整型数值分别代表要计算天数指定的年份和月份，方法返回一个整型数值，表示指定年和月中的天数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DaysInMonth。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于显示本月所包含的天数。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    int P_Count = DateTime.DaysInMonth(                                    //获取本月的天数
        DateTime.Now.Year, DateTime.Now.Month);
    MessageBox.Show("本月有" +                                             //显示本月的天数
        P_Count.ToString() + "天", "提示！ ");
}
```

秘笈心法

心法领悟 059：判断当前时间所属月份。

使用 DateTime 结构的 Now 静态属性，可以方便地获取系统日期对象，通过系统日期对象的 Month 属性可以判断出当前时间所属的月份。

实例 060　取得当前日期的前一天

光盘位置：光盘\MR\03\060　　中级　　趣味指数：★★★★☆

实例说明

使用 DateTime 结构的 Now 属性可以得到当前的日期信息，此时调用 ToString 方法，并在该方法中添加指定的格式化字符串，可以按照要求输出当前日期的信息，那么，怎样才可以获取当前日期前一天的信息呢？本实例中使用了 AddDays 方法实现此功能。实例运行效果如图 3.52 所示。

图 3.52　取得当前日期的前一天

关键技术

本实例重点介绍怎样使用 DateTime 结构的 AddDays 方法得到当前日期的前一天，下面对其进行详细讲解。

DateTime 对象的 AddDays 方法用于将指定的天数加到此 DateTime 对象的值上。语法格式如下：

```
public DateTime AddDays(double value)
```

参数说明

❶ value：双精度 Double 值，由整数和小数部分组成的天数。

❷ DateTime：返回 DateTime 对象，它的值是此 DateTime 对象所表示的日期和时间与 value 所表示的天数之和。

技巧：DateTime 对象的 AddDays 方法中的参数可以是正数也可以是负数，如果为负数，则方法返回当前日期以前的 DateTime 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetYesterDay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于显示前一天日期的信息。

（3）程序的主要代码如下：

```
private void btn_GetYesterday_Click(object sender, EventArgs e)
{
    MessageBox.Show(                                          //显示前一天日期
        "昨天是：" + DateTime.Now.AddDays(-1).
        ToString("yyyy 年 M 月 d 日"), "提示！");
}
```

秘笈心法

心法领悟 060：获取当前日期的后一天。

实例中使用了 DateTime 结构的 AddDays 方法得到了当前日期的前一天，同样也可以使用此方法得到当前日期的后一天，代码如下：

```
DateTime.Now.AddDays(1);
```

实例 061　将日期格式化为指定格式

光盘位置：光盘\MR\03\061　　初级　　趣味指数：★★★★

实例说明

程序设计过程中，经常需要输出日期信息，在输出日期信息时，可能会需要输出不同格式的日期格式。本实例介绍了几种不同的方式输出日期格式的信息，其中，自定义日期格式最为灵活。实例运行效果如图 3.53 所示。

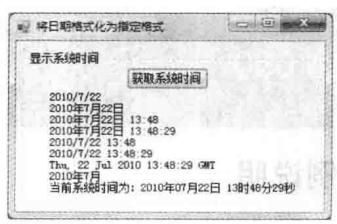


图 3.53　将日期格式化为指定格式

关键技术

本实例重点介绍将时间对象转换为多种字符串表示形式，下面对其进行详细讲解。

调用 DateTime 对象的 ToString 方法可以将当前 DateTime 对象的值转换为其等效的字符串表示形式，而且 ToString 方法提供了重载，可以在 ToString 方法中添加不同的参数，进而将时间转换为各种格式的字符串表示形式。语法格式如下：

```
DateTime.Now.ToString("yyyy ");
DateTime.Now.ToString("MM");
DateTime.Now.ToString("dd");
DateTime.Now.ToString("HH");
DateTime.Now.ToString("mm");
DateTime.Now.ToString("ss");
```

ToString 方法中的参数说明如表 3.3 所示。

表 3.3　ToString 方法中的参数说明

参　数	描　述	参　数	描　述
yyyy	年份，范围为 0000～9999	HH	时，范围为 00～23
MM	月份，范围为 01～12	mm	分钟，范围为 00～59
dd	日期，范围为 01～31	ss	秒，范围为 00～59

说明：DateTime 对象的 ToString 方法提供了多个重载，可以在 ToString 方法中使用其他参数方便地得到日期格式字符串，如“d”“D”“F”等。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TmrFormat。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Label 控件，用于显示系统时间；添加一个 Button 控件，用于获取系统时间。

（3）程序的主要代码如下：

```
private void btn_GetTime_Click(object sender, EventArgs e)
{
    lab_time.Text =
        DateTime.Now.ToString("d") + "\n" +                    //使用指定格式的字符串变量格式化日期字符串
        DateTime.Now.ToString("D") + "\n" +
        DateTime.Now.ToString("f") + "\n" +
        DateTime.Now.ToString("F") + "\n" +
        DateTime.Now.ToString("g") + "\n" +
        DateTime.Now.ToString("G") + "\n" +
        DateTime.Now.ToString("R") + "\n" +
        DateTime.Now.ToString("y") + "\n" +
        "当前系统时间为："+DateTime.Now.ToString(              //使用自定义格式格式化字符串
        "yyyy 年 MM 月 dd 日 HH 时 mm 分 ss 秒");
}
```

秘笈心法

心法领悟 061：DateTime 结构实现了 IFormattable 接口。

DateTime 结构实现了 IFormattable 接口，IFormattable 接口中定义了 ToString 方法，所以 DateTime 结构实现了这个 ToString 方法，此方法会使用指定的格式格式化当前时间信息。

实例 062　使用 DateDiff 方法获取日期时间的间隔数

光盘位置：光盘\MR\03\062　　中级　　趣味指数：★★★★☆

实例说明

在程序设计过程中，经常需要计算两个时间信息所间隔的天数，例如，计算上次商品入库到本次商品入库相隔几天。本实例使用 DateDiff 方法方便地计算出两个时间段的间隔。实例运行效果如图 3.54 所示。

图 3.54　使用 DateDiff 方法获取日期时间的间隔数

关键技术

本实例重点介绍使用 DateDiff 方法获取日期时间的间隔数，下面对其进行详细讲解。

说明：关于 DateDiff 方法的详细讲解请参见实例 055 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetDateDiff。

（2）由于 DateDiff 是 Visual Basic 中的方法，所以要添加 Visual Basic 程序集的引用。首先，在“解决方案资源管理器”面板中右击“引用”，在弹出的快捷菜单中选择“添加引用”命令，如图 3.55 所示。在弹出的“添加引用”对话框中选择 Microsoft.VisualBasic 程序集的引用，单击“确定”按钮，如图 3.56 所示。最后在代码中添加命名空间的引用“using Microsoft.VisualBasic;”。

图 3.55　选择“添加引用”命令

图 3.56　添加 Microsoft.VisualBasic 程序集的引用

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 DateTimePicker 控件，分别用于选择两个日期时间；添加一个 Button 控件，用于计算时间间隔。

（4）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    MessageBox.Show("间隔 "+
        DateAndTime.DateDiff(                                    //使用 DateDiff 方法获取日期间隔
        DateInterval.Day, dtpicker_first.Value, dtpicker_second.Value,
        FirstDayOfWeek.Sunday, FirstWeekOfYear.Jan1).ToString()+" 天", "间隔时间");
}
```

秘笈心法

心法领悟 062：使用 TimeSpan 也可以获取日期时间的间隔数。

使用 TimeSpan 也可以获取日期时间的间隔数，首先将两个 DateTime 对象相减，此时会返回 TimeSpan 对象，然后调用 TimeSpan 对象的 Days 属性就可以方便地得到两个 DateTime 对象所间隔的天数。

实例 063　使用 DateAdd 方法向指定日期添加一段时间间隔

中级

光盘位置：光盘\MR\03\063

趣味指数：★★★★☆

实例说明

可以使用 DateTime.Now 属性获取系统时间，使用 ToString 方法输出不同格式的时间信息，还可以对时间

信息做更多操作，如可以使用 DateAdd 方法方便地向指定日期添加一段时间间隔。实例运行效果如图 3.57 所示。

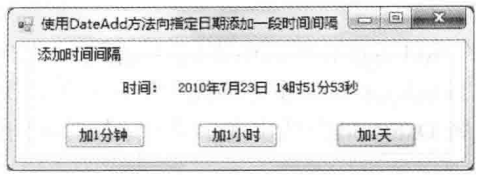


图 3.57　使用 DateAdd 方法向指定日期添加一段时间间隔

关键技术

本实例中主要用到了 DateAndTime 类的 DateAdd 静态方法，下面对其进行详细讲解。

使用 DateAndTime 类的 DateAdd 静态方法可以方便地向指定日期添加一段时间间隔。语法格式如下：

```
DateAndTime.DateAdd(DateInterval interval, double Number, DateTime DateValue);
```

参数说明

❶ interval：表示添加时间间隔的枚举值。

❷ Number：表示要添加的时间间隔数，如果 interval 的枚举值为 day，Number 值为 1，则向当前时间对象添加 1 天。

❸ DateValue：表示当前操作的时间对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddDate。

（2）由于 DataAdd 是 Visual Basic 中的方法，所以要添加 Visual Basic 程序集的引用，添加 Visual Basic 程序集引用及命名空间请参照实例 062。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Label 控件，用于显示时间信息；添加 3 个 Button 控件，分别用于将当前时间增加 1 分钟、1 小时和 1 天。

（4）程序的主要代码如下：

```
private DateTime G_datetime;                                          //定义时间字段
private void Frm_Main_Load(object sender, EventArgs e)
{
    G_datetime=DateTime.Now;                                          //得到系统当前时间
    lab_time.Text = G_datetime.ToString(                              //显示时间信息
        "时间：    yyyy 年 M 月 d 日 H 时 m 分 s 秒");
}
private void btn_AddM_Click(object sender, EventArgs e)
{
    G_datetime = DateAndTime.DateAdd(                                 //向时间字段中添加 1 分钟
        DateInterval.Minute, 1, G_datetime);
    lab_time.Text = G_datetime.ToString(                              //显示时间信息
        "时间：    yyyy 年 M 月 d 日 H 时 m 分 s 秒");
}
private void btn_AddH_Click(object sender, EventArgs e)
{
    G_datetime = DateAndTime.DateAdd(                                 //向时间字段中添加 1 小时
        DateInterval.Hour, 1, G_datetime);
    lab_time.Text = G_datetime.ToString(                              //显示时间信息
        "时间：    yyyy 年 M 月 d 日 H 时 m 分 s 秒");
}
private void btn_addD_Click(object sender, EventArgs e)
{
    G_datetime = DateAndTime.DateAdd(                                 //向时间字段中添加 1 天
      DateInterval.Day, 1, G_datetime);
    lab_time.Text = G_datetime.ToString(                              //显示时间信息
        "时间：    yyyy 年 M 月 d 日 H 时 m 分 s 秒");
}
```

秘笈心法

心法领悟 063：使用 DateTime 的 Add 方法也可以向指定日期添加一段时间间隔。

DateTime 的 Add 方法接收一个 TimeSpan 对象作为参数，用于向当前 DateTime 中添加一段时间间隔。所以也可以使用 DateTime 的 Add 方法代替 DateAdd 方法实现相同的添加时间间隔的效果。

实例 064	使用 TimeSpan 对象获取时间间隔	中级
	光盘位置：光盘\MR\03\064	趣味指数：★★★★☆

实例说明

使用 TimeSpan 对象可以方便地获取两个时间段的间隔。两个时间信息相减后会得到一个 TimeSpan 对象，该 TimeSpan 对象代表时间间隔，可以通过 TimeSpan 对象的 Days、Hours、Minutes、Seconds、Milliseconds 属性分别得到间隔的天、时、分、秒、毫秒数。实例运行效果如图 3.58 所示。

图 3.58　使用 TimeSpan 对象获取时间间隔

关键技术

本实例重点介绍 TimeSpan 对象的使用方法，下面对其进行详细讲解。

TimeSpan 对象表示时间间隔或持续时间，如果两个 DateTime 对象相减，则会得到一个 TimeSpan 对象，该 TimeSpan 对象表示在两个 DateTime 对象内所持续的时间，使用 TimeSpan 对象的 days 只读属性可以获取持续时间的天数，使用 TimeSpan 对象的 Hours 只读属性可以获取持续时间的小时数，使用 Minutes 只读属性可以获取持续的分钟数。

技巧：可以调用 TimeSpan 的 Add 方法，得到两个 TimeSpan 持续时间的和。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetInterval。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 Label 控件，分别用于显示系统时间和时间间隔；添加 3 个 Button 控件，分别用于获取系统时间和计算时间间隔。

（3）程序的主要代码如下：

```
private DateTime G_DateTime_First,                                    //定义两个时间字段
    G_DateTime_Second;
private void btn_First_Click(object sender, EventArgs e)
{
    G_DateTime_First = DateTime.Now;                                  //为时间字段赋值
    lab_first.Text = "系统时间：" +                                    //显示时间
        G_DateTime_First.ToString(
        "yyyy 年 M 月 d 日 H 时 m 分 s 秒 fff 毫秒");
}
private void btn_Second_Click(object sender, EventArgs e)
{
    G_DateTime_Second = DateTime.Now;                                 //为时间字段赋值
    lab_second.Text = "系统时间：" +                                   //显示时间
        G_DateTime_Second.ToString(
        "yyyy 年 M 月 d 日 H 时 m 分 s 秒 fff 毫秒");
}
private void btn_Result_Click(object sender, EventArgs e)
{
    TimeSpan P_timespan_temp =                                        //计算两个时间的时间间隔
```

```
            G_DateTime_First > G_DateTime_Second ?
            G_DateTime_First - G_DateTime_Second :
            G_DateTime_Second - G_DateTime_First;
        lab_result.Text = string.Format(                                    //显示时间间隔
            "间隔时间：{0}天{1}时{2}分{3}秒 {4}毫秒",
            P_timespan_temp.Days, P_timespan_temp.Hours,
            P_timespan_temp.Minutes, P_timespan_temp.Seconds,
            P_timespan_temp.Milliseconds);
    }
```

秘笈心法

心法领悟 064：方便灵活的 TimeSpan。

TimeSpan 对象代表两个时间段的间隔或跨度，使用 TimeSpan 对象的 Days 只读属性可以获取时间间隔的天数，TimeSpan 对象的 Hours 只读属性可以获取时间间隔的小时数，Minutes 只读属性可以获取时间间隔的分钟数，Seconds 只读属性可以获取秒数，而 Milliseconds 只读属性可以获取间隔的毫秒数。

实例 065　使用 Sleep 方法延迟时间

光盘位置：光盘\MR\03\065　　高级　　趣味指数：★★★★★

实例说明

在使用线程时，经常会用到 Sleep 方法，Sleep 方法用于将线程挂起指定的时间，时间的单位为毫秒。本实例中将会创建一条线程，并在线程的 While 循环中使用 Sleep 方法将线程挂起一秒钟，使线程每隔一秒钟获取一次系统时间，并将系统时间显示到 TextBox 控件中。实例运行效果如图 3.59 所示。

关键技术

本实例实现时主要用到了多线程技术和线程的 Sleep 方法，下面对其进行详细讲解。

（1）创建和使用多线程

在使用多线程前要先引用命名空间 System.Threading，引用命名空间后就可以在需要的地方方便地创建并使用线程。定义线程如图 3.60 所示。

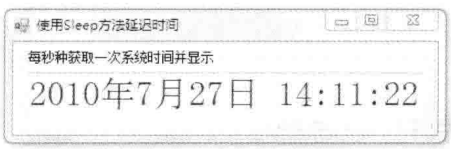


图 3.59　使用 Sleep 方法延迟时间

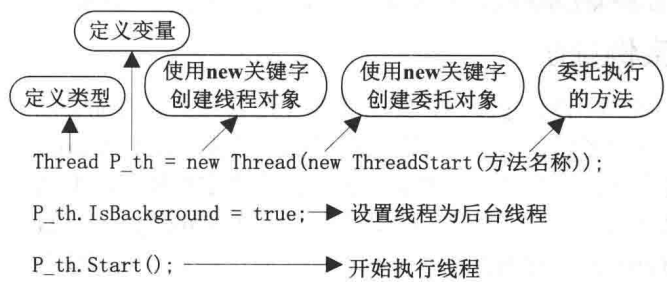


图 3.60　定义线程

从图 3.60 中可以看到，创建线程对象的构造方法中使用了 ThreadStart()委托，当线程开始执行时，将会执行委托中定义的方法。通过 IsBackground 属性设置 P_th 线程为后台线程，调用 P_th 对象的 Start 方法启动线程，开始执行，在线程开始后可以调用线程对象的 Abort 方法终止线程。

（2）Sleep 方法

Thread 类的 Sleep 方法用于将当前线程挂起指定的时间。语法格式如下：

```
public static void Sleep(int millisecondsTimeout)
```

参数说明

millisecondsTimeout：整型数值，表示线程被阻塞的毫秒数。

注意：实例中使用了 Lambda 表达式，实现了 Thread 类的构造方法中的 ThreadStart 委托所约定的方法。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseSleep。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于显示系统时间。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    Thread th = new Thread(                                          //创建线程对象
        () =>                                                        //使用 Lambda 表达式
        {
            while (true)                                             //无限循环
            {
                Invoke(                                              //在窗体线程中执行
                    (MethodInvoker)(() =>                            //使用 Lambda 表达式
                    {
                        txt_Time.Text =                              //显示系统时间
                            DateTime.Now.ToString("F");
                    }));
                Thread.Sleep(1000);                                  //线程挂起 1000 毫秒
            }
        });
    th.IsBackground = true;                                          //设置线程为后台线程
    th.Start();                                                      //开始执行线程
}
```

秘笈心法

心法领悟 065：在窗体线程中可以使用 Sleep 方法吗？

Sleep 方法用于挂起当前线程，在窗体线程中不应当调用 Sleep 方法，如果调用此方法则会出现窗体无响应的情况。

实例 066 如何确定程序的运行时间

光盘位置：光盘\MR\03\066　　高级　　趣味指数：★★★★★

实例说明

在程序设计过程中，经常需要在主窗体中动态地显示程序的运行时间，怎样实现此功能呢？首先，在窗体 Load 事件中获取系统时间。然后，使用线程获取系统时间并与窗体载入时获取的时间相减，会得到一个 TimeSpan 对象，此 TimeSpan 对象就是程序运行的时间。最后，使用线程在 StatusStrip 中动态显示程序的运行时间。实例运行效果如图 3.61 所示。

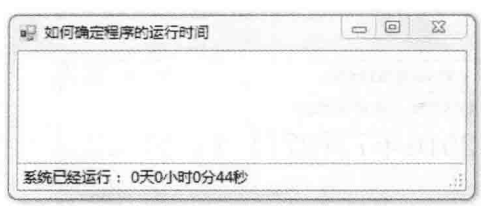


图 3.61　如何确定程序的运行时间

关键技术

本实例重点介绍 TimeSpan 结构的 Days、Hours、Minutes 及 Seconds 属性的使用，下面对其进行详细讲解。

（1）Days 属性

TimeSpan 结构的 Days 属性用来获取由当前 TimeSpan 结构表示的整天数。语法格式如下：

```
public int Days { get; }
```

参数说明

属性值：整型数值，表示此 TimeSpan 结构的天数部分。返回值可以是正数也可以是负数。

（2）Hours 属性

TimeSpan 结构的 Hours 属性用来获取由当前 TimeSpan 结构表示的整小时数。语法格式如下：

```
public int Hours { get; }
```

参数说明

属性值：整型数值，表示当前 TimeSpan 结构的小时分量。返回值的范围为-23～23。

（3）Minutes 属性

TimeSpan 结构的 Minutes 属性用来获取由当前 TimeSpan 结构表示的整分钟数。语法格式如下：

```
public int Minutes { get; }
```

参数说明

属性值：整型数值，表示当前 TimeSpan 结构的分钟分量。返回值的范围为-59～59。

（4）Seconds 属性

TimeSpan 结构的 Seconds 属性用来获取由当前 TimeSpan 结构表示的整秒数。语法格式如下：

```
public int Seconds { get; }
```

参数说明

属性值：整型数值，表示当前 TimeSpan 结构的秒分量。返回值的范围为-59～59。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayRunTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 StatusStrip 控件，向该控件中添加一个 Label 标签，用于显示程序的运行时间。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    G_DateTime = DateTime.Now;                                          //得到系统当前时间
    Thread P_th = new Thread(                                           //创建线程
        () =>                                                           //使用 Lambda 表达式
        {
            while (true)                                                //无限循环
            {
                TimeSpan P_TimeSpan =                                   //得到时间差
                    DateTime.Now - G_DateTime;
                Invoke(                                                 //调用窗体线程
                    (MethodInvoker)(() =>                               //使用 Lambda 表达式
                    {
                        tssLabel_Time.Text =                            //显示程序启动时间
                            string.Format(
                            "系统已经运行：  {0}天{1}小时{2}分{3}秒",
                            P_TimeSpan.Days, P_TimeSpan.Hours,
                            P_TimeSpan.Minutes, P_TimeSpan.Seconds);
                    }));
                Thread.Sleep(1000);                                     //线程挂起 1 秒钟
            }
        });
    P_th.IsBackground = true;                                           //设置为后台线程
    P_th.Start();                                                       //开始执行线程
}
```

秘笈心法

心法领悟 066：前台线程和后台线程的区别。

实例中使用了线程来动态显示程序的运行时间，线程分为前台线程和后台线程，通过线程对象的 IsBackground 属性可以设置线程为前台线程或后台线程。前台线程与后台线程的区别是，后台线程不会使托管程序处于运行状态，也就是说，如果进程停止所有前台线程，那么系统会认为此进程处于非运行状态，将会停止所有后台线程并关闭。

实例 067 使用 ParseExact 方法将字符串转化为日期格式

光盘位置：光盘\MR\03\067　　初级　　趣味指数：★★★★

实例说明

在程序设计过程中，经常需要显示系统日期，使用 DateTime 结构的 Now 属性可以获取系统的日期信息，此时调用 ToString 方法，将会返回日期的字符串表示，那么，怎样将字符串转换为日期格式呢？本实例介绍的 ParseExact 方法可以轻松地实现此功能。实例运行效果如图 3.62 所示。

图 3.62　使用 ParseExact 方法将字符串转化为日期格式

关键技术

本实例重点介绍怎样使用 DateTime 结构的 ParseExact 方法将字符串转换为日期格式，下面对其进行详细讲解。

DateTime 结构的 ParseExact 方法用于将日期和时间的指定字符串表示形式转换为其等效的 DateTime 对象。语法格式如下：

```
public static DateTime ParseExact(string s,string format,IFormatProvider provider)
```

ParseExact 方法中的参数说明如表 3.4 所示。

表 3.4　ParseExact 方法中的参数说明

参　数	描　述
s	字符串对象，表示包含要转换的日期和时间的字符串
format	字符串对象，表示 s 的预期格式
provider	实现 IFormatProvider 接口的对象，表示用于提供有关 s 的区域性特定格式信息的 IFormatProvider
DateTime	方法返回 DateTime 对象，等效于由 format 和 provider 所指定的 s 中包含的日期和时间

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ConvertToString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 TextBox 控件，分别用于输入时间信息；添加一个 Button 控件，用于将字符串转换为日期格式，并输出日期信息。

（3）程序的主要代码如下：

```
private void btn_Convert_Click(object sender, EventArgs e)
{
    string s=string.Format("{0}/{1}/{2}",                                    //得到日期字符串
        txt_Year.Text, txt_Month.Text, txt_Day.Text);
    DateTime P_dt = DateTime.ParseExact(                                     //将字符串转换为日期格式
        s, "yyyy/MM/dd", null);
    MessageBox.Show("输入的日期为：  "                                        //弹出消息对话框
        + P_dt.ToLongDateString(), "提示！ ");
}
```

秘笈心法

心法领悟 067：DateTime 结构的 Parse 方法。

实例中使用了 DateTime 结构的 ParseExact 方法将字符串转化为日期格式。同样地，也可以使用 DateTime 结构的 Parse 方法将字符串转化为日期格式，而且 Parse 方法的使用更加简单。

实例 068　使用 ToString 方法格式化日期

光盘位置：光盘\MR\03\068　　初级　　趣味指数：★★★★

实例说明

使用 DateTime 结构的 Now 静态属性，可以方便地获取系统日期信息，如果要显示此日期信息，就要将日期信息转换为字符串的形式。本实例使用了日期信息的 ToString 方法来显示不同格式的日期信息。实例运行效果如图 3.63 所示。

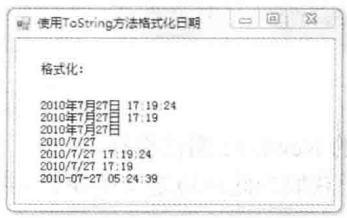


图 3.63　使用 ToString 方法格式化日期

关键技术

本实例重点介绍怎样使用 Environment 类的 NewLine 属性和 DateTime 对象的 ToString 方法，下面对其进行详细讲解。

（1）NewLine 属性

该属性用于获取为此环境定义的换行字符串。语法格式如下：

```
public static string NewLine { get; }
```

参数说明

属性值：字符串类型值。包含“\r\n”的字符串，用于非 UNIX 平台或 UNIX 平台。

说明：使用 Environment 类的 Exit 方法，可以快速地退出窗体应用程序。

（2）ToString 方法

DateTime 对象的 ToString 方法用于根据指定的区域性特定格式信息将 DateTime 对象的值转换为它的等效字符串表示形式。语法格式如下：

```
public string ToString(IFormatProvider provider)
```

参数说明

❶ provider：一个 IFormatProvider，它提供区域性特定的格式设置信息。

❷ string：返回字符串对象，表示由 provider 指定的当前 DateTime 对象的值的字符串表示形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ToStringFormat。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 label 标签，用于显示日期信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("F"));
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("f"));
```

```
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("D"));
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("d"));
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("G"));
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("g"));
    lb_Format.Text += Environment.NewLine;
    lb_Format.Text += string.Format("{0}",                          //使用指定日期格式化方式格式化字符串
        DateTime.Now.ToString("yyyy-MM-dd hh:ss:ff"));
}
```

秘笈心法

心法领悟 068：使用 Environment 类的 NewLine 属性换行。

Environment 类的 NewLine 属性用于获取为此环境定义的换行字符串，程序执行过程中，可以使用此属性方便地对字符串进行换行操作。

实例 069 使用 ToDateTime 方法转换日期显示格式

光盘位置：光盘\MR\03\069　　初级　　趣味指数：★★★★

实例说明

使用 Convert 类提供的多个静态方法可以实现数据的类型转换，使用 ToInt32 方法可以将指定的数值转换为整数类型，使用 ToString 方法可以将指定的数值转换为字符串，当然，如果转换不成功则会抛出异常信息。本实例介绍使用 Convert 类的 ToDateTime 静态方法将字符串转换为日期格式。实例运行效果如图 3.64 所示。

图 3.64　使用 ToDateTime 方法转换日期显示格式

关键技术

本实例重点介绍怎样使用 Convert 类的 ToDateTime 方法将字符串转换为 DateTime 对象，下面对其进行详细讲解。

Convert 类的 ToDateTime 方法用于将指定的值转换为 DateTime 对象。语法格式如下：

```
public static DateTime ToDateTime(string value)
```

参数说明

❶ value：字符串对象，日期和时间的字符串表示形式。

❷ DateTime：返回 DateTime 对象，等效于 value 值的 DateTime 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ConvertToString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 TextBox 控件，分别用于输入时间信息；添加一个 Button 控件，用于将字符串转换为日期格式，并输出日期信息。

（3）程序的主要代码如下：

```
private void btn_Convert_Click(object sender, EventArgs e)
{
    string P_DateTime=string.Format("{0}/{1}/{2}",                    //得到日期字符串
        txt_Year.Text, txt_Month.Text, txt_Day.Text);
    DateTime P_dt = Convert.ToDateTime(P_DateTime);
    MessageBox.Show("输入的日期为：  "                                  //弹出消息对话框
        + P_dt.ToLongDateString(), "提示！ ");
}
```

秘笈心法

心法领悟 069：Convert 类的 ToInt32 方法。

实例中介绍了怎样使用 Convert 类的 ToDateTime 方法将字符串转换为日期对象，Convert 类提供了多种用于转换的静态方法，如可以使用 Convert 类的 ToInt32 方法将字符串对象转换为整型数值。

3.5　常用数字验证技巧

实例 070	使用正则表达式验证电话号码	中级
	光盘位置：光盘\MR\03\070	趣味指数：★★★★☆

实例说明

正则表达式在程序设计中有着重要的位置，经常被用于处理字符串信息。本实例使用了正则表达式来验证电话号码是否合法。实例运行效果如图 3.65 所示。

图 3.65　使用正则表达式验证电话号码

关键技术

本实例主要用到了 Regex 类的 IsMatch 方法，下面对其进行详细讲解。

Regex 类的 IsMatch 方法用于指示正则表达式使用 pattern 参数中指定的正则表达式是否在输入字符串中找到匹配项。语法格式如下：

```
public static bool IsMatch(string input,string pattern)
```

参数说明

❶ input：字符串对象，表示要搜索匹配项的字符串。

❷ pattern：字符串对象，表示要匹配的正则表达式模式。

❸ bool：返回布尔值，如果正则表达式找到匹配项，则返回值为 true，否则返回值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidatePhone。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入电话号码；添加一个 Button 控件，用于验证电话号码。

（3）程序的主要代码如下：

```
public bool IsTelephone(string str_telephone)
{
    return System.Text.RegularExpressions.                            //使用正则表达式判断是否匹配
        Regex.IsMatch(str_telephone, @"^(\d{3,4}-)?\d{6,8}$");
}
```

秘笈心法

心法领悟 070：正则表达式中匹配位置的元字符“^”。

正则表达式中“^”用于匹配行首，如果正则表达式匹配以 First 开头的行，则正则表达式如下：

```
^First
```

实例 071 使用正则表达式验证输入密码条件

初级

光盘位置：光盘\MR\03\071

趣味指数：★★★★

实例说明

在注册用户时，经常需要填写密码信息，为保证用户信息的安全性，密码一般情况下要求输入 6 位以上，而且不可以使用纯字母或数字。本实例使用正则表达式来验证密码条件，只允许输入数字和英文字母组成的字符串。实例运行效果如图 3.66 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证输入密码条件，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.67 所示。

图 3.66　使用正则表达式验证输入密码条件

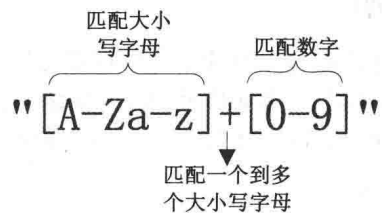


图 3.67　正则表达式

图 3.67 详细地描述了"[A-Za-z]+[0-9]"正则表达式，其中[A-Za-z]表示匹配一个大写或小写的字母，+表示最少匹配一个大写或小写的字母，[0-9]表示匹配一个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidatePassWord。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入密码信息；添加一个 Button 控件，用于验证密码是否符合条件。

（3）程序的主要代码如下：

```
public bool IsPassword(string str_password)
{
    return System.Text.RegularExpressions.                                    //使用正则表达式判断是否匹配
        Regex.IsMatch(str_password, @"[A-Za-z]+[0-9]");
}
```

秘笈心法

心法领悟 071：正则表达式中匹配位置的元字符“$”。

正则表达式中的“$”用于匹配行尾，如果正则表达式匹配以 End 结尾的行，则正则表达式如下：

```
End$
```

实例 072　使用正则表达式验证邮政编号

光盘位置：光盘\MR\03\072　　初级　　趣味指数：★★★★

实例说明

在写信时，需要填写邮政编号，邮政编号由 6 个数字组成，每一个地区的邮政编号都不相同，使用邮政编号的目的是提高信件的传递速度和准确性。本实例使用了正则表达式来验证输入的邮政编号是否合法。实例运行效果如图 3.68 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证邮政编号，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.69 所示。

图 3.68　使用正则表达式验证邮政编号

图 3.69　正则表达式

图 3.69 详细地描述了"^\d{6}$"正则表达式，其中^表示匹配行开始；\d 表示匹配数字；{6}是限定符，表示匹配 6 个数字；$表示匹配行结束。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateCode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入邮政编号；添加一个 Button 控件，用于验证邮政编号。

（3）程序的主要代码如下：

```
public bool IsPostalcode(string str_postalcode)
{
    return System.Text.RegularExpressions.
        Regex.IsMatch(str_postalcode, @"^\d{6}$");          //使用正则表达式判断是否匹配
}
```

秘笈心法

心法领悟 072：正则表达式中匹配字符的元字符“\w”和“\W”。

正则表达式中的“\w”用于匹配单词字符，单词字符包括了汉字、字母、数字和下划线；“\W”用于匹配任意的非单词字符。

实例 073　使用正则表达式验证手机号

光盘位置：光盘\MR\03\073　　中级　　趣味指数：★★★★☆

实例说明

在填写联系人信息时，如果错误地输入了电话号码，则会产生不必要的麻烦。本实例使用了正则表达式来

验证用户输入的手机号是否合法，如果输入的手机号格式不正确，则会弹出消息对话框，提示手机号不正确。实例运行效果如图 3.70 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证输入的手机号码，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.71 所示。

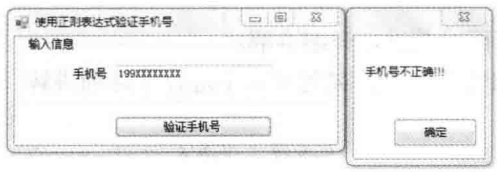


图 3.70　使用正则表达式验证手机号

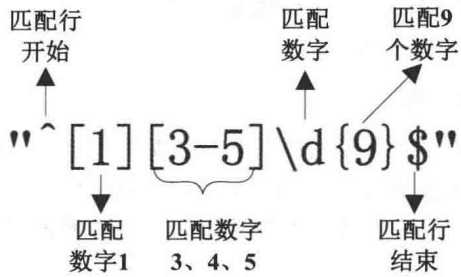


图 3.71　正则表达式

图 3.71 详细地描述了"^[1][3-5]\d{9}$"正则表达式，其中[1]表示匹配一个数字 1；[3-5]表示匹配数字 3、4、5 中的任何一个数字；\d 表示匹配数字；{9}是限定符，表示匹配 9 个数字；$表示匹配行结束。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MobileValidate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入手机号码；添加一个 Button 控件，用于验证手机号码。

（3）程序的主要代码如下：

```
public bool IsHandset(string str_handset)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_handset, @"^[1][3-5]\d{9}$");
}
```

秘笈心法

心法领悟 073：正则表达式中匹配字符的元字符“\s”和“\S”。

正则表达式中的“\s”用于匹配任意的空白字符，空白字符包括换行符、空格、制表符等；“\S”用于匹配任意的非空白字符。

实例 074　使用正则表达式验证身份证号

光盘位置：光盘\MR\03\074　　中级　　趣味指数：★★★★★

实例说明

身份证是用于证明持有人身份的证件，身份证号码分为 15 位和 18 位两种，到银行办理新建账户或办理宽带服务时都需要填写身份证号。本实例将使用正则表达式来验证用户输入的身份证号是否合法。实例运行效果如图 3.72 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证输入的身份证号，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.73 所示。

图 3.72　使用正则表达式验证身份证号

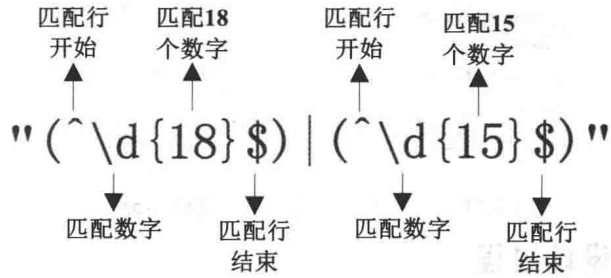


图 3.73　正则表达式

图 3.73 详细地描述了"(^\d{18}$)|(^\d{15}$)"正则表达式，其中(^\d{18}$)表示匹配 18 个数字，(^\d{15}$)表示匹配 15 个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidatePerson。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入身份证号；添加一个 Button 控件，用于验证身份证号。

（3）程序的主要代码如下：

```
public bool IsIDcard(string str_idcard)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_idcard, @"(^\d{18}$)|(^\d{15}$)");
}
```

秘笈心法

心法领悟 074：正则表达式中匹配字符的元字符“\d”和“\D”。

正则表达式中的“\d”用于匹配任意的数字，数字包括了 0、1、2、3、4、5、6、7、8 和 9；“\D”用于匹配任意的非数字字符。

实例 075　使用正则表达式验证两位小数

光盘位置：光盘\MR\03\075　　初级　　趣味指数：★★★★

实例说明

在程序运行过程中，经常需要用户输入一些数值信息，那么，怎样判断用户输入信息的格式是否正确呢？可以使用正则表达式来进行判断。本实例使用正则表达式来验证用户输入的是否为带有两位小数的数值。实例运行效果如图 3.74 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证输入的是两位小数，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.75 所示。

图 3.75 详细地描述了"^[0-9]+\.[0-9]{2}$"正则表达式，其中[0-9]+表示匹配一个或多个数字，\.表示匹配小数点“.”，[0-9]{2}表示匹配两个数字。

图 3.74　使用正则表达式验证两位小数

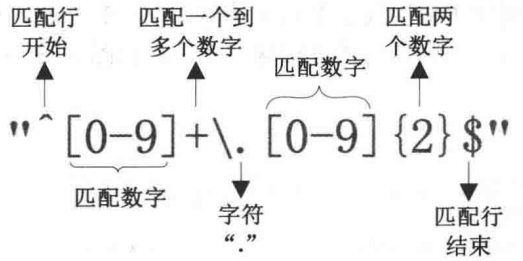


图 3.75　正则表达式

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateDecimal。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入小数数值；添加一个 Button 控件，用于验证小数数值。

（3）程序的主要代码如下：

```
public bool IsDecimal(string str_decimal)
{
    return System.Text.RegularExpressions.Regex.
        IsMatch(str_decimal, @"^[0-9]+\.[0-9]{2}$");          //使用正则表达式判断是否匹配
}
```

秘笈心法

心法领悟 075：正则表达式中匹配字符的元字符“.”。

正则表达式中的“.”用于匹配除换行符以外的任意字符。例如，正则表达式“^...$”匹配了字符串“abc”同时也会匹配“123”和“_1a”。

实例 076　使用正则表达式验证一年的 12 个月份

光盘位置：光盘\MR\03\076　　初级　　趣味指数：★★★★

实例说明

一年有 365 天，分为 12 个月，在程序中填写月份信息时，可以使用正则表达式判断输入的月份信息是否正确，如果用户输入的信息正确，例如，在月份信息中填写了 12，那么将会弹出消息对话框，提示用户输入信息正确。实例运行效果如图 3.76 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证一年的 12 个月份，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.77 所示。

图 3.76　使用正则表达式验证一年的 12 个月份

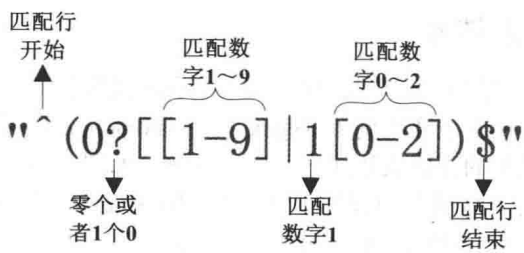


图 3.77　正则表达式

图 3.77 详细地描述了"^(0?[[1-9]|1[0-2]]$"正则表达式，其中 0?表示匹配零个或 1 个“0”，[1-9]表示匹配数字 1～9，1[0-2]表示匹配数字 10、11、12。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateMonth。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入月份信息；添加一个 Button 控件，用于验证月份信息。

（3）程序的主要代码如下：

```
public bool IsMonth(string str_Month)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_Month, @"^(0?[[1-9]|1[0-2]]$");
}
```

秘笈心法

心法领悟 076：正则表达式中的限定符。

正则表达式中的每一个元字符都只能匹配一个字符，如果要匹配多个字符，则需要使用限定符。限定符用于指定字符自身重复出现的次数。

实例 077　使用正则表达式验证一个月的 31 天

光盘位置：光盘\MR\03\077　　高级　　趣味指数：★★★★★

实例说明

一年有 365 天，分为 12 个月，每个月最多有 31 天。本实例将使用正则表达式来验证用户输入的每个月的天数是否正确，如果用户输入的天数小于 1 或大于 31，则弹出消息对话框，提示输入天数不正确。实例运行效果如图 3.78 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证一个月的 31 天，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.79 所示。

图 3.78　使用正则表达式验证一个月的 31 天

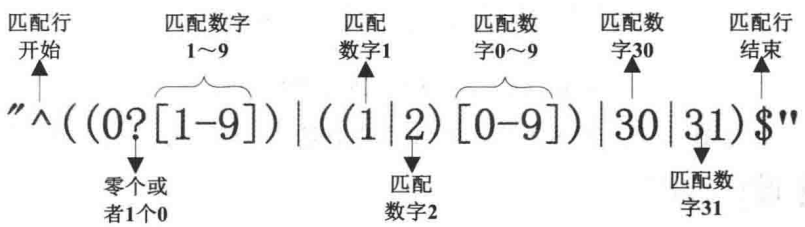


图 3.79　正则表达式

图 3.79 详细地描述了"^((0?[1-9])|((1|2)[0-9])|30|31)$"正则表达式，其中 0?表示匹配零个或 1 个“0”，[1-9]表示匹配数字 1～9，(1|2)表示匹配数字 1 或 2，|30|31 表示匹配数字 30 或 31。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateDay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入日期信

息；添加一个 Button 控件，用于验证日期信息。

（3）程序的主要代码如下：

```
public bool IsDay(string str_day)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_day, @"^((0?[1-9])|((1|2)[0-9])|30|31)$");
}
```

秘笈心法

心法领悟 077：正则表达式中的“？”限定符。

正则表达式中的“？”限定符用于限定指定的字符出现 0 次或 1 次。例如：

```
^abc?$
```

上面的正则表达式可以匹配字符串“abc”和“ab”。表达式 c?表示字母 c 可以出现 1 次或者不出现。

实例 078　使用正则表达式验证数字输入

光盘位置：光盘\MR\03\078　　初级　　趣味指数：★★★★

实例说明

在程序运行过程中，经常需要用户输入数字信息，如输入员工年龄、工资等。本实例中使用了正则表达式，有效地判断用户输入的信息是否为数字，如果用户输入的信息是数字，则会弹出消息对话框，提示用户输入信息正确。实例运行效果如图 3.80 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证数字，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.81 所示。

图 3.80　使用正则表达式验证数字输入

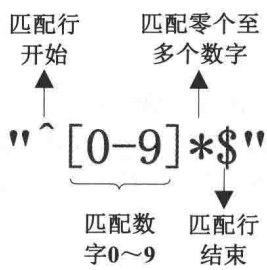


图 3.81　正则表达式

图 3.81 详细地描述了"^[0-9]*$"正则表达式，其中[0-9]*表示匹配零个至多个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateValue。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入数字信息；添加一个 Button 控件，用于验证数字信息。

（3）程序的主要代码如下：

```
public bool IsNumber(string str_number)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_number, @"^[0-9]*$");
}
```

秘笈心法

心法领悟 078：正则表达式中的“*”限定符。

正则表达式中的“*”限定符用于限定指定的字符至少出现 0 次。例如：

```
^abc*$
```

上面的正则表达式可以匹配字符串“ab”“abc”“abcc”等。表达式 c*表示字母 c 可以出现 0 次或无数次。

实例 079　使用正则表达式验证密码长度

光盘位置：光盘\MR\03\079　　初级　　趣味指数：★★★★

实例说明

在注册用户时，经常需要填写密码信息，为保证用户信息的安全性，密码一般情况下要求输入 6 位或 6 位以上。本实例中使用了正则表达式来验证密码长度，密码的长度应当为 6～18 位，如果用户输入的密码长度在此范围内，则会弹出消息对话框，提示用户输入信息正确。实例运行效果如图 3.82 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证输入的密码长度，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.83 所示。

图 3.82　使用正则表达式验证密码长度

图 3.83　正则表达式

图 3.83 详细地描述了"^\d{6,18}$"正则表达式，其中\d{6,18}表示匹配 6～18 个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateLength。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入密码信息；添加一个 Button 控件，用于验证密码长度。

（3）程序的主要代码如下：

```
public bool IsPasswLength(string str_Length)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_Length, @"^\d{6,18}$");
}
```

秘笈心法

心法领悟 079：正则表达式中的“+”限定符。

正则表达式中的“+”限定符用于限定指定的字符至少出现 1 次。例如：

```
^abc+$
```

上面的正则表达式可以匹配字符串“abc”“abcc”和“abccc”等。表达式 c+表示字母 c 至少出现 1 次。

实例 080 使用正则表达式验证非零的正整数

光盘位置：光盘\MR\03\080　　中级　　趣味指数：★★★★☆

实例说明

正则表达式经常被用于验证或处理字符串信息。本实例使用了正则表达式来验证用户输入的信息是否为非零的正整数，如果用户输入的数值正确，则会弹出消息对话框，提示用户输入信息正确。实例运行效果如图 3.84 所示。

图 3.84　使用正则表达式验证非零的正整数

关键技术

本实例重点介绍怎样使用正则表达式来验证非零的正整数，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.85 所示。

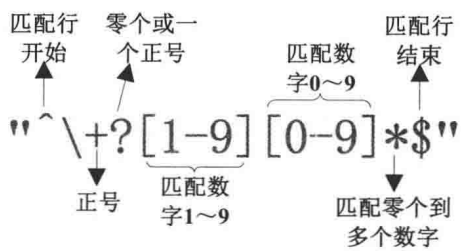


图 3.85　正则表达式

图 3.85 详细地描述了"^\+?[1-9][0-9]*$"正则表达式，其中\+?表示匹配零个或一个正号“+”，[0-9]*表示匹配零个到多个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateInteger。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入数值信息；添加一个 Button 控件，用于验证输入的数值是否为非零的正整数。

（3）程序的主要代码如下：

```
public bool IsIntNumber(string str_intNumber)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_intNumber, @"^\+?[1-9][0-9]*$");
}
```

秘笈心法

心法领悟 080：正则表达式中的“{n}”限定符。

正则表达式中的“{n}”限定符用于限定指定的字符出现 n 次。例如：

```
^c{5}$
```

上面的正则表达式可以匹配字符串“ccccc”。表达式 c{5}表示字母 c 可以出现 5 次。

实例 081　使用正则表达式验证非零的负整数

光盘位置：光盘\MR\03\081　　中级　　趣味指数：★★★★☆

实例说明

本实例中使用了正则表达式来验证用户输入的信息是否为非零的负整数，如果用户输入的数值正确，则会弹出消息对话框，提示用户输入信息正确。实例运行效果如图 3.86 所示。

图 3.86　使用正则表达式验证非零的负整数

关键技术

本实例重点介绍怎样使用正则表达式来验证非零的负整数，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.87 所示。

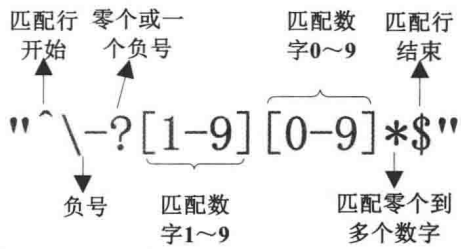


图 3.87　正则表达式

图 3.87 详细地描述了"^\-?[1-9][0-9]*$"正则表达式，其中\-?表示匹配零个或一个负号“-”，[0-9]*表示匹配零个到多个数字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateInteger。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入数值信息；添加一个 Button 控件，用于验证输入的数值是否为非零的负整数。

（3）程序的主要代码如下：

```
public bool IsIntNumber(string str_intNumber)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_intNumber, @"^\-[1-9][0-9]*$");
}
```

秘笈心法

心法领悟 081：正则表达式中的“{n , }”限定符。

正则表达式中的“{n,}”限定符用于限定指定的字符出现 n 次。例如：

```
^c{5,}$
```

上面的正则表达式可以匹配字符串“ccccc”和“cccccc”等。表达式 c{5}表示字母 c 可以出现 5 次或 5 次以上。

3.6 常用字符验证技巧

实例 082	使用正则表达式验证大写字母	初级
	光盘位置：光盘\MR\03\082	趣味指数：★★★★

实例说明

在程序设计过程中，经常会使用正则表达式判断用户输入的字符信息是否正确。本实例中将会使用正则表达式判断用户输入的字符串是否为大写字母，如果输入的字符为大写字母，则会弹出消息对话框，提示输入信息正确。实例运行效果如图 3.88 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证大写字母，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.89 所示。

图 3.88 使用正则表达式验证大写字母

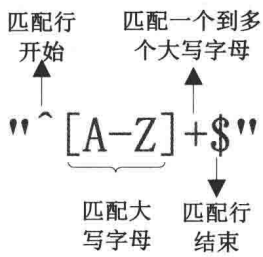


图 3.89 正则表达式

图 3.89 详细地描述了"^[A-Z]+$"正则表达式，其中[A-Z]+表示匹配一个到多个大写字母。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateCapitalLetter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入字符串；添加一个 Button 控件，用于验证字符串中的字符是否为大写字母。

（3）程序的主要代码如下：

```
public bool IsUpChar(string str_UpChar)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_UpChar, @"^[A-Z]+$");
}
```

秘笈心法

心法领悟 082：Char 结构的 IsUpper 方法。

实例中使用了正则表达式来验证用户输入的字符串是否为大写字母，同样地，使用 Char 结构的 IsUpper 方法也可以实现此验证功能。使用 Char 结构的 IsUpper 方法依次判断用户输入字符串中的每一个字符是否为大写，如果为大写则返回 true，否则返回 false。

实例 083　使用正则表达式验证小写字母

光盘位置：光盘\MR\03\083　　初级　　趣味指数：★★★★

实例说明

在程序运行过程中，经常需要输入字符信息，那么，怎样判断用户输入的字符是否为小写字母呢？本实例将会使用正则表达式来判断用户输入的字符串是否为小写字母，如果输入的字符为小写字母，则会弹出消息对话框，提示输入信息正确。实例运行效果如图 3.90 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证小写字母，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.91 所示。

图 3.90　使用正则表达式验证小写字母

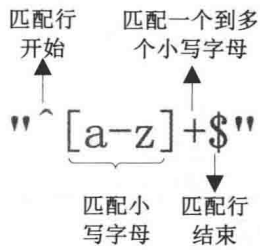


图 3.91　正则表达式

图 3.91 详细地描述了"^[a-z]+$"正则表达式，其中[a-z]+表示匹配一个到多个小写字母。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateSmallLetter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入字符串；添加一个 Button 控件，用于验证字符串中的字符是否为小写字母。

（3）程序的主要代码如下：

```
public bool IsUpChar(string str_UpChar)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_UpChar, @"^[a-z]+$");
}
```

秘笈心法

心法领悟 083：Char 结构的 IsLower 方法。

实例中使用了正则表达式来验证用户输入的字符串是否为小写字母，同样地，使用 Char 结构的 IsLower 方法也可以实现此验证功能。使用 Char 结构的 IsLower 方法依次判断用户输入字符串中的每一个字符是否为小写，如果为小写则返回 true，否则返回 false。

实例 084　使用正则表达式检查字符串中重复出现的词

光盘位置：光盘\MR\03\084　　中级　　趣味指数：★★★★☆

实例说明

可以将正则表达式理解为描述某些规则的工具，使用正则表达式可以方便地对字符串进行查找和替换的操

作，本实例中使用了正则表达式来检查字符串中重复出现的词。实例运行效果如图 3.92 所示。

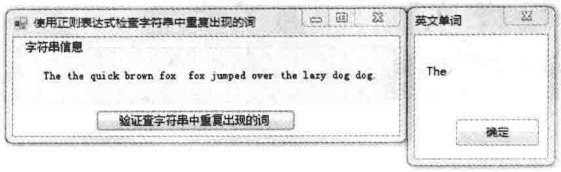


图 3.92　使用正则表达式检查字符串中重复出现的词

关键技术

本实例主要用到了 Regex 类的 Matches 方法，下面对其进行详细讲解。

Regex 类的 Matches 方法用于在输入字符串中搜索 pattern 参数中提供的正则表达式的匹配项。语法格式如下：

```
public static MatchCollection Matches(string input, string pattern, RegexOptions options)
```

Matches 方法中的参数说明如表 3.5 所示。

表 3.5　Matches 方法中的参数说明

参　数	描　述
input	要搜索匹配项的字符串
pattern	要匹配的正则表达式模式
options	RegexOption 枚举值的按位“或”组合
MatchCollection	返回找到的成功匹配的集合

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateWord。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Label 控件，用于显示字符串；添加一个 Button 控件，用于查找字符串中重复出现的单词。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    System.Text.RegularExpressions.MatchCollection matches =                    //使用正则表达式查找重复出现单词的集合
        System.Text.RegularExpressions.Regex.Matches(label1.Text,
        @"\b(?<word>\w+)\s+(\k<word>)\b", System.Text.
        RegularExpressions.RegexOptions.Compiled | System.Text.
        RegularExpressions.RegexOptions.IgnoreCase);
    if (matches.Count != 0)                                                     //如果集合中有内容
    {
        foreach (System.Text.RegularExpressions.Match                           //遍历集合
            match in matches)
        {
            string word = match.Groups["word"].Value;                           //获取重复出现的单词
            MessageBox.Show(word.ToString(), "英文单词");                        //弹出消息对话框
        }
    }
    else { MessageBox.Show("没有重复的单词"); }                                  //弹出消息对话框
}
```

秘笈心法

心法领悟 084：正则表达式中的“{n , m}”限定符。

正则表达式中的“{n , m}”限定符用于限定指定的字符至少出现 n 次，至多出现 m 次。例如：

```
^c{2,3}$
```

上面的正则表达式可以匹配字符串“cc”或“ccc”。表达式 c{2 , 3}表示字母 c 至少可以出现两次，最多可以出现 3 次。

实例 085　使用正则表达式替换字符串

光盘位置：光盘\MR\03\085　　中级　　趣味指数：★★★★☆

实例说明

使用字符串对象的 Replace 方法可以替换字符串中指定的字符或字符串，正则表达式也可以实现同样的替换功能。本实例将会使用正则表达式来替换指定字符串中的信息。实例运行效果如图 3.93 所示。

图 3.93　使用正则表达式替换字符串

关键技术

本实例主要用到了 Regex 类的 Replace 方法，下面对其进行详细讲解。

Regex 类的 Replace 方法用于在指定的输入字符串内，使用指定的替换字符串替换与指定正则表达式匹配的所有字符串。语法格式如下：

```
public static string Replace(string input,string pattern,string replacement)
```

Replace 方法中的参数说明如表 3.6 所示。

表 3.6　Replace 方法中的参数说明

参　数	描　述	参　数	描　述
input	要搜索匹配项的字符串	replacement	替换字符串
pattern	要匹配的正则表达式模式	string	返回被替换后的字符串对象

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReplaceStr。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入被替换的字符串和替换的字符串；添加两个 Button 控件，分别用于拆分字符串并显示字符串信息和退出当前程序。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string strResult = System.Text.RegularExpressions.Regex.
        Replace(textBox1.Text, @"[A-Za-z]\*?", textBox2.Text);          //使用正则表达式替换字符串
    MessageBox.Show("替换前字符:" + "\n" + textBox1.Text +
        "\n" + "替换的字符:" + "\n" + textBox2.Text + "\n" +
        "替换后的字符:" + "\n" + strResult,"替换");                      //弹出消息对话框
}
```

秘笈心法

心法领悟 085：字符串对象的 Replace 方法。

实例中使用了正则表达式替换用户输入的字母为指定的字符串，同样地，使用字符串对象的 Replace 方法也可以实现此功能。使用字符串对象的 Replace 方法可以方便地替换字符串对象中指定的字符或字符串。

实例 086 使用正则表达式拆分字符串

光盘位置：光盘\MR\03\086　　中级　　趣味指数：★★★★☆

实例说明

使用字符串对象的 Split 方法可以对字符串进行拆分操作，正则表达式也可以实现同样的拆分功能。本实例将会使用正则表达式来拆分指定字符串中的信息。实例运行效果如图 3.94 所示。

图 3.94　使用正则表达式拆分字符串

关键技术

本实例主要用到了 Regex 类的 Split 方法，下面对其进行详细讲解。

Regex 类的 Split 方法用于由正则表达式模式定义的位置拆分输入字符串。语法格式如下：

```
public static string[] Split(string input,string pattern)
```

参数说明

❶ input：字符串对象，表示要拆分的字符串。

❷ pattern：字符串对象，表示要匹配的正则表达式模式。

❸ string[]：返回字符串数组，表示拆分后的字符串数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateSplit。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 TextBox 控件，分别用于输入字符串和显示拆分后的字符串；添加一个 Button 控件，用于拆分字符串并显示字符串信息。

（3）程序的主要代码如下：

```
private void btn_Split_Click(object sender, EventArgs e)
{
    string[] P_Str = System.Text.RegularExpressions.
        Regex.Split(txt_Split.Text, "[1-9]");                    //使用正则表达式根据数字进行拆分
    foreach (string s in P_Str)                                  //遍历拆分后的字符串集合
    {
        txt_Result.Text += s;                                    //显示字符串
    }
}
```

秘笈心法

心法领悟 086：字符串对象的 Split 方法。

实例中使用了正则表达式拆分指定的字符串，同样地，使用字符串对象的 Split 方法也可以实现此功能。使用字符串对象的 Split 方法可以根据用户选择的拆分条件，方便地将字符串对象拆分为多个字符串。

实例 087 使用正则表达式验证输入字母

光盘位置：光盘\MR\03\087　　初级　　趣味指数：★★★★

实例说明

在程序运行过程中，经常需要输入字符信息，那么，怎样判断用户输入的字符是否为字母呢？本实例将会使用正则表达式来判断用户输入的字符串是否为字母，如果输入的字符为字母，则会弹出消息对话框，提示输入的是字母。实例运行效果如图 3.95 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证输入字母，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.96 所示。

图 3.95　使用正则表达式验证输入字母

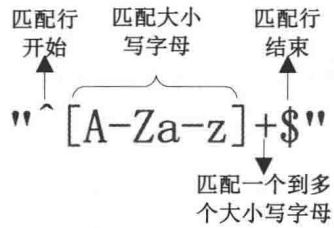


图 3.96　正则表达式

图 3.96 详细地描述了"^[A-Za-z]+$"正则表达式，其中[A-Za-z]+表示匹配一个到多个大写或小写的字母。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IsLetter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入字符串；添加一个 Button 控件，用于验证字符串是否由字母组成。

（3）程序的主要代码如下：

```
public bool IsLetter(string str_Letter)
{
    return System.Text.RegularExpressions.Regex.
        IsMatch(str_Letter, @"^[A-Za-z]+$");                    //使用正则表达式判断是否匹配
}
```

秘笈心法

心法领悟 087：Char 结构的 IsLetter 方法。

实例中使用了正则表达式验证用户输入的字符串是否为字母，同样地，使用 Char 结构的 IsLetter 方法也可以实现此验证功能。使用 Char 结构的 IsLetter 方法可以依次判断用户输入字符串中的每一个字符是否为字母，如果为字母则返回 true，否则返回 false。

实例 088　使用正则表达式验证中文汉字输入

高级

光盘位置：光盘\MR\03\088

趣味指数：★★★★★

实例说明

我们伟大的祖国拥有着五千年的悠久历史和灿烂的文化，而汉字正是这五千年文化的一种体现。在操作系统中，用户可以方便地输入汉字信息，那么，怎样判断用户输入的是否为汉字呢？本实例将使用正则表达式来实现此功能。实例运行效果如图 3.97 所示。

关键技术

本实例重点介绍怎样使用正则表达式来验证中文汉字输入，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.98 所示。

图 3.97　使用正则表达式验证中文汉字输入

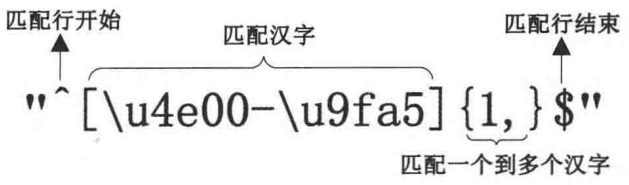


图 3.98　正则表达式

图 3.98 详细地描述了"^[\u4e00-\u9fa5]{1,}$"正则表达式，其中[\u4e00-\u9fa5]{1,}表示匹配一个到多个汉字。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IsChinese。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入字符串；添加一个 Button 控件，用于验证字符串中的字符是否为汉字。

（3）程序的主要代码如下：

```
public bool IsChinese(string str_chinese)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_chinese, @"^[\u4e00-\u9fa5]{1,}$");
}
```

秘笈心法

心法领悟 088：怎样验证字符是否由数字组成？

Char 结构的 IsDigit 方法用来验证指定的字符是否为十进制数字，可以使用 for 循环遍历字符串中的字符，依次判断每一个字符是否为数字，如果是数字则返回 true，否则返回 false。

实例 089　使用正则表达式验证输入字符串

光盘位置：光盘\MR\03\089　　中级　　趣味指数：★★★★☆

实例说明

通过字符串对象的 Length 属性可以有效地判断和限制用户输入的字符串长度。同样地，使用正则表达式也可以实现此功能。本实例将使用正则表达式来判断用户输入的字符串长度是否大于等于 8 位，如果大于等于 8 位，则会弹出消息对话框，提示输入正确。实例运行效果如图 3.99 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证输入字符串，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.100 所示。

图 3.99　使用正则表达式验证输入字符串

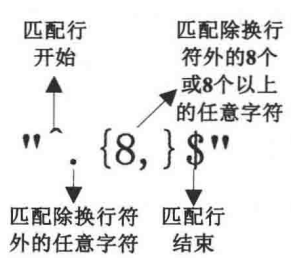


图 3.100　正则表达式

图 3.100 详细地描述了"^.{8,}$"正则表达式，其中.{8,}表示匹配除换行符外的 8 个或 8 个以上的字符。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LetterCount。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入字符串；添加一个 Button 控件，用于验证字符串中的字符数量是否大于 8 个。

（3）程序的主要代码如下：

```
public bool IsLength(string str_Length)
{
    return System.Text.RegularExpressions.Regex.                    //使用正则表达式判断是否匹配
        IsMatch(str_Length, @"^.{8,}$");
}
```

秘笈心法

心法领悟 089：字符串对象的 Length 属性。

实例中使用了正则表达式验证用户输入的字符串长度是否大于等于 8 位，同样地，使用字符串对象的 Length 属性也可以实现此验证功能。使用字符串对象的 Length 属性可以得到用户输入字符串的长度，只要判断 Length 属性是否大于等于 8 即可。

3.7　网络验证应用技巧

实例 090　使用正则表达式验证 E-mail 格式　　高级

光盘位置：光盘\MR\03\090　　趣味指数：★★★★★

实例说明

当用户填写联系信息时，如果填写的邮件地址不正确，则会出现很多不必要的麻烦。本实例将使用正则表达式来检查用户输入的邮件地址格式是否正确，如果格式正确，则会弹出消息对话框，提示 E-mail 格式正确。实例运行效果如图 3.101 所示。

图 3.101　使用正则表达式验证 E-mail 格式

关键技术

本实例重点介绍怎样使用正则表达式来验证 E-mail 格式，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.102 所示。

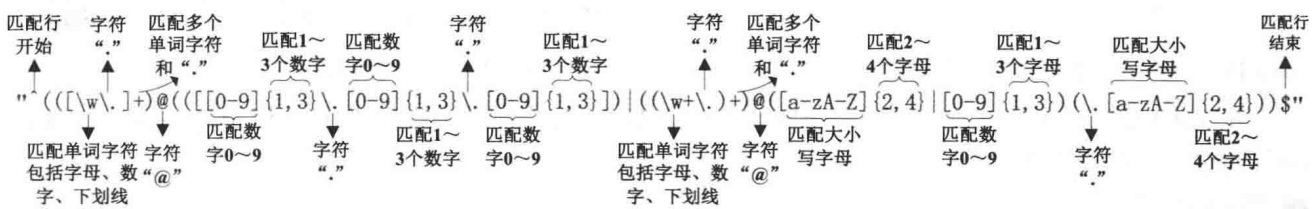


图 3.102　正则表达式

图 3.102 详细地描述了"^(([\w\.]+)@(([[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|((\w+\.)+)@([a-zA-Z] {2,4}|[0-9]{1,3})(\.[a-zA-Z]{2,4}))$"正则表达式，其中(([\w\.]+)@(([[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|((\w+\.)+)表示匹配邮件用户

名称，@表示匹配字符“@”，([a-zA-Z]{2,4}|[0-9]{1,3})(\.[a-zA-Z]{2,4}))表示匹配服务器域名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateEmail。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入 E-mail 地址；添加一个 Button 控件，用于验证 E-mail 地址格式是否正确。

（3）程序的主要代码如下：

```
public bool IsEmail(string str_Email)
{
    return System.Text.RegularExpressions.Regex.IsMatch(str_Email,                    //使用正则表达式判断是否匹配
@ "^(([\w\.]+)@(([[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|((\w+\.?)+)@([a-zA-Z]{2,4}|[0-9]{1,3})(\.[a-zA-Z]{2,4}))$");
}
```

秘笈心法

心法领悟 090：正则表达式中的转义字符。

在使用正则表达式的过程中，经常会验证带有小数点“.”的字符串，“.”在正则表达式中表示匹配除了换行符号外的任意字符。那么，要怎样才可以匹配字符“.”呢？可以使用转义字符“\”实现，\.代表字符串中的“.”，而不是正则表达式中的除换行符外的任意字符。

实例 091　使用正则表达式验证 IP 地址

光盘位置：光盘\MR\03\091　　高级　　趣味指数：★★★★★

实例说明

在应用程序的使用过程中，如果需要连接到服务器或客户端，那么，有可能会填写 IP 地址信息，IP 地址由 4 个 0～255 的数字组成，本实例使用了正则表达式来判断用户输入的 IP 地址格式是否正确。实例运行效果如图 3.103 所示。

关键技术

本实例重点介绍怎样使用正则表达式验证 IP 地址，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.104 所示。

图 3.103　使用正则表达式验证 IP 地址

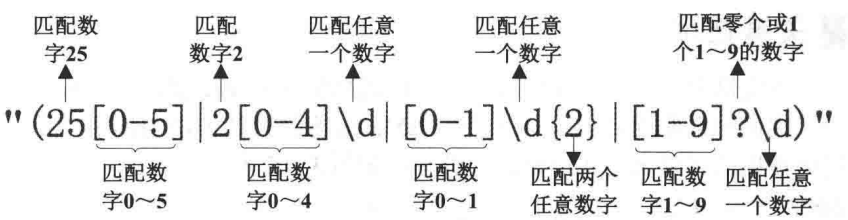


图 3.104　正则表达式

图 3.104 详细地描述了"(25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]?\d)"正则表达式，其中(25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]?\d)表示匹配数字 0～255。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateIP。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入 IP 地址；

添加一个 Button 控件，用于验证 IP 地址格式是否正确。

（3）程序的主要代码如下：

```
public bool IPCheck(string IP)
{
    string num = @"(25[0-5]|2[0-4]\d|[0-1]\d{2}|[1-9]?\d)";                    //创建正则表达式字符串
    return Regex.IsMatch(IP,                                                    //使用正则表达式判断是否匹配
        ("^" + num + "\\." + num + "\\." + num + "\\." + num + "$"));
}
```

秘笈心法

心法领悟 091：使用 Split 方法代替正则表达式验证 IP 地址。

实例中使用了正则表达式来验证 IP 地址格式是否正确，在实际应用中，也可以使用字符串对象的 Split 方法验证 IP 地址。首先，调用 Split 方法根据字符“.”拆分 IP 地址字符串为字符串数组，然后将字符串数组中的字符串转换为数值类型并判断数值范围即可。

实例 092　使用正则表达式验证 URL

光盘位置：光盘\MR\03\092　　高级　　趣味指数：★★★★★

实例说明

在浏览网页时，可以在浏览器的地址栏中输入网址信息并按 Enter 键跳转到指定网页，如果用户输入的网址不正确，则无法显示网页信息。本实例使用正则表达式来判断用户输入的网址信息是否正确。实例运行效果如图 3.105 所示。

图 3.105　使用正则表达式验证 URL

关键技术

本实例重点介绍怎样使用正则表达式来验证 URL，下面对其进行详细讲解。

实例中主要用到了 Regex 类的 IsMatch 方法，该方法的使用已经在实例 070 中进行了详细讲解。下面对实例中使用的正则表达式进行详细讲解，如图 3.106 所示。

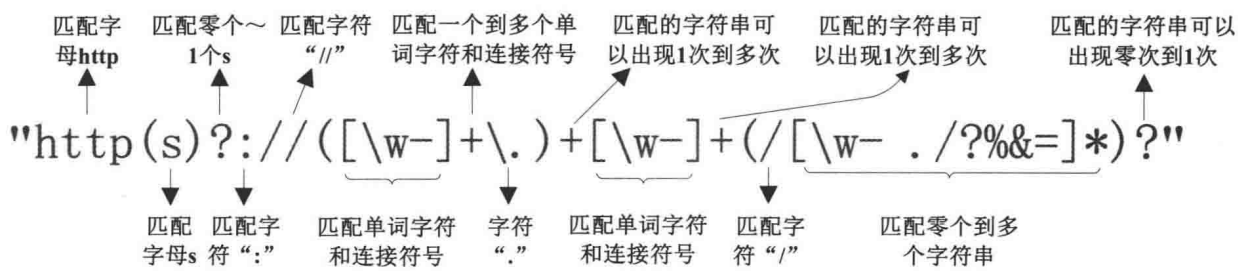


图 3.106　正则表达式

图 3.106 详细地描述了"http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?"正则表达式，其中 http(s)?表示匹配字符串“http”或“https”；[\w-]+\.表示匹配一个或多个以单词字符和连接符号“-”组成的字符串开头的，以字符“.”结尾的字符串；(/[\w- ./?%&=]*)?表示匹配出现零次或一次由单词字符“-”“/”“.”“?”“%”“&”“=”

组成的长度至少为 1 的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ValidateURL。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于输入网址；添加一个 Button 控件，用于验证网址格式是否正确。

（3）程序的主要代码如下：

```
public bool IsUrl(string str_url)
{
    return System.Text.RegularExpressions.Regex.IsMatch(str_url,                    //使用正则表达式判断是否匹配
@"http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?");
}
```

秘笈心法

心法领悟 092：使用正则表达式验证用户名格式是否正确。

在程序运行过程中，经常需要用户注册信息，以得到相应的操作权限。如果在用户注册的过程中，用户填写的用户名格式不正确，有可能会带来不必要的麻烦，使用正则表达式验证用户名格式将是一种不错的选择。

第 4 章

数组和集合的使用

- 数组的基本操作
- 常用数组排序算法
- 常用集合的使用

4.1 数组的基本操作

实例 093 获取多维数组的行数与列数

初级

光盘位置：光盘\MR\04\093

趣味指数：★★★★☆

实例说明

获取数组的行数与列数是使用多维数组时最常用的一种操作，本实例演示如何获取多维数组的行数和列数。运行本实例，单击“随机生成数组”按钮，即可生成一个数组，并且在窗体上方显示所生成数组的行数与列数。实例运行效果如图 4.1 所示。

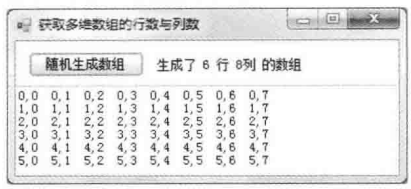


图 4.1 获取多维数组的行数与列数

关键技术

本实例在获取数组的行数与列数时，主要用到了 Array 类的 GetUpperBound 方法，下面对其进行详细讲解。

Array 类是公共语言运行库中所有数组的基类，提供了创建、操作、搜索和排序数组的方法，其 GetUpperBound 方法用来获取 Array 数组的指定维度的上限。语法格式如下：

```
public int GetUpperBound(int dimension)
```

参数说明

❶ dimension：Array 数组的从零开始的维度，其上限需要确定。

❷ 返回值：Array 中指定维度的上限。

说明：在 C#中，使用 GetUpperBound(0) + 1 获取数组的行数，使用 GetUpperBound(1) + 1 获取数组的列数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ArrayRank。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加一个 Button 控件，用来生成随机数组；添加一个 TextBox 控件，用来显示生成的随机数组；添加一个 Label 控件，用来显示生成数组的行数和列数。

（3）程序的主要代码如下：

```
private void btn_GetArray_Click(object sender, EventArgs e)
{
    txt_display.Clear();                                    //清空控件中的字符串
    G_str_array = new string[                               //随机生成二维数组
        G_Random_Num.Next(2, 10),
        G_Random_Num.Next(2, 10)];
    lab_Message.Text = string.Format(
        "生成了{0}行{1}列的数组",
        G_str_array.GetUpperBound(0) + 1,                   //获取数组的行数
        G_str_array.GetUpperBound(1) + 1);                  //获取数组的列数
    DisplayArray();                                         //调用显示数组方法
}
```

说明：上面的代码中用到了 DisplayArray 方法，该方法为自定义的无返回值类型方法，主要用来随机生成多维数组，并将生成的多维数组显示在文本框中。

秘笈心法

心法领悟 093：使用 Random 类生成随机数字。

Random 类表示伪随机数生成器，它是一种能够产生满足某些随机性统计要求的数字序列的设备，其 Next

方法用来返回一个指定范围内的随机数。

实例 094　按指定条件在数组中检索元素

光盘位置：光盘\MR\04\094　　初级　　趣味指数：★★★☆

实例说明

开发用户注册模块时，通常都有选择用户所在地的功能，开发人员可以将用户所在地存储到一个数组中，然后在程序中将用户选择的省份作为条件，并根据这个条件在数组中查找其所包含的所有城市，这时就需要用到如何在数组中检索元素。本实例通过使用 Array 类的 FindAll 方法来实现根据指定条件在数组中检索元素的功能。实例运行效果如图 4.2 所示。

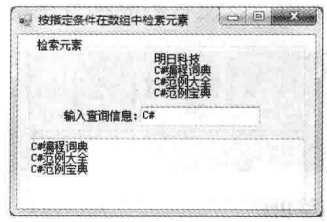


图 4.2　按指定条件在数组中检索元素

关键技术

本实例在数组中检索元素时，主要用到了 Array 类的 FindAll 方法，下面对其进行详细讲解。

FindAll 方法用来检索与指定谓词定义的条件匹配的所有元素。语法格式如下：

```
public static T[] FindAll<T>(T[] array,Predicate<T> match)
```

参数说明

❶ array：要搜索的从零开始的一维 Array 数组。

❷ match：Predicate<T>，定义要搜索的元素的条件。

❸ 返回值：如果找到一个其中所有元素均与指定谓词定义的条件匹配的 Array 数组，则为该数组；否则为一个空 Array 数组。

说明：FindAll 方法是一个泛型方法，该方法中用到了类型参数 T，用来表示数组元素的类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FindStr。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加一个 Label 控件，用来显示原始数组；添加两个 TextBox 控件，分别用来输入查询条件和显示查询结果。

（3）程序的主要代码如下：

```
private void txt_find_TextChanged(object sender, EventArgs e)
{
    if (txt_find.Text != string.Empty)                          //判断查找字符串是否为空
    {
        string[] P_str_temp = Array.FindAll                     //使用 FindAll 方法查找相应的字符串
            (G_str_array, (s) => s.Contains(txt_find.Text));
        if (P_str_temp.Length > 0)                              //判断是否查找到相应的字符串
        {
            txt_display.Clear();                                //清空控件中的字符串
            foreach (string s in P_str_temp)                    //向控件中添加字符串
            {
                txt_display.Text += s + Environment.NewLine;
            }
        }
        else
        {
            txt_display.Clear();                                //清空控件中的字符串
            txt_display.Text = "没有找到记录";                   //提示没有找到记录
        }
    }
    else
```

```
    {
        txt_display.Clear();                                    //清空控件中的字符串
    }
}
```

秘笈心法

心法领悟 094：泛型的使用。

泛型是用于处理算法、数据结构的一种编程方法。泛型的目标是采用广泛适用和可交互性的形式来表示算法和数据结构，以使它们能够直接用于软件构造。泛型通常用在集合和在集合上运行的方法中。

实例 095	在数组中添加一个元素	初级
	光盘位置：光盘\MR\04\095	趣味指数：★★★★☆

实例说明

本实例演示如何在数组中添加一个元素，运行本实例，首先单击“随机生成数组”按钮，生成一个数组；然后在文本框中输入要插入的元素，单击“确定”按钮，即可在数组的指定索引处插入一个元素。实例运行效果如图 4.3 所示。

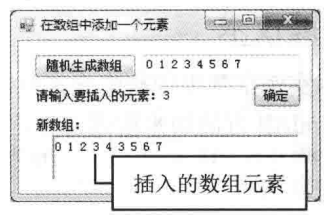


图 4.3　在数组中添加一个元素

关键技术

本实例在实现时，首先需要定义一个一维数组，然后修改数组的长度（这里使用 Length 属性获取数组的长度，然后加 1，作为新数组的长度），从而在其中增加一个元素，下面对本实例用到的关键技术进行详细讲解。

（1）一维数组的定义

数组是包含若干相同类型的变量，这些变量都可以通过索引进行访问。数组中的变量称为数组的元素，数组能够容纳元素的数量称为数组的长度。数组中的每个元素都具有唯一的索引与其相对应，数组的索引从零开始。

一维数组是具有相同数据类型的一组数据的线性集合，在程序中可以通过一维数组来完成一组相同数据类型数据的线性处理。一维数组的语法格式如下：

```
type[] arrayName;
```

参数说明

❶ type：数组存储数据的数据类型。

❷ arrayName：数组名称。

（2）Length 属性

该属性用来获取数组中的所有维数中元素的总数。语法格式如下：

```
public int Length { get; }
```

参数说明

属性值：一个 32 位整数，表示数组中的所有维数中元素的总数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddElementInArray。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示随机生成的数组和输入要插入的数组元素；添加两个 Button 控件，分别用来执行随机生成数组操作和向数组中插入元素操作；添加一个 RichTextBox 控件，用来显示插入元素后的新数组。

（3）程序的主要代码如下：

```
/// <summary>
///增加单个数组元素
```

```
/// </summary>
/// <param name="ArrayBorn">要向其中添加元素的一维数组</param>
/// <param name="Index">添加索引</param>
/// <param name="Value">添加值</param>
/// <returns></returns>
static int[] AddArray(int[] ArrayBorn, int Index, int Value)
{
    if (Index >= (ArrayBorn.Length))                    //判断添加索引是否大于数组的长度
        Index = ArrayBorn.Length - 1;                   //将添加索引设置为数组的最大索引
    int[] TemArray = new int[ArrayBorn.Length + 1];     //声明一个新的数组
    for (int i = 0; i < TemArray.Length; i++)           //遍历新数组的元素
    {
        if (Index >= 0)                                 //判断添加索引是否大于等于 0
        {
            if (i < (Index + 1))                        //判断遍历到的索引是否小于添加索引加 1
                TemArray[i] = ArrayBorn[i];             //交换元素值
            else if (i == (Index + 1))                  //判断遍历到的索引是否等于添加索引加 1
                TemArray[i] = Value;                    //为遍历到的索引设置添加值
            else
                TemArray[i] = ArrayBorn[i - 1];         //交换元素值
        }
        else
        {
            if (i == 0)                                 //判断遍历到的索引是否为 0
                TemArray[i] = Value;                    //为遍历到的索引设置添加值
            else
                TemArray[i] = ArrayBorn[i - 1];         //交换元素值
        }
    }
    return TemArray;                                    //返回插入元素后的新数组
}
```

秘笈心法

心法领悟 095：数组的由来及分类。

数组类型是从抽象基类型 Array 派生的引用类型，通过 new 运算符创建数组并将数组元素初始化为它们的默认值。数组可以分为一维数组、二维数组和多维数组等，而二维数组也是一种特殊的多维数组。

实例 096　在数组中添加一个数组

光盘位置：光盘\MR\04\096　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在数组中添加一个数组，运行本实例，首先单击“随机生成数组”按钮，生成一个数组；然后单击“生成插入数组”按钮，生成要插入的数组；最后单击“确定”按钮，即可在原数组的指定索引处插入一个数组，从而得到一个新的数组。实例运行效果如图 4.4 所示。

图 4.4　在数组中添加一个数组

关键技术

本实例在实现时，首先需要定义两个一维数组，分别用来作为原始数组和要插入的数组，然后修改原始数

组的长度（这里使用 Length 属性分别获取原始数组和要插入数组的长度，然后把获得的长度相加，作为新数组的长度），从而在其中增加一个数组。

📖 **说明：** 关于一维数组的定义及如何使用 Length 属性获取数组的长度，在实例 095 中已经做过详细讲解，详情请参见实例 095 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddArrayInArray。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示随机生成的数组和要插入的数组；添加 3 个 Button 控件，分别用来执行随机生成数组操作、随机生成要插入的数组操作和向数组中插入数组操作；添加一个 RichTextBox 控件，用来显示插入数组后的新数组。

（3）程序的主要代码如下：

```
/// <summary>
///向一维数组中添加一个数组
/// </summary>
/// <param name="ArrayBorn">源数组</param>
/// <param name="ArrayAdd">要添加的数组</param>
/// <param name="Index">添加索引</param>
/// <returns>新得到的数组</returns>
static int[] AddArray(int[] ArrayBorn, int[] ArrayAdd, int Index)
{
    if (Index >= (ArrayBorn.Length))                    //判断添加索引是否大于数组的长度
        Index = ArrayBorn.Length - 1;                   //将添加索引设置为数组的最大索引
    int[] TemArray = new int[ArrayBorn.Length + ArrayAdd.Length];  //声明一个新的数组
    for (int i = 0; i < TemArray.Length; i++)           //遍历新数组的元素
    {
        if (Index >= 0)                                 //判断添加索引是否大于等于 0
        {
            if (i < (Index + 1))                        //判断遍历到的索引是否小于添加索引加 1
                TemArray[i] = ArrayBorn[i];             //交换元素值
            else if (i == (Index + 1))                  //判断遍历到的索引是否等于添加索引加 1
            {
                for (int j = 0; j < ArrayAdd.Length; j++)   //遍历要添加的数组
                    TemArray[i + j] = ArrayAdd[j];          //为遍历到的索引设置添加值
                i = i + ArrayAdd.Length - 1;                //将遍历索引设置为要添加数组的索引最大值
            }
            else
                TemArray[i] = ArrayBorn[i - ArrayAdd.Length];   //交换元素值
        }
        else
        {
            if (i == 0)                                 //判断遍历到的索引是否为 0
            {
                for (int j = 0; j < ArrayAdd.Length; j++)   //遍历要添加的数组
                    TemArray[i + j] = ArrayAdd[j];          //为遍历到的索引设置添加值
                i = i + ArrayAdd.Length - 1;                //将遍历索引设置为要添加数组的索引最大值
            }
            else
                TemArray[i] = ArrayBorn[i - ArrayAdd.Length];   //交换元素值
        }
    }
    return TemArray;                                    //返回添加数组后的新数组
}
```

秘笈心法

心法领悟 096：数组的定义。

数组类型的值是对象，数组对象被定义为存储数组元素类型值的一系列位置。也就是说，数组是一个存储一系列元素位置的对象。数组中存储位置的数量由数组的秩和边界来确定。

实例 097　不改变长度删除数组中的元素

光盘位置：光盘\MR\04\097　　初级　　趣味指数：★★★☆

实例说明

在程序中删除数组元素时主要有两种情况，一是在不改变数组元素总数的情况下删除指定元素（也就是用删除元素后面的元素覆盖要删除的元素）；二是删除指定元素后，根据删除元素的个数 n，使删除后的数组长度减 n。本实例主要演示第一种情况，即在不改变长度的情况下，删除数组中的元素。实例运行效果如图 4.5 所示。

图 4.5　不改变长度删除数组中的元素

关键技术

本实例在实现时，首先需要定义一个一维数组、要删除的开始索引和要删除的长度，然后判断要删除的开始索引和删除的长度是否超出了数组的范围，如果超出，则返回；否则，使用数组中后面的值覆盖前面的值，并将删除长度后面的数组元素值全部初始化为 0，这样就实现了在不改变数组长度的情况下，删除数组中元素的功能。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DelArrayNoLength。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加 3 个 TextBox 控件，分别用来显示随机生成的数组和输入开始删除的索引、要删除的元素个数；添加两个 Button 控件，分别用来执行随机生成数组操作和删除数组中的元素操作；添加一个 RichTextBox 控件，用来显示删除元素后的新数组。

（3）程序的主要代码如下：

```
/// <summary>
///删除数组中的元素
/// </summary>
/// <param name="ArrayBorn">要从中删除元素的数组</param>
/// <param name="Index">删除索引</param>
/// <param name="Len">删除的长度</param>
static void DeleteArray(int[] ArrayBorn, int Index, int Len)
{
    if (Len <= 0)                                          //判断删除长度是否小于等于 0
        return;                                            //返回
    if (Index == 0 && Len >= ArrayBorn.Length)             //判断删除长度是否超出了数组范围
        Len = ArrayBorn.Length;                            //将删除长度设置为数组的长度
    else if ((Index + Len) >= ArrayBorn.Length)            //判断删除索引和长度的和是否超出了数组范围
        Len = ArrayBorn.Length - Index - 1;                //设置删除的长度
    int i = 0;                                             //定义一个 int 变量，用来标识开始遍历的位置
    for (i = 0; i < ArrayBorn.Length - Index - Len; i++)   //遍历删除的长度
        ArrayBorn[i + Index] = ArrayBorn[i + Len + Index]; //覆盖要删除的值
    //遍历删除长度后面的数组元素值
    for (int j = (ArrayBorn.Length - 1); j > (ArrayBorn.Length - Len - 1); j--)
        ArrayBorn[j] = 0;                                  //设置数组为 0
}
```

秘笈心法

心法领悟 097：了解什么是静态数组？

静态数组是指数组元素的个数是固定不变的，即它们占用的内存空间大小是固定不变的。

实例 098 删除数组元素后改变其长度

光盘位置：光盘\MR\04\098　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在删除数组元素后改变其长度。运行本实例，首先单击“随机生成数组”按钮，生成一个数组；然后分别输入开始删除的索引和要删除的元素个数；最后单击“确定”按钮，从生成的数组的指定索引处删除指定的元素个数，从而生成一个新的数组。实例运行效果如图 4.6 所示。

图 4.6　删除数组元素后改变其长度

关键技术

本实例在实现时，首先需要定义一个一维数组、要删除的开始索引和要删除的长度，然后判断要删除的开始索引和删除的长度是否超出了数组的范围。如果删除长度超出了数组范围，则将要删除的元素个数设置为数组的长度；如果删除的开始索引和删除长度超出了数组范围，则将要删除的元素个数设置为数组的长度减去删除开始索引，再减去 1；然后定义一个新的数组，其长度设置为原数组长度减去上述运算所得到的长度，最后遍历新数组，并且为新数组的每一个索引赋相应的值即可，这样就实现了删除数组元素后改变其长度的功能。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DelArrayYesLength。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加 3 个 TextBox 控件，分别用来显示随机生成的数组和输入开始删除的索引、要删除的元素个数；添加两个 Button 控件，分别用来执行随机生成数组操作和删除数组中的元素操作；添加一个 RichTextBox 控件，用来显示删除元素后的新数组。

（3）程序的主要代码如下：

```
/// <summary>
///删除数组中的元素，并改变数组的长度
/// </summary>
/// <param name="ArrayBorn">要从中删除元素的数组</param>
/// <param name="Index">删除索引</param>
/// <param name="Len">删除的长度</param>
/// <returns>得到的新数组</returns>
static int[] DeleteArray(int[] ArrayBorn, int Index, int Len)
{
    if (Len <= 0)                                          //判断删除长度是否小于等于 0
        return ArrayBorn;                                  //返回源数组
    if (Index == 0 && Len >= ArrayBorn.Length)             //判断删除长度是否超出了数组范围
        Len = ArrayBorn.Length;                            //将删除长度设置为数组的长度
    else if ((Index + Len) >= ArrayBorn.Length)            //判断删除索引和长度的和是否超出了数组范围
        Len = ArrayBorn.Length - Index - 1;                //设置删除的长度
    int[] temArray = new int[ArrayBorn.Length - Len];      //声明一个新的数组
    for (int i = 0; i < temArray.Length; i++)              //遍历新数组
    {
        if (i >= Index)                                    //判断遍历索引是否大于等于删除索引
            temArray[i] = ArrayBorn[i + Len];              //为遍历到的索引元素赋值
        else
            temArray[i] = ArrayBorn[i];                    //为遍历到的索引元素赋值
    }
    return temArray;                                       //返回得到的新数组
}
```

秘笈心法

心法领悟 098：了解什么是动态数组？

动态数组的声明实际上就是将数组的声明部分和初始化部分分别写在不同的语句中。动态数组的初始化也需要使用 new 关键字为数组元素分配内存空间，并为数组元素赋初值。

4.2　常用数组排序算法

实例 099	使用选择排序法对一维数组进行排序 光盘位置：光盘\MR\04\099	初级 趣味指数：★★★★

实例说明

本实例演示如何使用选择排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“选择排序”按钮，使用选择排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例运行效果如图 4.7 所示。

关键技术

本实例实现时主要用到了选择排序算法，下面对选择排序算法的实现原理进行详细讲解。

选择排序的基本思想是，每一趟在 n 个记录中选取关键字最小的记录作为有序序列的第 I 个记录，并且令 I 为 1～n-1，进行 n-1 趟选择操作。

使用选择排序法排序的过程如图 4.8 所示。

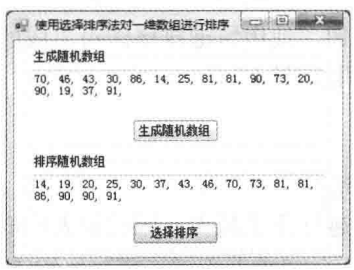


图 4.7　使用选择排序法对一维数组进行排序

原序列	94	35	61	53	77	9	12	39
第 1 遍选择	9	35	61	53	77	94	12	39
第 2 遍选择	9	12	61	53	77	94	35	39
第 3 遍选择	9	12	35	53	77	94	61	39
第 4 遍选择	9	12	35	39	77	94	61	53
第 5 遍选择	9	12	35	39	53	94	61	77
第 6 遍选择	9	12	35	39	53	61	94	77
第 7 遍选择	9	12	35	39	53	61	77	94

图 4.8　选择排序法排序的过程

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用选择排序法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和使用选择排序法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void btn_sort_Click(object sender, EventArgs e)
 {
     if (G_int_value != null)
     {
         int min;                                          //定义一个 int 变量，用来存储数组下标
         for (int i = 0; i < G_int_value.Length - 1; i++)  //循环访问数组中的元素值（除最后一个）
         {
             min = i;                                      //为定义的数组下标赋值
             for (int j = i + 1; j < G_int_value.Length; j++)  //循环访问数组中的元素值（除第一个）
```

```
            {
                if (G_int_value[j] < G_int_value[min])          //判断相邻两个元素值的大小
                    min = j;
            }
            int t = G_int_value[min];                           //定义一个 int 变量，用来存储比较大的数组元素值
            G_int_value[min] = G_int_value[i];                  //将小的数组元素值移动到前一位
            G_int_value[i] = t;                                 //将 int 变量中存储的较大的数组元素值向后移
        }
        txt_str2.Clear();                                       //清空控件内的字符串
        foreach (int i in G_int_value)                          //遍历字符串集合
        {
            txt_str2.Text += i.ToString() + ", ";               //向控件内添加字符串
        }
    }
    else
    {
        MessageBox.Show("首先应当生成数组，然后再进行排序。","提示！");
    }
}
```

秘笈心法

心法领悟 099：声明数组时注意数组的大小。

声明数组时，数组大小必须与大括号中的元素个数相匹配，否则会产生编辑错误。

实例 100　使用冒泡排序法对一维数组进行排序　初级

光盘位置：光盘\MR\04\100　趣味指数：★★★★

实例说明

本实例演示如何使用冒泡排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“冒泡排序”按钮，使用冒泡排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例运行效果如图 4.9 所示。

关键技术

本实例实现时主要用到了冒泡排序算法，下面对冒泡排序算法的实现原理进行详细讲解。

冒泡排序是一种最常用的排序方法，其过程很简单，就像气泡一样越往上走越大，因此被人们形象地称为冒泡排序法。冒泡排序的过程很简单，首先将第一个记录的关键字和第二个记录的关键字进行比较，若为逆序，则将两个记录交换，然后比较第二个记录和第 3 个记录的关键字，依此类推，直至第 n-1 个记录和第 n 个记录的关键字进行过比较为止，上述过程称为第一趟冒泡排序，执行 n-1 次上述过程后，排序即可完成。

使用冒泡排序法排序的过程如图 4.10 所示。

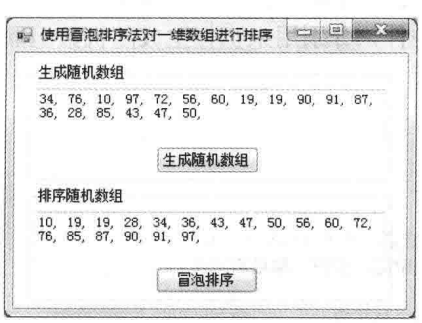


图 4.9　使用冒泡排序法对一维数组进行排序

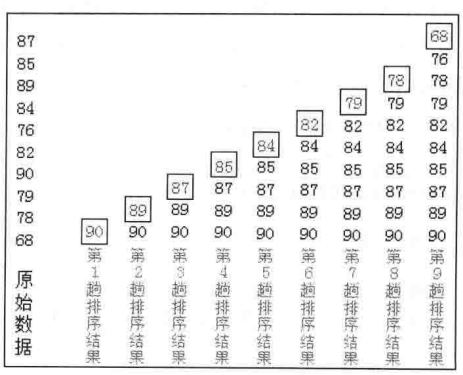


图 4.10　冒泡排序法排序的过程

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BubbleUpSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用冒泡排序法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和使用冒泡排序法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void btn_sort_Click(object sender, EventArgs e)
{
    if (G_int_value != null)
    {
        //定义两个 int 类型的变量，分别用来表示数组下标和存储新的数组元素
        int j, temp;
        for (int i = 0; i < G_int_value.Length - 1; i++)          //根据数组下标的值遍历数组元素
        {
            j = i + 1;
        id:                                                       //定义一个标识，以便从这里开始执行语句
            if (G_int_value[i] > G_int_value[j])                  //判断前后两个数的大小
            {
                temp = G_int_value[i];                            //将比较后大的元素赋值给定义的 int 变量
                G_int_value[i] = G_int_value[j];                  //将后一个元素的值赋值给前一个元素
                G_int_value[j] = temp;                            //将 int 变量中存储的元素值赋值给后一个元素
                goto id;                                          //返回标识，继续判断后面的元素
            }
            else
                if (j < G_int_value.Length - 1)                   //判断是否执行到最后一个元素
                {
                    j++;                                          //如果没有，则再往后判断
                    goto id;                                      //返回标识，继续判断后面的元素
                }
        }
        txt_str2.Clear();                                         //清空控件内的字符串
        foreach (int i in G_int_value)                            //遍历字符串集合
        {
            txt_str2.Text += i.ToString() + ", ";                 //向控件内添加字符串
        }
    }
    else
    {
        MessageBox.Show("首先应当生成数组，然后再进行排序。","提示！");
    }
}
```

秘笈心法

心法领悟 100：编译器可以自动计算数组长度。

实际上，初始化数组时可以省略 new 运算符和数组的长度，编译器将根据初始值的数量来自动计算数组长度，并创建数组。

实例 101　使用快速排序法对一维数组进行排序

光盘位置：光盘\MR\04\101　　初级　　趣味指数：★★★★

实例说明

本实例演示如何使用快速排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“快速排序”按钮，使用快速排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例运行效果如图 4.11 所示。

关键技术

本实例实现时主要用到了快速排序算法，下面对快速排序算法的实现原理进行详细讲解。

快速排序算法是对冒泡排序算法的一种改进，其基本思想是，通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此使整个数据变成有序序列。

假设要排序的数组是 A[1]…A[N]，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，递归调用此过程，即可实现数组的快速排序。

使用快速排序法排序的过程如图 4.12 所示。

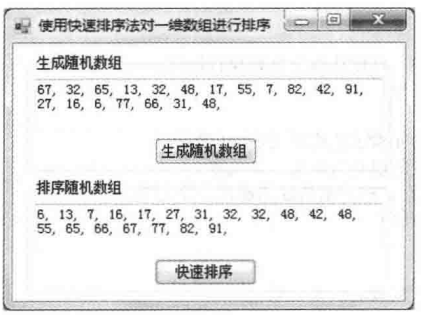


图 4.11　使用快速排序法对一维数组进行排序

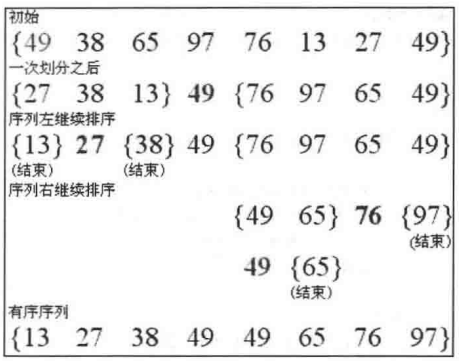


图 4.12　快速排序法排序的过程

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FastSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用快速排序法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和使用快速排序法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void swap(ref int l, ref int r)                    //交换数据
{
    int temp;                                              //临时值
    temp = l;                                              //记录前一个值
    l = r;                                                 //记录后一个值
    r = temp;                                              //前后交换数据
}
private void Sort(int[] list, int low, int high)
{
    int pivot;                                             //临时变量，用来存储最大值
    int l, r;                                              //分别用来记录遍历到的索引和最大索引
    int mid;                                               //中间索引
    if (high <= low)                                       //判断输入的值是否合法
        return;
    else if (high == low + 1)                              //判断两个索引是否相邻
    {
        if (list[low] > list[high])                        //判断前面的值是否大于后面的值
            swap(ref list[low], ref list[high]);           //交换前后索引的值
        return;
    }
    mid = (low + high) >> 1;                               //记录数组的中间索引
    pivot = list[mid];                                     //初始化临时变量的值
    swap(ref list[low], ref list[mid]);                    //交换第一个值和中间值的索引顺序
    l = low + 1;                                           //记录遍历到的索引值
```

```
    r = high;                                   //记录最大索引
    do                                          //使用 do…while 循环遍历数组，并比较前后值的大小
    {
        while (l <= r && list[l] < pivot)       //判断遍历到的索引是否小于最大索引
            l++;                                //索引值加 1
        while (list[r] >= pivot)                //判断最大值是否大于等于记录的分支点
            r--;                                //最大索引值减 1
        if (l < r)                              //如果当前遍历到的值小于最大值
            swap(ref list[l], ref list[r]);     //交换顺序
    } while (l < r);
    list[low] = list[r];                        //在最小索引处记录最小值
    list[r] = pivot;                            //在最大索引处记录最大值
    if (low + 1 < r)                            //判断最小索引是否小于最大索引
        Sort(list, low, r - 1);                 //调用自身进行快速排序
    if (r + 1 < high)                           //判断最大索引是否小于数组长度
        Sort(list, r + 1, high);                //调用自身进行快速排序
}
```

秘笈心法

心法领悟 101：如何评价排序算法的好坏。

评价排序算法好坏的标准主要有两条，分别为：

（1）指定时间和所需的辅助控件。

（2）算法本身的复杂程度。

实例 102　使用直接插入法对一维数组进行排序

光盘位置：光盘\MR\04\102　　初级　　趣味指数：★★★★

实例说明

本实例演示如何使用直接插入排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“插入排序”按钮，使用直接插入排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例运行效果如图 4.13 所示。

关键技术

本实例实现时主要用到了直接插入排序算法，下面对直接插入排序算法的实现原理进行详细讲解。

插入排序是将一个记录插入到有序数列中，使得到的新数列仍然有序。插入排序的算法思想是：将 n 个有序数存放在数组 a 中，要插入的数为 x，首先确定 x 插在数组中的位置 p，数组中 p 之后的元素都向后移一个位置，空出 a(p)，将 x 放入 a(p)。这样即可实现插入后数列仍然有序。

使用插入排序法排序的过程如图 4.14 所示。

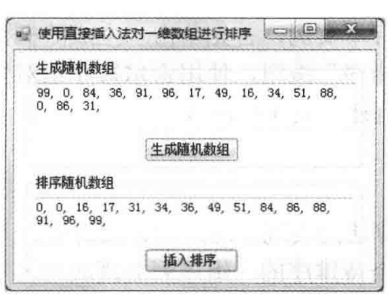


图 4.13　使用直接插入法对一维数组进行排序

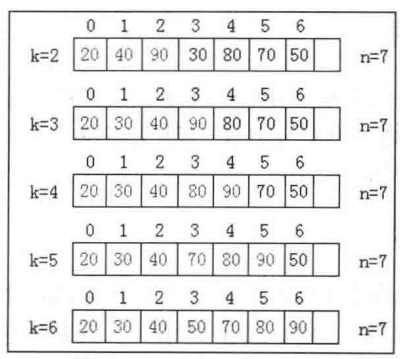


图 4.14　直接插入排序法排序的过程

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InsertSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用直接插入排序法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和使用直接插入排序法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void btn_sort_Click(object sender, EventArgs e)
{
    if (G_int_value != null)
    {
        for (int i = 0; i < G_int_value.Length; ++i)                    //循环访问数组中的元素
        {
            int temp = G_int_value[i];                                  //定义一个 int 变量，并使用获得的数组元素值赋值
            int j = i;
            while ((j > 0) && (G_int_value[j - 1] > temp))              //判断数组中的元素是否大于获得的值
            {
                G_int_value[j] = G_int_value[j - 1];                    //如果是，则将后一个元素的值提前
                --j;
            }
            G_int_value[j] = temp;                                      //最后将 int 变量存储的值赋值给最后一个元素
        }
        txt_str2.Clear();                                               //清空控件内的字符串
        foreach (int i in G_int_value)                                  //遍历字符串集合
        {
            txt_str2.Text += i.ToString() + ", ";                       //向控件内添加字符串
        }
    }
    else
    {
        MessageBox.Show("首先应当生成数组，然后再进行排序。", "提示！");
    }
}
```

秘笈心法

心法领悟 102：什么是排序算法的空间复杂度？

排序算法的空间复杂度是指排序算法运行所需要的额外消耗存储空间，一般使用 O()来表示。例如，直接插入排序法的空间复杂度为 O(1)。

实例 103　使用希尔排序法对一维数组进行排序

光盘位置：光盘\MR\04\103　　初级　　趣味指数：★★★★

实例说明

本实例演示如何使用希尔排序法对一维数组进行排序。运行本实例，首先单击“生成随机数组”按钮，生成一个随机数组，并显示在上方的文本框中；然后单击“希尔排序”按钮，使用希尔排序法对生成的一维数组进行排序，并将排序后的一维数组显示在下方的文本框中。实例运行效果如图 4.15 所示。

关键技术

本实例实现时主要用到了希尔排序算法，下面对希尔排序算法的实现原理进行详细讲解。

希尔排序又称“缩小增量排序”，其基本思想是，先将整个待排序的一组序列分割成为若干子序列，然后分别进行直接插入排序，待整个序列中的数“基本有序”时再对全体记录进行一次直接插入排序。

说明：子序列的构成不是简单地“逐段分割”，而是将相隔某个“增量”的数组成一个子序列，这是希尔排序的特点。

使用希尔排序法排序的过程如图 4.16 所示。

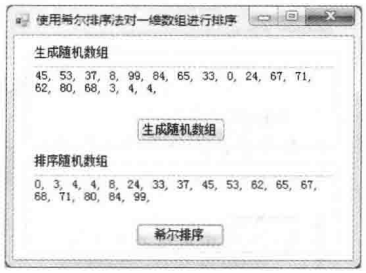


图 4.15　使用希尔排序法对一维数组进行排序

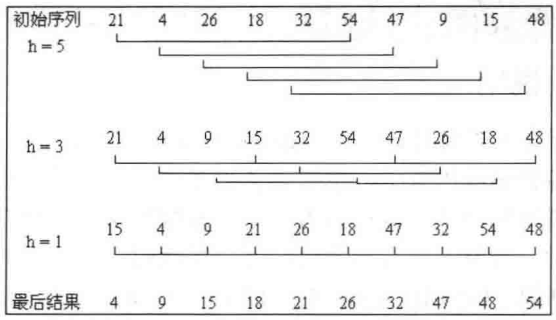


图 4.16　希尔排序法排序的过程

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 XESort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用希尔排序法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和使用希尔排序法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void btn_sort_Click(object sender, EventArgs e)
{
    if (G_int_value != null)
    {
        int inc;                                                    //定义一个 int 变量，用来确定每个有序序列的个数
        for (inc = 1; inc <= G_int_value.Length / 9; inc = 3 * inc + 1) ;   //为有序序列赋值
        for (; inc > 0; inc /= 3)
        {
            for (int i = inc + 1; i <= G_int_value.Length; i += inc)
            {
                int t = G_int_value[i - 1];                         //记录当前值
                int j = i;                                          //定义下一个索引
                while ((j > inc) && (G_int_value[j - inc - 1] > t))
                {
                    G_int_value[j - 1] = G_int_value[j - inc - 1];  //交换数据
                    j -= inc;
                }
                G_int_value[j - 1] = t;                             //将下一个元素值设置为当前值
            }
        }
        txt_str2.Clear();                                           //清空控件内的字符串
        foreach (int i in G_int_value)                              //遍历字符串集合
        {
            txt_str2.Text += i.ToString() + ", ";                   //向控件内添加字符串
        }
    }
    else
    {
        MessageBox.Show("首先应当生成数组，然后再进行排序。", "提示！");
    }
}
```

秘笈心法

心法领悟 103：排序算法的时间开销。

大多数排序算法的时间开销主要是关键字之间的比较和记录的移动，但有的排序算法的执行时间不仅依赖

于问题的规模，还取决于输入数据的状态。

实例 104　使用 Sort 方法对数组进行快速排序

光盘位置：光盘\MR\04\104　　初级　　趣味指数：★★★★

实例说明

实际开发项目时，经常需要对程序中用到的数组进行排序，而且在各种编程语言中，也提供了很多种对数组进行排序的算法，如冒泡排序法、直接插入法和选择排序法等，但在使用排序算法时，开发人员必须手动编写一堆代码，而且有的实现起来比较麻烦。C#中的 Array 类提供了一个 Sort 方法，使用这个方法，开发人员可以很方便地对各种数组进行排序，大大降低了数组排序的难度，本实例就将使用该方法来方便地对数组进行快速排序。实例运行效果如图 4.17 所示。

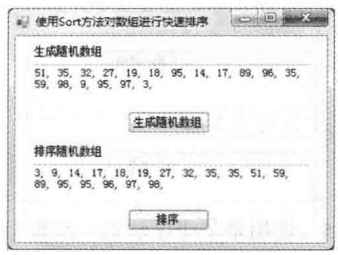


图 4.17　使用 Sort 方法对数组进行快速排序

关键技术

本实例在对数组进行快速排序时，主要用到了 Array 类的 Sort 方法，下面对其进行详细讲解。

Sort 方法使用 Array 类中每个元素的 IComparable 接口实现，用来对整个一维 Array 数组中的元素进行排序，该方法有数十种重载形式，本实例中用到的该方法的语法格式如下：

```
public static void Sort(Array array)
```

参数说明

array：要排序的一维 Array 数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SortArray。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 TextBox 控件，分别用来显示原始数组和使用 Sort 方法排序之后的数组；添加两个 Button 控件，分别用来执行生成随机数组操作和调用 Sort 方法对数组进行排序操作。

（3）程序的主要代码如下：

```
private void btn_sort_Click(object sender, EventArgs e)
{
    if (G_int_value != null)
    {
        Array.Sort(G_int_value);                          //使用 Sort 方法对数组排序
        txt_str2.Clear();                                 //清空控件内的字符串
        foreach (int i in G_int_value)                    //遍历字符串集合
        {
            txt_str2.Text += i.ToString() + ", ";         //向控件内添加字符串
        }
    }
    else
    {
        MessageBox.Show("首先应当生成数组，然后再进行排序。","提示！");
    }
}
```

秘笈心法

心法领悟 104：有效利用 Array 类。

Array 类提供了创建、操作、搜索和排序数组的方法，因而在公共语言运行库中用作所有数组的基类。在数

组中，一个元素就是 Array 类中的一个值。Array 类的长度是它可包含的元素总数，Array 类的秩是 Array 类中的维数，Array 类中维度的下限是 Array 类中该维度的起始索引。

实例 105　反转数组中元素的顺序

光盘位置：光盘\MR\04\105　　初级　　趣味指数：★★★★

实例说明

反转数组中元素的顺序，字面理解就是对数组中的元素进行倒序输出，实现该功能时，C#中的 Array 类提供了一个 Reverse 方法，使用该方法可以很方便地实现反转数组中的元素顺序功能。实例运行效果如图 4.18 所示。

图 4.18　反转数组中元素的顺序

关键技术

本实例在反转数组中元素的顺序时，主要用到了 Array 类的 Reverse 方法，下面对其进行详细讲解。

Reverse 方法用来反转一维 Array 数组或部分 Array 数组中元素的顺序，该方法有两种重载形式，其语法格式分别如下：

```
public static void Reverse(Array array)
public static void Reverse(Array array,int index,int length)
```

参数说明

❶ array：要反转的一维 Array 数组。

❷ index：要反转部分的起始索引。

❸ length：要反转部分中的元素数。

说明：本实例用到的是 Reverse 方法的第一种重载形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Reverse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 Label 控件，分别用来显示原始数组和反转之后的数组；添加一个 Button 控件，用来对数组执行反转操作。

（3）程序的主要代码如下：

```
private string[] G_str_array;                                    //定义字符串数组字段
private void btn_Reverse_Click(object sender, EventArgs e)
{
    lab_str2.Text = string.Empty;                                //清空信息
    foreach (string str in G_str_array.Reverse())                //遍历反转后的字符串数组
    {
        lab_str2.Text += str + Environment.NewLine;              //显示字符串数组
    }
}
```

秘笈心法

心法领悟 105：“反转”≠“倒序排序”。

“反转”并不等于“倒序排序”，“反转”只是将数组元素的顺序进行颠倒，而并不对其执行排序操作；而“倒序排序”则是对数组中的元素进行从大到小的排序。

4.3 常用集合的使用

实例 106 向班级集合中添加学生信息

光盘位置：光盘\MR\04\106　　中级　　趣味指数：★★★★☆

实例说明

集合在程序开发中经常用到，如可以将学生信息、商品信息等存储到集合中，以便随时更新。本实例将使用集合存储学生信息。实例运行效果如图 4.19 所示。

图 4.19　向班级集合中添加学生信息

关键技术

本实例实现时主要用到了 ArrayList 集合及其 Add 方法，下面对本实例用到的关键技术进行详细讲解。

（1）ArrayList 集合

ArrayList 集合类位于 System.Collections 命名空间下，它可以动态地添加和删除元素。ArrayList 集合类相当于一种高级的动态数组，它是 Array 类的升级版本，但它并不等同于数组。

与数组相比，ArrayList 集合类为开发人员提供了以下功能：

- 数组的容量是固定的，而 ArrayList 的容量可以根据需要自动扩充。
- ArrayList 提供了添加、删除和插入某一范围元素的方法，但在数组中，只能一次获取或设置一个元素的值。
- ArrayList 提供将只读和固定大小包装返回到集合的方法，而数组不提供。
- ArrayList 只能是一维形式，而数组可以是多维的。

ArrayList 集合类的默认构造器格式如下：

```
public ArrayList();
```

通过以上构造器声明 ArrayList 对象的语法格式如下：

```
ArrayList List = new ArrayList();
```

参数说明

List：ArrayList 对象名。

（2）ArrayList 类的 Add 方法

ArrayList 类的 Add 方法用来将对象添加到 ArrayList 集合的结尾处。语法格式如下：

```
public virtual int Add(Object value)
```

参数说明

❶ value：要添加到 ArrayList 的末尾处的 Object，该值可以为空引用。

❷ 返回值：ArrayList 索引，已在此处添加了 value。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddStudentINList。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加一个 DataGridView 控件，用来显示 ArrayList 集合中显示的学生信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    ArrayList P_list_StudentInfo = new ArrayList();                    //创建集合对象
    //定义两个字符串数组，分别用来记录学生整体信息和分解后的信息
    string[] P_str_Students, P_str_Info;
    string P_str_Student = "";                                         //定义一个字符串变量，用来记录所有学生信息
    //向集合中添加学生信息
    P_list_StudentInfo.Add("小王 男 1980-01-01");
    P_list_StudentInfo.Add("小刘 女 1981-01-01");
    P_list_StudentInfo.Add("小赵 男 1990-01-01");
    P_list_StudentInfo.Add("小吕 男 1995-01-01");
    P_list_StudentInfo.Add("小梁 男 2000-01-01");
    foreach (string Pc_str_Student in P_list_StudentInfo)              //遍历集合
    {
        P_str_Student += Pc_str_Student + ",";                         //记录所有学生信息
    }
    P_str_Students = P_str_Student.Split(',');                         //将学生信息存储在一个字符串数组中
    dgv_Info.Rows.Add(5);                                              //为 DataGridView 控件添加 5 行
    for (int i = 0; i < P_str_Students.Length - 1; i++)                //遍历存储学生整体信息的数组
    {
        P_str_Info = P_str_Students[i].Split(' ');                     //将学生整体信息进行分解
        dgv_Info.Rows[i].Cells[0].Value = P_str_Info[0];               //显示学生姓名
        dgv_Info.Rows[i].Cells[1].Value = P_str_Info[1];               //显示性别
        dgv_Info.Rows[i].Cells[2].Value = P_str_Info[2];               //显示出生年月
    }
}
```

秘笈心法

心法领悟 106：更高级的 List<T>泛型集合。

List<T>泛型集合表示可通过索引访问的对象的强类型列表，它提供了用于对列表进行搜索、排序和操作的方法，相对于 ArrayList 类来说，List<T>泛型集合在大多数情况下执行得更好并且是类型安全的。

实例 107　使用哈希表对 XML 文件进行查询

光盘位置：光盘\MR\04\107　　中级　　趣味指数：★★★★★

实例说明

开发网络电台应用程序时，可以将网络电台的地址及名称存放到 XML 文件中，这时如果需要将 XML 文件中存储的所有电台地址及对应名称显示出来，可以使用哈希表来实现。本实例将通过使用哈希表对 XML 文件进行查询。实例运行效果如图 4.20 所示。

图 4.20　使用哈希表对 XML 文件进行查询

关键技术

本实例实现时主要用到了 Hashtable 哈希表及其 Add 方法，下面对本实例用到的关键技术进行详细讲解。

（1）Hashtable 哈希表

Hashtable 通常称为哈希表，它表示键/值对的集合，这些键/值对根据键的哈希代码进行组织，它的每个元素都是一个存储于 DictionaryEntry 对象中的键/值对。

注意：Hashtable 哈希表中的键不能为空引用，但值可以。

Hashtable 哈希表的默认构造器格式如下：

```
public Hashtable();
```

通过以上构造器声明 Hashtable 对象的语法格式如下：

```
Hashtable HTable = new Hashtable();
```

参数说明

HTable：Hashtable 对象名。

（2）Hashtable 哈希表的 Add 方法

Hashtable 哈希表的 Add 方法用来将带有指定键和值的元素添加到 Hashtable 哈希表中。语法格式如下：

```
public virtual void Add(Object key,Object value)
```

参数说明

❶ key：要添加的元素的键。

❷ value：要添加的元素的值，该值可以为空引用。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectXMLByHashtable。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加两个 ComboBox 控件，分别用来显示 XML 文件中存储的网络电台名称和电台网址。

（3）程序的主要代码如下：

```
/// <summary>
///在 XML 文件中查找电台地址及名称
/// </summary>
/// <param name="strPath">XML 文件路径</param>
/// <returns>Hashtable 对象，用来记录找到的电台地址及名称</returns>
static Hashtable SelectXML(string strPath)
{
    Hashtable HTable = new Hashtable();                                    //创建哈希表对象
    XmlDocument doc = new XmlDocument();                                   //创建 XML 文档对象
    doc.Load(strPath);                                                     //加载 XML 文档
    XmlNodeList xnl = doc.SelectSingleNode("BCastInfo").ChildNodes;        //获取 NewDataSet 节点的所有子节点
    string strVersion = "";                                                //定义一个字符串，用来记录电台地址
    string strInfo = "";                                                   //定义一个字符串，用来记录电台名称
    foreach (XmlNode xn in xnl)                                            //遍历所有子节点
    {
        XmlElement xe = (XmlElement)xn;                                    //将子节点类型转换为 XmlElement 类型
        if (xe.Name == "DInfo")                                            //判断节点名为 DInfo
        {
            XmlNodeList xnlChild = xe.ChildNodes;                          //继续获取 xe 子节点的所有子节点
            foreach (XmlNode xnChild in xnlChild)                          //遍历
            {
                XmlElement xeChild = (XmlElement)xnChild;                  //转换类型
                if (xeChild.Name == "Address")
                {
                    strVersion = xeChild.InnerText;                        //记录电台地址
                }
                if (xeChild.Name == "Name")
                {
                    strInfo = xeChild.InnerText;                           //记录电台名称
                }
            }
            HTable.Add(strVersion, strInfo);                               //向哈希表中添加键值
        }
    }
    return HTable;
}
```

秘笈心法

心法领悟 107：XML 文件的使用。

XML 是 eXtensible Markup Language 的缩写，它是一种类似于 HTML 的标记语言。XML 以简易而标准的方式保存各种信息（如文字和数字等信息），适用于不同应用程序间的数据交换，而这种交换不以预先定义的一

组数据结构为前提，增强了可扩展性。

实例 108　计算两个矩形矩阵的乘积

光盘位置：光盘\MR\04\108　　高级　　趣味指数：★★★★☆

实例说明

矩阵相当于一个数组，主要用来存储一系列数，例如，m×n 矩阵是排列在 m 行和 n 列中的一系列数，m×n 矩阵可与一个 n×p 矩阵相乘，结果为一个 m×p 矩阵。这里需要注意的是，如果两个矩阵相乘，第一个矩阵的列数必须与第二个矩阵的行数相同。本实例将演示如何计算两个矩形矩阵的乘积。实例运行效果如图 4.21 所示。

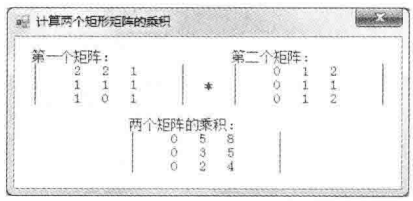


图 4.21　计算两个矩形矩阵的乘积

关键技术

本实例实现时主要用到了矩阵以及矩阵的乘法原理，下面对本实例用到的关键技术进行详细讲解。

（1）矩阵

矩阵是指纵横排列的二维数据表格，最早来自于方程组的系数及常数所构成的方阵，这一概念是由 19 世纪英国数学家凯利首先提出的。矩阵的概念在生产实践中有很多应用，如矩阵图法以及保护个人账号的矩阵卡系统等。

（2）矩阵的乘法原理

矩阵乘法是一种高效的算法，它可以把一些一维递归优化到 log(n)，还可以求路径方案等。在执行两个矩阵的乘法运算时，需要将前面矩阵的第 i 行与后面矩阵的第 j 列对应的元素相乘，然后再相加，最后将得到的结果放到结果矩阵的第(i,j)这个位置上即可。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CalcRMatrix。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并在窗体中添加 3 个 Label 控件，分别用来表示第一个矩阵、第二个矩阵和两个矩阵的乘积。

（3）程序的主要代码如下：

```
#region 矩阵乘法
public void MultiplyMatrix(float[,] MatrixEin, float[,] MatrixZwei, float[,] MatrixResult)
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            for (int k = 0; k < 3; k++)
            {
                MatrixResult[i, j] += MatrixEin[i, k] * MatrixZwei[k, j];        //计算矩阵的乘积
            }
        }
    }
}
#endregion
private void Form1_Load(object sender, EventArgs e)
```

```
{
    //定义 3 个 float 类型的二维数组，作为矩阵
    float[,] MatrixEin = new float[3, 3];
    float[,] MatrixZwei = new float[3, 3];
    float[,] MatrixResult = new float[3, 3];
    //为第一个矩阵中的各个项赋值
    MatrixEin[0, 0] = 2;
    MatrixEin[0, 1] = 2;
    MatrixEin[0, 2] = 1;
    MatrixEin[1, 0] = 1;
    MatrixEin[1, 1] = 1;
    MatrixEin[1, 2] = 1;
    MatrixEin[2, 0] = 1;
    MatrixEin[2, 1] = 0;
    MatrixEin[2, 2] = 1;
    //为第二个矩阵中的各个项赋值
    MatrixZwei[0, 0] = 0;
    MatrixZwei[0, 1] = 1;
    MatrixZwei[0, 2] = 2;
    MatrixZwei[1, 0] = 0;
    MatrixZwei[1, 1] = 1;
    MatrixZwei[1, 2] = 1;
    MatrixZwei[2, 0] = 0;
    MatrixZwei[2, 1] = 1;
    MatrixZwei[2, 2] = 2;
    lab_First.Text += "第一个矩阵：\n";
    //循环遍历第一个矩阵并输出
    for (int i = 0; i < 3; i++)
    {
        lab_First.Text += "|      ";
        for (int j = 0; j < 3; j++)
        {
            lab_First.Text += MatrixEin[i, j] + "    ";
        }
        lab_First.Text += "   |\r\n";
    }
    lab_Second.Text = "第二个矩阵：\n";
    //循环遍历第二个矩阵并输出
    for (int i = 0; i < 3; i++)
    {
        lab_Second.Text += "|      ";
        for (int j = 0; j < 3; j++)
        {
            lab_Second.Text += MatrixZwei[i, j] + "    ";
        }
        lab_Second.Text += "   |\r\n";
    }
    MultiplyMatrix(MatrixEin, MatrixZwei, MatrixResult);          //调用自定义方法计算两个矩阵的乘积
    lab_Result.Text = "两个矩阵的乘积：\n";
    //循环遍历新得到的矩阵并输出
    for (int i = 0; i < 3; i++)
    {
        lab_Result.Text += "|      ";
        for (int j = 0; j < 3; j++)
        {
            lab_Result.Text += MatrixResult[i, j] + "    ";
        }
        lab_Result.Text += "   |\r\n";
    }
}
```

秘笈心法

心法领悟 108：矩阵相乘的必要条件。

如果两个矩阵相乘，则第一个矩阵的列数必须与第二个矩阵的行数相同。

第 5 章

面向对象编程技术

- 属性和方法
- 结构与类
- 面向对象技术高级应用
- 迭代器和分部类的使用
- 泛型的使用

5.1 属性和方法

实例 109	使用属性存储用户编号和姓名	初级
	光盘位置：光盘\MR\05\109	趣味指数：★★★☆

实例说明

平时开发各种数据库管理系统时，都需要用到一个用户表，该表中一般都包括用户的编号和姓名。本实例使用属性来对用户的编号和姓名进行存储，并将存储的用户编号和姓名显示到Windows 窗体上。实例运行效果如图 5.1 所示。

图 5.1 使用属性存储用户编号和姓名

关键技术

本实例实现的关键是如何定义属性，而在属性定义的过程中需要用到 get 访问器和 set 访问器。下面对本实例中用到的关键技术进行详细讲解。

（1）属性

属性是对现实实体特征的抽象，它提供了对类或对象性质的访问。类的属性描述的是状态信息，在类的实例中，属性的值表示对象的状态值。属性不表示具体的存储位置，属性有访问器，这些访问器指定在它们的值被读取或写入时需要执行的语句。所以属性提供了一种机制，把读取和写入对象的某些特性与一些操作关联起来，程序员可以像使用公共数据成员一样使用它。属性的声明格式如下：

```
【修饰符】【类型】【属性名】
{
get  {get 访问器体}
set  {set 访问器体}
}
```

参数说明

❶ 修饰符：指定属性的访问级别。

❷ 类型：指定属性的类型，可以是任何的预定义或自定义类型。

❸ 属性名：一种标识符，命名规则与字段相同，但是，属性名的第一个字母通常都大写。

（2）get 访问器

get 访问器相当于一个具有属性类型返回值的无参数方法，它除了作为赋值的目标外，当在表达式中引用属性时，将调用该属性的 get 访问器计算属性的值。get 访问器体必须用 return 语句来返回，并且所有的 return 语句都必须返回一个可隐式转换为属性类型的表达式。

（3）set 访问器

set 访问器相当于一个具有单个属性类型值参数和 void 返回类型的方法。set 访问器的隐式参数始终命名为 value。当一个属性作为赋值的目标被引用时就会调用 set 访问器，所传递的参数将提供新值。不允许 set 访问体中的 return 语句指定表达式。由于 set 访问器存在隐式的参数 value，所以 set 访问器中不能自定义使用名称为 value 的局部变量或常量。

技巧：由于 set 访问器中可以包含大量的语句，因此可以对赋予的值进行检查，如果值不安全或者不符合要求，就给予提示，这样可以避免因为给属性设置了错误的值而导致的错误。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StoreIDAndName。

（2）在默认窗体 Form1 中添加两个 Label 控件，分别用来显示存储的用户编号和姓名。

（3）程序的主要代码如下：

```
private string id = "";                                    //定义一个 string 类型的变量，用来记录用户编号
private string name = "";                                  //定义一个 string 类型的变量，用来记录用户姓名
/// <summary>
///定义用户编号属性，该属性为可读可写属性
/// </summary>
public string ID
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}
/// <summary>
///定义用户姓名属性，该属性为可读可写属性
/// </summary>
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

秘笈心法

心法领悟 109：属性和字段的区别。

属性和字段都可以访问对象中包含的数据。对象数据用来区分不同的对象，同一个类的不同对象可能在属性和字段中存储了不同的值。字段和属性都可以输入，通常把信息存储在字段和属性中，但是属性和字段是不同的，属性不能直接访问数据，字段可以直接访问数据。在属性中可以添加对数据访问的限制，例如，有一个 int 类型的属性，可以限制它只能存储 1～5 的数字，但如果用字段就可以存储任何 int 类型的数值。

实例 110	通过定义方法求一个数的平方	初级
	光盘位置：光盘\MR\05\110	趣味指数：★★★☆

实例说明

在数学运算中，求一个数的平方是经常会遇到的一个问题，求一个数的平方，实际上就是将这个数乘以其自身所得到的结果。本实例将通过自定义的方法来计算一个数的平方，实例运行效果如图 5.2 所示。

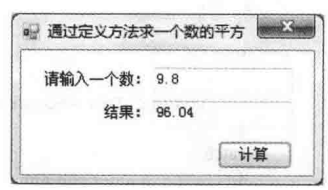


图 5.2　通过定义方法求一个数的平方

关键技术

本实例重点考查的是如何定义方法，下面对本实例用到的关键技术进行详细讲解。

方法主要用来完成类或对象的行为，在面向对象编程语言中，类或对象是通过方法来与外界交互的。所以，方法是类与外界交互的基本方式。方法是包含一系列语句的代码块，在 C#中，每个执行指令都是在方法的上下文中完成的。

声明方法时，需要为其指定修饰符，以指定其访问级别或使用限制，C#中常用的有 private、public、protected、

internal 4 个访问修饰符和 partial、new、static、virtual、override、sealed、abstract、extern 8 个声明修饰符，下面分别对它们进行简单介绍。

- private：私有访问是允许的最低访问级别，私有成员只有在声明它们的类和结构体中才可以访问。
- public：公共访问是允许的最高访问级别，对访问公共成员没有限制。
- protected：受保护成员在它的类中可访问并且可由派生类访问。
- internal：只有在同一程序集的文件中，内部类型或成员才是可访问的。
- partial：在整个同一程序集中定义分部类和结构。
- new：从基类成员隐藏继承的成员。
- static：声明属于类型本身而不是属于特定对象的成员。
- virtual：在派生类中声明其实现可由重写成员更改的方法或访问器。
- override：提供从基类继承的虚拟成员的新实现。
- sealed：指定类不能被继承。
- abstract：指示某个类只能是其他类的基类。
- extern：指示在外部实现方法。

声明方法时，除了需要指定上面所介绍的修饰符之外，还需要指定其返回值、方法名称及方法参数等。方法参数放在括号中，并用逗号隔开，如果括号中没有内容，则表示声明的方法没有参数。另外，方法声明的返回类型指定了由该方法计算和返回的值的类型，如果该方法并无返回值，则其返回类型为 void。

C#中声明方法的格式如下：

```
【修饰符】 【返回类型】 【方法名】(形参列表)
{
    【方法体】
}
```

注意：一个方法的返回类型和其形参列表中所引用的各个类型必须至少具有与该方法本身相同的可访问性。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SquareByMethod。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来输入一个数和显示输入数的平方值；添加一个 Button 控件，用来对输入的数进行求平方运算。

（3）程序的主要代码如下：

```
public double SquareNum(double num)
{
    return num * num;                                          //求一个数的平方
}
private void button1_Click(object sender, EventArgs e)
{
    string strNum = textBox1.Text.Trim();                      //记录 TextBox 文本框中的内容
    if (strNum != "")                                          //判断是否输入了数据
    {
        try
        {
            textBox2.Text = SquareNum(double.Parse(strNum)).ToString();   //调用自定义方法进行求平方运算
        }
        catch
        {
            MessageBox.Show("请输入正确的数字！");
        }
    }
}
```

秘笈心法

心法领悟 110：在 C#中计算一个数的平方的简便方法。

在 C#中计算一个数的平方时，可以直接使用 Math 类的 Pow 方法来实现。Math 类用来为三角函数、对数函数和其他通用数学函数提供常数和静态方法，而其 Pow 方法用来返回指定数字的指定次幂。

实例 111　使用重载方法实现不同类型数据的计算

光盘位置：光盘\MR\05\111　　初级　　趣味指数：★★★★

实例说明

实际生活中经常遇到不同类型数据的计算问题。例如，计算某公司某月的总工资时，就有可能遇到这种情况，假设公司员工的工资都是整数，如果没有请假和迟到早退现象，那么就是所有人的工资相加，这时用到的都是整数类型的数据相加；而如果这个月中出现了请假或者迟到早退现象，那么有的员工的工资就可能是小数类型的，这时再计算总工资时，就有可能是整数和小数类型或者小数和小数类型的数据相加。在 C#中实现以上功能最快捷的方式就是使用重载方法，然后可以设置重载方法中要计算的数据的类型不同，这样就实现了不同类型数据的计算功能。实例运行效果如图 5.3～图 5.5 所示。

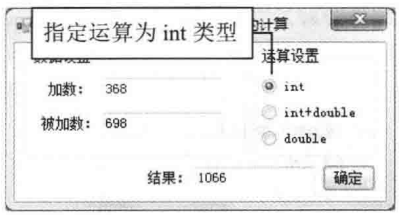


图 5.3　两个整数类型数的和

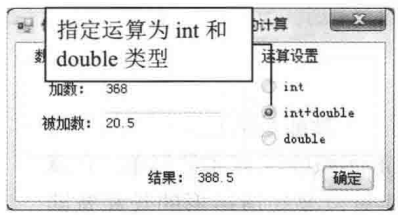


图 5.4　一个整数类型数和一个浮点类型数的和

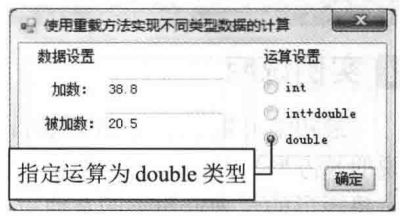


图 5.5　两个浮点类型数的和

关键技术

本实例在实现时主要用到了重载方法，下面对其进行详细讲解。

重载方法就是方法名称相同，但是每个方法中参数的数据类型、个数或顺序不同的方法。如果一个类中存在两个以上的同名方法，并且方法的参数类型、个数或者顺序不同，当调用这样的方法时，编译器会根据传入的参数自动进行判断，决定调用哪个方法。

说明： 实际应用中对重载方法有这样一种解释，重载方法就是名称相同、签名不同的方法。这里所说的方法签名，其实就是由方法的名称和它的每一个形参的类型和种类（值、引用或输出）组成的。这里需要注意的是，虽然方法签名包含 ref 和 out 参数，但是它并不区分这两种参数。例如，一个方法的重载形式有两种，一种里面只有一个 ref 参数，另一种里面只有一个 out 参数，那么这个方法在编译时就会出现异常，因为它违背了方法签名的唯一性原则。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TypesCalc。

（2）在默认窗体 Form1 中添加 3 个 TextBox 控件，分别用来输入加数、被加数和显示和；添加 3 个 RadioButton 控件，分别用来选择要计算的数据类型；添加一个 Button 控件，用来对输入的两个数进行求和运算。

（3）程序的主要代码如下：

```
public static int Add(int x, int y)                    //定义一个静态方法 Add，返回值为 int 类型，有两个 int 类型的参数
{
    return x + y;
}
public double Add(int x, double y)                     //重新定义方法 Add，它与第一个方法的返回值类型及参数类型不同
{
    return x + y;
}
```

```
public double Add(double x, double y)          //重新定义方法 Add，它与第一个方法的返回值类型及参数类型不同
{
    return x + y;
}
```

秘笈心法

心法领悟 111：重载的使用范围。

（1）除了类以外，重载也可以出现在接口或者结构中。

（2）除了方法以外，也可以对构造函数、索引器以及运算符等进行重载。

5.2 结 构 与 类

实例 112	通过结构计算矩形的面积 光盘位置：光盘\MR\05\112	初级 趣味指数：★★★★

实例说明

最初学习数学中关于面积的计算时，一般都是从矩形（即人们常说的长方形）的面积开始学起，因为矩形面积的计算非常简单，它就是将矩形的长乘以矩形的宽即可，因此只需要知道矩形的长和宽即可。本实例中定义了一个矩形结构，其中定义了矩形的长和宽，并且自定义了一个 Area 方法，用来计算矩形的面积；当在程序中具体使用时，只需要对结构中定义的矩形的长和宽赋值，然后调用自定义方法即可计算矩形的面积。实例运行效果如图 5.6 所示。

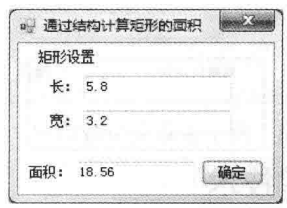


图 5.6 通过结构计算矩形的面积

关键技术

本实例重点考查的是如何定义和使用结构，下面对本实例用到的关键技术进行详细讲解。

结构是一种可以包含数据成员和方法成员的值类型数据结构。为结构分配数据时不需要从托管堆中分配内存，结构类型的变量直接包含了该结构的数据。结构中可以包括构造函数、常量、字段、方法、属性、运算符、事件和嵌套类型等，但如果要同时包括上述几种成员，则应该考虑使用类。

使用结构对于小的数据结构特别有用，能够节省大量的分配内存，如复数和坐标系中的点。

结构具有以下特点：

- 结构是值类型。
- 向方法传递结构时，是通过传值方式传递的，而不是作为引用传递的。
- 结构的实例化可以不使用 new 运算符。
- 结构可以声明构造函数，但它们必须带参数。
- 一个结构不能从另一个结构或类继承。所有结构都直接继承自 System.ValueType，而 System.ValueType 继承自 System.Object。
- 结构可以实现接口。
- 在结构中初始化实例字段是错误的。

说明： 由于结构的副本由编译器自动创建和销毁，因此不需要使用默认构造函数和析构函数。实际上，编译器通过为所有字段赋予默认值来实现默认构造函数。

C#中，使用 struct 关键字来定义结构，语法格式如下：

```
【结构修饰符】 struct 【结构名】
{
}
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetRAreaByStruct。

（2）在默认窗体 Form1 中添加 3 个 TextBox 控件，分别用来输入矩形的长、宽和显示矩形的面积；添加一个 Button 控件，用来根据用户输入的长和宽计算矩形的面积。

（3）程序的主要代码如下：

```
public struct Rect                                    //定义一个矩形结构
{
    public double width;                              //矩形的宽
    public double height;                             //矩形的高
    /// <summary>
    ///构造函数，初始化矩形的宽和高
    /// </summary>
    /// <param name="x">矩形的宽</param>
    /// <param name="y">矩形的高</param>
    public Rect(double x, double y)
    {
        width = x;
        height = y;
    }
    /// <summary>
    ///计算矩形面积
    /// </summary>
    /// <returns>矩形面积</returns>
    public double Area()
    {
        return width * height;
    }
}
```

秘笈心法

心法领悟 112：结构和类的区别。

结构是值类型，它在栈上分配数据，并且结构的赋值将分配产生一个新的对象；而类是引用类型，它在堆上分配数据，对类的对象进行赋值时只是复制了引用，它们都指向同一个实际对象分配的内存。

实例 113　通过类继承计算梯形面积

光盘位置：光盘\MR\05\113　　初级　　趣味指数：★★★☆

实例说明

梯形是指一组对边平行，而另一组对边不平行的四边形，其中平行的两边分别叫做梯形的上底和下底。梯形的面积计算是，首先用上底加上下底，然后乘以梯形的高，再除以 2。本实例通过类的继承来实现梯形面积的计算，其中首先定义一个父类，用来存储梯形的上底、下底和高；然后再自定义一个子类，并且使这个子类继承自父类，这个子类中自定义一个计算梯形面积的方法；在代码中使用时，只需要创建子类的对象，并通过子类的对象来调用父类中的上底、下底和高属性，再分别给它们赋值，然后再调用子类中的自定义方法即可实现梯形面积的计算功能。实例运行效果如图 5.7 所示。

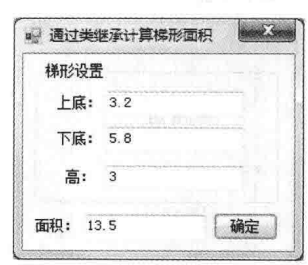


图 5.7　通过类继承计算梯形面积

关键技术

本实例实现时主要用到了面向对象的继承性，下面对其进行详细讲解。

继承是面向对象编程最重要的特性之一，任何类都可以从另外一个类继承，这就是说，这个类拥有它继承的类的所有成员。在面向对象编程中，被继承的类称为父类或基类。C#中提供了类的继承机制，但只支持单继承，而不支持多重继承，即在C#中一次只允许继承一个类，不能同时继承多个类。

利用类的继承机制，程序开发人员可以在已有类的基础上构造新类，这一性质使得类支持分类的概念。例如，用户可以通过增加、修改或替换类中的方法对这个类进行扩充，以适应不同的应用要求。在日常生活中很多东西都很有条理，那是因为它们有着很好的层次分类，如果不用层次分类，则需要对每个对象都定义其所有的性质。使用继承后，每个对象就可以只定义自己的特殊性质，每一层的对象只需定义本身的性质，其他性质可以从上一层继承下来。

继承一个类时，类成员的可访问性是一个重要的问题。子类（派生类）不能访问基类的私有成员，但是可以访问其公共成员，这就是说，只要使用 public 声明类成员，就可以让一个类成员被基类和子类（派生类）同时访问，同时也可以被外部的代码访问。

为了解决基类成员访问问题，C#还提供了另外一种可访问性，即 protected，只有子类（派生类）才能访问 protected 成员，基类和外部代码都不能访问 protected 成员。

除了可以定义成员的保护级别外，还可以为成员定义其继承行为。基类的成员可以是虚拟的，成员可以由继承它的类重写。子类（派生类）可以提供成员的其他执行代码，这种执行代码不会删除原来的代码，仍可以在类中访问原来的代码，但外部代码不能访问它们。如果没有提供其他执行方式，外部代码就直接访问基类中成员的执行代码。

另外，基类还可以定义为抽象类。抽象类不能直接实例化，要使用抽象类就必须继承这个类，然后再实例化。

C#中实现继承的语法格式如下：

```
class DerivedClass: BaseClass { }
```

说明：继承类时，必须在子类和基类之间使用冒号（:）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetTAreaByClass。

（2）在默认窗体 Form1 中添加 4 个 TextBox 控件，分别用来输入梯形的上底、下底、高和显示梯形的面积；添加一个 Button 控件，用来根据用户输入的上底、下底和高计算梯形的面积。

（3）程序的主要代码如下：

```
class Trapezia                                  //自定义类
{
    private double sd = 0;                      //定义 int 型变量，作为梯形的上底
    private double xd = 0;                      //定义 int 型变量，作为梯形的下底
    private double height = 0;                  //定义 int 型变量，作为梯形的高
    /// <summary>
    ///上底
    /// </summary>
    public double SD
    {
        get
        {
            return sd;
        }
        set
        {
            sd = value;
        }
    }
    /// <summary>
    ///下底
    /// </summary>
```

```
    public double XD
    {
        get
        {
            return xd;
        }
        set
        {
            xd = value;
        }
    }
    /// <summary>
    ///高
    /// </summary>
    public double Height
    {
        get
        {
            return height;
        }
        set
        {
            height = value;
        }
    }
}
class TrapeziaArea : Trapezia                    //自定义类，该类继承自 Trapezia
{
    /// <summary>
    ///求梯形的面积
    /// </summary>
    /// <returns>梯形的面积</returns>
    public double Area()
    {
        return (SD + XD) * Height / 2;
    }
}
```

秘笈心法

心法领悟 113：使用类继承提高程序的开发效率。

类可以从另外一个类继承，也就是说，这个类拥有它继承的类的所有成员（除构造方法外）。利用类的继承机制，程序开发人员可以在已有类的基础上构造新类，从而更快地提高程序的开发效率。

实例 114　封装类实现一个简单的计算器

光盘位置：光盘\MR\05\114　　初级　　趣味指数：★★★★

实例说明

编写计算器是编程学习人员开始学习编程时最常见的一个实例，很多高校在测试学生编程能力时，通常都使用编写计算器来考查学生。本实例将使用面向对象编程思想中的封装性来编写一个简单的计算器，实例运行效果如图 5.8 所示。

图 5.8　封装类实现一个简单的计算器

关键技术

本实例实现时主要用到了面向对象的封装性，下面对其进行详细讲解。

面向对象编程中，大多数都是以类作为数据封装的基本单位。类将数据和操作数据的方法结合成一个单位。设计类时，不希望直接存取类中的数据，而是希望通过方法来存取数据，这样就可以达到封装数据的目的，方

便以后的维护升级，也可以在操作数据时多一层判断。

此外，封装还可以解决数据存取的权限问题，可以使用封装将数据隐藏起来，形成一个封闭的空间，然后设置哪些数据只能在这个空间中使用，哪些数据可以在空间外部使用。如果一个类中包含敏感数据，有些人可以访问，有些人不能访问，如果不对这些数据的访问加以限制，后果将会非常严重。所以在编写程序时，要对类的成员使用不同的访问修饰符，从而定义它们的访问级别。

C#中可以使用类来达到数据封装的效果，这样就可以使数据与方法封装成单一元素，以便于通过方法存取数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CalcByClass。

（2）在默认窗体 Form1 中添加一个 TextBox 控件，用来显示输入的数和计算结果；添加 16 个 Button 控件，分别用来表示相应的数字和对输入的数进行加、减、乘、除运算。

（3）程序的主要代码如下：

```
public class CCount
{
    public int Sum(int a, int b, string type)                    //定义一个方法，用来计算两个数的和、差、积、商
    {
        switch (type)                                            //判断运算符类型
        {
            case "+":
                return a + b; break;                             //计算两个数的和
            case "-":
                return a - b; break;                             //计算两个数的差
            case "*":
                return a * b; break;                             //计算两个数的积
            case "/":
                return a / b; break;                             //计算两个数的商
            default:
                return 0; break;
        }
    }
}
```

秘笈心法

心法领悟 114：封装性的使用范围。

封装性是面向对象编程中最基本的一个特性，在 C#中使用封装性时，主要是针对接口和类来说的。对于一些程序中通用的属性、方法等，通常都封装到接口或者类中，从而提高代码的重用率。

实例 115　使用面向对象思想查找字符串中的所有数字

光盘位置：光盘\MR\05\115　　中级　　趣味指数：★★★★☆

实例说明

查找字符串中的所有数字时，首先需要将所有数字存储到一个字符串数组中，然后循环遍历要在其中查找数字的字符串，如果与定义的字符串数组中的某一项相匹配，则记录该项，循环执行该操作，最后得到的结果就是字符串中的所有数字。实例运行效果如图 5.9 所示。

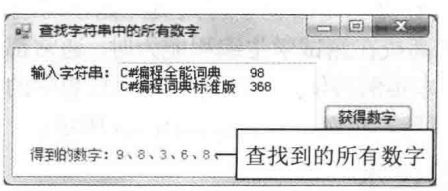


图 5.9　使用面向对象思想查找字符串中的所有数字

关键技术

本实例实现时主要用到了面向对象编程中的自定义方法，下面对其进行详细讲解。

自定义方法主要用来完成类或对象的行为，在面向对象编程语言中，类或对象是通过方法来与外界交互的。

所以，方法是类与外界交互的基本方式。

C#中声明方法的格式如下：

```
【修饰符】 【返回类型】 【方法名】(形参列表)
{
    【方法体】
}
```

说明：这里只对自定义方法进行了简单说明，其详细讲解请参见实例 110 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetNumInString。

（2）在默认窗体 Form1 中添加一个 TextBox 控件，并将其 Multiline 属性设置为 true，用来输入字符串；添加一个 Button 控件，用来从输入的字符串中查找数字；添加一个 Label 控件，用来显示查找到的所有数字。

（3）程序的主要代码如下：

```
public bool getNumeric(string str)                    //查找数字方法
{
    bool b = false;
    //将所有数字存储到一个字符串数组中
    string[] ArrayInt = new string[] { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" };
    foreach (string n in ArrayInt)                     //判断字符是否包含数组中指定的数字
    {
        if (n == str)                                  //如果找到了数字
        {
            b = true;
            break;
        }
    }
    return b;
}
```

秘笈心法

心法领悟 115：什么是面向对象编程技术？

面向对象编程技术也叫 OOP 技术，它是计算机软件开发中的一种新方法、新思想。在过去的面向过程编程中，经常将所有的代码都包含在几个模块中，这样，当做某些修改时，常常是牵一发而动全身，不容易开发和维护。使用面向对象技术开发软件时，也可以将软件分成几个模块，每个模块都有特定的功能，但是，模块之间是相互独立，同时又是相互联系的，模块的代码可以重用，这样大大增加了代码的使用率，有利于软件的开发和维护。

5.3　面向对象技术高级应用

实例 116　自定义抽象类计算圆形的面积

光盘位置：光盘\MR\05\116　　中级　　趣味指数：★★★☆

实例说明

众所周知，在数学计算领域，圆面积的计算非常简单，只要知道了圆半径的值，将其进行求平方运算之后，再乘以 π，得到的结果就是圆面积。本实例将通过自定义的抽象类来计算圆形的面积，实例运行效果如图 5.10 所示。

图 5.10　自定义抽象类计算圆形的面积

说明：计算圆面积时需要用到π，π在C#中使用Math.PI常量字段来表示，它的值为3.14159265358979323846。

关键技术

本实例实现时主要用到了抽象类，下面对其进行详细讲解。

如果一个类不与具体的事物相联系，而只是表达一种抽象的概念，仅是作为其派生类的一个基类，这样的类就是抽象类。在抽象类中声明方法时，如果加上abstract关键字，则为抽象方法。

抽象类主要用来提供多个派生类可共享的基类的公共定义，它与非抽象类的主要区别如下：

- 抽象类不能直接实例化。
- 抽象类中可以包含抽象成员，但非抽象类中不可以。
- 抽象类不能被密封。

C#中声明抽象类时需要使用abstract关键字，语法格式如下：

```
访问修饰符 abstract class 类名:基类或接口
{
    //类成员
}
```

说明：声明抽象类时，除abstract关键字、class关键字和类名外，其他的都是可选项。

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为CalcRAreaByAbstractClass。

（2）在默认窗体Form1中添加两个TextBox控件，分别用来输入圆半径和显示圆面积；添加一个Button控件，用来根据输入的圆半径计算圆面积。

（3）程序的主要代码如下：

```
public abstract class Roll
{
    private int r = 0;
    /// <summary>
    ///圆半径
    /// </summary>
    public int R
    {
        get
        {
            return r;
        }
        set
        {
            r = value;
        }
    }
    /// <summary>
    ///抽象方法，用来计算圆面积
    /// </summary>
    public abstract double Area();
}
public class CalcArea : Roll                    //继承抽象类
{
    /// <summary>
    ///重写抽象类中计算圆面积的方法
    /// </summary>
    public override double Area()
    {
        return Math.PI * R * R;
    }
}
```

秘笈心法

心法领悟 116：抽象类和接口的区别。

抽象类和接口的区别主要有以下几点：

- 它们的派生类只能继承一个基类，即只能直接继承一个抽象类，但可以继承任意多个接口。
- 抽象类中可以定义成员的实现，但接口中不可以。
- 抽象类中可以包含字段、构造函数、析构函数、静态成员或常量等，接口中不可以。
- 抽象类中的成员可以是私有的（只要它们不是抽象的）、受保护的、内部的或受保护的内部成员（受保护的内部成员只能在应用程序的代码或派生类中访问），但接口中的成员必须是公共的。

实例 117	利用接口实现选择不同的语言	中级
	光盘位置：光盘\MR\05\117	趣味指数：★★★★

实例说明

当人们面前站着两个人，分别是中国人和美国人时，如果想和这两个人交流，那么要对中国人说汉语，对美国人说英语。因此，在程序中可以建立一个接口，该接口定义一个方法用于对话，而对话这个方法是在类中实现的。分别创建一个中国人的类和一个美国人的类，这两个类都继承自接口，在中国人的类中说汉语，在美国人的类中说英语，当和不同国家的人交流时，实例化接口，并调用相应类中的方法即可。实例运行效果如图 5.11 和图 5.12 所示。

图 5.11　选择“中国人”对话方式

图 5.12　选择“美国人”对话方式

关键技术

本实例实现时主要用到了接口，下面对其进行详细讲解。

接口是一种用来定义程序的协议，它描述可属于任何类或结构的一组相关行为，可以把它看成是实现一组类的模板。接口可由方法、属性、事件和索引器或这 4 种成员类型的任何组合构成，但不能包含字段。

类和结构可以像类继承基类一样从接口继承，但是可以继承多个接口。当类或结构继承接口时，它继承成员定义但不继承实现。若要实现接口成员，类或结构中的对应成员必须是公共的、非静态的，并且与接口成员具有相同的名称和签名。类或结构的属性和索引器可以为接口中定义的属性或索引器定义额外的访问器。例如，接口可以声明一个带有 get 访问器的属性，而实现该接口的类可以声明同时带有 get 和 set 访问器的同一属性。但是，如果属性或索引器使用显式实现，则访问器必须匹配。

另外，接口也可以继承其他接口，类可以通过其继承的基类或接口多次继承某个接口，在这种情况下，如果将该接口声明为新类的一部分，则类只能实现该接口一次。如果没有将继承的接口声明为新类的一部分，其实现将由声明它的基类提供。基类可以使用虚拟成员实现接口成员。在这种情况下，继承接口的类可通过重写虚拟成员来更改接口行为。

综上所述，接口具有以下特征：

- 接口类似于抽象基类，继承接口的任何非抽象类型都必须实现接口的所有成员。

- 不能直接实例化接口。
- 接口可以包含事件、索引器、方法和属性。
- 接口不包含方法的实现。
- 类和结构可从多个接口继承。
- 接口自身可从多个接口继承。

C#中使用 interface 关键字声明接口，语法格式如下：

```
修饰符 interface 接口名称 : 继承的接口列表
{
    接口内容;
}
```

说明：（1）声明接口时，通常以大写字母“I”开头。

（2）声明接口时，除 interface 关键字和接口名称外，其他的都是可选项。

（3）可以使用 new、public、protected、internal 和 private 等修饰符声明接口，但接口成员必须是公共的。

接口通过类继承来实现，一个类虽然只能继承一个基类，但可以继承任意多个接口。声明实现接口的类时，需要在基类列表中包含类所实现的接口的名称。

C#中实现继承的语法格式如下：

```
class DerivedClass: BaseClass { }
```

说明：继承接口时，必须在子类和接口之间用冒号（:）。另外，如果继承多个接口，那么在继承的每个接口之间用逗号（,）分隔。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AppInterface。

（2）在默认窗体 Form1 中添加一个 ComboBox 控件，用来选择对话方式；添加一个 TextBox 控件，并将其 Multiline 属性设置为 true，用来输入对话内容；添加一个 Button 控件，用来执行对话操作。

（3）程序的主要代码如下：

```
/// <summary>
///声明一个接口，用于定义 Speak 方法，而具体 Speak 方法功能的实现是在类中进行的
/// </summary>
interface ISelectLanguage
{
    void Speak(string str);
}
/// <summary>
///如果与中国人对话，则说汉语
/// </summary>
class C_SpeakChinese : ISelectLanguage
{
    public void Speak(string str)
    {
        MessageBox.Show("您对中国友人说：" + str, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
/// <summary>
///如果与美国人对话，则说英语
/// </summary>
class C_SpeakEnglish : ISelectLanguage
{
    public void Speak(string str)
    {
        MessageBox.Show("您对美国友人说：" +str, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 117：有效使用接口进行组件编程。

接口使得服务的协议与实现相分离，它是组件编程的基础，在组件编程中，接口是组件向外公布其功能的唯一方法。

实例 118　使用密封类密封用户信息

光盘位置：光盘\MR\05\118　　初级　　趣味指数：★★★★

实例说明

密封类的突出特点是不能被继承，通过密封类封装用户信息可以增加用户信息的安全性。本实例将使用密封类密封用户的登录用户名和密码，以保证其安全性。实例运行效果如图 5.13 所示。

图 5.13　使用密封类密封用户信息

关键技术

本实例实现时主要用到了密封类，下面对其进行详细讲解。

密封类可以用来限制扩展性，如果密封了某个类，则其他类不能从该类继承；如果密封了某个成员，则派生类不能重写该成员的实现。默认情况下，不应该密封类型和成员。密封可以防止对库的类型和成员进行自定义，但也会影响某些开发人员对可用性的认识。

C#中使用密封类时，如果类满足如下条件，则应将其密封。

- 类是静态类。
- 类包含带有安全敏感信息的继承的受保护成员。
- 类继承多个虚成员，并且密封每个成员的开发和测试开销明显大于密封整个类。
- 类是一个要求使用反射进行快速搜索的属性，密封属性可提高反射在检索属性时的性能。

在 C#中声明密封类时需要使用 sealed 关键字，具体语法格式如下：

```
访问修饰符 sealed class 类名:基类或接口
{
    //类成员
}
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SealedUserInfo。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来输入用户名和密码；添加两个 Button 控件，分别用来执行用户登录和退出窗体功能。

（3）程序的主要代码如下：

```
/// <summary>
///通过 sealed 关键字声明密封类，防止类被继承，有效保护重要信息
/// </summary>
public sealed class myClass
{
    private string name = "";                    //string 类型变量，用来记录用户名
    private string pwd = "";                     //string 类型变量，用来记录密码
    /// <summary>
    ///用户名
    /// </summary>
    public string Name
    {
        get
```

```
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
    /// <summary>
    ///密码
    /// </summary>
    public string Pwd
    {
        get
        {
            return pwd;
        }
        set
        {
            pwd = value;
        }
    }
}
```

秘笈心法

心法领悟 118：使用密封类时的注意事项。

使用密封类时需要注意以下 3 点：

- 密封类不能作为基类被继承，但它可以继承别的类或接口。
- 在密封类中不能声明受保护成员或虚成员，因为受保护成员只能从派生类进行访问，而虚成员只能在派生类中重写。
- 由于密封类的不可继承性，因此密封类不能声明为抽象的。

实例 119　通过重写虚方法实现加法运算

初级

光盘位置：光盘\MR\05\119

趣味指数：★★★★

实例说明

通过 virtual 关键字修饰的方法被称为虚方法，虚方法可以被其子类重写。例如，程序中经常用到的 ToString 方法就是一个虚方法，它可以在其子类中进行重写，实现输出自定义格式的字符串。本实例将通过重写虚方法实现两个数相加，具体实现时，首先在 Operation 类中定义一个虚方法 operation，用于实现两个数相乘；然后创建一个子类 Addition，继承自 Operation 类，在子类中重写虚方法 operation，实现两个数相加。实例运行效果如图 5.14 所示。

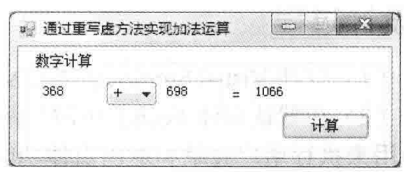


图 5.14　通过重写虚方法实现加法运算

关键技术

本实例实现时主要用到了虚方法和重写方法，下面对本实例用到的关键技术进行详细讲解。

（1）虚方法

虚方法就是允许被其子类重新定义的方法，在声明时，需要使用 virtual 修饰符。

注意：① virtual 修饰符不能与 static、abstract 或者 override 修饰符同时使用。

② 由于虚方法不能是私有的，所以 virtual 修饰符不能与 private 修饰符同时使用。

例如，使用 virtual 修饰符声明一个虚方法，用来计算两个数的和，代码如下：

```
public virtual int Add(int x, int y)                    //定义一个虚方法
{
    return x + y;                                       //返回两个数的和
}
```

（2）重写方法

如果一个方法声明中含有 override 修饰符，则称该方法为重写方法，它主要用来使用相同的签名重写继承的虚方法。虚方法主要用来引入新方法，而重写方法则使从基类继承而来的虚方法专用化（提供虚方法的具体实现）。

例如，使用 override 修饰符重写上面声明的虚方法，代码如下：

```
public override int Add(int x, int y)                   //重写虚方法
{
    return x + y;                                       //返回两个数的和
}
```

注意：override 修饰符不能与 new、static 或者 virtual 修饰符同时使用。另外，重写方法只能用于重写基类中的虚方法，不能用来单独声明方法。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddByVirtualMethod。

（2）在默认窗体 Form1 中添加 3 个 TextBox 控件，分别用来输入要计算的两个数和显示计算结果；添加一个 ComboBox 控件，用来选择计算方式；添加一个 Button 控件，用来按指定方式对输入的数进行计算。

（3）程序的主要代码如下：

```
/// <summary>
///建立的虚方法，可以在子类中被重写
/// </summary>
class Operation
{
    public virtual double operation(int x,int y)        //定义虚方法
    {
        return x * y;                                   //计算两个数的乘积
    }
}
/// <summary>
///在子类中重写虚方法
/// </summary>
class Addition : Operation
{
    public override double operation(int x, int y)      //重写虚方法
    {
        return (x + y);                                 //计算两个数的和
    }
}
```

秘笈心法

心法领悟 119：重写方法与重载方法的区别。

重写方法与重载方法是不相同的，重写方法是指在派生类中重写基类的虚方法；而重载方法是指编写一个与已有方法同名，但参数列表不同的方法。

实例 120	通过类的多态性确定人类的说话行为	中级
	光盘位置：光盘\MR\05\120	趣味指数：★★★★☆

实例说明

本实例通过使用类的多态性来确定人类的说话行为。运行本实例，首先在文本框中输入人的姓名，然后单

击“确定”按钮，在 RichTextBox 控件中分别显示人的“说汉语”和“说英语”行为。实例运行效果如图 5.15 所示。

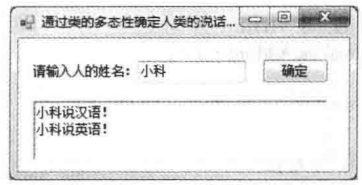


图 5.15　通过类的多态性确定人类的说话行为

关键技术

本实例实现时主要用到了面向对象的多态性，下面对其进行详细讲解。

多态性可以简单地概括为“一个接口，多种方法”，它是在程序运行的过程中才决定调用的方法，多态性是面向对象编程的核心概念。例如，日常生活中经常说的开电视、开电脑、开音箱等，这里的“开”其实就是多态。

多态使得子类（派生类）的实例可以直接赋予基类的对象（这里不需要进行强制类型转换），然后直接就可以通过这个对象调用子类（派生类）的方法。

说明：（1）多态性主要是用于实现接口重用，因为接口是程序开发中最耗费时间的资源，实质上设计一个接口要比设计一堆类要显得更有效率。

（2）多态性在 C#中主要通过虚方法和重写方法来体现，关于这两个方法的详细讲解请参见实例 119 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PeopleSpeakByMState。

（2）在默认窗体 Form1 中添加一个 TextBox 控件，用来输入人的姓名；添加一个 Button 控件，用来执行人的说话操作；添加一个 RichTextBox 控件，用来显示人的说话行为。

（3）程序的主要代码如下：

```
class People                                          //定义基类
{
    public virtual void Say(RichTextBox rtbox, string name)   //定义一个虚方法，用来表示人的说话行为
    {
        rtbox.Text += name;                           //输出人的名字
    }
}
class Chinese : People                                //定义派生类，继承于 People 类
{
    public override void Say(RichTextBox rtbox, string name)  //重写基类中的虚方法
    {
        base.Say(rtbox, name + "说汉语！ \n");
    }
}
class American : People                               //定义派生类，继承于 People 类
{
    public override void Say(RichTextBox rtbox, string name)  //重写基类中的虚方法
    {
        base.Say(rtbox, name + "说英语！ ");
    }
}
private void button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "")                          //判断是否输入了姓名
    {
        Console.WriteLine("请输入姓名：");
        return;
    }
    richTextBox1.Clear();                             //清空文本框中的内容
    string strName = textBox1.Text;                   //记录用户输入的名字
    People[] people = new People[2];                  //声明 People 类型数组
    people[0] = new Chinese();                        //使用第一个派生类的对象初始化数组的第一个元素
    people[1] = new American();                       //使用第二个派生类的对象初始化数组的第二个元素
    for (int i = 0; i < people.Length; i++)           //遍历赋值后的数组
```

```
    {
        people[i].Say(richTextBox1,strName);                    //根据数组元素调用相应派生类中的重写方法
    }
}
```

秘笈心法

心法领悟 120：如何使用虚方法实现多态性？

首先在基类中定义虚方法，每一个子类都重写基类的虚方法，采用基类的对象引用子类实例的方法创建对象，这样会产生很多基类的对象，使用每一个基类的对象调用虚方法时会调用不同子类重写基类的方法，这样就实现了多态性。

5.4　迭代器和分部类的使用

实例 121	使用迭代器显示公交车站点 光盘位置：光盘\MR\05\121	初级 趣味指数：★★★★★

实例说明

在人们去某地时，有时只知道地方的大概位置，并不知道具体怎么走，这时就要通过查找公交车的站点来明确坐几路公交车。本实例将使用 C#中的迭代器依次显示公交车的所有站点，以方便人们的出行。实例运行效果如图 5.16 所示。

图 5.16　使用迭代器显示公交车站点

关键技术

本实例实现时主要用到了迭代器，下面对其进行详细讲解。

迭代器是.NET 4.5 中的一个新特性，它是可以返回相同类型的值的有序序列的一段代码。迭代器可用作方法、运算符或 get 访问器的代码体。在迭代器的实现代码中，通常使用 yield return 语句依次返回每个元素，使用 yield break 语句终止迭代。

注意：迭代器的返回类型必须为 IEnumerable 或 IEnumerator 中的任意一种。

例如，通过对 IEnumerator 接口实现 GetEnumerator 方法创建迭代器的实现代码如下：

```
string[] MyFamily ={ "父亲","母亲","弟弟","妹妹"};              //创建一个 string 类型的数组用于存储家庭成员
public System.Collections.IEnumerator GetEnumerator()
{
    for (int i = 0; i < MyFamily.Length; i++)                   //使用 for 语句循环数组
    {
        yield return MyFamily[i];                               //使用 yield return 语句依次返回每个元素
    }
}
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShowBusStation。

（2）在默认窗体 Form1 中添加一个 ListView 控件，用来显示公交车站点列表；添加两个 Button 控件，分别用来获取公交车站点和关闭当前窗体。

（3）程序的主要代码如下：

```
public static IList<object> items = new List<object>();         //定义一个泛型对象，用于存储对象
/// <summary>
///通过迭代器获取泛型中的所有对象值
/// </summary>
```

```
/// <param Node="n">泛型对象</param>
/// <returns>IEnumerable<object></returns>
public static IEnumerable<object> GetValues()
{
    if (items != null)                                    //如果泛型不为空
    {
        foreach (object i in items)                       //遍历泛型中的对象
            yield return i;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    foreach (object i in GetValues())                     //遍历泛型集合
        listView1.Items.Add(i.ToString());                //在列表视图中显示公交车站点
}
```

秘笈心法

心法领悟 121：使用 ILDASM.exe 工具查看中间语言代码。

如果了解编译器在后台执行了什么操作，可以使用 Visual Studio 2012 开发环境自带的 ILDASM.exe 工具来查看为迭代器方法生成的中间语言（IL）代码。

实例 122　使用迭代器实现倒序遍历

光盘位置：光盘\MR\05\122　　初级　　趣味指数：★★★★

实例说明

开发一些游戏程序时，经常会用到对字符串或数字的倒序遍历功能，当然，该功能可以使用常用的排序算法进行实现，但是对于使用 C# 4.5 进行开发的程序员来说，迭代器显然是一种更加简单、高效的实现倒序遍历功能的方法。本实例将使用迭代器实现字符串或数字的倒序遍历功能，实例运行效果如图 5.17 所示。

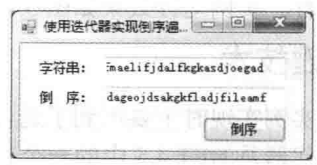


图 5.17　使用迭代器实现倒序遍历

关键技术

本实例主要使用 for 语句和迭代器的 IEnumerable 接口，以及 yield return 语句对指定的字符串进行倒序遍历。在迭代器中遍历字符串中的每个元素时，为了便于获取元素的类型，将迭代器的返回值设置为 object 对象。

说明：关于迭代器的详细讲解请参见实例 121 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReverseOrder。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来输入字符串和显示倒序排序之后的字符串；添加一个 Button 控件，用来对输入的字符串进行倒序排序。

（3）程序的主要代码如下：

```
/// <summary>
///通过迭代器实现字符串的倒序
/// </summary>
/// <param string="n">进行倒序的字符串</param>
/// <returns>以对象的方式倒序返回单个字符</returns>
public static IEnumerable<object> Transpose(string n)
{
    if (n.Length >0)                                      //如果泛型不为空
    {
        for (int i = n.Length-1; i >= 0; i--)             //从末尾开始遍历字符串
```

```
            yield return (object)n[i];                    //返回数据集合
        }
    }
    /// <summary>
    ///获取倒序后的字符串
    /// </summary>
    /// <param string="Str">进行倒序的字符串</param>
    /// <returns>返回倒序后的字符串</returns>
    public string GetValue(string Str)
    {
        if (Str.Length == 0)                              //判断字符串长度是否为 0
            return "";                                    //返回空
        string Tem_Str = "";                              //记录倒序之后的字符串
        foreach (object i in Transpose(Str))              //遍历迭代器
            Tem_Str += i.ToString();                      //获取迭代器中的每个字符
        return Tem_Str;                                   //返回倒序之后的字符串
    }
```

秘笈心法

心法领悟 122：实用的 IEnumerable 接口。

IEnumerable 接口公开了枚举数，该枚举数支持在非泛型集合上进行简单迭代。

实例 123　使用迭代器实现文字的动态效果

光盘位置：光盘\MR\05\123　　中级　　趣味指数：★★★★★

实例说明

开发项目时，为了使界面具有动态效果，可以在界面中实现一些特殊文字的动态效果。本实例主要使用迭代器遍历文本字符串中的每一个文字，然后使用 GDI+技术在窗体上以不同的字体样式依次绘制每一个文字，以便实现文字的动态效果。实例运行效果如图 5.18 所示。

图 5.18　使用迭代器实现文字的动态效果

关键技术

本实例实现时主要用到了迭代器以及 GDI+绘图技术，由于迭代器在实例 121 中已经做过详细讲解，所以这里主要讲解 GDI+绘图技术。

说明：关于迭代器的详细讲解请参见实例 121 中的关键技术。

GDI+是GDI（即Windows早期版本中附带的Graphics Device Interface）的后继者，它是一种构成Windows XP操作系统的子系统的应用程序编程接口（API）。GDI+主要用于在窗体上绘制各种图形图像。另外，它还可以在窗体程序中产生很多自定义的图形，便于开发人员展示各种图形化的数据。

GDI+的核心是Graphics类，该类表示GDI+绘图表面，它提供将对象绘制到显示设备的方法。Graphics类封装了绘制直线、曲线、图形、图像和文本的方法，它是进行一切GDI+操作的基础类。本实例使用Graphics类的DrawString方法来绘制动态的文字，该方法常用的语法格式如下：

```
public void DrawString(string s,Font font,Brush brush,PointF point)
```

参数说明

❶ s：要绘制的字符串。

❷ font：定义字符串的文本格式。

❸ brush：确定所绘制文本的颜色和纹理。

❹ point：指定所绘制文本的左上角。

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为DynamicLetter。

（2）在默认窗体Form1中添加一个Panel控件，用来显示动态文字。

（3）程序的主要代码如下：

```
/// <summary>
///通过迭代器实现字符串的遍历
/// </summary>
/// <param string="n">文字字符串</param>
/// <returns>返回单个文字</returns>
public static IEnumerable<object> Transpose(string n)
{
    if (n.Length > 0)                                          //如果泛型不为空
    {
        foreach (object i in n)                                //对字符串进行遍历
            yield return i;
    }
}
/// <summary>
///绘制动态文字
/// </summary>
/// <param string="C_Str">绘制的文字字符串</param>
public void DynamicText(Object C_Str)
{
    float tem_left = 0;                                        //获取当前文字的左端位置
    float tem_top = 0;                                         //获取当前文字的顶端位置
    float tem_w = 0;                                           //获取文字的宽度
    float tem_h = 0;                                           //获取文字的高度
    float tem_place = Str_Width;                               //获取起始文字的位置
    Font Tem_Font = new Font("黑体", FSize[0], FontStyle.Bold);//定义字体样式
    int p = 0;                                                 //记录字符串中文字的索引号
    int Str_Index = 0;
    try
    {
        foreach (object s in Transpose(C_Str.ToString()))      //遍历字符串
        {
            for (int i = 1; i < 5; i++)
            {
                if (i >= 3)
                    p = Convert.ToInt16(Math.Floor(i / 2F));
                else
                    p = i;
                ProtractText(C_Str.ToString(), Str_Index);
                Tem_Font = new Font("黑体", FSize[p], FontStyle.Bold);//定义字体样式
                SizeF TitSize = g.MeasureString(s.ToString(), Str_Font);//将绘制的单个文字进行格式化
                tem_w = TitSize.Width;                         //获取文字串的宽度
                tem_h = TitSize.Height;                        //获取文字串的高度
```

```
                tem_left = tem_place - (tem_w - Str_Odd_Width) / 2F;          //获取文字改变大小后的左端位置
                tem_top = Str_Height - (Str_Height - tem_h) / 2F;             //获取文字改变大小后的顶端位置
                ProtractOddText(s.ToString(), Tem_Font, tem_left, tem_top);   //绘制单个文字
                Thread.Sleep(200);                                            //等待 0.2 秒
                g.FillRectangle(new SolidBrush(Panel_C), 0, 0, Panel_W, Panel_H);  //清空绘制的文字
            }
            tem_place += Str_Odd_Width + Str_block;                           //计算下一个文字的左端位置
            Str_Index += 1;                                                   //将索引号定位到下一个文字
        }
        ProtractText(C_Str.ToString(), -1);                                   //恢复文字的原始绘制样式
        //创建 ParameterizedThreadStart 委托线程
        th = new Thread(new ParameterizedThreadStart(DynamicText));
        th.Start(C_Str);                                                      //传递一个字符串的参数
    }
    catch                                                                     //在关闭窗体时关闭线程
    {
        th.Abort();                                                           //关闭线程
    }
}
```

秘笈心法

心法领悟 123：有效避免程序“假死”现象。

运行 C#程序时，如果一个任务执行时间过长，会导致程序主窗体处于“假死”状态。为了避免这种情况发生，可以使用 Thread 类来创建多线程，即每一个线程完成一个功能，这样就可以有效地避免程序出现“假死”现象。

实例 124　使用分部类实现多种计算方法

光盘位置：光盘\MR\05\124　　中级　　趣味指数：★★★★☆

实例说明

在开发一些大型项目或者特殊部署时，可能需要把一个类、结构或接口放在几个文件中来分别进行处理。等到编译时，再自动把它们整合起来，这时就用到了分部类。本实例将使用分部类制作一个计算器，其中主要是用分部类来分别记录计算器的计算方法，如将实现加、减、乘和除的方法放在一个分部类中，而将实现正负、开方、百分比和倒数的方法放在另一个分部类中。实例运行效果如图 5.19 所示。

图 5.19　使用分部类实现多种计算方法

关键技术

本实例实现时主要用到了分部类，下面对其进行详细讲解。

分部类是 C# 4.5 中的一个新特性，它的出现使得程序的结构更加合理，代码组织更加紧密。开发人员可以将类、结构或接口的定义拆分到两个或多个源文件中，每个源文件包含类定义的一部分，编译应用程序时，Visual Studio 2012 会把所有部分组合起来，这样的类被称为分部类。分部类主要应用在以下两个方面：

- 当项目比较庞大时，使用分部类可以拆分一个类至几个文件中，这样可以使不同的开发人员同时进行工作，提高了工作效率。
- 使用自动生成的文件源时，无须重新创建源文件即可将代码添加到类中。Visual Studio 2012 在创建 Windows 窗体和 Web 服务包装代码等时都使用此方法。

分部类相当于将一个会计部门（类、结构或接口）分成两个部门，这两个部门可以单独对公司各部门的账目进行审核，而在繁忙时期，两个部门也可以相互调动人员（这里的人员相当于类中的方法、属性、变量等），

或是合成一个整体进行工作。

定义分部类时需要使用 partial 关键字，分部类的每个部分都必须包含一个 partial 关键字，并且其声明必须与其他部分位于同一命名空间。使用分部类时，要成为同一类型的各个部分的所有分部类型定义都必须在同一程序集和同一模块（.exe 或.dll 文件）中进行定义，分部类定义不能跨越多个模块。

📖 **说明**：使用分部类时，各个部分必须具有相同的可访问性，如 public、private 等。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BranchMeans。

（2）在默认窗体 Form1 中添加一个 TextBox 控件，用来显示输入的数和计算结果；添加 23 个 PictureBox 控件，分别用来表示相应的数字和对输入的数进行加、减、乘、除、求倒数、开方、相反数和百分比运算。

（3）程序的主要代码如下：

```
public partial class Account
{
    public double Addition(double Former, double After)                    //执行两个数的加法运算
    {
        return Former + After;                                             //返回相加后的结果
    }
    public double Subtration(double Former, double After)                  //执行两个数的减法运算
    {
        return Former - After;                                             //返回相减后的结果
    }
    public double Multiplication(double Former, double After)              //执行两个数的乘法运算
    {
        return Former * After;                                             //返回相乘后的结果
    }
    public double Division(double Former, double After)                    //执行两个数的除法运算
    {
        if (After == 0)                                                    //判断被除数是否为 0
        {
            MessageBox.Show("被除数不能为 0。");
            return 0;
        }
        return Former / After;                                             //返回相除后的结果
    }
}
public partial class Account
{
    public double Molecule(double num)                                     //计算一个数的倒数
    {
        return 1 / num;                                                    //返回倒数值
    }
    public double Evolution(double num)                                    //计算一个数的开方
    {
        return Math.Sqrt(num);                                             //返回开方后的值
    }
    public double Opposition(double num)                                   //求一个数的相反数
    {
        return -num;                                                       //返回相反数
    }
    public double Par(double num)                                          //计算一个数的百分比
    {
        return num / 100 * num;                                            //返回百分比
    }
}
```

秘笈心法

心法领悟 124：合理使用分部类分割类。

C#编码规范中规定，一个类中的代码最好不要超过 500 行，但是，如果实际开发中，确实需要在一个类中

包含超过 500 行以上的代码时，该怎么办呢？这时可以使用分部类将该类分成几部分，其中每个部分包含一部分实现代码。

实例 125　使用分部类记录学生信息

光盘位置：光盘\MR\05\125　　初级　　趣味指数：★★★★

实例说明

在开发项目时，不可能每开发一个项目，都一行代码一行代码地进行编辑，有时可以使用其他项目中的类、方法和结构。在定义结构时，可以用分部类将公共部分和非公共部分分开定义，以便于在开发其他项目时，与其他结构相组合。下面使用分部类结构记录学生的信息，并进行显示。实例运行效果如图 5.20 所示。

图 5.20　使用分部类记录学生信息

关键技术

本实例主要使用分部类来分别记录学生的相关信息，然后在调用时，通过创建的分部类的对象分别为学生的相关属性赋值，并将赋予的值显示在对应的文本框中。

说明：关于分部类的详细讲解请参见实例 124 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StoreStuInfo。

（2）在默认窗体 Form1 中添加 8 个 TextBox 控件，分别用来显示学生的编号、姓名、年龄、性别、出生年月、年级、班级和班主任；添加一个 Button 控件，用来获取分部类中存储的学生信息，并显示在 TextBox 控件中。

（3）程序的主要代码如下：

```
partial class CommInfo
{
    /// <summary>
    ///编号
    /// </summary>
    public object ID;
    /// <summary>
    ///姓名
    /// </summary>
    public object Name;
    /// <summary>
    ///性别
    /// </summary>
    object sex;
    public object Sex
    {
        get
        {
            if ((bool)sex == true)
                sex = "男";
            else
                sex = "女";
            return sex;
        }
        set { sex = value; }
    }
    /// <summary>
    ///年龄
```

```
    /// </summary>
    public object Age;
    /// <summary>
    ///出生年月
    /// </summary>
    public object Birthday;
}
partial class CommInfo
{
    /// <summary>
    ///年级
    /// </summary>
    public object Grade;
    /// <summary>
    ///班级
    /// </summary>
    public object Class;
    /// <summary>
    ///班主任
    /// </summary>
    public object Director;
}
```

秘笈心法

心法领悟 125：使用“///”标记给代码段添加说明。

在 Visual Studio 2012 开发环境中，可以使用“///”标记为代码段添加说明，这样可以使团队中的开发人员更加清楚地了解代码段的作用。

5.5 泛型的使用

实例 126　使用泛型存储不同类型的数据列表　　初级

光盘位置：光盘\MR\05\126　　趣味指数：★★★★

实例说明

泛型编程是一种编程方式，它利用“参数化类型”将类型抽象化，从而实现更为灵活的复用。本实例将使用泛型存储不同类型的数据，在实现时，首先定义一个泛型类，并在泛型类中定义多个泛型变量；然后使用这些变量记录不同类型的数据，这样就可以重复利用泛型变量来存储不同类型的数据。实例运行效果如图 5.21 所示。

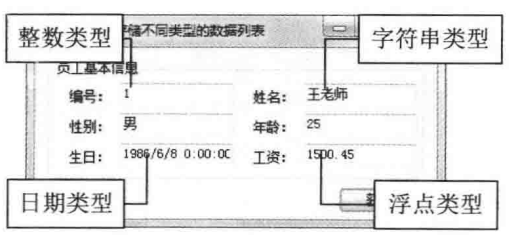


图 5.21　使用泛型存储不同类型的数据列表

关键技术

本实例实现时主要用到了泛型类，下面对其进行详细讲解。

泛型是用于处理算法、数据结构的一种编程方法。泛型的目标是采用广泛适用和可交互性的形式来表示算

法和数据结构，以使它们能够直接用于软件构造。泛型类、结构、接口、委托和方法可以根据它们存储和操作的数据类型来进行参数化。泛型能在编译时提供强大的类型检查，减少数据类型之间的显示转换、装箱操作和运行时的类型检查等。泛型类和泛型方法同时具备可重用性、类型安全和效率高等特性，这是非泛型类和非泛型方法无法具备的。

技巧：泛型通常用在集合和在集合上运行的方法中。

声明泛型类的语法与声明普通类是类似的，语法格式如下：

```
【类修饰符】 class 【类名】 <类型参数 T>
{
【类体】
}
```

从上面的语法中可以看出，泛型类的声明相比普通类多了一个类型参数 T，类型参数 T 可以看作是一个占位符，它不是一种类型，它仅代表了某种可能的类型。在定义泛型类时，T 出现的位置可以在使用时用任何类型来代替。类型参数 T 的命名规则如下：

- 使用描述性名称命名泛型类型参数，除非单个字母名称完全可以让人了解它表示的含义，而描述性名称不会有更多的意义。
- 将 T 作为描述性类型参数名的前缀。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExtensiveList。

（2）在默认窗体 Form1 中添加 6 个 TextBox 控件，分别用来显示员工的编号、姓名、年龄、性别、生日和工资；添加一个 Button 控件，用来通过泛型获取员工信息并显示。

（3）程序的主要代码如下：

```
class Types<T>
{
    public T Num;                                   //声明编号字段
    public T Name;                                  //声明姓名字段
    public T Sex;                                   //声明性别字段
    public T Age;                                   //声明年龄字段
    public T Birthday;                              //声明生日字段
    public T Salary;                                //声明工资字段
}
private void button1_Click(object sender, EventArgs e)
{
    Types<object> Exte = new Types<object>();       //创建泛型对象
    //为泛型类中声明的字段进行赋值，存储不同类型的值
    Exte.Num = 1;
    Exte.Name = "王老师";
    Exte.Sex = "男";
    Exte.Age = 25;
    Exte.Birthday = Convert.ToDateTime("1986-06-08");
    Exte.Salary = 1500.45F;
    //将泛型类中各字段的值显示在文本框中
    textBox1.Text = Exte.Num.ToString();
    textBox2.Text = Exte.Name.ToString();
    textBox3.Text = Exte.Sex.ToString();
    textBox4.Text = Exte.Age.ToString();
    textBox5.Text = Exte.Birthday.ToString();
    textBox6.Text = Exte.Salary.ToString();
}
```

秘笈心法

心法领悟 126：.NET 类库中的泛型类。

.NET 类库中提供了开发人员常用的一些泛型类，主要有 Dictionary<TKey, TValue>、List<T>、Queue<T>、Stack<T>和 SortedList<TKey, TValue>等，它们都位于 System.Collections.Generic 命名空间中。

实例 127 通过泛型查找不同数组中的值

光盘位置：光盘\MR\05\127　　初级　　趣味指数：★★★☆

实例说明

实际应用中，查找或遍历数组中的值时，有时因为数组类型的不同，需要对不同的数组进行操作，那么，可以使用同一种方法对不同类型的数组进行操作吗？答案是肯定的，本实例将使用一个泛型方法对不同类型的数组进行操作。实例运行效果如图 5.22 所示，当用户单击图 5.22 所示窗体中的“字符串”“整数”和“布尔型”按钮时，将分别弹出如图 5.23～图 5.25 所示的对话框。

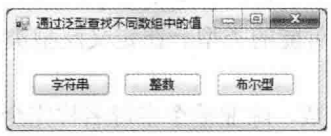


图 5.22　通过泛型查找不同数组中的值主窗体

图 5.23　在字符串中查找

图 5.24　在整数中查找

图 5.25　在布尔型数据中查找

关键技术

本实例实现时主要用到了泛型方法，下面对其进行详细讲解。

泛型方法是在声明中包括了类型参数 T 的方法。泛型方法可以在类、结构或接口声明中声明，这些类、结构或接口本身可以是泛型或非泛型。如果在泛型类型声明中声明泛型方法，则方法体可以同时引用该方法的类型参数 T 和包含该方法声明的类型参数 T。泛型方法的语法格式如下：

```
【修饰符】 【返回值类型】 【方法名】<类型参数 T>()
{
【方法体】
}
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ArrayInfo。

（2）在默认窗体 Form1 中添加 3 个 Button 控件，分别用来在字符串数组、整数数组和布尔型数组中查找指定值的索引。

（3）程序的主要代码如下：

```
public class Finder
{
    public static int Find<T>(T[] items, T item)          //定义一个泛型方法，用来查找指定值在数组中的索引
    {
        for (int i = 0; i < items.Length; i++)            //遍历泛型数组
        {
            if (items[i].Equals(item))                    //判断是否找到了指定值
            {
                return i;                                 //返回指定值在数组中的索引
            }
        }
        return -1;                                        //如果没有找到，返回-1
    }
}
```

秘笈心法

心法领悟 127：泛型集合的使用场合。

通常情况下，建议开发人员使用泛型集合，因为这样可以获得类型安全的直接优点，而不需要从基集合类型派生并实现类型特定的成员。此外，如果集合元素为值类型，泛型集合类型的性能通常优于对应的非泛型集合类型（并优于从非泛型基集合类型派生的类型），因为使用泛型时不必对元素进行装箱。

实例 128	通过继承泛型类实现输出学生信息	中级
	光盘位置：光盘\MR\05\128	趣味指数：★★★★

实例说明

在学生成绩管理系统中显示学生信息时，有时需要将学生信息分块进行存储，这样在使用时将会在不同情况下显示不同范围的学生信息。本实例通过继承泛型类来分块显示学生的个人信息以及成绩信息，实例运行效果如图 5.26 所示。

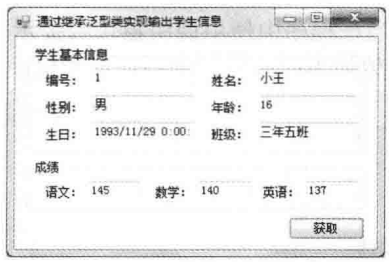


图 5.26　通过继承泛型类实现输出学生信息

关键技术

本实例实现时主要用到了泛型类的继承，下面对其进行详细讲解。

泛型类的继承与普通类的继承是类似的，只是在继承时多了一个类型参数 T，语法格式如下：

```
class DerivedClass <类型参数 T>: BaseClass <类型参数 T>
{
}
```

说明：关于类的继承的详细讲解请参见实例 113 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShowStuInfo。

（2）在默认窗体 Form1 中添加 9 个 TextBox 控件，分别用来显示学生的编号、姓名、性别、年龄、生日、班级、语文成绩、数学成绩和英语成绩；添加一个 Button 控件，用来通过访问继承的泛型派生类获取学生信息并显示。

（3）程序的主要代码如下：

```
class BStuInfo<T>
{
    public T ID;                          //声明学生编号字段
    public T Name;                        //声明姓名字段
    public T Sex;                         //声明性别字段
    public T Age;                         //声明年龄字段
    public T Birthday;                    //声明生日字段
    public T Grade;                       //声明班级字段
}
class HStuInfo<T> : BStuInfo<T>           //继承自 BStuInfo 泛型类
```

```
{
    public T Chinese;                    //声明语文成绩字段
    public T Math;                       //声明数学成绩字段
    public T English;                    //声明英语成绩字段
}
```

秘笈心法

心法领悟 128：泛型类型也可以可视化。

使用 Visual Studio 2012 开发环境中的“类设计器”可以对现有泛型类型进行可视化，但不能创建新的泛型类型，也不能修改现有泛型类型的类型参数。

实例 129 通过泛型实现子窗体的不同操作

光盘位置：光盘\MR\05\129　　中级　　趣味指数：★★★★★

实例说明

实际开发项目时，有时会因为调用窗体或提示窗体过多，而难于管理，这时，可以通过泛型方法的重载将调用窗体与提示窗体分开编写，这样，当在程序中使用调用窗体或提示窗体时，只需调用指定的泛型方法即可。本实例将通过泛型方法的重载实现子窗体的不同操作，实例运行效果如图 5.27 和图 5.28 所示。

图 5.27　显示子窗体

图 5.28　显示错误提示窗体

关键技术

本实例实现时主要用到了泛型方法的重载，下面对其进行详细讲解。

泛型方法的重载与普通方法的重载是类似的，只是需要在声明泛型方法时添加一个类型参数 T。

说明：（1）关于方法重载的详细讲解请参见实例 111 中的关键技术。

（2）关于声明泛型方法的详细讲解请参见实例 127 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FormDisOperate。

（2）在默认窗体 Form1 中添加两个 Button 控件，分别用来指定调用子窗体和调用提示窗体操作。

（3）在项目中添加一个 Windows 窗体，默认名称为 Form2.cs，用来作为要调用的子窗体。

（4）程序的主要代码如下：

```
public void FormOperate<T>()
{
    Form2 Frm_2 = new Form2();                    //创建 Form2 窗体对象
    Frm_2.ShowDialog();                           //以对话框形式显示 Form2 窗体
}
public void FormOperate<T>(string strError)
```

```
{
    MessageBoxIcon messIcon = MessageBoxIcon.Error;                    //创建提示框中显示图标对象
    MessageBox.Show(strError, "提示", MessageBoxButtons.OK, messIcon);  //显示错误提示框
}
```

秘笈心法

心法领悟 129：使用泛型更好地实现通用化。

在公共语言运行库和 C#语言的早期版本中，通用化是通过在类型与通用基类型 Object 之间进行强制转换来实现的，而泛型则提供了针对这种限制的解决方案。通过创建泛型类，开发人员可以创建一个在编译时类型安全的集合，从而更加灵活地实现了程序代码的通用化。

实例 130	使用泛型去掉数组中的重复数字	初级
	光盘位置：光盘\MR\05\130	趣味指数：★★★☆

实例说明

本实例使用泛型去掉了数组中的重复数字，具体实现时，首先需要对数组进行排序，排序之后重复的数字肯定是相邻的，这时只需要比较邻近的数字是否相同即可。实例运行效果如图 5.29 所示。

图 5.29　使用泛型去掉数组中的重复数字

关键技术

本实例实现时主要用到了 List<T>泛型集合及其 Add 方法、ToArray 方法，下面对本实例用到的关键技术进行详细讲解。

（1）List<T>泛型集合

List<T>泛型集合表示可通过索引访问的对象的强类型列表，提供了用于对列表进行搜索、排序和操作的方法。

说明：List<T>泛型集合位于 System.Collections.Generic 命名空间下。

（2）List<T>泛型集合的 Add 方法

List<T>泛型集合的 Add 方法用于将对象添加到 List<T>的结尾处。语法格式如下：

```
public void Add(T item)
```

参数说明

item：要添加到 List<T>的末尾处的对象。对于引用类型，该值可以为 null。

（3）List<T>泛型集合的 ToArray 方法

List<T>泛型集合的 ToArray 方法用于将 List<T>的元素复制到新数组中。语法格式如下：

```
public virtual void Add(Object key,Object value)
```

参数说明

返回值：一个数组，它包含 List<T>的元素的副本。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RemoveSameNum。

（2）在默认窗体 Form1 中添加两个 Label 控件，分别用来显示原始一维数组和去掉重复数字之后的一维数组。

（3）程序的主要代码如下：

```
/// <summary>
///去掉数组中的重复数字
/// </summary>
/// <param name="P_int_Data">要去除重复数字的 int 数组</param>
/// <returns>取出重复数字之后的数组</returns>
static int[] RemoveNum(int[] P_int_Data)
{
    List<int> P_list_Arrs = new List<int>();                    //创建泛型集合对象
    for (int i = 0; i < P_int_Data.Length - 1; i++)             //循环访问源数组元素
    {
        if (P_int_Data[i] != P_int_Data[i + 1])                 //判断相邻的值是否相同
        {
            P_list_Arrs.Add(P_int_Data[i]);                     //如果不同，将值添加到泛型集合中
        }
    }
    P_list_Arrs.Add(P_int_Data[P_int_Data.Length - 1]);         //将数组的最后一个元素添加到泛型集合中
    return P_list_Arrs.ToArray();                               //将泛型集合转换为数组，并返回
}
```

秘笈心法

心法领悟 130：C#泛型和 C++模板的区别。

C#泛型和 C++模板都是用于提供参数化类型支持的语言功能，然而，这两者之间存在许多差异。在语法层面上，C#泛型是实现参数化类型的更简单方法，不具有 C++模板的复杂性。在实现层面上，主要区别在于 C#泛型类型替换是在运行时执行的，从而为创建的对象保留了泛型类型信息。

第 6 章

数据结构与算法

- 数据结构的实现
- 常见算法的实际应用

6.1 数据结构的实现

实例 131	单向链表的实现 光盘位置：光盘\MR\06\131	初级 趣味指数：★★★★☆

实例说明

链表是一种特殊的数据结构，能够动态地存储一种结构类型数据。在开发复杂的系统时，经常会使用链表存储数据。在本实例中，笔者设计了一个单向链表，并将其编译为类库，以供大家在以后的开发中使用。

关键技术

链表是一种重要的数据结构，该结构由节点组成。每个节点包含两部分数据，第一部分是节点本身的数据，第二部分是指向下一个节点的指针。对于单向链表，链表中存在两个特殊的节点，分别为“头节点”和“尾节点”。头节点本身没有数据，只存储下一个节点的指针，尾节点只存储数据。单向链表结构如图 6.1 所示。

图 6.1 单向链表结构

本例中为了描述节点，笔者设计了一个节点类，代码如下：

```
public class ListNode                                    //节点类
{
    public ListNode(int NewValue)
    {
        Value=NewValue;
    }
    public ListNode Previous;                            //前一个
    public ListNode Next;                                //后一个
    public int Value;                                    //值
}
```

其中，私有成员 Value 用于储存节点本身的数据，Next 用于存储下一个节点的指针，Previous 用于存储上一个节点的指针。对于链表的操作，无非是进行节点的查找、插入和删除操作。

（1）查找节点

在进行节点查找时，通常根据节点的数据查找节点。要实现节点的查找，首先需要解决的问题就是遍历链表中的所有节点。可以从链表头节点开始，利用循环的方式向下查找，如果当前节点指向的下一个节点不为空，则将下一个节点设置为当前节点。

（2）插入节点

插入节点需要考虑两种情况。第一种情况是在链表尾节点处插入节点，这种情况比较简单，只要将当前尾节点的 FNextNode（指向下一个节点的指针）指向新插入的节点即可。第二种情况是在链表的中间位置插入节点，首先需要确定在哪个节点后插入新的节点，这就应该由用户来确定。本例中为了演示的需要，笔者将树视图控件中的当前选项设置为当前节点；然后需要记录当前节点的下一个节点（避免发生“脱节”），可以定义一个临时的节点进行记录；最后将当前节点的下一个节点指向新添加的节点，再将新添加节点的下一个节点指向临时节点。整个插入节点的流程如图 6.2 所示。

（3）删除节点

删除节点与插入节点类似，也需要考虑两种情况。第一种情况是删除尾节点，这种情况操作相对比较简单，首先找到尾节点之前的一个节点，将该节点指向的下一个节点设置为空，然后释放尾节点资源。第二种情况是

删除链表中间部分的节点，首先需要找到当前节点的上一个节点，并定义一个临时节点 PriorNode 进行记录，然后再定义一个临时节点 TempNode 记录当前节点的下一个节点，最后将 PriorNode 的下一个节点指向 TempNode，并释放当前节点。整个删除节点的流程如图 6.3 所示。

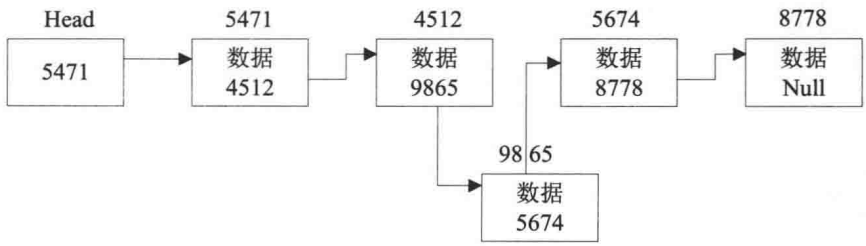


图 6.2　插入节点的流程图

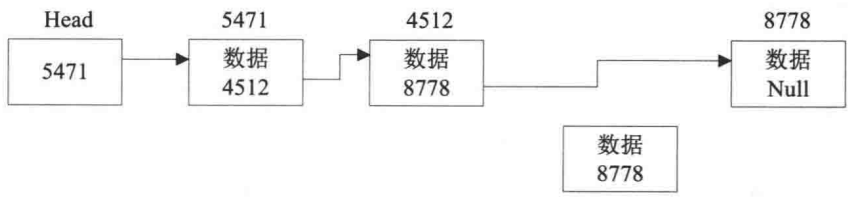


图 6.3　删除节点的流程图

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个类库项目，并将其命名为 UnilateralismChainTable。

（2）将 Class1.cs 文件重命名为 Clist.cs。

（3）程序的主要代码如下。

在构造函数中初始化并定义指针变量，代码如下：

```
public Clist()                                          //构造函数
{
    ListCountValue = 0;                                 //初始化
    Head = null;
    Tail = null;
}
private ListNode Head;                                  //头指针
private ListNode Tail;                                  //尾指针
private ListNode Current;                               //当前指针
private int ListCountValue;                             //链表数据的个数
```

尾部添加数据的代码如下：

```
public void Append(int DataValue)
{
    ListNode NewNode = new ListNode(DataValue);
    if (IsNull())                                       //如果头指针为空
    {
        Head = NewNode;
        Tail = NewNode;
    }
    else
    {
        Tail.Next = NewNode;
        NewNode.Previous = Tail;
        Tail = NewNode;
    }
    Current = NewNode;
    ListCountValue += 1;                                //链表数据个数加 1
}
```

删除当前数据的代码如下：

```
public void Delete()
{
```

```
    if (!IsNull())                                          //若为空链表
    {
        if (IsBof())                                        //若删除头
        {
            Head = Current.Next;
            Current = Head;
            ListCountValue -= 1;
            return;
        }
        if (IsEof())                                        //若删除尾
        {
            Tail = Current.Previous;
            Current = Tail;
            ListCountValue -= 1;
            return;
        }
        Current.Previous.Next = Current.Next;               //若删除中间数据
        Current = Current.Previous;
        ListCountValue -= 1;
        return;
    }
}
```

移动数据的代码如下：

```
public void MoveNext()
{
    if (!IsEof()) Current = Current.Next;                   //向后移动一个数据
}
public void MovePrevious()
{
    if (!IsBof()) Current = Current.Previous;               //向前移动一个数据
}
public void MoveFrist()
{
    Current = Head;                                         //移动到第一个数据
}
public void MoveLast()
{
    Current = Tail;                                         //移动到最后一个数据
}
public bool IsNull()
{
    if (ListCountValue == 0)                                //判断是否为空链表
        return true;
    return false;
}
public bool IsEof()
{
    if (Current == Tail)                                    //判断是否为到达尾部
        return true;
    return false;
}
public bool IsBof()
{
    if (Current == Head)                                    //判断是否为到达头部
        return true;
    return false;
}
public int GetCurrentValue()
{
    return Current.Value;                                   //获取节点值
}
public int ListCount
{
    get
    {
        return ListCountValue;                              //取得链表的数据个数
    }
}
public void Clear()                                         //清空链表
```

```
{
    MoveFrist();
    while (!IsNull())
    {
        Delete();                                          //若不为空链表，从尾部删除
    }
}
```

在当前位置前插入数据的代码如下：

```
public void Insert(int DataValue)
{
    ListNode NewNode = new ListNode(DataValue);
    if (IsNull())
    {
        Append(DataValue);                                 //如果为空表，则添加
        return;
    }
    if (IsBof())
    {
        //为头部插入
        NewNode.Next = Head;
        Head.Previous = NewNode;
        Head = NewNode;
        Current = Head;
        ListCountValue += 1;
        return;
    }
    //中间插入
    NewNode.Next = Current;
    NewNode.Previous = Current.Previous;
    Current.Previous.Next = NewNode;
    Current.Previous = NewNode;
    Current = NewNode;
    ListCountValue += 1;
}
```

升序插入的代码如下：

```
public void InsertAscending(int InsertValue)
{
    //参数：InsertValue 插入的数据
    if (IsNull())                                          //为空链表
    {
        Append(InsertValue);                               //添加
        return;
    }
    MoveFrist();                                           //移动到头
    if ((InsertValue < GetCurrentValue()))
    {
        Insert(InsertValue);                               //满足条件，则插入，退出
        return;
    }
    while (true)
    {
        if (InsertValue < GetCurrentValue())
        {
            Insert(InsertValue);                           //满足条件，则插入，退出
            break;
        }
        if (IsEof())
        {
            Append(InsertValue);                           //尾部添加
            break;
        }
        MoveNext();                                        //移动到下一个指针
    }
}
```

降序插入的代码如下：

```
public void InsertUnAscending(int InsertValue)
{
    //参数：InsertValue 插入的数据
```

```
    if (IsNull())                                    //为空链表
    {
        Append(InsertValue);                         //添加
        return;
    }
    MoveFrist();                                     //移动到头
    if (InsertValue > GetCurrentValue())
    {
        Insert(InsertValue);                         //满足条件，则插入，退出
        return;
    }
    while (true)
    {
        if (InsertValue > GetCurrentValue())
        {
            Insert(InsertValue);                     //满足条件，则插入，退出
            break;
        }
        if (IsEof())
        {
            Append(InsertValue);                     //尾部添加
            break;
        }
        MoveNext();                                  //移动到下一个指针
    }
}
```

秘笈心法

心法领悟 131：算法的概念。

算法是解决问题的方法和步骤，它实际上就是用自然语言或其他方式描述的一个计算机程序，编写计算机程序也就是把用某种方式描述的算法，通过程序设计语言重新对其进行描述。

实例 132 双向链表的实现

光盘位置：光盘\MR\06\132

初级

趣味指数：★★★★☆

实例说明

在单向链表的存储结构中，只有一个指示直接后继的指针域，由此从某个节点出发只能顺时针向后查询其他节点。如果要查询节点的直接前驱，则需要从表头指针出发。为了克服单链表这种单向性的缺点，可以使用双向链表。在双向链表的节点中有两个指针域，其一指向直接前驱，另一个指向直接后继。本实例用 C#实现了双向链表。

关键技术

双向链表的算法描述和单向链表基本相同，具体基础知识可以参照实例 131，但是双向链表在删除和插入节点时与单向链表有很大的不同。双向链表在删除节点时，不但要修改节点的直接后继指针，还要同时修改节点的直接前驱指针。在插入时更是要修改插入节点的前驱和后继的两个方向上的指针。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个类库项目，并将其命名为 BothChainTable。

（2）将 Class1.cs 文件重命名为 DoubleLink.cs。

（3）程序的主要代码如下。

构造函数代码如下：

```
public Objects(int num, string Name, int count)          //构造函数
{
```

```
    //获取参数值
    number = num;
    name = Name;
    counter = count;
}
public int Number/                                    //定义变量
{
    get                                               //访问器
    {
        return number;                                //获取值
    }
    set                                               //访问器
    {
        number = value;                               //设置值
    }
}
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
public int Counter
{
    get
    {
        return counter;
    }
    set
    {
        counter = value;
    }
}
```

定义链表中节点的实现代码如下：

```
public class ListNode//类
{
    public ListNode(Objects bugs)                     //构造函数
    {
        goods = bugs;
    }
    public ListNode Previous;                         //前一个
    public ListNode Next;                             //后一个
    public ListNode next
    {
        get
        {
            return Next;
        }
        set
        {
            Next = value;
        }
    }
    public Objects goods;                             //值
    public Objects Goods
    {
        get
        {
            return goods;
        }
        set
        {
            goods = value;
```

```
        }
    }
}
```

定义链表类的实现代码如下：

```
public Clists()                                          //构造函数
{
    ListCountValue = 0;                                  //初始化
    Head = null;
    Tail = null;
}
private string clistname = "";                           //表名
public string ClistName
{
    get
    {
        return clistname;
    }
    set
    {
        clistname = value;
    }
}
private ListNode Head;                                   //头指针
private ListNode Tail;                                   //尾指针
private ListNode Current;                                //当前指针
public ListNode current
{
    get
    {
        return Current;
    }
    set
    {
        Current = value;
    }
}
private int ListCountValue;                              //链表数据的个数
```

在链表尾部添加数据的实现代码如下：

```
public void Append(Objects DataValue)
{
    ListNode NewNode = new ListNode(DataValue);
    if (IsNull())                                        //如果头指针为空
    {
        Head = NewNode;
        Tail = NewNode;
    }
    else
    {
        Tail.Next = NewNode;
        NewNode.Previous = Tail;
        Tail = NewNode;
    }
    Current = NewNode;
    ListCountValue += 1;                                 //链表数据个数加 1
}
```

删除当前数据的实现代码如下：

```
public void Delete()
{
    if (!IsNull())                                       //若为空链表
    {
        if (IsBof())                                     //若删除头
        {
            Head = Current.Next;
            Current = Head;
            ListCountValue -= 1;
            return;
        }
```

```
            if (IsEof())                                    //若删除尾
            {
                Tail = Current.Previous;
                Tail.next = null;
                Current = Tail;
                ListCountValue -= 1;
                return;
            }
            Current.Previous.Next = Current.Next;
            Current = Current.Previous;
            ListCountValue -= 1;
            return;
        }
    }
    public void MoveNext()
    {
        if (!IsEof()) Current = Current.Next;               //向后移动一个数据
    }
    public void MovePrevious()
    {
        if (!IsBof()) Current = Current.Previous;           //向前移动一个数据
    }
    public void MoveFrist()
    {
        Current = Head;                                     //移动到第一个数据
    }
    public void MoveLast()
    {
        Current = Tail;                                     //移动到最后一个数据
    }
    public bool IsNull()
    {
        if (ListCountValue == 0)                            //判断是否为空链表
        return true;
        else
        return false;
    }
    public bool IsEof()
    {
        if (Current == Tail)                                //判断是否到达尾部
        return true;
        else
        return false;
    }
    public bool IsBof()
    {
        if (Current == Head)                                //判断是否到达头部
          return true;
        else
        return false;
    }
    public Objects GetCurrentValue()                        //获取值
    {
        return Current.goods;
    }
    public int ListCount
    {
        get
        {
            return ListCountValue;                          //取得链表的数据个数
        }
    }
```

清空链表的实现代码如下：

```
public void Clear()
{
    MoveFrist();
    while (!IsNull())
    {
```

```
            Delete();                                    //若不为空链表，从尾部删除
        }
    }
    public void Insert(Objects DataValue)                //在当前位置前插入数据
    {
        ListNode NewNode = new ListNode(DataValue);
        if (IsNull())
        {
            Append(DataValue);                           //为空表，则添加
            return;
        }
        if (IsBof())
        {
            //为头部插入
            NewNode.Next = Head;
            Head.Previous = NewNode;
            Head = NewNode;
            Current = Head;
            ListCountValue += 1;
            return;
        }
        NewNode.Next = Current;                          //中间插入
        NewNode.Previous = Current.Previous;
        Current.Previous.Next = NewNode;
        Current.Previous = NewNode;
        Current = NewNode;
        ListCountValue += 1;
    }
```

升序插入的实现代码如下：

```
    public void InsertAscending(Objects InsertValue)
    {
        //参数：InsertValue 插入的数据
        if (IsNull())                                    //为空链表
        {
            Append(InsertValue);                         //添加
            return;
        }
        MoveFrist();                                     //移动到头
        if ((InsertValue.Number < GetCurrentValue().Number))
        {
            Insert(InsertValue);                         //满足条件，则插入，退出
            return;
        }
        while (true)
        {
            if (InsertValue.Number < GetCurrentValue().Number)
            {
                Insert(InsertValue);                     //满足条件，则插入，退出
                break;
            }
            if (IsEof())
            {
                Append(InsertValue);                     //尾部添加
                break;
            }
            MoveNext();                                  //移动到下一个指针
        }
    }
```

降序插入的实现代码如下：

```
    public void InsertUnAscending(Objects InsertValue)
    {
        //参数：InsertValue 插入的数据
        if (IsNull())                                    //为空链表
        {
            Append(InsertValue);                         //添加
            return;
        }
```

```
    MoveFrist();                                          //移动到头
    if (InsertValue.Number > GetCurrentValue().Number)
    {
        Insert(InsertValue);                              //满足条件，则插入，退出
        return;
    }
    while (true)
    {
        if (InsertValue.Number > GetCurrentValue().Number)
        {
            Insert(InsertValue);                          //满足条件，则插入，退出
            break;
        }
        if (IsEof())
        {
            Append(InsertValue);                          //尾部添加
            break;
        }
        MoveNext();                                       //移动到下一个指针
    }
}
```

秘笈心法

心法领悟 132：算法的特点。

一个算法应该具有以下特点。

- 有穷性：一个算法必须在执行有穷多个计算机步骤后终止。
- 确定性：一个算法给出的每个计算步骤，必须有精确定义、无二义性。
- 有效性：算法中的每一个步骤必须有效地执行，并能得到确定结果。
- 输入：一个算法中可以没有输入，也可以有一个或多个输入信息。这些输入信息是算法所需的初始数据。
- 输出：一个算法应有一个或多个输出，一个算法得到的结果（中间结果或最后结果）就是算法的输出，没有输出的算法是没有意义的。

实例 133	堆栈的实现	初级
	光盘位置：光盘\MR\06\133	趣味指数：★★★★☆

实例说明

栈是一种重要的数据结构。从数据结构的角度看，栈也是线性表，其特殊性在于栈的基本操作是线性表操作的子集，它们是操作受限的线性表，因此可以称为限定性的数据结构。本实例用 C#实现了栈的数据结构，并编译为类库供大家以后使用。

关键技术

栈是限定仅在表尾进行插入或删除操作的线性表。因此对栈来说，表尾端有其特殊含义，称为“栈顶（top）”，相应地，表头端称为“栈底（bottom）”，不含元素的空表称为空栈。栈的修改是按照后进先出的原则进行的。因此栈又称为后进先出（last in first out）的线性表，在实现栈的数据结构时一定要注意这个特点。栈的基本操作除了在栈顶进行插入或删除外，还有栈的初始化，判断是否为空以及取栈顶元素等。

说明：因为在算法中引用了链表的操作，所以要添加对实例 131 所生成类库的引用。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个类库项目，并将其命名为 StackApply。

（2）将 Class1.cs 文件重命名为 CStack.cs。

（3）程序的主要代码如下：

```
public class CStack
{
    private Clist m_List;                                    //调用链表类
    public CStack()
    {
        m_List = new Clist();                                //构造函数
    }
    /// <summary>
    ///压入堆栈
    /// </summary>
    public void Push(int PushValue)
    {
        m_List.Append(PushValue);                            //参数：int PushValue 压入堆栈的数据
    }
    /// <summary>
    ///弹出堆栈数据，如果为空，则取得 2147483647 为 int 的最大值
    /// </summary>
    public int Pop()
    {
        int PopValue;                                        //功能：弹出堆栈数据
        if (!IsNullStack())                                  //不为空堆栈
        {
            MoveTop();                                       //移动到顶
            PopValue = GetCurrentValue();                    //取得弹出的数据
            Delete();                                        //删除
            return PopValue;
        }
        return 2147483647;                                   //空时为 int 类型的最大值
    }
    /// <summary>
    ///判断是否为空的堆栈
    /// </summary>
    public bool IsNullStack()
    {
        if (m_List.IsNull())
        return true;
        return false;
    }
    /// <summary>
    ///堆栈的个数
    /// </summary>
    public int StackListCount
    {
        get
        {
            return m_List.ListCount;
        }
    }
    /// <summary>
    ///移动到堆栈的底部
    /// </summary>
    public void MoveBottom()
    {
        m_List.MoveFrist();
    }
    /// <summary>
    ///移动到堆栈的顶部
    /// </summary>
    public void MoveTop()
    {
        m_List.MoveLast();
    }
    /// <summary>
    ///向上移动
    /// </summary>
    public void MoveUp()
    {
```

```
            m_List.MoveNext();
        }
        /// <summary>
        ///向下移动
        /// </summary>
        public void MoveDown()
        {
            m_List.MovePrevious();
        }
        /// <summary>
        ///取得当前的值
        /// </summary>
        public int GetCurrentValue()
        {
            return m_List.GetCurrentValue();
        }
        /// <summary>
        ///删除取得当前的节点
        /// </summary>
        public void Delete()
        {
            m_List.Delete();
        }
        /// <summary>
        ///清空堆栈
        /// </summary>
        public void Clear()
        {
            m_List.Clear();
        }
    }
```

秘笈心法

心法领悟 133：C#中的堆栈类——Stack 类。

C#中提供了堆栈类，用 Stack 类来表示，该类表示对象的简单的后进先出非泛型集合，其位于 System.Collections 命名空间下。

实例 134　队列的实现

光盘位置：光盘\MR\06\134　　初级　　趣味指数：★★★★☆

实例说明

本实例在 C#中实现队列类，其实队列也是链表的扩展，它是一种特殊的链表，如堆栈一样。它与堆栈的不同在于，堆栈采用的是先进后出原则，而队列采用的是先进先出原则。什么是先进先出呢？在日常生活中也到处可见，例如，买火车票，大家都要排队，先来的先买票，后来的在后面排队，在队伍中随便插入是不合法的，等到前面买完之后，自己才可以买票，这是一个典型的应用，队列在程序设计中是基础的编程技术。

关键技术

和栈相反，队列是先进先出的线性表，只允许在表的一端进行插入，而在另一端删除元素。在队列中，允许插入的一端叫做队尾，允许删除的一端称为队头。在队列的实现过程中主要有两个操作，即入队和出队，具体如下：

（1）入队就是在队列的尾部添加数据，队列数据个数加 1，尾指针后移。

（2）出队就是在队列的头部取数据，然后删除该数据，头指针后移。

说明：因为在算法中引用了链表的操作，所以要添加对实例 131 所生成类库的引用。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个类库项目，并将其命名为 Alignment。

（2）将 Class1.cs 文件重命名为 Queue.cs。

（3）程序的主要代码如下：

```
/// <summary>
///队列类
/// </summary>
public class CQueue
{
    private Clist m_List;
    public CQueue()                                          //构造函数
    {
        m_List = new Clist();                                //这里使用到前面编写的 List
    }
    /// <summary>
    ///入队
    /// </summary>
    public void EnQueue(int DataValue)
    {
        //功能：加入队列，这里使用 List 类的 Append 方法
        m_List.Append(DataValue);                            //尾部添加数据，数据个数加 1
    }
    /// <summary>
    ///出队
    /// </summary>
    public int DeQueue()
    {
        //功能：出队
        //返回值：2147483647 表示为空队列无返回
        int QueValue;
        if (!IsNull())                                       //不为空的队列
        {
            m_List.MoveFrist();                              //移动到队列的头
            QueValue = m_List.GetCurrentValue();             //取得当前的值
            m_List.Delete();                                 //删除出队的数据
            return QueValue;
        }
        return 2147483647;
    }
    /// <summary>
    ///判断队列是否为空
    /// </summary>
    public bool IsNull()
    {
        return m_List.IsNull();                              //判断是否为空的队列
    }
    /// <summary>
    ///清空队列
    /// </summary>
    public void Clear()
    {
        m_List.Clear();                                      //清空链表
    }
    /// <summary>
    ///取得队列的数据个数
    /// </summary>
    public int QueueCount
    {
        get
        {
            return m_List.ListCount;                         //取得队列的个数
        }
    }
}
```

秘笈心法

心法领悟 134：C#中的队列类——Queue 类。

C#中提供了队列类，用 Queue 类来表示，该类表示对象的先进先出集合，其位于 System.Collections 命名空间下。队列在按接收顺序存储消息方面非常有用，以便于进行顺序处理。Queue 类将队列作为循环数组实现，存储在 Queue 类中的对象在一端插入，从另一端移除。

实例 135	树的实现	初级
	光盘位置：光盘\MR\06\135	趣味指数：★★★★☆

实例说明

树形结构是一类重要的非线性数据结构，其中以树和二叉树最为常用。直观来看，树是以分支关系定义的层次结构。树结构在客观世界中广泛存在，如人类社会的族谱和各种社会组织机构都可以用树来形象地表示。树在计算机领域中也得到了广泛的应用，如在编译程序中，可以用树来表示源程序的语法结构。又如在数据库系统中，树形结构也是信息的重要组织形式之一。本实例使用 C#实现了树的数据结构。

关键技术

树的抽象数据类型定义包含树的结构定义和基本操作，下面分别进行介绍。

（1）结构定义

树是 n（n≥1）个节点的有限集，在任意一棵非空树中，都应该符合以下条件：

- 有且仅有一个特定的节点称为根的节点。
- 当 n>1 时，其余节点可分为 m 个互不相交的有限集，其中每一个集合本身又是一棵树，并且称为根的子树。

（2）基本操作

树的基本操作主要包括构造空树、销毁树、清空树、求树的深度、访问节点、插入子树和删除子树等。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个类库项目，并将其命名为 Arbor。

（2）将 Class1.cs 文件重命名为 Ctree.cs。

（3）程序的主要代码如下：

```
class TravelBinTree : ITravelBinTree, IInsertBinTree
{
    const int INIT_TREE_SIZE = 20;
    private IBinNode tree;
    private BinNodeLine head;                          //线索化后的头指针
    private IBinNode prenode;                          //指向最近访问过的前驱节点
    public TravelBinTree()
    {
        tree = new BinNodePtr();
    }
    public TravelBinTree(IBinNode INode)
    {
        tree = INode;
    }
    /// <summary>
    ///先序遍历树，用非递归算法实现
    /// </summary>
    /// <remarks>非递归实现</remarks>
    public void PreOrderTravel()
    {
```

```
        IBinNode temptree;
        Stack stk = new Stack(INIT_TREE_SIZE);
        if (tree == null)
            throw (new InvalidOperationException("访问的树为空"));
        temptree = tree;
        stk.Push(tree);
        while (stk.Count != 0)
        {
            while (temptree != null)
            {
                Print(temptree);
                stk.Push(temptree.Left);
                temptree = temptree.Left;
            }
            stk.Pop(); //空指针退栈
            if (stk.Count != 0)
            {
                temptree = (IBinNode)stk.Pop();
                stk.Push(temptree.Right);
                temptree = temptree.Right;
            }
        }
    }
    public void InOrderTravel()
    {
        InOrderTravel(tree);
    }
    private void InOrderTravel(IBinNode t)
    {
        if (t == null) return;
        InOrderTravel(t.Left);
        Print(t);
        InOrderTravel(t.Right);
    }
    public void RevOrderTravel()
    {
        RevOrderTravel(tree);
    }
    private void RevOrderTravel(IBinNode t)
    {
        if (t == null) return;
        RevOrderTravel(t.Left);
        RevOrderTravel(t.Right);
        Print(t);
    }
    public void Print(IBinNode t)
    {
        Console.Write(t.Element + ",");
    }
    public void Insert(IBinNode node, Position pos)
    {
        if (node == null)
            throw (new InvalidOperationException("不能将空节点插入树"));
        switch (pos)
        {
            case Position.LEFT: tree.Left = node; break;
            case Position.RIGHT: tree.Right = node; break;
        }
    }
    /// <summary>
    ///按照先序遍历顺序遍历树
    /// </summary>
    public void TreeBuilder()
    {
        Stack stk = new Stack(INIT_TREE_SIZE);
        stk.Push(tree);
        Position pos;
        string para;
        pos = Position.LEFT;
```

```
        IBinNode baby, temp;
        while (true)
        {
            para = Console.ReadLine();
            if (para == "")
            {
                if (pos == Position.RIGHT)
                {
                    stk.Pop();
                    while (stk.Count != 0 && ((IBinNode)stk.Peek()).Right != null)
                        stk.Pop();
                    if (stk.Count == 0) break;
                }
                else
                    pos = Position.RIGHT;
            }
            else
            {
                // if (tree.GetType().Equals(baby) == true)
                baby = new BinNodePtr(para);
                temp = (IBinNode)stk.Peek();
                if (pos == Position.LEFT)
                    temp.Left = baby;
                else
                    temp.Right = baby;
                pos = Position.LEFT;
                stk.Push(baby);
            }
        }
    }
    /// <summary>
    ///中序线索化
    /// </summary>
    public void InOrderThreading()
    {
        head = new BinNodeLine("");
        head.RTag = Tag.THREAD;
        head.Right = head;
        if (tree == null) head.Left = head;
        else
        {
            head.Left = tree; prenode = head;
        }
    }
    /// <summary>
    ///中序线索化的递归实现
    /// </summary>
    /// <param name="t"></param>
    private void InThreading(IBinNode t)
    {
        if (t == null)
            return;
        else
        {
            InThreading(t.Left);
        }
    }
}
```

秘笈心法

心法领悟 135：树在 C#中的形象表示。

树在 C#中使用 TreeView 控件来表示。TreeView 控件又称为树视图控件，它可以为用户显示节点层次结构，而每个节点又可以包含子节点，包含子节点的节点叫父节点，其效果就像在 Windows 操作系统的 Windows 资源管理器功能的左窗口中显示文件和文件夹一样。

6.2 常见算法的实际应用

实例 136	计算 $1+2^2+3^3+4^4+\cdots+n^n$ 的值	初级
	光盘位置：光盘\MR\06\136	趣味指数：★★★★☆

实例说明

本实例演示如何使用 C#计算 $1+2^2+3^3+4^4+\cdots+n^n$ 的值。运行本实例，输入要计算的最大整数，按 Enter 键，即可显示表达式及结果。实例运行效果如图 6.4 所示。

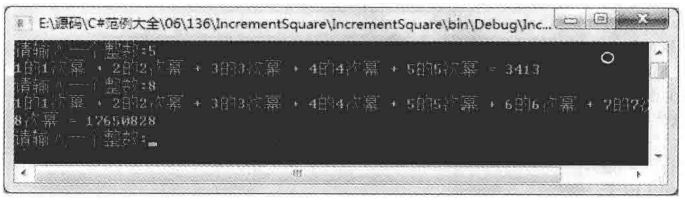


图 6.4 计算 $1+2^2+3^3+4^4+\cdots+n^n$ 的值

关键技术

本实例实现时主要用到了 Math 类的 Pow 方法，下面对其进行详细介绍。

Math 类为三角函数、对数函数和其他通用数学函数提供常数和静态方法，其 Pow 方法用来返回指定数字的指定次幂。语法格式如下：

```
public static double Pow(double x,double y)
```

参数说明

❶ x：要乘幂的双精度浮点数。

❷ y：指定幂的双精度浮点数。

❸ 返回值：double 类型值，表示数字 x 的 y 次幂。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 IncrementSquare。

（2）程序的主要代码如下：

```
public string sum(int num)
{
    string P_str_Expression = "";                                   //定义一个字符串，用来记录表达式
    double sum = 0;                                                 //定义一个变量，用来记录结果
    for (int i = 1; i <= num; i++)                                  //遍历所有要计算的数
    {
        sum += Convert.ToDouble(Math.Pow(i, i));                    //计算结果并记录
        P_str_Expression += i + "的" + i + "次幂" + " + ";          //记录表达式
    }
    return P_str_Expression.Remove(P_str_Expression.Length - 3) + " = " + sum;    //返回表达式和结果
}
```

秘笈心法

心法领悟 136：return 语句的使用。

return 语句用于退出类的方法，控制返回方法的调用者，如果方法有返回类型，return 语句必须返回这个类型的值；如果方法没有返回类型，应使用没有表达式的 return 语句。

实例 137　计算 10！的值

光盘位置：光盘\MR\06\137　　初级　　趣味指数：★★★★☆

实例说明

10！表示求 10 的阶乘，即 1*2*3*…*10。本实例采用递归方法实现数字的阶乘运算。运行本实例，输入要计算的数字，按 Enter 键，即可将指定数字的阶乘结果显示出来。实例运行效果如图 6.5 所示。

图 6.5　计算 10！的值

关键技术

本实例实现时主要是使用乘法运算符（*）计算数字的阶乘结果，下面对乘法运算符（*）进行详细介绍。

乘法运算符（*）用来通过两个或多个数相乘来执行标准的乘法运算，例如，本实例中使用乘法运算符实现阶乘运算的代码如下：

```
return num * factorial(num - 1);                    //计算阶乘结果
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 TenFactiorial。

（2）程序的主要代码如下：

```
public double factorial(int num)                    //计算指定数的阶乘
{
    switch (num)                                    //判断输入的数
    {
        case 1:                                     //如果是 1
            return 1;                               //返回 1
        default:
            return num * factorial(num - 1);        //计算阶乘结果
    }
}
```

秘笈心法

心法领悟 137：使用 switch 语句的注意事项。

使用 switch 语句时需要注意以下 3 点：

（1）每个 case 后面的“常量表达式”的值必须是与“表达式”的类型相同的一个常量，不能是变量。

（2）同一个 switch 语句中的两个或多个 case 标签中指定同一个常数值，会导致编译出错。

（3）一个 switch 语句中最多只能有一个 default 标签，并且每一个标签后面都需要一个 break 语句跳过 switch 语句的其他标签。

实例 138　求最大公约数

光盘位置：光盘\MR\06\138　　初级　　趣味指数：★★★★☆

实例说明

最大公约数是能被两个整数整除的最大整数，例如，24 与 15 两个数的最大公约数为 3。本实例使用 C#实现了求两个数的最大公约数的功能，运行本实例，输入要计算其最大公约数的两个数，按 Enter 键，即可将最大公约数显示出来。实例运行效果如图 6.6 所示。

图 6.6　求最大公约数

关键技术

本实例实现时主要用到了 Math 类的 Max 方法和 Min 方法，下面分别对它们进行详细介绍。

（1）Max 方法

该方法用来返回两个指定数字中较大的一个，该方法为可重载方法。语法格式如下：

```
public static int Max(int val1,int val2)
```

参数说明

❶ val1：要比较的两个 32 位有符号的整数中的第一个。

❷ val2：要比较的两个 32 位有符号的整数中的第二个。

❸ 返回值：val1 或 val2 参数中较大的一个。

（2）Min 方法

该方法用来返回两个指定数字中较小的一个，该方法为可重载方法。语法格式如下：

```
public static int Min(int val1,int val2)
```

参数说明

❶ val1：要比较的两个 32 位有符号的整数中的第一个。

❷ val2：要比较的两个 32 位有符号的整数中的第二个。

❸ 返回值：val1 或 val2 参数中较小的一个。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 GetMaxCommonDivisor。

（2）程序的主要代码如下：

```
public float maxGongYueShu(int n1, int n2)
{
    int temp = Math.Max(n1, n2);                    //求两个数中的最大值
    n2 = Math.Min(n1, n2);                          //求两个数中的最小值
    n1 = temp;                                      //记录临时值
    while (n2 != 0)
    {
        n1 = n1 > n2 ? n1 : n2;                     //使 n1 中的数大于 n2 中的数
        int m = n1 % n2;                            //记录 n1 求余 n2 的结果
        n1 = n2;                                    //交换两个数
        n2 = m;                                     //记录求余结果
    }
    return n1;                                      //得到最大公约数
}
```

秘笈心法

心法领悟 138：do while 循环与 while 循环有什么不同吗？

do while 与 while 语句的运行方式基本相同，只是 do while 语句要先执行一次循环体的内容，然后再判断布尔条件，如果满足条件则执行循环体，不满足布尔条件时退出循环体。

实例 139 求最小公倍数

光盘位置：光盘\MR\06\139　　初级　　趣味指数：★★★★☆

实例说明

最小公倍数可以通过两个数的乘积除以这两个数的最大公约数得到。例如，12 与 9 的最大公约数为 3，则

最小公倍数的计算方法为 12*9/3，计算的结果就是这两个数的最小公倍数。本实例使用 C#实现了求两个数的最小公倍数的功能，运行本实例，输入要计算其最小公倍数的两个数，按 Enter 键，即可将最小公倍数显示出来。实例运行效果如图 6.7 所示。

图 6.7　求最小公倍数

关键技术

本实例实现时主要用到了 Math 类的 Max 方法和 Min 方法。

说明：关于 Math 类的 Max 方法和 Min 方法的详细讲解，请参见实例 138 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 GetMinCommonMultiple。

（2）程序的主要代码如下：

```
public float minGongBeiShu(int n1, int n2)
{
    int temp = Math.Max(n1, n2);                    //求两个数中的最大值
    n2 = Math.Min(n1, n2);                          //求两个数中的最小值
    n1 = temp;                                      //记录临时值
    int product = n1 * n2;                          //求两个数的乘积
    while (n2 != 0)
    {
        n1 = n1 > n2 ? n1 : n2;                     //使 n1 中的数大于 n2 中的数
        int m = n1 % n2;                            //记录 n1 求余 n2 的结果
        n1 = n2;                                    //交换两个数
        n2 = m;                                     //记录求余结果
    }
    return (product / n1);                          //得到最小公倍数
}
```

秘笈心法

心法领悟 139：如何实现无限循环？

在编写循环流程的过程中，很可能出现永远不终止的循环，即所谓的无限循环。出现无限循环的代码可能有下列 3 种情况：

```
For(;;){};
while(true){};
do{}while(true);
```

实例 140　判断素数的算法

光盘位置：光盘\MR\06\140　　初级　　趣味指数：★★★★☆

实例说明

本实例主要判断一个数是否是素数，运行本实例，在窗体中输入数值，单击“判断”按钮，窗体下方即显示出判断结果。实例运行效果如图 6.8 所示。

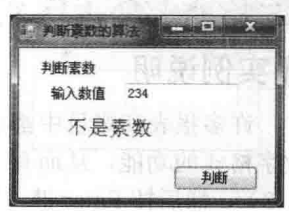


图 6.8　判断素数的算法

关键技术

本实例的重点是什么是素数，素数也叫质数，它是指在一个大于 1 的自然数中，除了 1 和其自身外，不能被其他自然数整除的数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PrimeNumber。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来判断素数；添加一个 TextBox 控件，用来输入信息；添加两个 Label 控件，用来显示信息和输出结果。

（3）程序的主要代码如下：

```
bool prime(int a)                                    //判断素数的方法
{
    int i, count = 0;
    if (a == 2)
        return true;
    else
    {
        for (i = 2; i < a / 2; i++)
        {
            if (a % i == 0)
                return false;
            else
                count++;
        }
        if (count == a / 2 - 2)
            return true;
        else
            return false;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    int j = Convert.ToInt32(textBox1.Text);
    if (prime(j)==true)
    {
        label2.Text = "是素数";
    }
    else
    {
        label2.Text = "不是素数";
    }
}
```

秘笈心法

心法领悟 140：巧用位移运算符。

位移运算符可分为右移运算符和左移运算符两种。右移运算符（>>）将第一个操作数向右移动第二个操作数所指定的位数，左移运算符（<<）将第一个操作数向左移动第二个操作数指定的位数。使用位移运算符时需要注意的是，第二个操作数的类型必须是整数值。

实例 141　按要求生成指定位数的编号

初级

光盘位置：光盘\MR\06\141

趣味指数：★★★★☆

实例说明

许多报表和账目中都指定了数字的格式，如果格式错误，填写的信息将会作废。本实例实现了按要求生成数字格式的功能，从而有效地避免了这种情况的发生。运行本实例，在窗体的文本框中输入数字（本例输入数字 8），然后按 Enter 键，输入的数字将会按指定的格式显示在文本框中（数字按规定显示为 00000008）。实例运行效果如图 6.9 所示。

图 6.9　按要求生成指定位数的编号

关键技术

本实例实现时主要用到了 TextBox 控件的 KeyPress 事件及 KeyPressEventArgs 事件的 Keychar 属性，下面分别对它们进行详细介绍。

（1）KeyPress 事件

该事件在控件有焦点的情况下按下键时发生。语法格式如下：

```
public event KeyPressEventHandler KeyPress
```

（2）KeyPressEventArgs 事件的 Keychar 属性

该属性用来获取或设置与按下的键对应的字符。语法格式如下：

```
public char KeyChar { get; set; }
```

参数说明

属性值：撰写的 ASCII 字符。例如，如果用户按下 Shift + K 组合键，则该属性返回一个大写的 K。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BuildNumber。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入初始编号信息。

（3）程序的主要代码如下：

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == (Char)Keys.Return)                    //如果按下 Enter 键
    {
        if (textBox1.Text.Length > 8)                      //如果位数大于 8
        {
            textBox1.Text = textBox1.Text.Substring(0, 8); //获取前 8 位数
        }
        else
        {
            int j = 8 - textBox1.Text.Length;              //确定增加的位数
            for (int i = 0; i < j; i++)
            {
                textBox1.Text = "0" + textBox1.Text;
            }
        }
    }
}
```

秘笈心法

心法领悟 141：如何处理转义字符？

为了避免转义序列元素转义，可以通过以下两种方案避免转义。

（1）可以通过@符实现，例如：

```
@"C:\Temp\myInfo\hwork.doc"
```

（2）可以通过逐字指定字符串字面值（两个反斜杠）实现，例如：

```
C:\\Temp\\myInfo\\hwork.doc
```

实例 142 身份证号从 15 位升到 18 位的算法

光盘位置：光盘\MR\06\142

初级

趣味指数：★★★★☆

实例说明

2005 年，大部分地区都办理了新身份证，这就涉及身份证从 15 位升级到 18 位的算法。本实例就是针对这一情况编写了一个小程序，用来实现身份证号从 15 位升到 18 位。运行本实例，在窗体的“15 位身份证”文本框中输入 15 位身份证号码，然后单击“转换”按钮，在“18 位身份证”文本框中将显示转换后的身份证号码。实例运行效果如图 6.10 所示。

图 6.10 身份证号从 15 位升到 18 位的算法

关键技术

本实例实现时主要使用了 Math 类的 DivRem 方法进行数学运算，下面对其进行详细介绍。

DivRem 方法用来计算两个 32 位有符号整数的商，并通过输出参数返回余数。语法格式如下：

```
public static int DivRem(int a,int b,out int result)
```

DivRem 方法中的参数说明如表 6.1 所示。

表 6.1 DivRem 方法中的参数说明

参　数	描　述
a	包含被除数的 System.Int32 值
b	包含除数的 System.Int32 值
result	接收余数的 System.Int32 值
返回值	包含指定数字的商的 System.Int32 值

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IDCardRemove。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来输入 15 位身份证号和显示 18 位身份证号；添加一个 Button 控件，用来实现身份证号的转换。

（3）程序的主要代码如下：

```
public string Shen(string id)
{
    int[] w = new int[]{7,9,10,5,8,4,2,1,6,3,7,9,10,5,8,4,2,1};
    char[] a = new char[]{'1','0','x','9','8','7','6','5','4','3','2'};          //设置 18 位最后一位的值
    string newID="";
    if (id.Length == 15)                                                          //判断位数
    {
        int s = 0;
        newID = id.Insert(6, "19");                                               //插入字符串
        for (int i = 0; i < 17; i++)                                              //生成前 17 位
        {
            int k = Convert.ToInt32(newID[i]) * w[i];
            s = s + k;
        }
        int h = 0;
        Math.DivRem(s, 11, out h);                                                //取余数
        newID = newID + a[h];                                                     //生成 18 位
    }
    return newID;
}
```

秘笈心法

心法领悟 142：“;”结束符问题。

与 C/C++一样，C#中的所有代码行都必须以分号结尾。如果在编写代码时，代码行中忘记了写分号，Visual Studio 开发环境的语法检测功能会提示程序开发人员在代码行中输入分号。

实例 143　哥德巴赫猜想的算法实现

光盘位置：光盘\MR\06\143　　中级　　趣味指数：★★★★★

实例说明

如果任意一个大于 6 的偶数都可以写成两个素数的和，就将其称为符合哥德巴赫猜想。下面通过一个实例讲解哥德巴赫猜想算法的实现，实例运行效果如图 6.11 所示。

图 6.11　歌德巴赫猜想的算法实现

关键技术

本实例实现时主要用到了 Math 类的 Sqrt 方法，下面对其进行详细介绍。

Sqrt 方法用来返回指定数字的平方根。语法格式如下：

```
public static double Sqrt(double d)
```

参数说明

d：数字。

返回值：double 类型值，如果 d 为正数或者 0，返回 d 的正平方根；如果 d 为负数，返回 NaN。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 GDBHArith。

（2）程序的主要代码如下：

```
#region  判断一个数是否是素数
/// <summary>
///判断一个数是否是素数
/// </summary>
/// <param name="intNum">要判断的数</param>
/// <returns>如果是，返回 true，否则返回 false</returns>
static bool IsPrimeNumber(int intNum)
{
    bool blFlag = true;                                    //标识是否是素数
    if (intNum == 1 || intNum == 2)                        //判断输入的数字是否是 1 或者 2
```

```
            blFlag = true;                                          //为 bool 类型变量赋值
        else
        {
            int sqr = Convert.ToInt32(Math.Sqrt(intNum));           //对要判断的数字进行开方运算
            for (int i = sqr; i >= 2; i--)                          //从开方后的数进行循环
            {
                if (intNum % i == 0)                                //对要判断的数字和指定数字进行求余运算
                {
                    blFlag = false;                                 //如果余数为 0，说明不是素数
                }
            }
        }
        return blFlag;                                              //返回 bool 型变量
    }
    #endregion
    #region 判断一个数是否符合哥德巴赫猜想
    /// <summary>
    ///判断一个数是否符合哥德巴赫猜想
    /// </summary>
    /// <param name="intNum">要判断的数</param>
    /// <returns>如果符合，返回 true，否则返回 false</returns>
    static bool ISGDBHArith(int intNum)
    {
        bool blFlag = false;                                        //标识是否符合哥德巴赫猜想
        if (intNum % 2 == 0 && intNum > 6)                          //对要判断的数字进行判断
        {
            for (int i = 1; i <= intNum / 2; i++)
            {
                bool bl1 = IsPrimeNumber(i);                        //判断 i 是否为素数
                bool bl2 = IsPrimeNumber(intNum - i);               //判断 intNum-i 是否为素数
                if (bl1 & bl2)
                {
                    //输出等式
                    Console.WriteLine("{0}={1}+{2}", intNum, i, intNum - i);
                    blFlag = true;                                  //符合哥德巴赫猜想
                }
            }
        }
        return blFlag;                                              //返回 bool 型变量
    }
    #endregion
```

秘笈心法

心法领悟 143：正确使用“&&”和“||”运算符。

“&&”运算符与“||”运算符要比“&”运算符与“|”运算符的功能更强。例如，使用“&&”运算符运算时，当“&&”运算符的一个操作数是 false 时，则不需要再考虑后面操作数的值，其结果一定是 false；同样，如果第一个操作数是 true，“||”运算符就一定返回 true。而“&”运算符与“|”运算符则需要把所有的操作数都运算完之后才返回结果。所以，推荐使用“&&”运算符与“||”运算符。

实例 144　八皇后问题的算法实现

光盘位置：光盘\MR\06\144　　中级　　趣味指数：★★★★★

实例说明

八皇后问题是一个古老而著名的问题，它实质上就是使棋盘上的 8 个皇后不能在同一行、同一列或同一条斜线上，共有 92 种方法。本实例使用 C#实现了解决八皇后问题的算法，实例运行效果如图 6.12 所示。

图 6.12　八皇后问题的算法实现

关键技术

本实例在实现八皇后问题时主要用到了回溯算法，其具体实现流程如下：

（1）从第一列开始，为皇后找到安全位置，然后跳转到下一列。

（2）如果在第 n 列出现死胡同，判断该列是第几列，如果为第一列，则棋局失败，否则返回到上一列，再进行回溯。

（3）如果在第 8 列上找到了安全位置，则棋局成功。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 EightQueen。

（2）程序的主要代码如下：

```
#region 解决八皇后问题
/// <summary>
///解决八皇后问题
/// </summary>
/// <param name="size">皇后数量</param>
static void QueenArithmetic(int size)
{
    int[] Queen = new int[size];                          //每行皇后的位置
    int y, x, i, j, d, t = 0;
    y = 0;
    Queen[0] = -1;
    while (true)
    {
        for (x = Queen[y] + 1; x < size; x++)
        {
            for (i = 0; i < y; i++)
            {
                j = Queen[i];
                d = y - i;
                if ((j == x) || (j == x - d) || (j == x + d))   //检查新皇后是否能与以前的皇后相互攻击
                    break;
            }
            if (i >= y)
                break;                                    //不攻击
        }
        if (x == size)                                    //没有合适的位置
        {
            if (0 == y)
            {
                Console.WriteLine("Over");                //回溯到了第一行
                break;                                    //结束
            }
            Queen[y] = -1;                                //回溯
            y--;
        }
        else
        {
            Queen[y] = x;                                 //确定皇后的位置
```

```
            y++;                                        //下一个皇后
            if (y < size)
                Queen[y] = -1;
            else
            {
                Console.WriteLine("\n" + ++t + ':');    //所有的皇后都排完了，输出
                for (i = 0; i < size; i++)
                {
                    for (j = 0; j < size; j++)
                        Console.Write(Queen[i] == j ? 'Q' : '*');
                    Console.WriteLine();
                }
                y = size - 1;                           //回溯
            }
        }
    }
    Console.ReadLine();
}
#endregion
```

秘笈心法

心法领悟 144：程序中尽量不要使用太多嵌套 for 语句。

由于嵌套 for 语句将消耗很大的资源，所以在实际开发项目时，尽量不要使用嵌套 for 语句。

实例 145　百钱百鸡的算法实现　　中级

光盘位置：光盘\MR\06\145　　趣味指数：★★★★★

实例说明

本实例主要计算如何用 100 元钱买 100 只鸡，其中，公鸡 5 元一只，母鸡 3 元一只，小鸡 3 只一元。实例运行效果如图 6.13 所示。

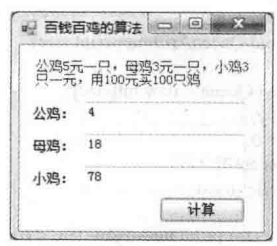


图 6.13　百钱百鸡的算法实现

关键技术

本实例主要使用了 Math 类的 DivRem 方法进行数学运算。

说明：关于 Math 类的 DivRem 方法的详细讲解，请参见实例 142 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 HundredChicken。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 TextBox 控件，分别用来显示公鸡、母鸡和小鸡的个数；添加一个 Button 控件，用来实现百钱百鸡的算法功能。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int a = 0, b = 0, c = 0, p = 0;                     //定义变量
    for (a = 1; a <= 19; a++)                           //公鸡在百元中的遍历
    {
        for (b = 1; b <= 33; b++)                       //母鸡在百元中的遍历
        {
            c = 100 - a - b;                            //获取百元中除了公鸡和母鸡后，小鸡的总钱数
            Math.DivRem(c, 3, out p);                   //计算小鸡的个数
            if (((5 * a + 3 * b + c / 3) == 100) && p == 0)   //如果公鸡、母鸡和小鸡的总钱数加起来为 100
            {
                textBox1.Text = a.ToString();           //显示公鸡的个数
                textBox2.Text = b.ToString();           //显示母鸡的个数
```

```
                textBox3.Text = c.ToString();                    //显示小鸡的个数
                return;
            }
        }
    }
}
```

秘笈心法

心法领悟 145：使用 params 关键字指定多个参数。

在编写一个方法时会遇到这样的问题，不知道这个方法要传入多少个参数，或者随着程序的开发进度，方法的参数不能满足程序的要求，这时使用 params 关键字可以更加灵活地定义该方法的参数。

实例 146　韩信点兵的算法实现

光盘位置：光盘\MR\06\146　　中级　　趣味指数：★★★★★

实例说明

韩信点兵是一道古代数学题，内容是：韩信带兵不足百人，3 人一行排列多一人，7 人一行排列少两人，5 人一行排列正好。本实例通过 C#算法计算出韩信究竟点了多少兵，实例运行效果如图 6.14 所示。

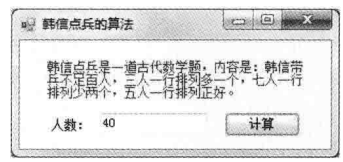


图 6.14　韩信点兵的算法实现

关键技术

本实例主要使用了 Math 类的 DivRem 方法进行数学运算。

说明：关于 Math 类的 DivRem 方法的详细讲解，请参见实例 142 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ALitterArms。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示符合条件的人数；添加一个 Button 控件，用来实现韩信点兵的算法功能。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int a = 0, b = 0, c = 0;                                    //定义变量
    for (int i = 1; i < 100; i++)                               //遍历
    {
        Math.DivRem(i, 3, out a);                               //3 行一列时取余
        Math.DivRem(i, 5, out b);                               //5 行一列时取余
        Math.DivRem(i, 7, out c);                               //7 行一列时取余
        if (a == 1 && b == 0 && c == 5)                         //如果 3 种方式的余数符合要求
        {
            textBox1.Text = i.ToString();                       //显示人数
            return;
        }
    }
}
```

秘笈心法

心法领悟 146：变量的作用域。

变量的作用域是可以访问该变量的代码范围，可以根据以下规则确定作用域：

（1）字段（也称成员变量）所属的类在某个作用域内，则字段也在该作用域内。

（2）在 while、for、do 或类似语句中声明的局部变量存在于该循环体内。

（3）局部变量存在于表示声明该变量的块语句或方法结束的封闭括号之前的作用域内，一旦离开这个作用域后，如果要再次使用这个局部变量名，则要求用户重新声明，虽然这两个变量同名但完全是两个变量，因为它们有各自的作用域。

实例 147　实现斐波纳契数列求和

光盘位置：光盘\MR\06\147　　中级　　趣味指数：★★★★★

实例说明

斐波纳契数列的定义是，它的第一项和第二项均为 1，以后各项都为前两项之和。本实例主要对斐波纳契数列进行求和，实例运行效果如图 6.15 所示。

图 6.15　实现斐波纳契数列求和

关键技术

本实例实现时主要用到了 Convert 类的 ToInt32 方法，下面对其进行详细介绍。

ToInt32 方法用来将数字的指定 String 表示形式转换为等效的 32 位有符号整数。语法格式如下：

```
public static int ToInt32(string value)
```

参数说明

❶ string：包含要转换的数字的 String。

❷ 返回值：与 value 的值等效的 32 位有符号整数。如果 value 为空引用，则为零。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AgreeArithmetic。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用于输入求和的范围；添加一个 ListBox 控件，用于显示求和结果；添加一个 Button 控件，用来实现斐波纳契数列求和算法。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();                                    //清空 ListBox 控件
    int p = 0, n = 0, a = 1, b = 1;                            //定义变量
    try
    {
        p = Convert.ToInt32(textBox1.Text);                    //将字符型转换成数值型
    }
    catch                                                      //出现错误
    {
        MessageBox.Show("请输入数值型数据");                    //弹出提示框
        textBox1.Text = "15";
        return;
    }
    for (n = 1; n <= p; n++)                                   //对输入的数值进行遍历
    {
```

```
        listBox1.Items.Add(a.ToString());                    //输出数值
        listBox1.Items.Add(b.ToString());                    //输出数值
        a = a + b;                                           //获取前两个数的和
        b = a + b;                                           //获取前两个数的和
    }
}
```

秘笈心法

心法领悟 147：使用 MessageBox 显示消息对话框。

向用户显示信息和消息的对话框称为消息对话框。MessageBox 是一个预定义对话框，主要用于向用户显示与应用程序相关的信息及请求来自用户的信息，其 Show 方法用来显示消息对话框。

实例 148　求水仙花数的算法实现

光盘位置：光盘\MR\06\148　　中级　　趣味指数：★★★★★

实例说明

水仙花数是一个 3 位数，每一位数的立方相加等于该数本身。本实例使用 C#实现求水仙花数的算法，实例运行效果如图 6.16 所示。

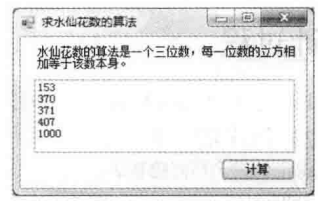


图 6.16　求水仙花数的算法实现

关键技术

本实例主要使用 Math 类的 DivRem 方法进行数学运算。

说明：关于 Math 类的 DivRem 方法的详细讲解，请参见实例 142 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DaffodilAccount。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用于显示计算所得的水仙花数结果；添加一个 Button 控件，用来实现水仙花数的算法功能。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int a = 0, b = 0, c = 0;                                 //定义变量
    listBox1.Items.Clear();                                  //清空 listBox1 控件
    for (int i = 100; i <= 1000; i++)                        //遍历所有 3 位数
    {
        a = i / 100;                                         //获取 3 位数中的第一个数
        Math.DivRem(i, 100, out b);                          //获取 3 位数中的后两位数
        b = b / 10;                                          //获取 3 位数中的第二位数
        Math.DivRem(i, 10, out c);                           //获取 3 位数中的第 3 位数
        a = a * a * a;                                       //计算第一位数的立方
        b = b * b * b;                                       //计算第二位数的立方
        c = c * c * c;                                       //计算第 3 位数的立方
        if ((a + b + c) == i)                                //如果符合水仙花数
            listBox1.Items.Add(i.ToString());                //显示当前 3 位数
    }
}
```

秘笈心法

心法领悟 148：对 ListBox 控件中的数据进行排序。

对 ListBox 控件中的数据进行排序时，只需要将其 Sorted 属性设置为 true 即可，代码如下：

```
listBox1.Sorted = true;
```

实例 149 约瑟夫环问题的算法实现

光盘位置：光盘\MR\06\149

中级

趣味指数：★★★★★

实例说明

本实例使用 C#实现了解决约瑟夫环问题的算法。运行本实例，程序从 12 个人围成的圈中的第 3 个人开始报数，数到第 4 个的人出列，如此反复循环，从而按照约瑟夫环使所有人都出列。实例运行效果如图 6.17 所示。

图 6.17 约瑟夫环问题的算法实现

关键技术

实现约瑟夫环算法时，最重要的是约瑟夫环的流程。约瑟夫环即设有 n 个人坐成一个圈，从某个人开始报数，数到 m 的人出列，接着从出列的下一个人开始重新报数，数到 m 的人再次出列，如此反复循环，直到所有人都出列为止，最后按出列顺序输出。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 YSFHResult。

（2）程序的主要代码如下：

```
#region 约瑟夫环问题算法
/// <summary>
///约瑟夫环问题算法
/// </summary>
/// <param name="total">总人数</param>
/// <param name="start">开始报数的人</param>
/// <param name="alter">要出列的人</param>
/// <returns>返回一个 int 类型的一维数组</returns>
static int[] Jose(int total, int start, int alter)
{
    int j, k = 0;
    int[] intCounts = new int[total + 1];                    //intCounts 数组存储按出列顺序的数据
    int[] intPers = new int[total + 1];                      //intPers 数组存储初始数据
    for (int i = 0; i < total; i++)                          //对数组 intPers 赋初值，第一个人序号为 0，第二个人为 1，依此类推
    {
        intPers[i] = i;
    }
    for (int i = total; i >= 2; i--)                         //按出列次序依次存于数组 intCounts 中
    {
        start = (start + alter - 1) % i;
        if (start == 0)
            start = i;
        intCounts[k] = intPers[start];
        k++;
        for (j = start + 1; j <= i; j++)
            intPers[j - 1] = intPers[j];
    }
    intCounts[k] = intPers[1];
    return intCounts;                                        //结果返回
}
#endregion
```

秘笈心法

心法领悟 149：List<T>泛型集合的使用。

List<T>泛型集合表示可通过索引访问的对象的强类型列表，它是 ArrayList 类的泛型等效类，提供用于对列表进行搜索、排序和操作的方法。

实例 150　C#实现杨辉三角算法

光盘位置：光盘\MR\06\150　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#实现了杨辉三角算法。运行本实例，程序会自动在控制台窗口中绘制一个 10 行的杨辉三角。实例运行效果如图 6.18 所示。

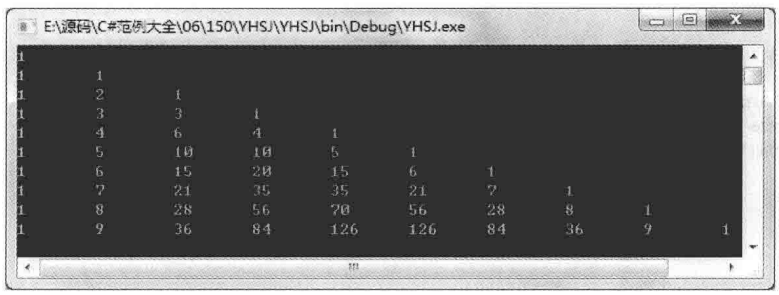


图 6.18　C#实现杨辉三角算法

关键技术

实现杨辉三角算法时，首先需要了解杨辉三角的原理。杨辉三角是一个由数字排列成的三角形数表，其最本质的特征是它的两条边都是由数字 1 组成的，而其余的数则等于它上方的两个数之和。杨辉三角有两种常用的表示形式，分别如图 6.19 和图 6.20 所示。

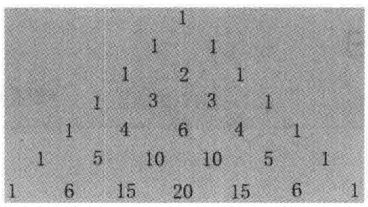


图 6.19　杨辉三角表示一

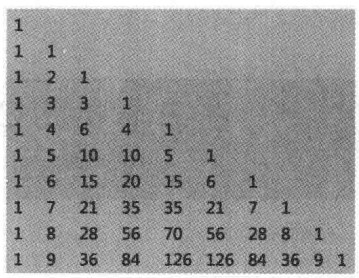


图 6.20　杨辉三角表示二

说明：图 6.19 和图 6.20 这两种表现形式并没有本质上的区别，因为杨辉三角的概念中只是提到了由数字排列成的三角形数表，而没有具体说是哪种形式的三角形，所以只要三角形的两条边都是由数字 1 组成的，而其余的数等于它上方的两个数之和，那么就可以说是一个杨辉三角形。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 YHSJ。

（2）程序的主要代码如下：

```
static void Main(string[] args)
{
    int[][] Array_int = new int[10][];                          //定义一个 10 行的二维数组
    //向数组中记录杨辉三角形的值
    for (int i = 0; i < Array_int.Length; i++)                  //遍历行数
    {
        Array_int[i] = new int[i + 1];                          //定义二维数组的列数
        for (int j = 0; j < Array_int[i].Length; j++)           //遍历二维数组的列数
        {
```

```
            if (i <= 1)                                                    //如果是数组的前两行
            {
                Array_int[i][j] = 1;                                       //将其设置为 1
                continue;
            }
            else
            {
                if (j == 0 || j == Array_int[i].Length - 1)                //如果是行首或行尾
                    Array_int[i][j] = 1;                                   //将其设置为 1
                else
                    Array_int[i][j] = Array_int[i - 1][j - 1] + Array_int[i - 1][j];   //根据杨辉三角算法进行计算
            }
        }
    }
    for (int i = 0; i < Array_int.Length; i++)                             //输出杨辉三角形
    {
        for (int j = 0; j < Array_int[i].Length; j++)
            Console.Write("{0}\t", Array_int[i][j]);
        Console.WriteLine();
    }
    Console.ReadLine();
}
```

秘笈心法

心法领悟 150：杨辉三角的发展历史。

北宋人贾宪约 1050 年首先使用“贾宪三角”进行高次开方运算。13 世纪，中国宋代数学家杨辉在《详解九章算术》里讨论了这种形式的数表，并说明此表引自 11 世纪前半叶贾宪的《释锁算术》，并绘画了“古法七乘方图”，因此，杨辉三角又被称为“贾宪三角”。元朝数学家朱世杰在《四元玉鉴》（1303 年）扩充 “贾宪三角”成了“古法七乘方图”。意大利人称之为“塔塔利亚三角形”（Triangolo di Tartaglia），以纪念在 16 世纪发现一元三次方程解的塔塔利亚。在欧洲直到 1623 年以后，法国数学家帕斯卡在 13 岁时发现了“帕斯卡三角”。

实例 151　如何将 B 转换成 GB、MB 和 KB

初级

光盘位置：光盘\MR\06\151

趣味指数：★★★★☆

实例说明

KB、MB 和 GB 之间的进位关系是 1KB=1024B，1MB=1024KB，1GB=1024MB。本实例将根据这个关系将 B 转换成相关格式的 KB、MB 或 GB，实例运行效果如图 6.21 所示。

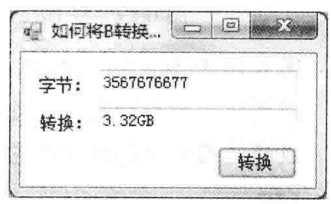


图 6.21　如何将 B 转换成 GB、MB 和 KB

关键技术

本实例实现时主要使用了 Math 类的 Round 方法对转换后的单精度小数进行四舍五入运算，下面对该方法进行详细介绍。

Round 方法将双精度浮点值舍入到指定精度。语法格式如下：

```
public static double Round(double value, int digits)
```

参数说明

❶ value：要舍入的双精度浮点数。

❷ digits：返回值中的有效数字位数（精度）。

❸ 返回值：精度等于 digits，最接近 value 的数字。如果 value 位于两个数字的中间，其中一个为偶数，另一个为奇数，则返回偶数。如果 value 的精度小于 digits，则返回 value 而不做更改。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ByteConversion。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来输入字节值和显示转换后的值；添加一个 Button 控件，用于实现字节转换功能。

（3）程序的主要代码如下：

```
const int GB = 1024 * 1024 * 1024;                          //定义 GB 的计算常量
const int MB = 1024 * 1024;                                 //定义 MB 的计算常量
const int KB = 1024;                                        //定义 KB 的计算常量
public string ByteConversionGBMBKB(Int64 KSize)
{
    if (KSize / GB >= 1)                                    //如果当前 Byte 的值大于等于 1GB
        return (Math.Round(KSize / (float)GB, 2)).ToString() + "GB";   //将其转换成 GB
    else if (KSize / MB >= 1)                               //如果当前 Byte 的值大于等于 1MB
        return (Math.Round(KSize / (float)MB, 2)).ToString() + "MB";   //将其转换成 MB
    else if (KSize / KB >= 1)                               //如果当前 Byte 的值大于等于 1KB
        return (Math.Round(KSize / (float)KB, 2)).ToString() + "KB";   //将其转换成 KB
    else
        return KSize.ToString() + "Byte";                   //显示 Byte 值
}
```

秘笈心法

心法领悟 151：硬盘空间的单位表示法。

硬盘空间的单位表示主要有 B、KB、MB 和 GB 4 种方式，其中 B 表示字节，这 4 种表示法之间的换算公式为：1KB=1024B，1MB=1024KB，1GB=1024MB。

实例 152　0～N 位数的任意组合

光盘位置：光盘\MR\06\152　　初级　　趣味指数：★★★★☆

实例说明

本实例将 0～5 的数字按指定的位数进行任意的组合，位数不能超过数字的个数，否则结果无效。实例运行效果如图 6.22 所示。

图 6.22　0～N 位数的任意组合

关键技术

本实例实现时主要使用 ArrayList 类实现了 N 位数的任意组合，下面对 ArrayList 类进行详细介绍。

（1）ArrayList 类的声明

ArrayList 提供了 3 个构造器，通过这 3 个构造器可以有 3 种声明方式，分别如下：

- 默认的构造器，将会以默认（16）的大小来初始化内部的数组。构造器格式如下：

```
public ArrayList();
```

通过以上构造器，ArrayList 声明的语法格式如下：

```
ArrayList List = new ArrayList();
```

参数说明

List：数组名。

- 用一个 ICollection 对象来构造，并将该集合的元素添加到 ArrayList 中。构造器语法格式如下：

```
public ArrayList(ICollection);
```

通过以上构造器，ArrayList 声明的语法格式如下：

```
ArrayList List = new ArrayList(arryName);
```

参数说明

❶ List：数组名。

❷ arryName：要添加集合的数组名。

❑ 用指定的大小来初始化内部的数组。构造器语法格式如下：

```
public ArrayList(int);
```

通过以上构造器，ArrayList 声明的语法格式如下：

```
ArrayList List = new ArrayList(n);
```

参数说明

❶ List：数组名。

❷ n：数组的空间大小。

📖 **说明**：程序中使用 ArrayList 类时，首先需要在命名空间区域添加 System.Collections 命名空间。

（2）ArrayList 类的常用方法

ArrayList 类的常用方法及说明如表 6.2 所示。

表 6.2 ArrayList 类的常用方法及说明

方　法	说　明
Add	用于添加一个元素到当前列表的末尾
AddRange	用于添加一批元素到当前列表的末尾
Remove	用于删除一个元素，通过元素本身的引用来删除
RemoveAt	用于删除一个元素，通过索引值来删除
RemoveRange	用于删除一批元素，通过指定开始的索引和删除的数量来删除
Insert	用于添加一个元素到指定位置，列表后面的元素依次向后移动
InsertRange	用于从指定位置开始添加一批元素，列表后面的元素依次向后移动
Clear	用于清除现有的所有元素
Contains	用来查找某个对象是否在列表中
ToArray	将 ArrayList 中的元素复制到一个新的数组中

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ArbitrarinessCombination。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来显示组合结果；添加一个 TextBox 控件，用来输入要组合的位数；添加一个 Button 控件，用于实现指定位数的任意组合。

（3）程序的主要代码如下。

自定义方法 SetArbitrariness 用于实现 N 位数的任意组合，参数 n 表示要组合的位数，参数 List 表示空的 ArrayList 类，用于记录组合数。SetArbitrariness 方法的实现代码如下：

```
static string[] a =new string[6]{ "0", "1", "2", "3", "4", "5" };          //定义数组
int num = 6;                                                               //定义最大的个数
int nIdx = 0, nSidx = 0;
public void SetArbitrariness(int n, ArrayList List)
{
    ArrayList SList = new ArrayList(a);                                    //创建 ArrayList 对象
    SList.Clear();                                                         //清空 ArrayList
    try
    {
        if (n >= 1)                                                        //如果大于等于 1
        {
            if (List.Count == 0)                                           //如果 List 为空
```

```
            {
                for (nIdx = 0; nIdx <= num - 1; nIdx++)                    //添加单位数
                    SList.Add(a[nIdx]);
            }
            else
            {
                //添加多位数
                for (nIdx = 0; nIdx <= num - 1; nIdx++)
                    for (nSidx = 0; nSidx <= List.Count - 1; nSidx++)
                        if (List[nSidx].ToString().IndexOf(a[nIdx]) == -1)  //如果当前的值没有添加
                            SList.Add(a[nIdx] + List[nSidx].ToString());    //添加
            }
            SetArbitrariness(n - 1, SList);                                //递归该方法
        }
        if (SList.Count > 0)                                               //如果有数值
        {
            List.Clear();                                                  //清空 List
            for (int i = 0; i < SList.Count; i++)                          //添加单位数
                List.Add(SList[i].ToString());
        }
    }
    catch
    {
        SList.Clear();
    }
}
```

“计算”按钮的 Click 事件用于实现指定位数的任意组合，实现代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    ArrayList List = new ArrayList(a);                                     //创建 ArrayList 对象
    List.Clear();                                                          //清空 ArrayList 对象
    listBox1.Items.Clear();                                                //清空 listBox1 控件
    SetArbitrariness(Convert.ToInt32(textBox1.Text), List);                //调用自定义方法
    //将不同的数值添加到 listBox1 控件上
    for (int i = 0; i < List.Count; i++)
        listBox1.Items.Add(List[i].ToString());
    listBox1.Items.Add("Total:" + listBox1.Items.Count.ToString());        //显示一共执行了几行
}
```

秘笈心法

心法领悟 152：更高级的 List<T>泛型集合。

List<T>泛型集合表示可通过索引访问的对象的强类型列表，它提供用于对列表进行搜索、排序和操作的方法，相对于 ArrayList 类来说，List<T>泛型集合在大多数情况下执行得更好并且是类型安全的。

实例 153　在数组中快速查找近似值

光盘位置：光盘\MR\06\153　　初级　　趣味指数：★★★★☆

实例说明

本实例使用二分查找方法在一个数组中查找已知数字的近似值。运行本实例，在“近似值”文本框中输入要查找的数字，单击“查找”按钮，即可在已知数组中查找输入值的近似值，并显示出来。实例运行效果如图 6.23 所示。

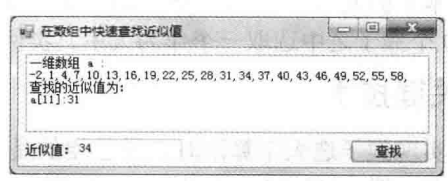


图 6.23　在数组中快速查找近似值

关键技术

本实例实现时主要用到了二分查找法，下面对其进行详细介绍。

二分查找法又称折半查找法，它是一种效率较高的查找方法。二分查找的基本思路是，首先将节点按关键

字排序，其次将待查找值与中间位置的值进行比较，如果相等则查找成功，否则说明中间数据值不是大于就是小于待查找值，然后继续进行这样的二分查找，直到查找结束。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LookupNear。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用于显示要查找的一组数字，以及查找到的近似值；添加一个 TextBox 控件，用来输入要查找的数值；添加一个 Button 控件，用于使用二分查找法在已知数组中查找指定值的近似值。

（3）程序的主要代码如下：

```
public string LOCATE(int[] XX, int N, int X)                    //N 表示数值的个数，X 表示查找的数值
{
    int tem_n;                                                  //定义变量
    int jl, ju, jm;
    jl=0;
    ju=N + 1;
    //利用二分查找法进行查询
    rebound:                                                    //goto 语句
    if (ju - jl > 1)                                            //如果没有遍历完
    {
        jm = (ju + jl) / 2;                                     //获取中间的位置
        if (XX[N] > XX[1] == X > XX[jm])                        //利用二分查找进行判断
            jl = jm;
        else
            ju = jm;
        goto rebound;
    }
    tem_n = jl;                                                 //获取近似值的位置
    if (X - XX[tem_n] > (XX[tem_n + 1] - N))                    //判断左右哪个更接近
        tem_n = jl + 1;
    return "a[" + tem_n.ToString() + "]:" + XX[tem_n].ToString();   //返回近似值
}
```

秘笈心法

心法领悟 153：尽量避免使用 goto 语句。

虽然 goto 语句有一定的使用价值，但是目前对它的使用存在争议。有人建议避免使用它，有人建议把它用来作为排除错误的基本工具，各种观点截然不同。虽然许多人不用 goto 语句也能够编程，但是仍然有人使用它。所以要小心使用，同时一定要确保程序是可维护的。

实例 154　猴子选大王算法的实现

光盘位置：光盘\MR\06\154　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#实现了猴子选大王的算法。运行本实例，程序自动在 10 个猴子之中选取一个作为大王。实例运行效果如图 6.24 所示。

图 6.24　猴子选大王算法的实现

关键技术

实现猴子选大王算法时，最重要的是猴子选大王的流程。猴子选大王即一群（M）猴子排成一列，数到 N 的退出，直到剩下一个。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 SelectMonkeyKing。

（2）程序的主要代码如下：

```
public int King(int M, int N)                    //总人数 M，数到第 N 个排除
{
    int k = 0;                                   //定义一个变量，用来存储号码
    for (int i = 2; i <= M; i++)                 //遍历猴子个数
        k = (k + N) % i;                         //计算号码值
    return ++k;                                  //返回猴子号码
}
```

秘笈心法

心法领悟 154：使用 ThreadPool 创建线程。

在应用程序开发中，如果做一些比较耗时的工作或启动其他应用程序，那么可以通过线程池创建一个新的线程来处理这些任务，这样可以防止因主线程繁忙而出现“假死”现象。

实例 155　使用 MD5 算法对密码进行加密

光盘位置：光盘\MR\06\155　　高级　　趣味指数：★★★★★

实例说明

MD5（Message Digest Algorithm 5）是一种被广泛使用的“消息-摘要算法”。“消息-摘要算法”实际上就是一个单项散列函数，数据块经过单向散列函数得到一个固定长度的散列值，数据块的签名就是计算数据块的散列值，MD5 算法的散列值为 128 位。本实例演示如何使用 MD5 算法对用户输入的密码进行加密，实例运行效果如图 6.25 所示。

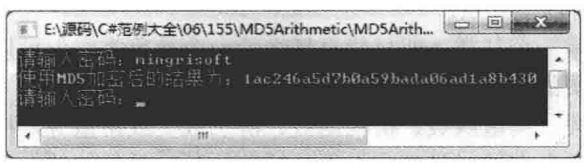


图 6.25　使用 MD5 算法对密码进行加密

关键技术

本实例在实现时主要用到了 MD5 类的 ComputeHash 方法，下面对其进行详细讲解。

MD5 类表示 MD5 哈希算法的所有实现均从中继承的抽象类，该类位于 System.Security.Cryptography 命名空间下，其 ComputeHash 方法有以下 3 种重载形式。

❑　计算指定字节数组的哈希值，语法格式如下：

```
public byte[] ComputeHash(byte[] buffer)
```

参数说明

❶ buffer：要计算其哈希代码的输入。

❷ 返回值：计算所得的哈希代码。

❑　计算指定 Stream 对象的哈希值，语法格式如下：

```
public byte[] ComputeHash(Stream inputStream)
```

参数说明

❶ inputStream：要计算其哈希代码的输入。

❷ 返回值：计算所得的哈希代码。

❑　计算指定字节数组的指定区域的哈希值，语法格式如下：

```
public byte[] ComputeHash(byte[] buffer,int offset,int count)
```

ComputeHash 方法中的参数说明如表 6.3 所示。

表 6.3　ComputeHash 方法中的参数说明

参　数	描　述	参　数	描　述
buffer	要计算其哈希代码的输入	count	数组中用作数据的字节数
offset	字节数组中的偏移量，从该位置开始使用数据	返回值	计算所得的哈希代码

说明：本实例用到了 ComputeHash 方法的第一种重载形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 MD5Arithmetic。

（2）程序的主要代码如下：

```
public string Encrypt(string strPwd)
{
    MD5 md5 = new MD5CryptoServiceProvider();                    //创建 MD5 对象
    byte[] data = System.Text.Encoding.Default.GetBytes(strPwd); //将字符编码为一个字节序列
    byte[] md5data = md5.ComputeHash(data);                      //计算 data 字节数组的哈希值
    md5.Clear();                                                 //清空 MD5 对象
    string str = "";                                             //定义一个变量，用来记录加密后的密码
    for (int i = 0; i < md5data.Length - 1; i++)                 //遍历字节数组
    {
        str += md5data[i].ToString("x").PadLeft(2, '0');         //对遍历到的字节进行加密
    }
    return str;                                                  //返回得到的加密字符串
}
```

秘笈心法

心法领悟 155：抽象类和接口的区别。

抽象类和接口的区别主要有以下几点：

- 它们的派生类只能继承一个基类，即只能直接继承一个抽象类，但可以继承任意多个接口。
- 抽象类中可以定义成员的实现，但接口中不可以。
- 抽象类中可以包含字段、构造函数、析构函数、静态成员或常量等，接口中不可以。
- 抽象类中的成员可以是私有的（只要它们不是抽象的）、受保护的、内部的或受保护的内部成员（受保护的内部成员只能在应用程序的代码或派生类中访问），但接口中的成员必须是公共的。

实例 156　凯撒密码算法的实现

光盘位置：光盘\MR\06\156　　高级　　趣味指数：★★★★★

实例说明

凯撒密码据传是古罗马凯撒大帝用来保护重要军情的加密系统，它是一种置换密码，通过将字母顺序推后起到加密作用。例如，将字母顺序推后 3 位，字母 A 将被推作字母 D，字母 B 将被推作字母 E。本实例使用 C#实现了凯撒加密的算法，实例运行效果如图 6.26 所示。

图 6.26　凯撒密码算法的实现

关键技术

本实例实现时主要用到了 String 类的 ToCharArray 方法和 Convert 类的 ToChar 方法，下面分别对它们进行详细介绍。

（1）String 类的 ToCharArray 方法

String 类的 ToCharArray 方法用来将字符串中的字符复制到 Unicode 字符数组，该方法有两种重载形式，本

实例中用到的它的重载形式如下：

```
public char[] ToCharArray()
```

参数说明

返回值：元素为此字符串的各字符的 Unicode 字符数组。如果此字符串是空字符串，则返回的数组为空且长度为零。

（2）Convert 类的 ToChar 方法

Convert 类的 ToChar 方法用来将指定的值转换为 Unicode 字符，该方法为可重载方法，本实例中用到的它的重载形式如下：

```
public static char ToChar(int value)
```

参数说明

❶ value：32 位有符号整数。

❷ 返回值：等效于 value 的值的 Unicode 字符。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个控制台应用程序，并将其命名为 CaesarArithmetic。

（2）程序的主要代码如下：

```
public string Caesar(string str)                                                    //凯撒加密算法的实现
{
    char[] c = str.ToCharArray();                                                   //创建字符数组
    string strCaesar = "";                                                          //定义一个变量，用来存储加密后的字符串
    for (int i = 0; i < str.Length; i++)                                            //遍历字符串中的每一个字符串
    {
        string ins = c[i].ToString();                                               //记录遍历到的字符
        string outs = "";                                                           //定义一个变量，用来记录加密后的字符串
        //判断指定的字符串中是否包含遍历到的字符
        bool isChar = "0123456789abcdefghijklmnopqrstuvwxyz".Contains(ins.ToLower());
        bool isToUpperChar = isChar && (ins.ToUpper() == ins);                      //判断遍历到的字符是否是大写
        ins = ins.ToLower();                                                        //将遍历到的字符转换为小写
        if (isChar)                                                                 //判断指定的字符串中是否包含遍历到的字符
        {
            int offset = (AscII(ins) + 5 - AscII("a")) % (AscII("z") - AscII("a") + 1);  //获取字符的 ASCII 码
            outs = Convert.ToChar(offset + AscII("a")).ToString();                  //转换为字符并记录
            if (isToUpperChar)                                                      //判断是否大写
            {
                outs = outs.ToUpper();                                              //全部转换为大写
            }
        }
        else
        {
            outs = ins;                                                             //记录遍历的字符
        }
        strCaesar += outs;                                                          //添加到加密字符串中
    }
    return strCaesar;                                                               //返回加密后的字符串
}
```

秘笈心法

心法领悟 156：如何获取字符的 ASCII 码？

获取字符的 ASCII 码时，主要用到 Encoding 类的 ASCII 属性的 GetBytes 方法。Encoding 类表示字符编码，其 ASCII 属性用来获取 ASCII（7 位）字符集的编码，该属性的 GetBytes 方法用来将一组字符编码为一个字节序列。例如，本实例中获取字符的 ASCII 码的实现代码如下：

```
byte[] array = new byte[1];                                                         //创建字节数组
array = System.Text.Encoding.ASCII.GetBytes(str);                                   //为字节数组赋值
int asciicode = (short)(array[0]);                                                  //获取字节数组的第一项
return asciicode;                                                                   //返回字节数组的第一项
```

Windows 窗体篇

- 第 7 章　Windows 窗体基础
- 第 8 章　特色窗体界面
- 第 9 章　窗体控制技术
- 第 10 章　MDI 窗体和继承窗体

第 7 章

Windows 窗体基础

- 设置窗体位置
- 设置窗体大小
- 设置窗体的标题栏
- 设置窗体的背景

7.1　设置窗体位置

实例 157	控制窗体加载时的位置 光盘位置：光盘\MR\07\157	初级 趣味指数：★★★★☆

实例说明

第一次运行 Windows 窗体应用程序时，窗体一般都有一个默认的显示位置，如在桌面居中显示、在桌面上的任意位置显示等。本实例将通过 C#代码控制窗体加载时在桌面上居中显示，实例运行效果如图 7.1 所示。

图 7.1　控制窗体加载时的位置

关键技术

本实例在控制窗体加载时的位置时，主要用到了窗体的 StartPosition 属性，下面对其进行详细讲解。

StartPosition 属性用来获取或设置运行时窗体的起始位置。语法格式如下：

```
public FormStartPosition StartPosition { get; set; }
```

参数说明

属性值：FormStartPosition 枚举值之一，表示窗体的起始位置。FormStartPosition 枚举值及说明如表 7.1 所示。

表 7.1　FormStartPosition 枚举值及说明

枚 举 值	说　明
CenterParent	窗体在其父窗体中居中
CenterScreen	窗体在当前显示窗口中居中，尺寸在窗体大小中指定
Manual	窗体的位置由 Location 属性确定
WindowsDefaultBounds	窗体定位在 Windows 默认位置，边界也由 Windows 默认决定
WindowsDefaultLocation	窗体定位在 Windows 默认位置，尺寸在窗体大小中指定

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LoadPosition。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.StartPosition = FormStartPosition.CenterScreen;                    //设置窗体居中显示
}
```

秘笈心法

心法领悟 157：Visual Studio 开发环境的发展。

Visual Studio 开发环境是微软为了配合.NET 战略推出的 IDE 开发环境，同时也是开发 C#应用程序最好的工具。Visual Studio 开发环境从最初的 2002 版本，经过 2003 和 2005 两个版本的过渡，现在普遍使用的是 2012

版本，而且其 2010 版本的正式版已经于 2010 年 4 月 12 日正式发布。

实例 158　设置窗体在屏幕中的位置

光盘位置：光盘\MR\07\158　　初级　　趣味指数：★★★☆

实例说明

在窗体的属性中可以设置窗体为居中显示，本实例通过设置窗体的 Left 属性和 Top 属性可以准确设置窗体的位置。实例运行效果如图 7.2 所示。

图 7.2　设置窗体在屏幕中的位置

关键技术

本实例设置窗体在屏幕中的位置时，主要通过设置窗体的 Left 属性和 Top 属性来实现，下面分别对它们进行详细讲解。

（1）Left 属性

该属性用来获取或设置窗体左边缘与其容器的工作区左边缘之间的距离。语法格式如下：

```
[BrowsableAttribute(false)]
public int Left { get; set; }
```

参数说明

属性值：窗体左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。

（2）Top 属性

该属性用来获取或设置窗体上边缘与其容器的工作区上边缘之间的距离。语法格式如下：

```
[BrowsableAttribute(false)]
public int Top { get; set; }
```

参数说明

属性值：窗体上边缘与其容器的工作区上边缘之间的距离（以像素为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetLocation。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来输入窗体距屏幕左侧和上方的距离；添加一个 Button 控件，用来设置窗体在屏幕上的位置。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.Left = Convert.ToInt32(textBox1.Text);          //设置窗体左边缘与屏幕左边缘之间的距离
    this.Top = Convert.ToInt32(textBox2.Text);           //设置窗体上边缘与屏幕上边缘之间的距离
}
```

秘笈心法

心法领悟 158：什么是 Windows 窗体？

在 Windows 窗体应用程序中，Windows 窗体是向用户显示信息的可视界面，它是 Windows 窗体应用程序的基本单元。Windows 窗体具有自己的特征，开发人员可以通过编程来进行设置。Windows 窗体也是对象，窗

体类定义了生成窗体的模板，每创建一个窗体对象，就产生一个 Windows 窗体。

实例 159　从上次关闭位置启动窗体

光盘位置：光盘\MR\07\159　　初级　　趣味指数：★★★★☆

实例说明

实际开发中，有很多软件都有一个通用的功能，即从上次关闭位置启动窗体，那么可不可以用 C#语言实现这样的功能？答案是肯定的，本实例就将使用 C#语言实现从上次关闭位置启动窗体的功能。实例运行效果如图 7.3 所示。

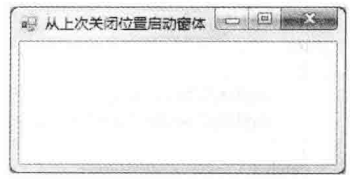


图 7.3　从上次关闭位置启动窗体

关键技术

本实例主要是通过在注册表中读写窗体的 Location 属性来实现的。实例中，在窗体关闭前处理窗体的 FormClosed 事件，将窗体的 Location 属性值写入注册表，然后在窗体的 Load 事件中从注册表中读取保存的数据，设置窗体的 Location 属性值，从而实现了从上次关闭位置启动窗体的功能。下面对本实例中用到的关键技术进行详细讲解。

（1）Location 属性

该属性用来获取或设置窗体的左上角相对于桌面的左上角坐标。语法格式如下：

```
[SettingsBindableAttribute(true)]
public Point Location { get; set; }
```

参数说明

属性值：Point 结构，表示窗体的左上角相对于桌面的左上角坐标。

（2）读写注册表

C#中对注册表进行读写，主要是通过 RegistryKey 类的 GetValue 方法和 SetValue 方法来实现的，其中，GetValue 方法用来检索与指定名称关联的值，其最常用的语法格式如下：

```
public Object GetValue(string name)
```

参数说明

❶ name：要检索的值的名称。

❷ 返回值：与 name 关联的值；如果未找到 name，则为 null。

SetValue 方法用来设置注册表项中的名称/值对的值，其最常用的语法格式如下：

```
public void SetValue(string name,Object value)
```

参数说明

❶ name：要存储的值的名称。

❷ value：要存储的数据。

说明：程序中使用 RegistryKey 类时，首先需要在命名空间区域添加 Microsoft.Win32 命名空间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StartFormByLClosePosition。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    RegistryKey myReg1, myReg2;                    //声明注册表对象
    myReg1 = Registry.CurrentUser;                 //获取当前用户注册表项
```

```
    try
    {
        myReg2 = myReg1.CreateSubKey("Software\\MySoft");          //在注册表项中创建子项
        this.Location = new Point(Convert.ToInt16(myReg2.GetValue("1")), Convert.ToInt16(myReg2.GetValue("2")));   //设置窗体的显示位置
    }
    catch { }
}
private void Frm_Main_FormClosed(object sender, FormClosedEventArgs e)
{
    RegistryKey myReg1, myReg2;                                    //声明注册表对象
    myReg1 = Registry.CurrentUser;                                 //获取当前用户注册表项
    myReg2 = myReg1.CreateSubKey("Software\\MySoft");              //在注册表项中创建子项
    try
    {
        myReg2.SetValue("1", this.Location.X.ToString());          //将窗体关闭位置的 x 坐标写入注册表
        myReg2.SetValue("2", this.Location.Y.ToString());          //将窗体关闭位置的 y 坐标写入注册表
    }
    catch { }
}
```

秘笈心法

心法领悟 159：注册表的使用。

注册表的层次结构和文件系统非常类似，它记录了用户账号、服务器硬件以及应用程序的设置信息等。与 INI 文件相比，注册表可以控制的数据更多，而且不仅仅限于处理字符串类型的数据。注册表也包含一些系统配置的信息，这些信息根据操作系统的不同而不同。用户选择“开始”/“运行”命令，在弹出对话框的“打开”文本框中输入“regedit”后，单击“确定”按钮即可打开注册表编辑器。

实例 160　始终在桌面最顶层显示的窗体

光盘位置：光盘\MR\07\160　　初级　　趣味指数：★★★★

实例说明

Windows 桌面上允许多个窗体同时显示，但是只有一个窗体能够得到焦点，当该窗体得到焦点后在其上面的窗体会被得到焦点的窗体遮挡，得到焦点的窗体会显示在最上层，这样被覆盖的窗体就不能完全地显示给用户，如果该窗体中具有实时性和比较重要的信息时，需要该窗口始终在最上层。本实例就实现了此功能，运行本实例后，主窗体会始终显示在桌面的最顶层。实例运行效果如图 7.4 所示。

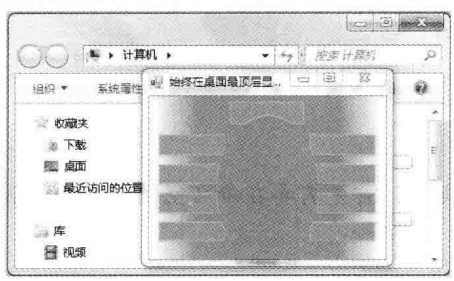


图 7.4　始终在桌面最顶层显示的窗体

说明：在图 7.4 中，“计算机”窗体获得了鼠标焦点，但是本实例的主窗体依然显示在最顶层。

关键技术

在其他开发语言中实现窗体始终在最顶层比较复杂，但在 C#中实现则非常简单，只要将窗体的 TopMost 属性设为 true 即可。下面对 TopMost 属性进行详细讲解。

Form 类的 TopMost 属性用来获取或设置一个值，指示该窗体是否应显示为最顶层窗体。语法格式如下：

```
public bool TopMost { get; set; }
```

参数说明

属性值：如果将窗体显示为最顶层窗体，则为 true；否则为 false。默认为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AlwaysActiveWindows。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加背景图片，并在“属性”窗口中设置窗体的 TopMost 属性为 true。

秘笈心法

心法领悟 160：“属性”窗口的使用。

“属性”窗口是 Visual Studio 2012 中一个重要的工具，该窗口中为 Windows 窗体应用程序的开发提供了简单的属性修改方式。“属性”窗口同时采用了两种方式管理属性和方法，分别为按分类方式和按字母顺序方式。读者可以根据自己的习惯采用不同的方式。另外，窗口的下方还有简单的帮助，方便开发人员对控件的属性进行操作和修改，“属性”面板的左侧是属性名称，对应的右侧是属性值。

7.2　设置窗体大小

实例 161	限制窗体大小	初级
	光盘位置：光盘\MR\07\161	趣味指数：★★★☆

实例说明

用户打开软件后，首先看到的就是软件窗体的大小，那么如何设置窗体的大小就成了摆在开发者面前的一个首要问题。本实例将告诉读者如何使用 C#语言实现限制窗体大小的功能，实例运行效果如图 7.5 所示。

图 7.5　限制窗体大小

关键技术

本实例在实现限制窗体大小的功能时，主要用到了窗体的 Width 属性、Height 属性和 FormBorderStyle 属性。下面对实例中用到的关键技术进行详细介绍。

（1）Width 属性

该属性用于获取或设置窗体的宽度。语法格式如下：

```
public int Width { get; set; }
```

参数说明

属性值：窗体的宽度（以像素为单位）。

（2）Height 属性

该属性用于获取或设置窗体的高度。语法格式如下：

```
public int Height { get; set; }
```

参数说明

属性值：窗体的高度（以像素为单位）。

（3）FormBorderStyle 属性

该属性用于获取或设置窗体的边框样式。语法格式如下：

```
public FormBorderStyle FormBorderStyle { get; set; }
```

参数说明

属性值：FormBorderStyle 枚举值之一，表示窗体的起始位置，默认为 FormBorderStyle.Sizable。FormBorderStyle 枚举值及说明如表 7.2 所示。

表 7.2 FormBorderStyle 枚举值及说明

枚 举 值	说 明
None	无边框
FixedSingle	固定的单行边框
Fixed3D	固定的三维边框
FixedDialog	固定的对话框样式的粗边框
Sizable	可调整大小的边框
FixedToolWindow	不可调整大小的工具窗口边框。工具窗口不会显示在任务栏中，也不会显示在当用户按 Alt+Tab 快捷键时出现的窗口中
SizableToolWindow	可调整大小的工具窗口边框。工具窗口不会显示在任务栏中，也不会显示在当用户按 Alt+Tab 快捷键时出现的窗口中

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ControlFormSize。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.Width = 300;                                    //设置窗体的宽度
    this.Height = 150;                                   //设置窗体的高度
    this.FormBorderStyle = FormBorderStyle.FixedDialog;  //设置窗体的边框样式
}
```

秘笈心法

心法领悟 161：注意 FormBorderStyle 属性的 FixedToolWindow 枚举值的使用。

尽管指定 FormBorderStyle 属性为 FixedToolWindow 枚举值的窗体通常不显示在任务栏中，还是必须确保窗体的 ShowInTaskbar 属性设置为 false，因为其默认值为 true。

实例 162 根据桌面大小调整窗体大小

光盘位置：光盘\MR\07\162　　初级　　趣味指数：★★★★

实例说明

窗体与桌面的大小比例是软件运行时用户经常会注意到的一个问题。例如，在 1634×768 的桌面上，如果放置一个很大（如 1280×1634）或者很小（如 10×10）的窗体，会显得非常不协调。正是基于以上情况，所以大部分软件的窗体界面都可以根据桌面的大小进行自动调整。本实例就实现了这样的功能，实例运行效果如图 7.6 所示。

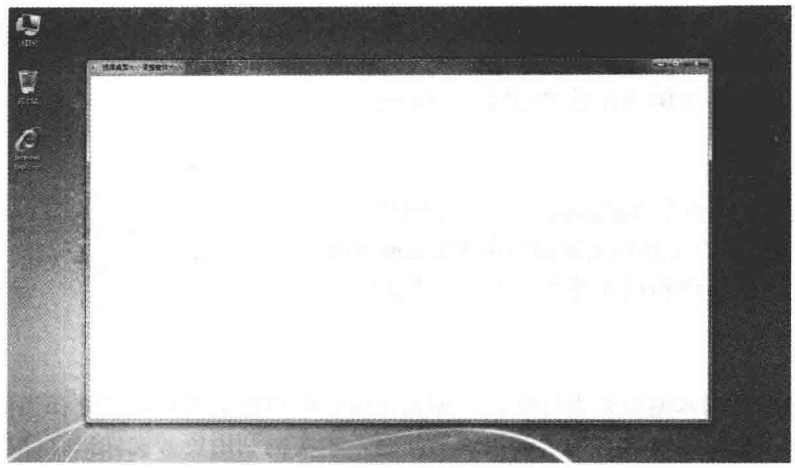


图 7.6　根据桌面大小调整窗体大小

关键技术

本实例实现的重点是如何获取桌面的大小，而获取桌面大小时，主要用到 Screen 类，下面对本实例中用到的关键技术进行详细介绍。

Screen 类表示单个系统上的一个或多个显示设备，其 PrimaryScreen 属性用来获取主显示，该属性返回一个 Screen 对象，而调用 Screen 对象的 WorkingArea 属性可以获取显示器的工作区。WorkingArea 属性的语法格式如下：

```
public Rectangle WorkingArea { get; }
```

参数说明

属性值：一个 Rectangle 结构，表示显示器的工作区。

说明：显示器的工作区是指显示器的桌面区域，不包括任务栏、停靠窗口和停靠工具栏等。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetFormSizeByDeskSize。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    int DeskWidth = Screen.PrimaryScreen.WorkingArea.Width;          //获取桌面宽度
    int DeskHeight = Screen.PrimaryScreen.WorkingArea.Height;        //获取桌面高度
    this.Width = Convert.ToInt32(DeskWidth * 0.8);                   //设置窗体宽度
    this.Height = Convert.ToInt32(DeskHeight * 0.8);                 //设置窗体高度
}
```

秘笈心法

心法领悟 162：有效使用窗体的 Load 事件。

Load 事件是在窗体加载时触发的一个事件，该事件中，一般用来实现默认资源、数据的加载等功能。

实例 163　自定义最大化、最小化和关闭按钮

光盘位置：光盘\MR\07\163　　中级　　趣味指数：★★★★★

实例说明

用户在制作应用程序时，为了使用户界面更加美观，一般都自己设计窗体的外观，以及窗体的最大化、最

小化和关闭按钮。本实例通过资源文件来存储窗体的外观，以及最大化、最小化和关闭按钮的图片，再通过鼠标移入、移出事件来实现按钮的动态效果。实例运行效果如图 7.7 所示。

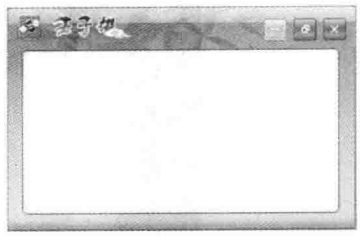


图 7.7　自定义最大化、最小化和关闭按钮

关键技术

本实例首先使用资源文件来存储窗体的外观、“最大化”、“最小化”和“关闭”按钮的图片，然后使用窗体的 WindowState 属性实现窗体的最大化、最小化和还原操作。下面对本实例中用到的关键技术进行详细说明。

（1）资源文件

在加载图片时，如果是以本地资源进行加载，那么在运行可执行文件时，必须在当前目录下存放指定的图片，这样会给用户带来不必要的麻烦。为了解决这一问题，本实例利用资源文件来存储相关的图片。下面对资源文件的应用进行详细说明。

❑ 将图片存入资源文件中

将图片存入资源文件中的步骤如下：

✧ 在“解决方案资源管理器”面板中打开 Properties 中的 Resources.resx 文件。

✧ 在“添加资源”下拉列表框中选择“添加现有文件”选项，弹出“将现有文件添加到资源中”窗体，在该窗体中将指定的图片添加到资源文件中。

❑ 读取资源文件中的图片

读取资源文件中的图片主要是用 Properties 命名空间下的 Resources 资源类。实现代码如下：

```
pictureBox.Image = Properties.Resources.最大化按钮;
```

注意：将图片添加到 Resources.resx 文件后，必须对整个项目进行保存，否则无法在 Resources 资源类中调用已添加的图片。

（2）WindowState 属性

该属性用于获取或设置窗体的窗口状态。语法格式如下：

```
public FormWindowState WindowState { get; set; }
```

参数说明

属性值：FormWindowState 枚举值之一，表示窗体的窗口状态。FormWindowState 枚举值及说明如表 7.3 所示。

表 7.3　FormWindowState 枚举值及说明

枚 举 值	描　述
Maximized	最大化的窗口
Minimized	最小化的窗口
Normal	默认大小的窗口

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ControlFormStatus。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Panel 控件，分别用来显示窗体标题栏和标题栏下面的窗体部分；添加 3 个 PictureBox 控件，分别用来表示“最大化”“最小化”和“关闭”按钮。

（3）程序的主要代码如下：

```
#region  设置窗体的最大化、最小化和关闭按钮的单击事件
/// <summary>
```

```
/// 设置窗体的“最大化”“最小化”和“关闭”按钮的单击事件
/// </summary>
/// <param Frm_Tem="Form">窗体</param>
/// <param n="int">标识</param>
public void FrmClickMeans(Form Frm_Tem, int n)
{
    switch (n)                                                    //窗体的操作样式
    {
        case 0:                                                   //窗体最小化
            Frm_Tem.WindowState = FormWindowState.Minimized;      //窗体最小化
            break;
        case 1:                                                   //窗体最大化和还原的切换
            {
                if (Frm_Tem.WindowState == FormWindowState.Maximized)   //如果窗体当前是最大化
                    Frm_Tem.WindowState = FormWindowState.Normal;       //还原窗体大小
                else
                    Frm_Tem.WindowState = FormWindowState.Maximized;    //窗体最大化
                break;
            }
        case 2:                                                   //关闭窗体
            Frm_Tem.Close();
            break;
    }
}
#endregion
```

秘笈心法

心法领悟 163：通过属性控制窗体的最大化和最小化。

Windows 窗体提供了“最大化”和“最小化”按钮，开发人员可以根据需要设置这两个按钮可用或不可用，该功能主要通过设置 Windows 窗体的 MaximizeBox 属性和 MinimizeBox 属性来实现，其中 MaximizeBox 属性用来设置窗体的“最大化”按钮是否可用，MinimizeBox 属性用来设置窗体的“最小化”按钮是否可用。

实例 164　手动改变自制窗体的大小

光盘位置：光盘\MR\07\164　　中级　　趣味指数：★★★★

实例说明

当用户去除窗体边框，自行设置窗体外观时，用户就不能对窗体的大小进行随意改变了。本实例将实现如何手动改变无边框窗体的大小，实例运行效果如图 7.8 所示。

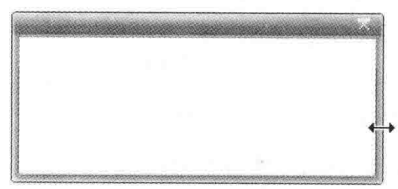


图 7.8　手动改变自制窗体的大小

关键技术

本实例在改变自制窗体的大小时，主要用到了 Cursor 类的 Position 属性以及窗体的 Width、Height、Top 和 Left 属性，另外，还用到了改变窗体大小的计算方法。下面对本实例中用到的关键技术进行详细介绍。

（1）Cursor 类的 Position 属性

Cursor 类代表用于绘制鼠标指针的图像，其 Position 属性用于获取或设置光标位置。该属性的语法格式如下：

```
public static Point Position { get; set; }
```

参数说明

属性值：代表光标位置的 Point（采用屏幕坐标）结构。

说明：① 关于窗体的 Width 属性和 Height 属性的详细讲解请参见实例 161 中的关键技术。

② 关于窗体的 Top 属性和 Left 属性的详细讲解请参见实例 158 中的关键技术。

（2）改变窗体大小的计算方法

本实例中最重要的部分就是如何根据鼠标的移动来设置窗体的大小，下面以改变窗体宽度来进行讲解。首先，用当前鼠标在屏幕上的 X 坐标值减去窗体左边距与屏幕左边距的距离，然后再加上鼠标与边框右端的距离，并将值设为当前窗体的宽度。实现代码如下：

```
Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
```

其中，Frm 为窗体名称，Pan 为显示窗体边框的控件名称，Example_X 为鼠标按下时的 X 坐标值。

说明：以上计算方法是在 Panel 控件的 MouseMove 事件中执行的。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChangeFormSize。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 11 个 Panel 控件，分别用来表示窗体标题栏和底边框的各个部分；添加一个 PictureBox 控件，用来作为“关闭”按钮。

（3）程序的主要代码如下：

```
/// <summary>
/// 改变窗体的大小（用于鼠标的移动事件）
/// </summary>
/// <param Frm="Form">窗体</param>
/// <param Pan="Panel">设置窗体边框的控件</param>
/// <param e="MouseEventArgs">窗体上有关鼠标的一些信息</param>
public void FrmScreen_EnlargeSize(Form Frm, Panel Pan, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        switch (Pan.Name)
        {
            case "panel_Right":                                          //如果移动的是窗体的右边框
                {
                    if (this.Width <= 70)                                //如果窗体的宽度小于等于 70
                    {
                        Frm.Width = 70;                                  //设置窗体的宽度为 70
                        //如果用鼠标向右移动窗体的右边框
                        if (Cursor.Position.X - Frm.Left + (Pan.Width - Example_X) > Frm.Width)
                        {
                            //根据鼠标的移动值，增加窗体的宽度
                            Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
                        }
                        break;
                    }
                    //根据鼠标的移动值，增加窗体的宽度
                    Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
                    break;
                }
            case "panel_BR":                                             //如果移动的是窗体的右下角
                {
                    //如果窗体的大小不为窗体大小的最小值
                    if (this.Width > 70 && this.Height > (panel_Title.Height + panel_Bn.Height + 1))
                    {
                        //根据鼠标的移动改变窗体的大小
                        Frm.Height = Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y);
                        Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
                    }
                    else
                    {
                        if (this.Width <= 70)                            //如果窗体的宽度小于等于 70
                        {
                            Frm.Width = 70;                              //设置窗体的宽度为 70
                            //如果窗体的高度小于最小值
                            if (this.Height <= (panel_Title.Height + panel_Bn.Height + 1))
                            {
                                //设置窗体的最小高度
```

```
                        Frm.Height = panel_Title.Height + panel_Bn.Height + 1;
                        //如果用鼠标向下移动窗体的底边框
                        if (Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y) > Frm.Height)
                        {
                            //根据鼠标的移动值，增加窗体的高度
                            Frm.Height = Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y);
                        }
                        break;
                    }
                    //如果用鼠标向右移动窗体
                    if (Cursor.Position.X - Frm.Left + (Pan.Width - Example_X) > Frm.Width)
                    {
                        //增加窗体的宽度
                        Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
                    }
                    break;
                }
                //如果窗体的高度小于等于最小值
                if (this.Height <= (panel_Title.Height + panel_Bn.Height + 1))
                {
                    //设置窗体的高度为最小值
                    Frm.Height = panel_Title.Height + panel_Bn.Height + 1;
                    //改变窗体的宽度
                    Frm.Width = Cursor.Position.X - Frm.Left + (Pan.Width - Example_X);
                    //如果用鼠标向下移动窗体的边框
                    if (Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y) > Frm.Height)
                    {
                        //增加窗体的高度
                        Frm.Height = Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y);
                    }
                    break;
                }
            }
            break;
        }
        case "panel_Bn":                                    //如果移动的是窗体的底边框
        {
            //如果窗体的高度小于等于最小值
            if (this.Height <= (panel_Title.Height + panel_Bn.Height + 1))
                {
                //设置窗体的高度为最小值
                Frm.Height = panel_Title.Height + panel_Bn.Height + 1;
                //如果用鼠标向下移动窗体的下边框
                if (Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y) > Frm.Height)
                {
                    //增加窗体的高度
                    Frm.Height = Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y);
                }
                break;
                }
                //增加窗体的高度
            Frm.Height = Cursor.Position.Y - Frm.Top + (Pan.Height - Example_Y);
            break;
        }
    }
  }
}
```

秘笈心法

心法领悟 164：如何快速修改窗体的大小？

在窗体的属性中，通过 Size 属性可以设置窗体的大小。双击窗体属性面板中的 Size 属性，可以看到其下拉菜单中有 Width 和 Height 两个属性，分别用于设置窗体的宽和高。修改窗体的大小，只需更改 Width 和 Height 属性的值即可。

实例 165 禁止改变窗体的大小

光盘位置：光盘\MR\07\165　　初级　　趣味指数：★★★★☆

实例说明

本实例主要实现禁止改变窗体大小的功能。运行本实例，默认可以通过鼠标拖拽的方式改变窗体大小，但是当用户单击“禁止改变窗体的大小”按钮后，窗体将会以一种对话框的方式进行显示，这时就不可以再用鼠标拖拽的方式改变窗体的大小了。实例运行效果如图 7.9 所示。

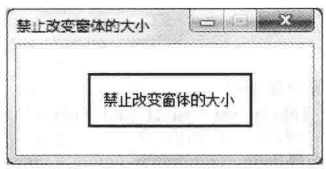


图 7.9　禁止改变窗体的大小

关键技术

本实例在实现禁止改变窗体的大小功能时，主要通过将窗体的 FormBorderStyle 属性设置为 FormBorderStyle.FixedDialog 枚举值来实现。

说明：关于窗体的 FormBorderStyle 属性的详细讲解请参见实例 161 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NChangeFormSize。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行禁止改变窗体大小的功能。

（3）程序的主要代码如下：

```
private void btn_Stop_Click(object sender, EventArgs e)
{
    this.MaximizeBox = false;                          //禁用“最大化”按钮
    this.FormBorderStyle = FormBorderStyle.FixedDialog;  //设置窗体边框样式为对话框样式
}
```

秘笈心法

心法领悟 165：将窗体以对话框形式显示。

使用 ShowDialog 方法可以将窗体以对话框形式进行显示。在显示时，如果作为激活窗体，则其他窗体不可用，只有在将模式窗体关闭之后，其他窗体才能恢复可用状态。

7.3　设置窗体的标题栏

实例 166 使窗体标题栏文字右对齐

光盘位置：光盘\MR\07\166　　初级　　趣味指数：★★★★☆

实例说明

窗体标题栏中的文字是窗体的重要说明，该文字可以体现出窗体的功能、状态或名称等信息，一般该文字

是居左显示的。在本实例中将设计一个标题栏文字右对齐的窗体，实例运行效果如图 7.10 所示。

图 7.10　使窗体标题栏文字右对齐

关键技术

本实例主要通过设置窗体的 RightToLeft 属性来实现，下面对其进行详细讲解。

RightToLeft 属性用来获取或设置一个值，该值指示是否将控件的元素对齐，以支持使用从右向左的字体的区域设置。语法格式如下：

```
public virtual RightToLeft RightToLeft { get; set; }
```

参数说明

属性值：RightToLeft 枚举值之一，默认为 Inherit。RightToLeft 枚举值及说明如表 7.4 所示。

表 7.4　RightToLeft 枚举值及说明

枚 举 值	说　明
No	文本从左至右读取，这是默认选项
Yes	文本从右到左读取
Inherit	文本的读取方向从父控件继承

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WindowsTitleRight。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加背景图片，并设置 BackgroundImageLayout 属性为 Stretch，以便使背景图片能够拉伸显示。

（3）设置 Form1 窗体的 RightToLeft 属性为 Yes。

秘笈心法

心法领悟 166：如何设置窗体的背景？

为使窗体设计更加美观，通常会设置窗体的背景，开发人员可以设置窗体的背景颜色，也可以设置窗体的背景图片。设置窗体的背景图片时可以通过设置窗体的 BackgroundImage 属性实现。

实例 167　没有标题栏也可以更改窗体的大小

初级

光盘位置：光盘\MR\07\167

趣味指数：★★★★

实例说明

隐藏 Windows 窗体的标题栏之后，窗体只剩下一个客户区域，有点像 Panel 控件在窗口中的样子，而这样的窗体通常是不能够改变大小的，因为屏蔽其标题栏之后，窗体默认将边框也去除了。本实例将用特殊的方法建立一个没有标题栏但可以改变其大小的窗体，实例运行效果如图 7.11 所示。

图 7.11　没有标题栏也可以更改窗体的大小

关键技术

窗体的样式是在窗体建立时确定的，在 C#中实现没有标题栏但是可以改变大小的窗体，有一个巧妙的方法就是将窗体的 Text 属性设为空，同时将 ControlBox 属性设为 false。下面

对本实例中用到的关键技术进行详细讲解。

ControlBox 属性用来获取或设置一个值，该值指示在该窗体的标题栏中是否显示控件框。语法格式如下：

```
public bool ControlBox { get; set; }
```

参数说明

属性值：如果在该窗体的左上角显示控件框，则为 true；否则为 false。默认为 true。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EditFormSize。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行退出程序功能。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "";                          //设置标题栏文本为空
    ControlBox = false;                      //不在窗体标题栏中显示控件
}
private void button1_Click(object sender, EventArgs e)
{
    this.Close();                            //关闭当前窗体
}
```

秘笈心法

心法领悟 167：如何隐藏窗体的标题栏？

开发人员可以通过设置窗体的 FormBorderStyle 属性为 None 来实现隐藏窗体标题栏功能。

实例 168　设置闪烁的标题栏

光盘位置：光盘\MR\07\168　　初级　　趣味指数：★★★★

实例说明

在 Windows 系统中，当程序在后台运行时，如果某个窗体的提示信息需要用户浏览，该窗体就会不停地闪烁，这样就会吸引用户的注意。同样，如果在自己的程序中使某个窗体不停地闪烁也会吸引用户的注意。本实例设计了一个闪烁的窗体标题栏，运行本实例，单击“开始闪烁”按钮，窗体标题栏就会不停地闪烁；单击“停止闪烁”按钮，窗体标题栏就会停止闪烁。实例运行效果如图 7.12 所示。

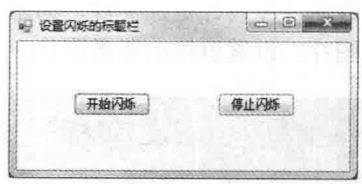


图 7.12　设置闪烁的标题栏

关键技术

本实例实现时主要用到了 API 函数 FlashWindow，下面对其进行详细讲解。

API 函数 FlashWindow 可以使窗体闪烁一下，语法格式如下：

```
[System.Runtime.InteropServices.DllImportAttribute("user32.dll")]
public static extern bool FlashWindow(IntPtr handle, bool bInvert);
```

参数说明

❶ handle：表示将要闪烁的窗体。

❷ bInvert：是否恢复状态。

说明：（1）重写 API 函数时，首先需要在命名空间区域中添加 System.Runtime.InteropServices 命名空间。

（2）利用 FlashWindow 函数只能使窗体闪烁一下，如果要让窗体不停地闪烁，就需要使用一个 Timer 控件每隔一段时间调用该函数使窗体闪烁。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FlashWindowBar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来使窗体开始闪烁和停止闪烁。

（3）程序的主要代码如下：

```
//重写 API 函数，用来实现窗体标题栏的闪烁功能
[System.Runtime.InteropServices.DllImportAttribute("user32.dll")]
public static extern bool FlashWindow(IntPtr handle, bool bInvert);
private void timer1_Tick(object sender, EventArgs e)
{
    FlashWindow(this.Handle,true);                    //启用窗体闪烁函数
}
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;                            //启动计时器
}
private void button2_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;                           //关闭计时器
}
```

秘笈心法

心法领悟 168：启动和关闭 Timer 计时器的两种方法。

启动 Timer 计时器时，可以将其 Enabled 属性设置为 true，或者调用其 Start 方法；而关闭 Timer 计时器时，则需要将其 Enabled 属性设置为 false，或者调用其 Stop 方法。

7.4　设置窗体的背景

实例 169　设置窗体背景颜色为淡蓝色　　初级

光盘位置：光盘\MR\07\169　　趣味指数：★★★★☆

实例说明

开发程序时，Windows 窗体的背景颜色默认为灰色，为了能够使窗体看起来更加美观，可以通过编码为 Windows 窗体设置更好看的颜色。本实例将 Windows 窗体的背景颜色设置成了淡蓝色，实例运行效果如图 7.13 所示。

图 7.13　设置窗体背景颜色为淡蓝色

关键技术

本实例在设置窗体的背景颜色时，主要用到了窗体的 BackColor 属性，下面对其进行详细讲解。

BackColor 属性用来获取或设置窗体的背景色。语法格式如下：

```
public override Color BackColor { get; set; }
```

参数说明

属性值：表示控件背景色的 Color 颜色。默认为 DefaultBackColor 属性的值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetFormBackColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackColor = Color.LightBlue;                    //设置窗体背景颜色
}
```

秘笈心法

心法领悟 169：通过“属性”窗口更快地设置窗体背景色。

在设置窗体的背景时，可以直接在“属性”窗口中进行设置。步骤是，选中窗体，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出“属性”窗口，然后找到 BackColor 属性，单击即可设置指定的背景颜色。

实例 170　设置窗体背景为指定图片

初级

光盘位置：光盘\MR\07\170

趣味指数：★★★★

实例说明

开发 Windows 窗体应用程序时，界面的美观是程序的一个重要组成部分，一般的应用程序界面背景都是非常漂亮或者代表实际意义的图片，那么如何为窗体设置背景图片呢？本实例将通过 C#代码为窗体背景设置为指定的图片，实例运行效果如图 7.14 所示。

图 7.14　设置窗体背景为指定图片

关键技术

本实例在设置窗体的背景图片时，主要用到了窗体的 BackgroundImage 属性，下面对其进行详细讲解。

BackgroundImage 属性用来获取或设置在窗体中显示的背景图像。语法格式如下：

```
public virtual Image BackgroundImage { get; set; }
```

参数说明

属性值：一个 Image 图像，它表示在窗体的背景中显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetFormBackImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("test.jpg");          //设置窗体的背景图片
}
```

提示：上面代码中的 test.jpg 图片需要放置在程序的 Debug 文件夹中，以便使代码能够自动识别。

秘笈心法

心法领悟 170：通过“属性”窗口更快地设置窗体背景图片。

在设置窗体的背景图片时，可以直接在“属性”窗口中进行设置。步骤是，选中窗体，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出“属性”窗口，然后找到 BackgroundImage 属性，单击即可设置指定的背景图片。

实例 171　使背景图片自动适应窗体的大小

光盘位置：光盘\MR\07\171　　初级　　趣味指数：★★★★

实例说明

开发人员在开发 Windows 窗体应用程序时，有时为一个窗体设置了背景图片，但是由于图片的大小与窗体的大小并不一定相同，所以就可能导致图片显示不全，那么如何来避免这种情况的发生呢？本实例将通过编写 C#代码来使背景图片能够自动适应窗体的大小，实例运行效果如图 7.15 所示。

图 7.15　使背景图片自动适应窗体的大小

关键技术

本实例在实现使背景图片自动适应窗体的大小功能时，主要用到了窗体的 BackgroundImage 和 BackgroundImageLayout 属性。下面对本实例中用到的关键技术进行详细讲解。

BackgroundImageLayout 属性用来获取或设置在 ImageLayout 枚举中定义的背景图像布局。语法格式如下：

```
public virtual ImageLayout BackgroundImageLayout { get; set; }
```

参数说明

属性值：ImageLayout 枚举值之一。

ImageLayout 枚举值及说明如表 7.5 所示。

表 7.5　ImageLayout 枚举值及说明

枚 举 值	说　明
None	图像沿控件的矩形工作区顶部左对齐
Tile	图像沿控件的矩形工作区平铺
Center	图像在控件的矩形工作区中居中显示
Stretch	图像沿控件的矩形工作区拉伸
Zoom	图像在控件的矩形工作区中放大

说明：（1）关于 BackgroundImage 属性的详细讲解请参见实例 170 中的关键技术。

（2）本实例中需要将窗体的 BackgroundImageLayout 属性设置为 ImageLayout.Stretch 枚举值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AutoImageSizeByForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("test.jpg");                    //设置窗体的背景图片
    this.BackgroundImageLayout = ImageLayout.Stretch;                     //使图片自动适应窗体大小
}
```

秘笈心法

心法领悟 171：自动识别图片或者文件路径。

程序中用到图片或者文件时，可以将其放置在程序的 Debug 文件夹中，这样在程序代码中可以直接通过编写图片或者文件名来使代码自动识别图片或者文件的路径。

实例 172	使背景图片在窗体中居中显示 光盘位置：光盘\MR\07\172	初级 趣味指数：★★★★

实例说明

在为 Windows 操作系统设置桌面图片时，通常都会看到“居中”“平铺”和“拉伸”等多种方式，那么，能不能用 C#代码来实现类似的多种设置图片的方式呢？答案是肯定的。本实例就将带领读者一起学习如何将背景图片在窗体中居中显示，实例运行效果如图 7.16 所示。

图 7.16　使背景图片在窗体中居中显示

关键技术

本实例在实现使背景图片居中显示时，首先需要通过窗体的 BackgroundImage 属性设置图片，然后将窗体的 BackgroundImageLayout 属性设置为 ImageLayout.Center 枚举值。

说明：（1）关于 BackgroundImage 属性的详细讲解请参见实例 170 中的关键技术。

（2）关于 BackgroundImageLayout 属性的详细讲解请参见实例 171 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImageInFormCenter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("test.jpg");                    //设置窗体的背景图片
    this.BackgroundImageLayout = ImageLayout.Center;                      //使图片居中显示
}
```

秘笈心法

心法领悟 172：通过“属性”窗口更快地设置窗体背景图片。

在设置窗体的背景图片时，可以直接在“属性”窗口中进行设置。步骤是，选中窗体，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出“属性”窗口，然后找到 BackgroundImage 属性，单击即可设置指定的背景图片。

实例 173　使背景图片在窗体中平铺显示

光盘位置：光盘\MR\07\173　　初级　　趣味指数：★★★★

实例说明

本实例演示如何使用 C#代码使背景图片在窗体中平铺显示，实例运行效果如图 7.17 所示。

图 7.17　使背景图片在窗体中平铺显示

关键技术

本实例在实现使背景图片居中显示时，首先需要通过窗体的 BackgroundImage 属性设置图片，然后将窗体的 BackgroundImageLayout 属性设置为 ImageLayout.Tile 枚举值。

说明：（1）关于 BackgroundImage 属性的详细讲解请参见实例 170 中的关键技术。

（2）关于 BackgroundImageLayout 属性的详细讲解请参见实例 171 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExpandImageInForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("test.jpg");                    //设置窗体的背景图片
    this.BackgroundImageLayout = ImageLayout.Tile;                        //使图片平铺显示
}
```

秘笈心法

心法领悟 173：自动识别图片或者文件路径。

程序中用到图片或者文件时，可以将其放置在程序的 Debug 文件夹中，这样在程序代码中可以直接通过编写图片或者文件名来使代码自动识别图片或者文件的路径。

实例 174	清空窗体的背景 光盘位置：光盘\MR\07\174	初级 趣味指数：★★★★

实例说明

在数据库管理系统中，如果需要绘制统计分析图表，常常需要以一种指定的颜色清空窗体中的背景，然后再在上面绘制统计分析图表，那么如何来清空窗体的背景呢？本实例就使用 C#代码实现了清空窗体背景的功能。运行本实例，运行效果如图 7.18 所示。单击“清空背景”按钮，即可以白色清空窗体中的背景图片，如图 7.19 所示。

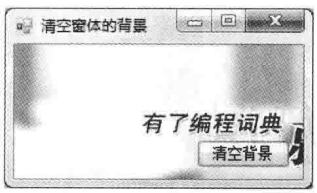


图 7.18　清空背景前

图 7.19　清空背景后

关键技术

本实例在实现清空窗体背景功能时，主要用到了 Graphics 对象的 Clear 方法，下面对本实例中用到的关键技术进行详细讲解。

Graphics 位于 System.Drawing 命名空间下，它封装一个 GDI+绘图图面，其 Clear 方法用来清除整个绘图面并以指定背景色填充。Clear 方法的语法格式如下：

```
public void Clear(Color color)
```

参数说明

color：Color 结构，表示绘图面的背景色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ClearFormBack。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行清空窗体背景操作。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.BackgroundImage = Image.FromFile("test.jpg");                    //设置窗体的背景图片
}
private void btn_Clear_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                            //创建绘图对象
    graphics.Clear(Color.White);                                          //清空背景
    graphics.Dispose();                                                   //释放绘图资源
}
```

秘笈心法

心法领悟 174：什么是 GDI+?

GDI+是 GDI（即 Windows 早期版本中附带的 Graphics Device Interface）的后继者，它是一种构成 Windows XP 操作系统的子系统的应用程序编程接口（API），它有 3 种基本类型的绘图界面，分别为 Windows 窗体上的控件、要发送给打印机的页面和内存中的位图图像。

第 8 章

特色窗体界面

- 导航菜单界面
- 特殊形状的窗体
- 窗体效果
- 窗体动画
- 特色程序界面

8.1 导航菜单界面

实例 175	创建类似 OutLook 的窗体 光盘位置：光盘\MR\08\175	初级 趣味指数：★★★★

实例说明

程序主界面包括菜单栏、工具栏、状态栏和树状视图等，而 OutLook 界面美观、友好，是一个很实用的程序主界面，并且菜单栏和工具栏是可移动的。本实例将演示如何创建一个类似 OutLook 的窗体。实例运行效果如图 8.1 所示。

图 8.1 创建类似 OutLook 的窗体

关键技术

本实例实现时主要用到了 MenuStrip 控件和 ToolStrip 控件的 AllowItemRecorder 属性，下面分别对它们进行详细讲解。

（1）MenuStrip 控件的 AllowItemRecorder 属性

该属性用来获取或设置一个值，该值指示是否由 MenuStrip 类独自处理拖放和项重新排序。语法格式如下：

```
public bool AllowItemReorder { get; set; }
```

参数说明

属性值：如果让 MenuStrip 类自动处理拖放和项重新排序，则为 true；否则为 false。默认值为 false。

（2）ToolStrip 控件的 AllowItemRecorder 属性

该属性用来获取或设置一个值，该值指示是否由 ToolStrip 类独自处理拖放和项重新排序。语法格式如下：

```
public bool AllowItemReorder { get; set; }
```

参数说明

属性值：如果让 ToolStrip 类自动处理拖放和项重新排序，则为 true；否则为 false。默认值为 false。

注意：在移动菜单栏和工具栏时，需要按住 Alt 键，同时用鼠标进行拖动。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OutLookForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计主菜单；添加一个 ToolStrip 控件，用来设计工具栏；添加一个 StatusStrip 控件，用来设计状态栏；添加一个 ImageList 控件和一个 TreeView 控件，用来设计树结构。

（3）分别为 MenuStrip、ToolStrip、ImageList 和 TreeView 控件添加子项，将 MenuStrip 和 ToolStrip 控件的 AllowItemRecorder 属性设为 true，并将 ToolStrip 控件的每个子项的 DisplayStyle 属性设置为 ImageAndText。

秘笈心法

心法领悟 175：MenuStrip 控件的使用。

MenuStrip 控件主要用来设计程序的菜单栏，C#中的 MenuStrip 控件支持多文档界面、菜单合并、工具提示和溢出等功能，开发人员可以通过添加访问键、快捷键、选中标记、图像和分隔条来增强菜单的可用性和可读性。

实例 176	带导航菜单的主界面 光盘位置：光盘\MR\08\176	初级 趣味指数：★★★★

实例说明

在窗体界面中，菜单栏是不可缺少的重要组成部分。本实例使用 Windows 标准控件来制作一个模拟菜单栏。运行程序，单击窗体上面的按钮，将会在按钮的下面显示一个下拉列表。实例运行效果如图 8.2 所示。

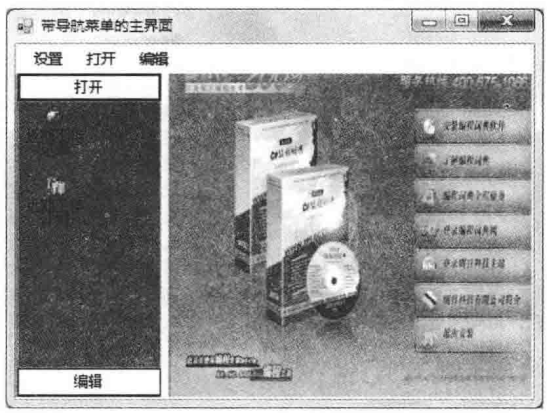


图 8.2　带导航菜单的主界面

关键技术

本实例实现时主要用到了 Button 控件的 SendToBack 方法和 ListView 控件的 Items.Add 方法，下面分别对它们进行详细讲解。

（1）Button 控件的 SendToBack 方法

该方法将控件发送到 Z 顺序的后面。语法格式如下：

```
public void SendToBack()
```

（2）ListView 控件的 Items.Add 方法

ListView 控件的 Items 属性用来获取包含控件中所有项的集合，其 Add 方法用来将项添加到 ListView 列表中。Add 方法是一个重载方法，其最常用的语法格式如下：

```
public virtual ListViewItem Add(string key,string text,int imageIndex)
```

Add 方法中的参数说明如表 8.1 所示。

表 8.1　Add 方法中的参数说明

参　数	说　明	参　数	说　明
key	项的名称	imageIndex	要为该项显示的图像的索引
text	项的文本	返回值	添加到集合中的 ListViewItem

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NavigationForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计菜单栏；添加一个 SplitContainer 控件，用来将窗体分为两部分；添加一个 ListView 控件和 3 个 Button 控件，并将它们加入 SqlitContainer1 的左侧部分，用来制作窗体左侧的导航栏；添加一个 ImageList 组件，用来为 ListView

列表项提供图标。

（3）程序的主要代码如下：

```
private void button1_Click_1(object sender, EventArgs e)
{
    listView1.Dock = DockStyle.None;                                    //设置 listView1 的绑定属性为未绑定
    button1.Dock = DockStyle.Top;                                       //设置 button1 的绑定属性为上端绑定
    button2.Dock = DockStyle.Bottom;                                    //设置 button2 的绑定属性为底端绑定
    button3.SendToBack();                                               //将 button3 发送到 Z 顺序的后面
    button3.Dock = DockStyle.Bottom;                                    //设置 button3 的绑定属性为底端绑定
    listView1.BringToFront();                                           //将 listView1 带到 Z 顺序的前面
    listView1.Dock = DockStyle.Bottom;                                  //设置 listView1 的绑定属性为底端绑定
    listView1.Clear();                                                  //清空 listView1 中的原有内容
    listView1.Items.Add("设置上下班时间", "设置上下班时间", 0);          //向 listView1 中添加“设置上下班时间”
    listView1.Items.Add("是否启用短信提醒", "是否启用短信提醒", 1);      //向 listView1 中添加“是否启用短信提醒”
    listView1.Items.Add("设置密码", "设置密码", 2);                     //向 listView1 中添加“设置密码”
}
private void button2_Click_1(object sender, EventArgs e)
{
    listView1.Dock = DockStyle.None;                                    //设置 listView1 的绑定属性为未绑定
    button2.Dock = DockStyle.Top;                                       //设置 button2 的绑定属性为上端绑定
    button1.SendToBack();                                               //将控件 button1 发送到 Z 顺序的后面
    button1.Dock = DockStyle.Top;                                       //设置 button1 的绑定属性为上端绑定
    button3.Dock = DockStyle.Bottom;                                    //设置 button3 的绑定属性为底端绑定
    listView1.Dock = DockStyle.Bottom;                                  //设置 listView1 的绑定属性为底端绑定
    listView1.Clear();                                                  //清空 listView1 中的原有内容
    listView1.Items.Add("近期工作记录", "近期工作记录", 3);             //向 listView1 中添加“近期工作记录”
    listView1.Items.Add("近期工作计划", "近期工作计划", 4);             //向 listView1 中添加“近期工作计划”
}
private void button3_Click_1(object sender, EventArgs e)
{
    listView1.Dock = DockStyle.None;                                    //设置 listView1 的绑定属性为未绑定
    button3.SendToBack();                                               //将 button3 发送到 Z 顺序的后面
    button3.Dock = DockStyle.Top;                                       //设置 button3 的绑定属性为上端绑定
    button2.SendToBack();                                               //将 button2 发送到 Z 顺序的后面
    button2.Dock = DockStyle.Top;                                       //设置 button2 的绑定属性为上端绑定
    button1.SendToBack();                                               //将 button1 发送到 Z 顺序的后面
    button1.Dock = DockStyle.Top;                                       //设置 button1 的绑定属性为上端绑定
    listView1.Dock = DockStyle.Bottom;                                  //设置 listView1 的绑定属性为底端绑定
    listView1.Clear();                                                  //清空 listView1 中的原有内容
    listView1.Items.Add("编辑工作进度报告", "编辑工作进度报告", 5);     //向 listView1 中添加“编辑工作进度报告”
    listView1.Items.Add("编辑项目设计图", "编辑项目设计图", 6);         //向 listView1 中添加“编辑项目设计图”
}
```

秘笈心法

心法领悟 176：有效利用 ListView 控件。

ListView 是一种列表控件，在实现诸如显示文件详细信息这样的功能时，推荐使用该控件；另外，由于 ListView 有多种显示样式，因此在实现类似 Windows 系统的“缩略图”“平铺”“图标”“列表”和“详细信息”等功能时，经常需要使用 ListView 控件。

实例 177　图形化的导航界面

初级

光盘位置：光盘\MR\08\177

趣味指数：★★★★☆

实例说明

图形化的导航界面，顾名思义就是在窗体中使用图形导航代替传统的文字导航。与传统的文字导航界面相比，图形化导航界面的优势在于可以使得窗体的界面更加美观、吸引人。本实例使用 C#制作了一个图形化的导航界面。实例运行效果如图 8.3 所示。

图 8.3　图形化的导航界面

关键技术

本实例主要用到了 Button 控件的 BackColor、FlatStyle 和 TextImageRelation 属性，下面分别对它们进行详细讲解。

（1）BackColor 属性

该属性主要用来获取或设置控件的背景色。语法格式如下：

```
public override Color BackColor { get; set; }
```

参数说明

属性值：一个表示背景色的 Color 值。

（2）FlatStyle 属性

该属性主要用来获取或设置按钮控件的平面样式外观。语法格式如下：

```
public FlatStyle FlatStyle { get; set; }
```

参数说明

属性值：FlatStyle 值之一。默认值为 Standard。

（3）TextImageRelation 属性

该属性主要用来获取或设置文本和图像相互之间的相对位置。语法格式如下：

```
public TextImageRelation TextImageRelation { get; set; }
```

参数说明

属性值：TextImageRelation 的值之一。默认值为 Overlay。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImageNavigationForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计菜单栏；添加一个 ToolStrip 控件，用来设计工具栏；添加两个 Panel 控件，用来将窗体分割成两部分；添加 12 个 Button 控件，将它们的 BackColor 属性设置为 Transparent、FlatStyle 属性设置为 Flat、TextImageRelation 属性设置为 ImageBeforeText，这些 Button 控件用来设计图形化的导航按钮。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    button5.Visible = true;                    //设置 button5 控件可见
    button6.Visible = true;                    //设置 button6 控件可见
    button7.Visible = true;                    //设置 button7 控件可见
}
private void button2_Click(object sender, EventArgs e)
{
```

```
    button8.Visible = true;                          //设置 button8 控件可见
    button9.Visible = true;                          //设置 button9 控件可见
    button10.Visible = true;                         //设置 button10 控件可见
}
private void button3_Click(object sender, EventArgs e)
{
    button11.Visible = true;                         //设置 button11 控件可见
    button12.Visible = true;                         //设置 button12 控件可见
    button13.Visible = true;                         //设置 button13 控件可见
}
```

秘笈心法

心法领悟 177：ToolStrip 控件的使用。

ToolStrip 控件主要用来设计窗体的工具栏，使用该控件可以创建具有 Windows XP、Office、Internet Explorer 或自定义的外观和行为的工具栏及其他用户界面元素，这些元素支持溢出及运行时项重新排序。

8.2 特殊形状的窗体

实例 178	创建非矩形窗体	初级
	光盘位置：光盘\MR\08\178	趣味指数：★★★★☆

实例说明

大部分 Windows 窗体都是一个矩形区域，读者是否已经厌倦了这种常规的矩形窗体呢？本实例中的窗体是一个打破传统矩形的窗体，运行本实例，会看到一个非常漂亮的非矩形窗体。实例运行效果如图 8.4 所示。

图 8.4 创建非矩形窗体

关键技术

本实例在创建非矩形窗体时，主要是通过重写窗体 OnPaint 方法，并在其中对窗体进行重绘，然后使用透明色将窗体设置为透明来实现的。下面对本实例中用到的关键技术进行详细讲解。

（1）OnPaint 方法

该方法主要用来重新绘制窗体图像。语法格式如下：

```
protected override void OnPaint(PaintEventArgs e)
```

参数说明

PaintEventArgs：为 Paint 事件提供数据。

（2）将窗体设置为透明

将窗体设置为透明时用到了 Bitmap 类的 MakeTransparent 方法，该方法主要用来使指定的颜色对 Bitmap 位图透明。语法格式如下：

```
public void MakeTransparent(Color transparentColor)
```

参数说明

transparentColor：Color 结构，它表示要使之透明的颜色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RectangleForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将该窗体的 FormBorderStyle 属性设置为 None。在 Frm_Main 窗体中添加一个 Label 控件，用来执行关闭窗体操作。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    bit = new Bitmap("bccd.png");                    //从指定的图像初始化 Bitmap 对象
    bit.MakeTransparent(Color.Blue);                 //使用默认的透明颜色对 Bitmap 位图透明
}
protected override void OnPaint(PaintEventArgs e)
{
    e.Graphics.DrawImage((Image)bit, new Point(0, 0));   //在窗体上绘制图片
}
```

秘笈心法

心法领悟 178：OnPaint 方法的使用。

OnPaint 是窗体的一个重绘方法，它对应窗体的 Paint 事件，主要在窗体进行重绘时触发，一般在一些需要绘图的应用中需要使用该方法。

实例 179　建立字体形状窗体

光盘位置：光盘\MR\08\179　　初级　　趣味指数：★★★★☆

实例说明

字体形状窗体一般应用在屏幕提示中，如收款机屏幕等，那么如何制作一个字体形状窗体呢？本实例将使用 C#语言制作一个字体形状窗体。实例运行效果如图 8.5 所示。

图 8.5　建立字体形状窗体

关键技术

本实例在建立字体形状窗体时，首先需要将字体画在一幅图上，然后重写窗体的 OnPaint 方法，并在该方法中对窗体进行重绘，最后使用透明色将窗体设置为透明即可。

说明：（1）关于 OnPaint 方法的详细讲解请参见实例 178 中的关键技术。

（2）关于如何将窗体设置为透明的详细讲解请参见实例 178 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FontForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将该窗体的 FormBorderStyle 属性设置为 None。在 Frm_Main 窗体中添加一个 Label 控件，用来执行关闭窗体操作。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    bit = new Bitmap("font.bmp");                    //从指定的图像创建 Bitmap 对象
    bit.MakeTransparent(Color.Blue);                 //使用默认的透明颜色对 Bitmap 位图透明
}
protected override void OnPaint(PaintEventArgs e)
{
    e.Graphics.DrawImage((Image)bit, new Point(0, 0));   //在指定位置按指定大小绘制图片的指定部分
}
```

秘笈心法

心法领悟 179：创建字体形状窗体的另一种方法。

本实例中通过调用带有文字的图片实现了字体形状窗体，另外，还可以通过调用 Graphics 类的 DrawString

方法在透明的窗体上绘制文字来实现。

实例 180 使控件大小随窗体自动调整

光盘位置：光盘\MR\08\180　　初级　　趣味指数：★★★☆

实例说明

在软件开发中，随着窗体大小的变化，界面会和设计时出现较大的差异，这样控件和窗体的大小会不成比例，从而出现非常不美观的界面。本实例将演示如何使控件的大小能够随着窗体的变化而自动调整。实例运行效果如图 8.6 所示。

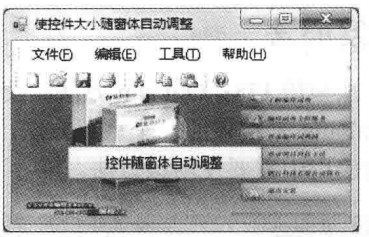


图 8.6　使控件大小随窗体自动调整

关键技术

本实例实现时主要用到了控件的 Anchor 属性，下面对其进行详细讲解。

Anchor 属性用来获取或设置控件绑定到容器的边缘，并确定控件如何随其父级一起调整大小。语法格式如下：

```
public virtual AnchorStyles Anchor { get; set; }
```

参数说明

属性值：AnchorStyles 枚举值的按位组合。默认值是 Top 和 Left。AnchorStyles 枚举值及说明如表 8.2 所示。

表 8.2　AnchorStyles 枚举值及说明

枚 举 值	说　明	枚 举 值	说　明
Top	控件锚定到其容器的上边缘	Right	控件锚定到其容器的右边缘
Bottom	控件锚定到其容器的下边缘	None	控件未锚定到其容器的任何边缘
Left	控件锚定到其容器的左边缘		

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChangeControlSizeByForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计菜单栏；添加一个 ToolStrip 控件，用来设计工具栏；添加一个 Button 控件，设置其 Anchor 属性为“Top, Bottom, Left, Right”，以便使其能够随窗体自动调整大小。

秘笈心法

心法领悟 180：Control 类的作用。

Control 类是所有 Windows 标准控件的基类，它主要实现向用户显示信息的类所需的最基本功能，并处理用户通过键盘和指针设备所进行的输入；另外，它还定义控件的边界（如位置和大小等）。

实例 181 带分隔栏的窗体

光盘位置：光盘\MR\08\181　　初级　　趣味指数：★★★☆

实例说明

软件开发中经常需要将界面分成几个部分，而且这几个部分又可以自由调整大小，这就是通常所说的“分

隔栏窗体”，那么如何来制作一个“分隔栏窗体”呢？本实例将给出满意的答案。实例运行效果如图 8.7 所示。

图 8.7　带分隔栏的窗体

关键技术

本实例在制作带分隔栏的窗体时，主要是使用 SplitContainer 控件来实现的，下面对其进行详细讲解。

可以将 Windows 窗体的 SplitContainer 控件看作是一个复合体，它是由一个可移动的拆分条分隔的两个面板。当鼠标指针悬停在该拆分条上时，鼠标指针将变为校正状态，以显示该拆分条是可移动的。因此只要在窗体中加入 SplitContainer 控件，就可以自动将窗体分成两部分。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SpliterForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计菜单栏；添加一个 ToolStrip 控件，用来设计工具栏；添加一个 SplitContainer 控件，用来作为窗体的分隔栏。

秘笈心法

心法领悟 181：SplitContainer 控件的作用。

使用 SplitContainer 控件可以创建复合的用户界面（通常，在一个面板中的选择决定了在另一个面板中显示哪些对象），这种排列对于显示和浏览信息非常有用。拥有两个面板使用户可以聚合不同区域中的信息，并且用户可以轻松地使用拆分条（也称为“拆分器”）调整面板的大小。

8.3　窗 体 效 果

实例 182	制作半透明渐显窗体 光盘位置：光盘\MR\08\182	中级 趣味指数：★★★★☆

实例说明

很多专业软件在启动前都会显示一个说明该软件信息或用途的窗口，有的则是一个漂亮的启动界面，如 Adobe 公司的 Acrobat 等。本实例将演示如何使用 C#制作一个半透明渐显窗体，运行本实例，窗体界面会以从完全透明慢慢到半透明的效果显示在用户面前。实例运行效果如图 8.8 和图 8.9 所示。

图 8.8　窗体显示过程中

图 8.9　窗体显示完毕

关键技术

本实例在制作半透明渐显窗体时主要用到了窗体的 Opacity 属性，下面对其进行详细讲解。

Opacity 属性用来获取或设置窗体的不透明度级别。语法格式如下：

```
[TypeConverterAttribute(typeof(OpacityConverter))]
public double Opacity { get; set; }
```

参数说明

属性值：窗体的不透明度级别。默认值为 1.00。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GraduallyForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并为该窗体设置背景图片。在 Frm_Main 窗体中添加一个 Timer 组件，并设置其 Enabled 属性为 true，Interval 属性为 1000，该组件主要用来控制窗体背景从透明到不透明的逐渐显示。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    this.Opacity += 0.1;                    //设置窗体的不透明级别
}
```

秘笈心法

心法领悟 182：将窗体设置为完全透明。

将窗体设置为完全透明时，只需要将其 Opacity 属性设置为 0 即可。

实例 183　使窗体背景色渐变

光盘位置：光盘\MR\08\183　　中级　　趣味指数：★★★★☆

实例说明

在程序设计时，可以通过设置窗体的 BackColor 属性来改变窗口的背景颜色，但是该属性改变后整个窗体的客户区都会变成这种颜色，这样显得非常单调。如果窗体的客户区可以像标题栏一样能够体现颜色的渐变效果，那么窗体风格将会另有一番风味。本实例将带领读者一起来制作一个背景色渐变的窗体。实例运行效果如图 8.10 所示。

图 8.10　使窗体背景色渐变

关键技术

本实例在实现窗体背景色渐变功能时主要用到了 Color 结构的 FromArgb 方法，下面对其进行详细讲解。

Color 结构表示一种 ARGB 颜色（alpha、红色、绿色和蓝色），其 FromArgb 方法用来从指定的 8 位颜色值（红色、绿色和蓝色）创建 Color 结构，该方法为可重载方法，其最常用的语法格式如下：

```
public static Color FromArgb(int red,int green,int blue)
```

FromArgb 方法中的参数说明如表 8.3 所示。

表 8.3　FromArgb 方法中的参数说明

参　数	说　明
red	新 Color 的红色分量值，有效值为 0～255
green	新 Color 的绿色分量值，有效值为 0～255
blue	新 Color 的蓝色分量值，有效值为 0～255
返回值	创建的 Color 结构

说明：FromArgb 方法就是用 3 种不同的色值来返回一个颜色，而稍微调整某一种颜色值就可以使整体的颜色发生细微的变化，在窗体中至上而下每行填充一种稍微调整后的颜色，这样整体看来就会产生渐变的效果。可以利用窗体的 Graphics 对象对窗体进行绘图，该对象可以完全操控窗体的客户区。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GraduallyBackColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
protected override void OnPaintBackground(PaintEventArgs e)
{
    int intLocation, intHeight;                                    //定义两个 int 型的变量 intLocation、intHeight
    intLocation = this.ClientRectangle.Location.Y;                 //为变量 intLocation 赋值
    intHeight = this.ClientRectangle.Height / 200;                 //为变量 intHeight 赋值
    for (int i =255; i >= 0; i--)
    {
        Color color = new Color();                                 //定义一个 Color 类型的实例 color
        color = Color.FromArgb(1, i, 100);                         //为实例 color 赋值
        SolidBrush SBrush = new SolidBrush(color);                 //创建一个单色画笔对象 SBrush
        Pen pen = new Pen(SBrush, 1);                              //创建一个用于绘制直线和曲线的对象 pen
        //绘制图形
        e.Graphics.DrawRectangle(pen, this.ClientRectangle.X, intLocation, this.Width, intLocation + intHeight);
        intLocation = intLocation + intHeight;                     //重新为变量 intLocation 赋值
    }
}
```

秘笈心法

心法领悟 183：实现窗体背景渐变的渐变方法。

本实例实现了窗体背景的颜色渐变，并且渐变的效果是由代码通过算法实现的。其实，实现窗体背景渐变不一定使用这种方法，最简单且最有效的方法就是使用一个最有渐变色的位图作为窗体的背景。

实例 184　随机更换主界面背景

光盘位置：光盘\MR\08\184　　中级　　趣味指数：★★★★☆

实例说明

如果开发的软件使用频率非常高，开发人员可以设计为随机更换背景的窗体，这样不但可以使用户心情愉快，也增加了软件的人性化设计。本实例就制作了一个随机更换主界面背景的程序。实例运行效果如图 8.11 所示。

图 8.11　随机更换主界面背景

关键技术

本实例实现时主要用到了 Random 类的 Next 方法和窗体的 BackgroundImage 属性，下面分别对它们进行详细讲解。

（1）Random 类的 Next 方法

Random 类表示伪随机数生成器，它是一种能够产生满足某些随机性统计要求的数字序列的设备，其 Next 方法用来返回一个小于所指定最大值的非负随机数，语法格式如下：

```
public virtual int Next(int maxValue)
```

参数说明

❶ maxValue：要生成的随机数的上界（随机数不能取该上界值）。maxValue 必须大于或等于零。

❷ 返回值：大于或等于零且小于 maxValue 的 32 位带符号整数，即返回的值范围包括零但不包括 maxValue。

（2）窗体的 BackgroundImage 属性

BackgroundImage 属性用来获取或设置在窗体中显示的背景图像。语法格式如下：

```
public virtual Image BackgroundImage { get; set; }
```

参数说明

属性值：一个 Image 图像，它表示在窗体的背景中显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RandomBackGround。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    //定义一个字符串数组，用来存储背景图片列表
    string[] strImages = new string[] { "01.jpg", "02.jpg", "03.jpg", "04.jpg", "05.jpg" };
    Random rdn = new Random();                                    //定义一个伪随机数生成器对象
    int intIndex = rdn.Next(0, strImages.Length - 1);             //产生一个随机数
    this.BackgroundImage = Image.FromFile(strImages[intIndex]);   //设置窗体的背景图片
    this.BackgroundImageLayout = ImageLayout.Stretch;             //设置背景图片拉伸显示
}
```

秘笈心法

心法领悟 184：使用 ImageLayout 枚举设置图片的显示方式。

ImageLayout 枚举用来指定控件上图像的位置，其 Stretch 枚举值用来指示图像沿控件的矩形工作区拉伸显示；另外，还可以通过该枚举的其他枚举值设置图像以居中、平铺等方式显示。

实例 185　磁性窗体的设计

光盘位置：光盘\MR\08\185　　高级　　趣味指数：★★★★★

实例说明

在多媒体播放软件中，经常会遇到一种情况，即当拖动其中一个窗体（主窗体）时，其他窗体（子窗体）也可以随着该窗体移动，当拖动子窗体时，其他窗体将不跟随移动，这就是磁性窗体。本实例将制作一个磁性窗体，当拖动主窗体移动时，两个子窗体如果相连，则跟随移动。实例运行效果如图 8.12 所示。

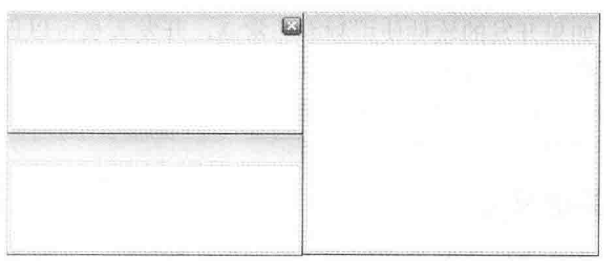


图 8.12　磁性窗体的设计

关键技术

本实例主要是在主窗体移动时，改变跟随窗体的 Left 和 Top 属性值，下面对本实例中用到的关键技术进行详细讲解。

（1）无标题窗体的移动

无标题窗体的移动主要是通过控件来移动窗体，下面用 Panel 控件来进行详细说明。

首先，在 Panel 控件的 MouseDown 事件中将鼠标按下时的位置值（负值）存入到全局变量 CPoint 中，代码

如下：

```
private void panel_Title_MouseDown(object sender, MouseEventArgs e)
{
    CPoint = new Point(-e.X, -e.Y);                    //获取鼠标按下时的位置
}
```

然后，在 Panel 控件的 MouseMove 事件中按照 CPoint 变量的值，以屏幕坐标平移指定的量，并用平移后的结果设置窗体的 DesktopLocation 属性，代码如下：

```
private void panel_Title_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        Point myPosittion = Control.MousePosition;          //获取当前鼠标的屏幕坐标
        myPosittion.Offset(CPoint.X, CPoint.Y);             //以屏幕坐标平移指定的量
        this.DesktopLocation = myPosittion;                 //设置当前窗体在屏幕上的位置
    }
}
```

（2）Left 属性

该属性用于获取或设置控件左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。语法格式如下：

```
public int Left { get; set; }
```

参数说明

属性值：窗体左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。

（3）Top 属性

该属性用于获取或设置控件上边缘与其容器的工作区上边缘之间的距离（以像素为单位）。语法格式如下：

```
public int Top { get; set; }
```

参数说明

属性值：窗体上边缘与其容器的工作区上边缘之间的距离（以像素为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MagnetismForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Play，并将该窗体的 FormBorderStyle 属性设置为 None。在 Frm_Play 窗体中添加两个 Panel 控件，分别命名为 panel_Title 和 panel_Close，并将 panel_Title 控件的 Dock 属性设置为 Top。

（3）在该项目中添加两个窗体，分别命名为 Frm_ListBox 和 Frm_Libretto。

说明： Frm_ListBox 和 Frm_Libretto 窗体中所用到的控件及其设置与 Frm_Play 窗体相同，在这里不作详解。

（4）程序的主要代码如下。

说明： 磁性窗体的设置实际上就是将相关联的窗体位置和大小记录在公共类 FrmClass 中，然后在移动窗体时，通过已记录的窗体信息，实现窗体的磁性效果。下面以 Frm_Play 窗体为例，介绍磁性窗体的实现过程。

在 Frm_Play 窗体标题栏上按下鼠标时，获取其他两个窗体的相关信息（这两个窗体与该窗体具有磁性效果）。代码如下：

```
private void panel_Title_MouseDown(object sender, MouseEventArgs e)
{
    int Tem_Y = 0;
    if (e.Button == MouseButtons.Left)                          //按下的是否为鼠标左键
    {
        Cla_FrmClass.FrmBackCheck();                            //检测各窗体是否连在一起
        Tem_Y = e.Y;
        FrmClass.FrmPoint = new Point(e.X, Tem_Y);              //获取鼠标在窗体上的位置，用于磁性窗体
        FrmClass.CPoint = new Point(-e.X, -Tem_Y);              //获取鼠标在屏幕上的位置，用于窗体的移动
        if (FrmClass.Example_List_AdhereTo)                     //如果与 Frm_ListBox 窗体相连接
        {
```

```
                Cla_FrmClass.FrmDistanceJob(this, FrmClass.F_List);                    //计算窗体的距离差
                if (FrmClass.Example_Assistant_AdhereTo)                               //两个辅窗体是否连接在一起
                {
                    Cla_FrmClass.FrmDistanceJob(this, FrmClass.F_Libretto);            //计算窗体的距离差
                }
            }
            if (FrmClass.Example_Libretto_AdhereTo)                                    //如果与 Frm_ Libretto 窗体相连接
            {
                Cla_FrmClass.FrmDistanceJob(this, FrmClass.F_Libretto);                //计算窗体的距离差
                if (FrmClass.Example_Assistant_AdhereTo)                               //两个辅窗体是否连接在一起
                {
                    Cla_FrmClass.FrmDistanceJob(this, FrmClass.F_List);                //计算窗体的距离差
                }
            }
        }
    }
```

自定义方法 FrmBackCheck 的主要功能是通过各窗体的位置和大小，检测各窗体是否连接在一起。代码如下：

```
#region  检测各窗体是否连接在一起
/// <summary>
///检测各窗体是否连接在一起
/// </summary>
public void FrmBackCheck()
{
    bool Tem_Magnetism = false;
    Tem_Magnetism = false;
    if ((Example_Play_Top - Example_List_Top) == 0)                                    //两个窗体是否顶对齐
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_List_Left) == 0)                                  //两个窗体是否左对齐
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_List_Left - Example_List_Width) == 0)             //两个窗体是否左右相连
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_List_Left + Example_List_Width) == 0)             //两个窗体是否左右相连
        Tem_Magnetism = true;
    if ((Example_Play_Top - Example_List_Top - Example_List_Height) == 0)              //两个窗体是否上下相连
        Tem_Magnetism = true;
    if ((Example_Play_Top - Example_List_Top + Example_List_Height) == 0)              //两个窗体是否上下相连
        Tem_Magnetism = true;
    if (Tem_Magnetism)                                                                 //如果连接
        Example_List_AdhereTo = true;                                                  //表示 Frm_ListBox 与主窗体是磁性窗体
    //Frm_Libretto 与主窗体
    Tem_Magnetism = false;
    if ((Example_Play_Top - Example_Libretto_Top) == 0)                                //两个窗体是否顶对齐
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_Libretto_Left) == 0)                              //两个窗体是否左对齐
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_Libretto_Left - Example_Libretto_Width) == 0)     //两个窗体是否左右相连
        Tem_Magnetism = true;
    if ((Example_Play_Left - Example_Libretto_Left + Example_Libretto_Width) == 0)     //两个窗体是否左右相连
        Tem_Magnetism = true;
    if ((Example_Play_Top - Example_Libretto_Top - Example_Libretto_Height) == 0)      //两个窗体是否上下相连
        Tem_Magnetism = true;
    if ((Example_Play_Top - Example_Libretto_Top + Example_Libretto_Height) == 0)      //两个窗体是否上下相连
        Tem_Magnetism = true;
    if (Tem_Magnetism)                                                                 //如果连接
        Example_Libretto_AdhereTo = true;                                              //Frm_Libretto 与主窗体是磁性窗体
    //两个辅窗体
    Tem_Magnetism = false;
    if ((Example_List_Top - Example_Libretto_Top) == 0)
        Tem_Magnetism = true;
    if ((Example_List_Left - Example_Libretto_Left) == 0)
        Tem_Magnetism = true;
    if ((Example_List_Left - Example_Libretto_Left - Example_Libretto_Width) == 0)
        Tem_Magnetism = true;
    if ((Example_List_Left - Example_Libretto_Left + Example_Libretto_Width) == 0)
        Tem_Magnetism = true;
    if ((Example_List_Top - Example_Libretto_Top - Example_Libretto_Height) == 0)
        Tem_Magnetism = true;
```

```
    if ((Example_List_Top - Example_Libretto_Top + Example_Libretto_Height) == 0)
        Tem_Magnetism = true;
    if (Tem_Magnetism)                                          //如果连接
        Example_Assistant_AdhereTo = true;                      //两个辅窗体是磁性窗体
}
#endregion
```

自定义方法 FrmDistanceJob 用于计算 Frm_Play 窗体与 Frm_ListBox 窗体、Frm_Libretto 窗体之间的距离。代码如下：

```
#region  计算窗体之间的距离差
/// <summary>
///计算窗体之间的距离差
/// </summary>
/// <param Frm="Form">窗体</param>
/// <param Follow="Form">跟随窗体</param>
public void FrmDistanceJob(Form Frm, Form Follow)
{
    switch (Follow.Name)
    {
        case "Frm_ListBox"                                      //Frm_ListBox 子窗体
            {
                //“列表”窗体与“播放”窗体的位置差
                Example_List_space_Top = Follow.Top - Frm.Top;
                Example_List_space_Left = Follow.Left - Frm.Left;
                break;
            }
        case "Frm_Libretto":                                    //Frm_Libretto 子窗体
            {
                //Frm_Libretto 子窗体与主窗体的位置差
                Example_Libretto_space_Top = Follow.Top - Frm.Top;
                Example_Libretto_space_Left = Follow.Left - Frm.Left;
                break;
            }
    }
}
#endregion
```

当鼠标按下后，通过鼠标的移动事件来移动窗体的位置，在移动窗体时，与该窗体相关联的磁性窗体也随之移动。代码如下：

```
private void panel_Title_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)                          //当鼠标按下的是左键
    {
        Cla_FrmClass.FrmMove(this, e);                          //移动当前窗体
        if (FrmClass.Example_List_AdhereTo)                     //如果 Frm_ListBox 窗体是磁性窗体
        {
            Cla_FrmClass.ManyFrmMove(this, e, FrmClass.F_List); //移动相关联的磁性窗体
            Cla_FrmClass.FrmInitialize(FrmClass.F_List);        //设置 Frm_ListBox 窗体的位置
            if (FrmClass.Example_Assistant_AdhereTo)            //如果两个辅窗体是磁性窗体
            {
                Cla_FrmClass.ManyFrmMove(this, e, FrmClass.F_Libretto); //移动相关联的磁性窗体
                Cla_FrmClass.FrmInitialize(FrmClass.F_Libretto);        //设置 Frm_Libretto 窗体的位置
            }
        }
        if (FrmClass.Example_Libretto_AdhereTo)                 //如果 Frm_Libretto 窗体是磁性窗体
        {
            Cla_FrmClass.ManyFrmMove(this, e, FrmClass.F_Libretto); //移动相关联的磁性窗体
            Cla_FrmClass.FrmInitialize(FrmClass.F_Libretto);        //设置 Frm_Libretto 窗体的位置
            if (FrmClass.Example_Assistant_AdhereTo)            //如果两个辅窗体是磁性窗体
            {
                Cla_FrmClass.ManyFrmMove(this, e, FrmClass.F_List); //移动相关联的磁性窗体
                Cla_FrmClass.FrmInitialize(FrmClass.F_List);        //设置 Frm_ListBox 窗体的位置
            }
        }
        Cla_FrmClass.FrmInitialize(this);                       //设置当前窗体的位置
    }
}
```

自定义方法 ManyFrmMove 是在主窗体移动时获取鼠标在屏幕上的位置，然后计算出相关磁性窗体的移动位置，并实现磁性窗体的移动效果。代码如下：

```
#region  磁性窗体的移动
/// <summary>
///磁性窗体的移动
/// </summary>
/// <param Frm="Form">窗体</param>
/// <param e="MouseEventArgs">控件的移动事件</param>
/// <param Follow="Form">跟随窗体</param>
//Form 或 MouseEventArgs 添加命名空间 using System.Windows.Forms;
public void ManyFrmMove(Form Frm, MouseEventArgs e, Form Follow)
{
    if (e.Button == MouseButtons.Left)
    {
        int Tem_Left = 0;
        int Tem_Top = 0;
        Point myPosittion = Control.MousePosition;                    //获取当前鼠标的屏幕坐标
        switch (Follow.Name)
        {
            case "Frm_ListBox":                                       //Frm_ListBox 子窗体
                {
                    //“列表”窗体与“播放”窗体的位置差
                    Tem_Top = Example_List_space_Top - FrmPoint.Y;
                    Tem_Left = Example_List_space_Left - FrmPoint.X;
                    break;
                }
            case "Frm_Libretto":                                      //Frm_Libretto 子窗体
                {
                    //Frm_Libretto 子窗体与主窗体的位置差
                    Tem_Top = Example_Libretto_space_Top - FrmPoint.Y;
                    Tem_Left = Example_Libretto_space_Left - FrmPoint.X;
                    break;
                }
        }
        myPosittion.Offset(Tem_Left, Tem_Top);                        //获取设置后的坐标
        Follow.DesktopLocation = myPosittion;                         //设置跟随窗体的位置
    }
}
#endregion
```

自定义方法 FrmInitialize 用于获取各窗体的初始化位置，以及移动后的位置。代码如下：

```
#region  对窗体的位置进行初始化
/// <summary>
///对窗体的位置进行初始化
/// </summary>
/// <param Frm="Form">窗体</param>
public void FrmInitialize(Form Frm)
{
    switch (Frm.Name)
    {
        case "Frm_Play":                                              //主窗体
            {
                Example_Play_Top = Frm.Top;                           //设置主窗体的上边距
                Example_Play_Left = Frm.Left;                         //设置主窗体的左边距
                Example_Play_Width = Frm.Width;                       //设置主窗体的宽度
                Example_Play_Height = Frm.Height;                     //设置主窗体的高度
                break;
            }
        case "Frm_ListBox":                                           //Frm_ListBox 子窗体
            {
                Example_List_Top = Frm.Top;
                Example_List_Left = Frm.Left;
                Example_List_Width = Frm.Width;
                Example_List_Height = Frm.Height;
                break;
            }
        case "Frm_Libretto":                                          //Frm_Libretto 子窗体
```

```
            {
                Example_Libretto_Top = Frm.Top;
                Example_Libretto_Left = Frm.Left;
                Example_Libretto_Width = Frm.Width;
                Example_Libretto_Height = Frm.Height;
                break;
            }
        }
    }
    #endregion
```

当鼠标松开时，存储各窗体的当前信息，以便于计算是否为磁性窗体。代码如下：

```
private void panel_Title_MouseUp(object sender, MouseEventArgs e)
{
    Cla_FrmClass.FrmPlace(this);
}
```

自定义方法 FrmPlace 用于存储当前窗体的位置和大小，并通过设置后的信息计算当前窗体是否与其他窗体成为磁性窗体。代码如下：

```
#region  存储各窗体的当前信息
/// <summary>
///存储各窗体的当前信息
/// </summary>
/// <param Frm="Form">窗体</param>
public void FrmPlace(Form Frm)
{
    FrmInitialize(Frm);                                    //对窗体的位置进行初始化
    FrmMagnetism(Frm);                                     //磁性窗体的设置
}
#endregion
```

自定义方法 FrmMagnetism 是根据指定的窗体名称，计算当前窗体与其他两个窗体是否能成为磁性窗体。代码如下：

```
#region  窗体的磁性设置
/// <summary>
///窗体的磁性设置
/// </summary>
/// <param Frm="Form">窗体</param>
public void FrmMagnetism(Form Frm)
{
    if (Frm.Name != "Frm_Play")                            //如果不是主窗体
    {
        FrmMagnetismCount(Frm, Example_Play_Top, Example_Play_Left, Example_Play_Width,
Example_Play_Height, "Frm_Play");                          //计算当前窗体与主窗体是否为磁性窗体
    }
    if (Frm.Name != "Frm_ListBox")                         //如果不是 Frm_ListBox 子窗体
    {
        FrmMagnetismCount(Frm, Example_List_Top, Example_List_Left, Example_List_Width,
Example_List_Height, "Frm_ListBox");                       //计算当前窗体与 Frm_ListBox 子窗体是否为磁性窗体
    }
    if (Frm.Name != "Frm_Libretto")                        //如果不是 Frm_Libretto 子窗体
    {
        FrmMagnetismCount(Frm, Example_Libretto_Top, Example_Libretto_Left, Example_Libretto_Width,
Example_Libretto_Height, "Frm_Libretto");                  //计算当前窗体与 Frm_Libretto 子窗体是否为磁性窗体
    }
    FrmInitialize(Frm);
}
#endregion
```

自定义方法 FrmMagnetismCount 用于计算两个窗体是否为磁性窗体，主要是当两个窗体上、下、左、右的间隔小于 10 时，将两个窗体自动粘贴到一起。代码如下：

```
#region  磁性窗体的计算
/// <summary>
///磁性窗体的计算
/// </summary>
/// <param Frm="Form">子窗体</param>
/// <param top="int">主窗体上边距</param>
/// <param left="int">主窗体左边距</param>
```

```
/// <param width="int">主窗体的宽度</param>
/// <param height="int">主窗体的高度</param>
/// <param Mforms="string">主窗体</param>
public void FrmMagnetismCount(Form Frm, int top, int left, int width, int height, string Mforms)
{
    bool Tem_Magnetism = false;                    //判断是否有磁性发生
    string Tem_MainForm = "";                      //临时记录主窗体
    string Tem_AssistForm = "";                    //临时记录辅窗体
    bool Tem_UpDown = false;                       //鼠标是否按下
    bool Tem_LeftRight = false;
    //下面进行磁性窗体计算
    if ((Frm.Top + Frm.Height - top) <= Example_FSpace && (Frm.Top + Frm.Height - top) >= -Example_FSpace)
    {
        //当一个主窗体不包含辅窗体时
        if ((Frm.Left >= left && Frm.Left <= (left + width)) || ((Frm.Left + Frm.Width) >= left && (Frm.Left + Frm.Width) <= (left + width)))
        {
            Frm.Top = top - Frm.Height;
            if ((Frm.Left - left) <= Example_FSpace && (Frm.Left - left) >= -Example_FSpace)
                Frm.Left = left;
            Tem_Magnetism = true;
        }
        //当一个主窗体包含辅窗体时
        if (Frm.Left <= left && (Frm.Left + Frm.Width) >= (left + width))
        {
            Frm.Top = top - Frm.Height;
            if ((Frm.Left - left) <= Example_FSpace && (Frm.Left - left) >= -Example_FSpace)
                Frm.Left = left;
            Tem_Magnetism = true;
        }
        if (Tem_Magnetism == true)
        {
            if (Frm.Left == left)
                Tem_Magnetism = false;
        }
    }
    else
        Tem_Magnetism = false;
    //下面进行磁性窗体计算
    if ((Frm.Top - (top + height)) <= Example_FSpace && (Frm.Top - (top + height)) >= -Example_FSpace)
    {
        //当一个主窗体不包含辅窗体时
        if ((Frm.Left >= left && Frm.Left <= (left + width)) || ((Frm.Left + Frm.Width) >= left && (Frm.Left + Frm.Width) <= (left + width)))
        {
            Frm.Top = top + height;
            if ((Frm.Left - left) <= Example_FSpace && (Frm.Left - left) >= -Example_FSpace)
                Frm.Left = left;
            Tem_Magnetism = true;
        }
        //当一个主窗体包含辅窗体时
        if (Frm.Left <= left && (Frm.Left + Frm.Width) >= (left + width))
        {
            Frm.Top = top + height;
            if ((Frm.Left - left) <= Example_FSpace && (Frm.Left - left) >= -Example_FSpace)
                Frm.Left = left;
            Tem_Magnetism = true;
        }
        if (Tem_Magnetism == true)
        {
            if (Frm.Left == left)
                Tem_Magnetism = false;
        }
    }
    else
        Tem_Magnetism = false;
    //下面进行磁性窗体计算
    if ((Frm.Left + Frm.Width - left) <= Example_FSpace && (Frm.Left + Frm.Width - left) >= -Example_FSpace)
    {
        //当一个主窗体不包含辅窗体时
```

```
        if ((Frm.Top > top && Frm.Top <= (top + height)) || ((Frm.Top + Frm.Height) >= top && (Frm.Top + Frm.Height) <= (top + height)))
        {
            Frm.Left = left - Frm.Width;
            if ((Frm.Top - top) <= Example_FSpace && (Frm.Top - top) >= -Example_FSpace)
                Frm.Top = top;
            Tem_Magnetism = true;
        }
        //当一个主窗体包含辅窗体时
        if (Frm.Top <= top && (Frm.Top + Frm.Height) >= (top + height))
        {
            Frm.Left = left - Frm.Width;
            if ((Frm.Top - top) <= Example_FSpace && (Frm.Top - top) >= -Example_FSpace)
                        Frm.Top = top;
            Tem_Magnetism = true;
        }
    }
    else
        Tem_Magnetism = false;
    //下面进行磁性窗体计算
    if ((Frm.Left - (left + width)) <= Example_FSpace && (Frm.Left - (left + width)) >= -Example_FSpace)
    {
        //当一个主窗体不包含辅窗体时
        if ((Frm.Top > top && Frm.Top <= (top + height)) || ((Frm.Top + Frm.Height) >= top && (Frm.Top + Frm.Height) <= (top + height)))
        {
            Frm.Left = left + width;
            if ((Frm.Top - top) <= Example_FSpace && (Frm.Top - top) >= -Example_FSpace)
                Frm.Top = top;
            Tem_Magnetism = true;
        }
        //当一个主窗体包含辅窗体时
        if (Frm.Top <= top && (Frm.Top + Frm.Height) >= (top + height))
        {
            Frm.Left = left + width;
            if ((Frm.Top - top) <= Example_FSpace && (Frm.Top - top) >= -Example_FSpace)
                Frm.Top = top;
            Tem_Magnetism = true;
        }
    }
    else
        Tem_Magnetism = false;
    //下面是用来判断两个辅窗体是否为磁性窗体
    if (Frm.Name == "Frm_Play")                          //如果主窗体是要靠近的窗体
        Tem_MainForm = "Frm_Play";
    else
        Tem_AssistForm = Frm.Name;
    if (Mforms == "Frm_Play")                            //如果主窗体是移动的窗体
        Tem_MainForm = "Frm_Play";
    else
        Tem_AssistForm = Mforms;
    if (Tem_MainForm == "")                              //如果是两个辅窗体间的移动
    {
        Example_Assistant_AdhereTo = Tem_Magnetism;
    }
    else
    {
        switch (Tem_AssistForm)
        {
            case "Frm_ListBox":                          //移动的是 Frm_ListBox 子窗体
                Example_List_AdhereTo = Tem_Magnetism;
                break;
            case "Frm_Libretto":                         //移动的是 Frm_Libretto 子窗体
                Example_Libretto_AdhereTo = Tem_Magnetism;
                break;
        }
    }
}
#endregion
```

秘笈心法

心法领悟 185：磁性窗体的实现原理。

为了实现磁性窗体功能，主要就是判断两个窗体之间的位置关系，当移动其中一个窗体时，如果和另一个窗体相连接则一起移动，否则只是单独移动；当移动当前窗体靠近另一个窗体时，判断两个窗体之间的距离，如果两个窗体之间的距离小于 20 像素，则自动将两个窗体连接在一起，这样就实现了磁性窗体功能。

实例 186　制作鼠标穿透窗体

光盘位置：光盘\MR\08\186　　高级　　趣味指数：★★★★★

实例说明

用户在对桌面进行操作时，为了使桌面更加美观，可以在桌面的上面加一层类似于玻璃的效果，用户可以用鼠标透过“玻璃”对桌面进行操作。本实例通过使用鼠标穿透窗体来实现以上功能。实例运行效果如图 8.13 所示。

图 8.13　制作鼠标穿透窗体

关键技术

本实例实现时主要用到了 API 函数 SetWindowLong 和 GetWindowLong，下面分别对它们进行详细讲解。

（1）SetWindowLong 函数

该函数主要用于在窗口结构中为指定的窗口设置信息。语法格式如下：

```
[DllImport("user32", EntryPoint = "SetWindowLong")]
private static extern uint SetWindowLong(IntPtr hwnd, int nIndex, uint dwNewLong);
```

参数说明

❶ hwnd：IntPtr，欲为其取得信息的窗口的句柄。

❷ nIndex：int，欲取回的信息，其常量如表 8.4 所示。

❸ dwNewLong：uint，由 nIndex 指定的窗口信息的新值。

❹ 返回值：uint，指定数据的前一个值。

表 8.4　nIndex 参数的常量值及说明

常　量	值	说　明
GWL_EXSTYLE	-20	扩展窗口样式
GWL_STYLE	-16	窗口样式
GWL_WNDPROC	-4	该窗口的窗口函数的地址
GWL_HINSTANCE	-6	拥有窗口实例的句柄
GWL_HWNDPARENT	-8	该窗口之父的句柄。不要用 SetWindowWord 来改变这个值
GWL_ID	-12	对话框中一个子窗口的标识符
GWL_USERDATA	-21	含义由应用程序规定
DWL_DLGPROC	4	这个窗口的对话框函数地址
DWL_MSGRESULT	0	在对话框函数中处理的一条消息返回的值
DWL_USER	8	含义由应用程序规定

说明：程序中使用系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间，下面再遇到类似的情况时将不再提示。

（2）GetWindowLong 函数

该函数主要从指定窗口的结构中取得信息。语法格式如下：

```
[DllImport("user32", EntryPoint = "GetWindowLong")]
private static extern uint GetWindowLong(IntPtr hwnd, int nIndex);
```

参数说明

❶ hwnd：IntPtr，欲为其取得信息的窗口的句柄。

❷ nIndex：该参数的值请参见表 8.4。

❸ 返回值：uint，由 nIndex 决定。零表示出错。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MouseThroughForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将该窗体的 FormBorderStyle 属性设置为 None。在 Frm_Main 窗体中添加一个 NotifyIcon 控件，并设置其 Icon 属性为指定的图标，用来显示提示信息；添加一个 ContextMenuStrip 控件，用来作为程序的快捷菜单。

（3）程序的主要代码。

Frm_Main 窗体中，首先自定义一个方法 CanPenetrate，该方法用来通过 API 函数 SetWindowLong 和 GetWindowLong 来实现鼠标的穿透效果。CanPenetrate 方法的实现代码如下：

```
private void CanPenetrate()
{
    uint intExTemp = GetWindowLong(this.Handle, GWL_EXSTYLE);                    //从当前窗口的结构中取得信息
    //在窗口结构中为当前窗口设置信息
    uint oldGWLEx = SetWindowLong(this.Handle, GWL_EXSTYLE, WS_EX_TRANSPARENT | WS_EX_LAYERED);
}
```

Frm_Main 窗体加载时，使窗体不出现在 Windows 任务栏中，并且使鼠标穿透窗体。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    this.ShowInTaskbar = false;                                                  //窗体不出现在 Windows 任务栏中
    CanPenetrate();
    this.TopMost = true;                                                         //使窗体始终在其他窗体之上
}
```

秘笈心法

心法领悟 186：将窗体设置为最顶层显示。

在将窗体设置为最顶层显示时，只需要将窗体的 TopMost 属性设为 true 即可。

实例 187　窗体换肤程序

光盘位置：光盘\MR\08\187　　高级　　趣味指数：★★★★★

实例说明

在默认情况下，Windows 窗体的皮肤由操作系统的主题和外观设置来决定，但有些应用软件为了摆脱操作系统的这种束缚，已经开发出自定义窗体皮肤的功能，如常见的播放软件暴风影音、PPLive 等。在本实例的窗体中单击鼠标右键，将弹出一个用于更换窗体皮肤的快捷菜单，选择“换皮肤”菜单下的任意子菜单，程序将为当前窗体更换皮肤。实例运行效果如图 8.14 所示。

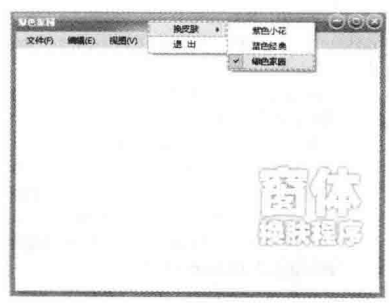


图 8.14　窗体换肤程序

关键技术

本实例的基本原理是给窗体的各个组成部分更换图片，基于该原

理，首先需要分析窗体的组成部分，主要包括标题栏、左边框、右边框、下边框、窗体中间区域及可能存在的菜单栏，其中标题栏和 3 个边框无法通过设置相关属性来达到更换背景图片的目的，对于这个问题，可以通过取消窗体的 FormBorderStyle 属性，同时在标题栏和 3 个边框的位置添加 Panel 控件来解决，然后通过选择不同的皮肤类型，程序为窗体的各个组成部分设置图片，最终达到窗体换肤的效果。在以上过程中，最主要的技术问题是如何从指定的文件创建 Image 对象，以及如何获取图片的路径。下面对本实例用到的关键技术进行详细讲解。

（1）从指定的文件创建 Image 对象

从指定的文件创建 Image 对象时主要用到 Image 类的 FromFile 方法，该方法的重载形式有两种，具体如下：

```
public static Image FromFile(string filename)                    //该方法从指定的文件创建 Image 对象
//该方法使用该文件中的嵌入颜色管理信息，从指定的文件创建 Image 对象
public static Image FromFile(string filename, bool useEmbeddedColorManagement)
```

参数说明

❶ filename：指定位置的图片名称。

❷ useEmbeddedColorManagement：若要使用图像文件中嵌入的颜色管理信息，则设置为 true；否则设置为 false。

说明：由于 FromFile 方法是 Image 类中的一个静态方法，所以可直接使用 Image 类来调用。

（2）获取图片路径

程序要实现动态加载图片文件，就需要获取图片文件的路径，本实例通过读取 Application.StartupPath 属性的值和调用 String 类的 LastIndexOf 方法来获取图片文件夹的路径。其中，Application.StartupPath 属性用于获取应用程序的可执行文件的路径（但不包括可执行文件的名称），String 类的 LastIndexOf 方法用于实现在当前字符串中获取最后一个匹配项的索引位置，该方法的重载形式有多种，其最常用的语法格式如下：

```
public int LastIndexOf(string value)                    //获取当前字符串中最后一个匹配项的索引位置
```

参数说明

❶ value：要查找的字符串，即在当前字符串中要匹配的字符串。

❷ 返回值：如果找到该字符串，则返回值为当前字符串中最后一个匹配项的索引位置；如果未找到该字符串，则返回值为-1。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WinCusSkin。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 6 个 Panel 控件，分别用来作为窗体的标题栏、下边框、左边框、右边框、左下角边框和右下角边框；添加一个 ContextMenuStrip 控件，用来作为更换皮肤的快捷菜单；添加 3 个 PictureBox 控件，分别用来显示最大化、最小化和关闭图片。

（3）程序的主要代码。

在 Frm_Main 窗体中单击鼠标右键，会弹出一个用于更换窗体皮肤的快捷菜单，选择“换皮肤”菜单下的任意子菜单，程序将为窗体更换皮肤。程序为窗体更换皮肤主要是通过设置 Panel 控件的 BackgroundImage 属性和 PictureBox 控件的 Image 属性来实现的，这里以选择“紫色小花”选项为例，选择该选项将触发它的 Click 事件。代码如下：

```
private void menItemSkin1_Click(object sender, EventArgs e)
{
    //设置窗体顶部 Panel 控件的 BackgroundImage 属性
    this.panel_Top.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\top.png");
    //设置窗体左侧 Panel 控件的 BackgroundImage 属性
    this.panel_Left.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\left.png");
    //设置窗体右侧 Panel 控件的 BackgroundImage 属性
    this.panel_Right.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\right.png");
    //设置窗体底部 Panel 控件的 BackgroundImage 属性
    this.panel_Bottom.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\bottom.png");
    //设置最小化图片控件的 Image 属性
    this.picMinimize.Image = Image.FromFile(strImagesPath + @"\images\purple\min.png");
```

```
    if (bol == true)                                                    //若当前窗体处于最大化状态
    {
        //设置最大化图片控件的 Image 属性
        this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\purple\max.png");
    }
    else                                                                //若当前窗体处于普通状态
    {
        //设置最大化图片控件的 Image 属性
        this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\purple\max_normal.png");
    }
    //设置关闭图片控件的 Image 属性
    this.picClose.Image = Image.FromFile(strImagesPath + @"\images\purple\close.png");
    this.menItemSkin1.Checked = true;                                   //设置菜单的选中标记
    this.menItemSkin2.Checked = false;                                  //取消菜单的选中标记
    this.menItemSkin3.Checked = false;                                  //取消菜单的选中标记
    //设置窗体主菜单的背景图像属性
    this.menuStrip1.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\menu.gif");
    //设置窗体的背景图像属性
    this.BackgroundImage = Image.FromFile(strImagesPath + @"\images\purple\background.gif");
}
```

在 Frm_Main 窗体中单击“最大化”按钮，窗体的大小将会在最大化状态和普通状态之间切换。窗体实现这种大小状态的切换，主要是通过设置其 Height 属性和 Width 属性来实现的，“最大化”按钮的 Click 事件代码如下：

```
private void picMaximize_Click(object sender, System.EventArgs e)
{
    if (!bol)                                                           //若窗体处于普通状态
    {
        top = this.Top;                                                 //获取窗体的 Top 属性值
        left = this.Left;                                               //获取窗体的 Left 属性值
        hei = this.Height;                                              //获取窗体的 Height 属性值
        wid = this.Width;                                               //获取窗体的 Width 属性值
        this.Top = 0;                                                   //设置窗体的 Top 属性值为零
        this.Left = 0;                                                  //设置窗体的 Left 属性值为零
        int hg = SystemInformation.MaxWindowTrackSize.Height;           //获取窗口的默认最大高度
        int wh = SystemInformation.MaxWindowTrackSize.Width;            //获取窗口的默认最大宽度
        this.Height = hg;                                               //设置窗体的 Height 属性值
        this.Width = wh;                                                //设置窗体的 Width 属性值
        bol = true;                                                     //设置窗体标记表示最大化
        if (menItemSkin1.Checked)                                       //若选择“紫色小花”选项
            //设置最大化图片的 Image 属性
            this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\purple\max.png");
        if (menItemSkin2.Checked)                                       //若选择“蓝色经典”选项
            this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\blue\max.png");
        if (menItemSkin3.Checked)                                       //若选择“绿色家园”选项
            //设置最大化图片的 Image 属性
            this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\green\max.png");
    }
    else                                                                //若窗体处于最大化状态
    {
        this.Top = top;                                                 //设置窗体的 Top 属性值
        this.Left = left;                                               //设置窗体的 Left 属性值
        this.Height = hei;                                              //设置窗体的 Height 属性值
        this.Width = wid;                                               //设置窗体的 Width 属性值
        bol = false;                                                    //设置窗体标记表示普通状态
        if (menItemSkin1.Checked)
            this.picMaximize.Image = Image.FromFile(strImagesPath +
@"\images\purple\max_Normal.png");
        if (menItemSkin2.Checked)
            this.picMaximize.Image = Image.FromFile(strImagesPath + @"\images\blue\max_Normal.png");
        if (menItemSkin3.Checked)
            this.picMaximize.Image = Image.FromFile(strImagesPath +
@"\images\green\max_Normal.png");
    }
}
```

秘笈心法

心法领悟 187：实现“最小化”按钮、“最大化”按钮或“关闭”按钮的动态效果。

本实例中，当鼠标进入窗体的“最小化”“最大化”或“关闭”按钮时，如果相应按钮能够有动态的效果显示，那么本实例会更加完美一些。当鼠标进入图片控件的区域时，将触发图片控件的 MouseEnter 事件；当鼠标离开图片按钮的区域时，将触发图片控件的 MouseLeave 事件，在图片控件的这两个事件中，通过设置图片控件的 Image 属性即可达到上面想要的动态效果。

8.4 窗体动画

实例 188	窗体中的滚动字幕 光盘位置：光盘\MR\08\188	初级 趣味指数：★★★★

实例说明

普通窗体中的文字位置都是固定的，但在一些窗体中需要让文字动起来，如一些广告性较强的界面中需要做一些滚动的字幕。本实例实现了一个具有滚动字幕效果的窗体，运行本实例，单击“演示”按钮，将看到窗口中的文字开始滚动；单击“暂停”按钮，可以使字幕停止滚动。实例运行效果如图 8.15 所示。

图 8.15 窗体中的滚动字幕

关键技术

本实例主要是通过 Timer 组件控制 Label 控件的移动来实现的，而 Label 控件的移动主要是通过设置其与窗体左边距的相对位置来实现的。下面对本实例用到的关键技术进行详细讲解。

（1）Timer 组件

Timer 计时器可以按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，其 Interval 属性用来获取或设置在相对于上一次发生的 Tick 事件引发 Tick 事件之前的时间（以毫秒为单位）。Interval 属性的语法格式如下：

```
public int Interval { get; set; }
```

参数说明

属性值：指定在相对于上一次发生的 Tick 事件引发 Tick 事件之前的毫秒数，该值不能小于 1。

Timer 组件的 Enabled 属性用来获取或设置计时器是否正在运行。语法格式如下：

```
public virtual bool Enabled { get; set; }
```

参数说明

属性值：如果计时器当前处于启用状态，则为 true；否则为 false。默认为 false。

（2）Label 控件的 Left 属性

该属性用来获取或设置控件左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。语法格式如下：

```
[BrowsableAttribute(false)]
public int Left { get; set; }
```

参数说明

属性值：表示控件左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。

说明：Left 属性的值等效于 Label 控件的 Location 属性值的 Point.X 属性。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MoveFontInForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示要滚动的文字信息；添加 3 个 Button 控件，分别用来执行开始滚动、停止滚动和关闭窗体操作；添加一个 Timer 组件，用来控制字幕的滚动。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)   //用 Timer 来控制字幕的滚动
{
    label1.Left -= 2;                                   //设置 label1 左边缘与其容器的工作区左边缘之间的距离
    if (label1.Right < 0)                               //当 label1 右边缘与其容器的工作区左边缘之间的距离小于 0 时
    {
        label1.Left = this.Width;                       //设置 label1 左边缘与其容器的工作区左边缘之间的距离为该窗体的宽度
    }
}
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;                              //开始滚动
}
private void button2_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;                             //停止滚动
}
```

秘笈心法

心法领悟 188：启动和关闭 Timer 计时器的两种方法。

启动 Timer 计时器时，可以将其 Enabled 属性设置为 true，或者调用其 Start 方法；而关闭 Timer 计时器时，则需要将其 Enabled 属性设置为 false，或者调用其 Stop 方法。

实例 189　动画显示窗体

光盘位置：光盘\MR\08\189　　中级　　趣味指数：★★★★

实例说明

当用户启动程序后，普通的程序窗体都是瞬间显示到屏幕上，这样未免有些生硬。如果窗体能够慢慢地展现在用户面前，将会是什么样的效果呢？本实例设计的是一个动画显示的窗体，该程序运行后，窗体是慢慢地以拉伸的效果显示到用户的面前；窗体关闭时，也是一样慢慢地消失。实例运行效果如图 8.16 所示。

图 8.16　动画显示窗体

关键技术

本实例实现时主要用到了 API 函数 AnimateWindow，下面对其进行详细讲解。

AnimateWindow 函数主要用来实现窗体的动画效果。语法格式如下：

```
[DllImportAttribute("user32.dll")]
private static extern bool AnimateWindow(IntPtr hwnd, int dwTime, int dwFlags);
```

参数说明

❶ hwnd：IntPtr，窗口句柄。

❷ dwTime：动画的持续时间，数值越大动画效果的时间就越长。

❸ dwFlags：动画效果类型选项，其常量值及说明如表 8.5 所示。

表 8.5 dwFlags 参数的常量值及说明

常　量	值	说　明
AW_SLIDE	0x00040000	使用滑动类型。默认则为滚动动画类型。当使用 AW_CENTER 标志时，这个标志就被忽略
AW_ACTIVATE	0x00020000	激活窗口。在使用 AW_HIDE 标志后不要使用这个标志
AW_BLEND	0x00080000	使用淡入效果。只有当 hWnd 为顶层窗口时才可以使用此标志
AW_HIDE	0x00010000	隐藏窗口，默认则显示窗口
AW_CENTER	0x00000010	若使用 AW_HIDE 标志，则使窗口向内重叠；若未使用 AW_HIDE 标志，则使窗口向外扩展
AW_HOR_POSITIVE	0x00000001	自左向右显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略
AW_HOR_NEGATIVE	0x00000002	自右向左显示窗口。当使用 AW_CENTER 标志时该标志被忽略
AW_VER_POSITIVE	0x00000004	自顶向下显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略
AW_VER_NEGATIVE	0x00000008	自下向上显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CartoonForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示窗体背景图片。

（3）程序的主要代码如下：

```
public Frm_Main()
{
    InitializeComponent();
    AnimateWindow(this.Handle, 3000, AW_SLIDE + AW_VER_NEGATIVE);                    //开始窗体动画
}

[DllImportAttribute("user32.dll")]
private static extern bool AnimateWindow(IntPtr hwnd, int dwTime, int dwFlags);    //重写 API 函数，用来执行窗体动画显示操作
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    AnimateWindow(this.Handle, 3000, AW_SLIDE + AW_VER_NEGATIVE + AW_HIDE);       //结束窗体动画
}
```

秘笈心法

心法领悟 189：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 190　制作类似 Office 的提示精灵

光盘位置：光盘\MR\08\190　　中级　　趣味指数：★★★★☆

实例说明

用过 Office 的用户都知道，Office 助手是一个非常实用、漂亮的小工具，有了它即使对 Office 不太熟悉的用户也可以操作自如。本实例将使用 C#制作一个类似 Office 的提示精灵。实例运行效果如图 8.17 所示。

关键技术

本实例在制作类似 Office 的提示精灵时，主要用到了 Microsoft Agent Control 2.0 COM 组件，下面对如何将 Microsoft Agent Control 2.0 组件添加到工具箱中进行详细讲解。

在 Visual Studio 2012 开发环境的工具箱中，单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令，弹出“选择工具箱项”对话框，如图 8.18 所示。在该对话框中切换到“COM 组件”选项卡，在列表中选中 Microsoft Agent Control 2.0 复选框，单击“确定”按钮，即可将 Microsoft Agent Control 2.0 组件添加到工具箱中。

图 8.17　制作类似 Office 的提示精灵

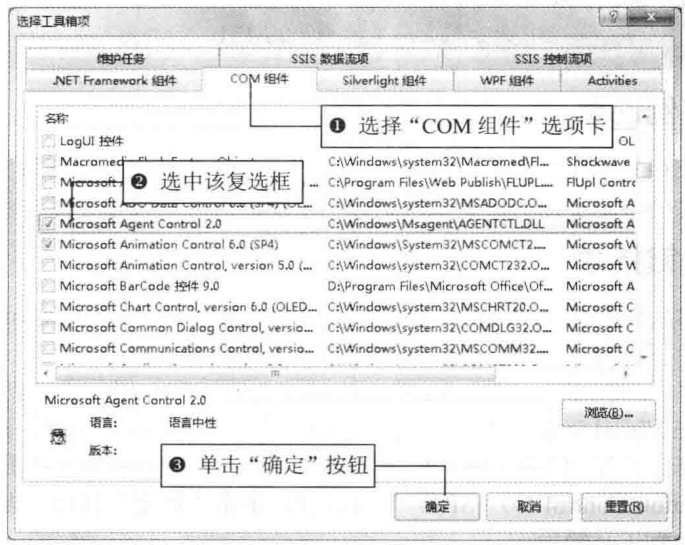


图 8.18　“选择工具箱项”对话框

说明：将 Microsoft Agent Control 2.0 组件添加到工具箱中后，即可像使用 Windows 标准控件一样使用它。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OfficeAgent。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来选择提示精灵的动作；添加一个 Microsoft Agent Control 2.0 组件，用来显示 Office 提示精灵的表情。

（3）程序的主要代码如下：

```
IAgentCtlCharacterEx ICCE;                                          //定义一个类 IAgentCtlCharacterEx 对象
IAgentCtlRequest ICR;                                               //定义一个类 IAgentCtlRequest 对象
//定义一个字符串数组，用来存储精灵的各种动作
string[] strAgents = new string[10] { "Acknowledge", "LookDown", "Sad", "Alert", "LookDownBlink", "Search", "Announce", "LookUp", "Think",
"Blink" };
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < strAgents.Length; i++)                      //循环遍历
    {
        listBox1.Items.Add(strAgents[i]);                           //向控件 listBox1 中添加字符串数组中的内容
    }
    ICR = axAgent1.Characters.Load("merlin", "merlin.acs");         //加载指定文件
    ICCE = axAgent1.Characters.Character("merlin");                 //设定模拟 Office 助手的表情
    ICCE.Show(0);                                                   //显示模拟 Office 助手的表情
}
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    ICCE.StopAll("");                                               //停止所有模拟 Office 助手的表情
    ICCE.Play(strAgents[listBox1.SelectedIndex]);                   //显示控件 listBox1 中选定的表情
}
```

秘笈心法

心法领悟 190：ListBox 控件的使用。

ListBox 控件又称为列表控件，主要用于显示一个列表，用户可以从中选择一项或多项，如果选项总数超出可以显示的项数，则控件会自动添加滚动条。

实例 191 动画形式的程序界面

光盘位置：光盘\MR\08\191　　中级　　趣味指数：★★★★☆

实例说明

在很多程序界面中，都是以菜单或工具栏的形式显示窗体界面，这种显示方式是以静止状态显示的，界面不够生动，本实例将讲解如何制作一个动画形式的程序界面。实例运行效果如图 8.19 所示。

关键技术

本实例在实现动画形式的程序界面时，主要用到了 Microsoft Animation Control 6.0（SP4）COM 组件的 Open 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）将 Microsoft Animation Control 6.0（SP4）组件添加到工具箱中

在 Visual Studio 2012 开发环境的工具箱中单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令，弹出“选择工具箱项”对话框，如图 8.20 所示。在该对话框中切换到“COM 组件”选项卡，在列表中选中 Microsoft Animation Control 6.0（SP4）”复选框，单击“确定”按钮，即可将 Microsoft Animation Control 6.0（SP4）组件添加到工具箱中。

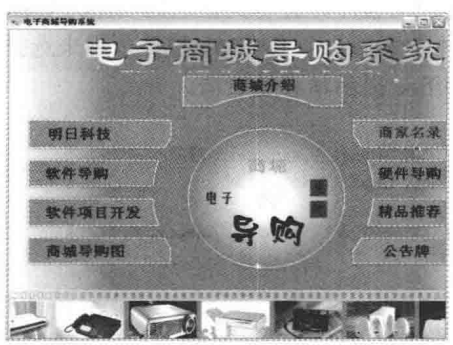


图 8.19　动画形式的程序界面

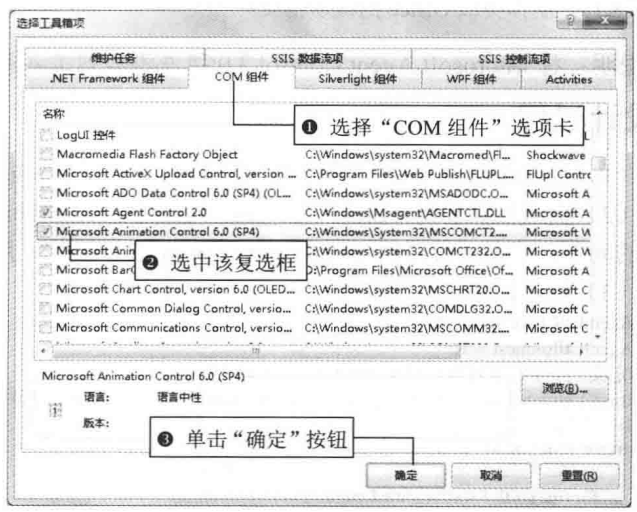


图 8.20　“选择工具箱项”对话框

（2）AxAnimation 组件的 Open 方法

AxAnimation 组件的 Open 方法用来播放动画文件，语法格式如下：

```
public void AxAnimation.Open(string bstrFilename)
```

参数说明

bstrFilename：将要播放的文件名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LeadWindows。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 Microsoft Animation Control 6.0（SP4）组件，用来播放 AVI 动画。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    axAnimation1.Open("Electron.avi");                    //播放动画文件 Electron.avi
    axAnimation2.Open("zybiao.avi");                      //播放动画文件 zybiao.avi
    axAnimation3.Open("gd.avi");                          //播放动画文件 gd.avi
}
```

秘笈心法

心法领悟 191：Microsoft Animation Control 6.0（SP4）组件的使用范围。

Microsoft Animation Control 6.0（SP4）组件只能用在 Windows XP、Windows 2000 及其早期的系统中，在 Windows 7 或者更高版本的系统中不能再使用。

8.5　特色程序界面

实例 192	仿 QQ 抽屉式窗体	高级
	光盘位置：光盘\MR\08\192	趣味指数：★★★★

实例说明

QQ 软件对于绝大多数的人来说再熟悉不过了，它以使用方便、界面美观及功能完善而著称。本实例仿照 QQ 软件界面的基本操作设计了一个抽屉式的窗体，实例运行效果如图 8.21 所示。在该窗体中单击任意按钮，程序将显示被单击按钮对应的列表，同时隐藏其他两个按钮对应的列表；用鼠标拖曳该窗体到屏幕的任意边缘，窗体会自动隐藏到该边缘内，当鼠标划过隐藏窗体的边缘时，窗体会显示出来；当鼠标离开窗体时，窗体再次被隐藏。

图 8.21　仿 QQ 抽屉式窗体

关键技术

本实例实现时主要用到了 API 函数 WindowFromPoint 和 GetParent，下面分别对它们进行详细说明。

（1）WindowFromPoint 函数

该函数用于获得包含指定点坐标的窗口的句柄。语法格式如下：

```
[DllImport("user32.dll")]                                 //需要引入 user32.dll 动态链接库
public static extern int WindowFromPoint(int xPoint, int yPoint)   //获得包含指定点坐标的窗口的句柄
```

参数说明

❶ xPoint：被检测点的横坐标。

❷ yPoint：被检测点的纵坐标。

❸ 返回值：为包含指定点坐标的窗口的句柄，若包含指定点坐标的窗口不存在，则返回值为 null；若该坐标对应的点在静态文本控件之上，则返回值是在该静态文本控件下面的窗口的句柄。

（2）GetParent 函数

该函数用于获取指定句柄的父级。语法格式如下：

```
[DllImport("user32.dll", ExactSpelling = true, CharSet = CharSet.Auto)]   //需要引入 user32.dll 动态链接库
public static extern IntPtr GetParent(IntPtr hWnd);                       //获取指定句柄的父级
```

参数说明

❶ hWnd：指定窗口的句柄。

❷ 返回值：若果函数执行成功，则返回指定窗口句柄的父级；若函数执行失败，则返回值为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 HideToolBar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 Button 控件，分别用来执行显示好友列表、显示陌生人和显示黑名单操作；添加一个 ListView 控件，用来显示好友、陌生人和黑名单列表；添加一个 ImageList 组件，用来为列表视图提供图标；添加两个 Timer 组件，分别用来判断窗体是否进入屏幕边界区域和控制窗体的隐藏。

（3）程序的主要代码如下：

```
private void JudgeWinMouPosition_Tick(object sender, EventArgs e)
{
    if (this.Top < 3)                                                    //当本窗体距屏幕的上边距小于 3px 时
    {
        if (this.Handle == MouseNowPosition(Cursor.Position.X, Cursor.Position.Y))   //当鼠标在该窗体上时
        {
            WindowFlag = 1;                                              //设定当前的窗体状态
            HideWindow.Enabled = false;                                  //设定计时器 HideWindow 为不可用状态
            this.Top = 0;                                                //设定窗体上边缘与容器工作区上边缘之间的距离
        }
        else                                                             //当鼠标没在窗体上时
        {
            WindowFlag = 1;                                              //设定当前的窗体状态
            HideWindow.Enabled = true;                                   //启动计时器 HideWindow
        }
    }                                                                    //当本窗体距屏幕的上边距大于 3px 时
    else
    {
        //当本窗体在屏幕的最左端或者最右端、最下端时
        if (this.Left < 3 || (this.Left + this.Width) > (GetSystemMetrics(0) - 3) || (this.Top + this.Height) > (Screen.AllScreens[0].Bounds.Height - 3))
        {
            if (this.Left < 3)                                           //当窗体处于屏幕左侧时
               {
                 //当鼠标在该窗体上时
                 if (this.Handle == MouseNowPosition(Cursor.Position.X, Cursor.Position.Y))
                 {
                     this.Height = Screen.AllScreens[0].Bounds.Height - 40;
                     this.Top = 3;
                     WindowFlag = 2;                                     //设定当前的窗体状态
                     HideWindow.Enabled = false;                         //设定计时器 HideWindow 为不可用状态
                     this.Left = 0;                                      //设定窗体的左边缘与容器工作区的左边缘之间的距离
                 }
                 else                                                    //当鼠标没在该窗体上时
                 {
                     WindowFlag = 2;                                     //设定当前的窗体状态
                     HideWindow.Enabled = true;                          //设定计时器 HideWindow 为可用状态
                 }
            }
            if ((this.Left + this.Width) > (GetSystemMetrics(0) - 3))    //当窗体处于屏幕的最右侧时
               {
                 //当鼠标处于窗体上时
                 if (this.Handle == MouseNowPosition(Cursor.Position.X, Cursor.Position.Y))
                 {
                     this.Height = Screen.AllScreens[0].Bounds.Height - 40;
                     this.Top = 3;
                     WindowFlag = 3;                                     //设定当前的窗体状态
                     HideWindow.Enabled = false;                         //设定计时器 HideWindow 为不可用状态
                     this.Left = GetSystemMetrics(0) - this.Width;       //设定该窗体与容器工作区左边缘之间的距离
                 }
                 else                                                    //当鼠标离开窗体时
                 {
```

```
                    WindowFlag = 3;                                          //设定当前的窗体状态
                    HideWindow.Enabled = true;                               //设定计时器 HideWindow 为可用状态
                }
            }
            if ((this.Top + this.Height) > (Screen.AllScreens[0].Bounds.Height - 3))    //当窗体距屏幕最下端的距离小于 3px 时
                {
                //当鼠标在该窗体上时
                if (this.Handle == MouseNowPosition(Cursor.Position.X, Cursor.Position.Y))
                {
                    WindowFlag = 4;                                          //设定当前的窗体状态
                    HideWindow.Enabled = false;                              //设定计时器 HideWindow 为不可用状态
                    this.Top = Screen.AllScreens[0].Bounds.Height - this.Height;    //设定该窗体与容器工作区上边缘之间的距离
                }
                else
                {
                    if ((this.Left > this.Width + 3) && (GetSystemMetrics(0) - this.Right) > 3)
                    {
                        WindowFlag = 4;                                      //设定当前的窗体状态
                        HideWindow.Enabled = true;                           //设定计时器 HideWindow 为可用状态
                    }
                }
            }
        }
    }
}
private void HideWindow_Tick(object sender, EventArgs e)
{
    switch (Convert.ToInt32(WindowFlag.ToString()))                          //判断当前窗体处于哪个状态
    {
        case 1:                                                              //当窗体处于最上端时
            if (this.Top < 3)                                                //当窗体与容器工作区的上边缘的距离小于 5px 时
                this.Top = -(this.Height - 2);                               //设定当前窗体距容器工作区上边缘的值
            break;
        case 2:                                                              //当窗体处于最左端时
            if (this.Left < 3)                                               //当窗体与容器工作区的左边缘的距离小于 5px 时
                this.Left = -(this.Width - 2);                               //设定当前窗体距容器工作区左边缘的值
            break;
        case 3:                                                              //当窗体处于最右端时
            if ((this.Left + this.Width) > (GetSystemMetrics(0) - 3) )       //当窗体与容器工作区的右边缘的距离小于 5px 时
                this.Left = GetSystemMetrics(0) - 2;                         //设定当前窗体距容器工作区左边缘的值
            break;
        case 4:                                                              //当窗体处于最底端时
            if (this.Bottom > Screen.AllScreens[0].Bounds.Height - 3 )       //当窗体与容器工作区的下边缘的距离小于 5px 时
                this.Top = Screen.AllScreens[0].Bounds.Height - 5;           //设定当前窗体距容器工作区上边缘之间的距离
            break;
    }
}
```

秘笈心法

心法领悟 192：为本实例添加快速启动图标。

腾讯公司 QQ 软件启动后，可以在桌面的右下角显示一个快速启动图标，双击该图标可以显示隐藏的窗口，右击该图标可以弹出快捷菜单。本实例通过在窗口中添加 NotifyIcon 控件也可实现这种功能。

实例 193	设计软件启动界面	中级
	光盘位置：光盘\MR\08\193	趣味指数：★★★★

实例说明

启动欢迎窗体是系统登录时常见的一个功能，主要用来等待系统加载资源文件所需要的时间，本实例将制作一个软件启动界面。实例运行效果如图 8.22 所示。

图 8.22　设计软件启动界面

关键技术

本实例实现时主要用到了 Timer 组件的 Interval 属性、Start 方法和窗体的 FormBorderStyle 属性、BackgroundImage 属性，下面对本实例用到的关键技术进行详细讲解。

（1）Timer 组件的 Start 方法

该方法用来启动计时器。语法格式如下：

```
public void Start()
```

说明：① 也可以通过将 Timer 组件的 Enabled 属性设置为 true 来启动计时器。

② 关于 Timer 组件的 Interval 属性的详细讲解请参见实例 188 中的关键技术。

（2）窗体的 FormBorderStyle 属性

该属性用来获取或设置窗体的边框样式。语法格式如下：

```
public FormBorderStyle FormBorderStyle { get; set; }
```

参数说明

属性值：FormBorderStyle 枚举值之一，表示窗体的起始位置，默认为 FormBorderStyle.Sizable。FormBorderStyle 枚举值及说明如表 8.6 所示。

表 8.6　FormBorderStyle 枚举值及说明

枚 举 值	说　明
None	无边框
FixedSingle	固定的单行边框
Fixed3D	固定的三维边框
FixedDialog	固定的对话框样式的粗边框
Sizable	可调整大小的边框
FixedToolWindow	不可调整大小的工具窗口边框。工具窗口不会显示在任务栏中，也不会显示在当用户按 Alt+Tab 快捷键时出现的窗口中
SizableToolWindow	可调整大小的工具窗口边框。工具窗口不会显示在任务栏中，也不会显示在当用户按 Alt+Tab 快捷键时出现的窗口中

说明：尽管指定 FormBorderStyle 属性为 FixedToolWindow 枚举值的窗体通常不显示在任务栏中，还是必须确保 ShowInTaskbar 属性设置为 false，因为其默认值为 true。

（3）窗体的 BackgroundImage 属性

该属性用来获取或设置在窗体中显示的背景图像。语法格式如下：

```
public virtual Image BackgroundImage { get; set; }
```

参数说明

属性值：一个 Image 图像，表示在窗体的背景中显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StartForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）在项目中添加一个 Frm_Start 窗体，用来作为启动欢迎窗体，在该窗体中添加一个 Timer 计时器组件，用来控制启动窗体的显示时间。

（4）程序的主要代码如下：

```
private void Frm_Start_Load(object sender, EventArgs e)
{
    this.FormBorderStyle = FormBorderStyle.None;                    //设置启动窗体为无标题栏窗体
    this.BackgroundImage = Image.FromFile("start.jpg");             //设置启动窗体的背景图片
    this.BackgroundImageLayout = ImageLayout.Stretch;               //设置图片自动适应窗体大小
    this.timer1.Start();                                            //启动计时器
    this.timer1.Interval = 10000;                                   //设置启动窗体的停留时间
}
```

秘笈心法

心法领悟 193：如何隐藏窗体的标题栏？

开发人员可以通过设置窗体的 FormBorderStyle 属性为 None 来实现隐藏窗体标题栏功能。

实例 194　以树形显示的程序界面

光盘位置：光盘\MR\08\194　　中级　　趣味指数：★★★★

实例说明

以树形来显示程序的菜单，可以更直观、更快捷地对窗体进行操作，而且树型菜单比菜单栏更加美观实用。本实例将详细介绍树形程序界面的设计方法。实例运行效果如图 8.23 所示。

图 8.23　以树形显示的程序界面

关键技术

本实例在制作以树形显示的程序界面时，主要用到了 TreeView 控件及其 ExpandAll 方法，下面对其进行详细讲解。

TreeView 控件又称为树控件，它可以为用户显示节点层次结构，而每个节点又可以包含子节点，包含子节点的节点叫父节点，其效果就像在 Windows 操作系统的 Windows 资源管理器功能的左窗口中显示文件和文件夹一样。TreeView 控件的 ExpandAll 方法用来展开所有树节点，语法格式如下：

```
public void ExpandAll()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TreeForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来设计菜单栏；添加一个 PictureBox 控件，用来显示主窗体背景图片；添加一个 TreevVew 控件，用来设计左侧树形导航。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    treeView1.ExpandAll();                                    //展开所有节点
}
```

秘笈心法

心法领悟 194：TreeView 控件的使用技巧。

TreeView 控件经常用来设计 Windows 窗体的左侧导航菜单。

实例 195　动态按钮的窗体界面

光盘位置：光盘\MR\08\195　　中级　　趣味指数：★★★★

实例说明

在窗体界面中，通常以按钮来代替菜单栏的功能，这种形式虽然给用户一种直观、界面风格各异的感觉，但通常按钮都是以静止的形式显示，当光标移到按钮上时，可以使按钮上的图片和文字说明动态化，使用户快捷地找到所选按钮，这时就需要一个动态按钮的窗体界面。本实例将详细介绍动态按钮窗体界面的制作过程。实例运行效果如图 8.24 所示。

图 8.24　动态按钮的窗体界面

关键技术

本实例在实现过程中，首先需要通过 Button 控件中的 Image 属性设置图片，然后将 Button 控件的 ImageAlign 属性设置为 MiddleCenter，以便使图片居中。下面对本实例中用到的关键技术进行详细讲解。

（1）Image 属性

该属性用来获取或设置显示在按钮控件上的图像。语法格式如下：

```
public Image Image { get; set; }
```

参数说明

属性值：按钮控件上显示的 Image 图像，默认值为空引用。

说明：本实例在设置 Button 控件的动态图片时，必须在相应控件的 MouseMove 事件中设置。

（2）ImageAlign 属性

该属性用来获取或设置按钮控件上的图像对齐方式。语法格式如下：

```
public ContentAlignment ImageAlign { get; set; }
```

参数说明

属性值：ContentAlignment 枚举值之一，默认值为 MiddleCenter。ContentAlignment 枚举值及说明如表 8.7 所示。

表 8.7　ContentAlignment 枚举值及说明

枚 举 值	说　明
TopLeft	内容在垂直方向上顶部对齐，在水平方向上左边对齐
TopCenter	内容在垂直方向上顶部对齐，在水平方向上居中对齐
TopRight	内容在垂直方向上顶部对齐，在水平方向上右边对齐
MiddleLeft	内容在垂直方向上中间对齐，在水平方向上左边对齐
MiddleCenter	内容在垂直方向上中间对齐，在水平方向上居中对齐
MiddleRight	内容在垂直方向上中间对齐，在水平方向上右边对齐
BottomLeft	内容在垂直方向上底边对齐，在水平方向上左边对齐
BottomCenter	内容在垂直方向上底边对齐，在水平方向上居中对齐
BottomRight	内容在垂直方向上底边对齐，在水平方向上右边对齐

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DynamicButtonForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示主窗体背景图片；添加 6 个 Button 控件，并分别为每个 Button 控件设置相应的图片和文字，这些 Button 控件用来作为窗体界面中的动态按钮。

（3）程序的主要代码如下：

```
private void button1_MouseMove(object sender, MouseEventArgs e)
{
    button1.ImageAlign = ContentAlignment.MiddleLeft;          //鼠标移动到 button1 时改变图片位置
}
private void button1_MouseLeave(object sender, EventArgs e)
{
    button1.ImageAlign = ContentAlignment.MiddleCenter;        //鼠标移动到 button1 时改变图片位置
}
```

秘笈心法

心法领悟 195：Button 控件的重要性。

Button 控件是程序开发中最常用到的一个控件，几乎窗体中的每个功能都需要直接或者间接地通过 Button 控件来实现，所以读者一定要熟练掌握它的使用。

第 9 章

窗体控制技术

- 窗体控制技术
- 其他窗体技术

9.1　窗体常用操作

实例 196	以全屏方式显示程序窗体 光盘位置：光盘\MR\09\196	初级 趣味指数：★★★☆

实例说明

全屏显示窗体在实际开发中经常用到，如人们常用的播放器或者网页视频等，全屏使得窗体能够显示得足够大，从而使人们更清楚地看到窗体中显示的内容。本实例实现了以全屏方式显示程序窗体的功能，运行本实例，单击“全屏”按钮，窗体即可全屏显示；单击“退出全屏”按钮，窗体即可恢复到原来状态。实例运行效果如图 9.1 所示。

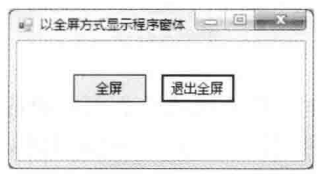


图 9.1　以全屏方式显示程序窗体

关键技术

本实例在实现时主要用到了窗体的 WindowState 属性和 FormBorderStyle 属性，下面对本实例用到的关键技术进行详细讲解。

WindowState 属性用来获取或设置窗体的窗口状态。语法格式如下：

```
public FormWindowState WindowState { get; set; }
```

参数说明

属性值：FormWindowState 枚举值之一，表示窗体的窗口状态。FormWindowState 枚举值及说明如表 9.1 所示。

表 9.1　FormWindowState 枚举值及说明

枚 举 值	描　述
Maximized	最大化的窗口
Minimized	最小化的窗口
Normal	默认大小的窗口

说明：（1）尽管指定 FormBorderStyle 属性为 FixedToolWindow 枚举值的窗体通常不显示在任务栏中，还是必须确保 ShowInTaskbar 属性设置为 false，因为其默认值为 true。

（2）关于 FormBorderStyle 属性的详细讲解请参见实例 193 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FullScreenForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来使窗体进入全屏模式和退出全屏模式。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.FormBorderStyle = FormBorderStyle.None;                    //设置窗体为无边框样式
```

```
    this.WindowState = FormWindowState.Maximized;                    //最大化显示窗体
}
private void button2_Click(object sender, EventArgs e)
{
    this.FormBorderStyle = FormBorderStyle.Sizable;                  //设置窗体为有边框样式
    this.WindowState = FormWindowState.Normal;                       //正常显示窗体
}
```

秘笈心法

心法领悟 196：全屏窗体的应用场合。

一般在开发视频播放软件时，需要用到全屏功能。

实例 197　通过子窗体刷新父窗体　　中级

光盘位置：光盘\MR\09\197　　趣味指数：★★★★★

实例说明

在进销存管理系统中添加销售单信息时，每个销售单都可能对应多种商品，而且在向销售单中添加商品时，一般都是在新弹出的窗体中选择商品，这时就涉及通过子窗体刷新父窗体的问题。本实例将使用 C#语言实现通过子窗体刷新父窗体的功能。实例运行效果如图 9.2 所示。

图 9.2　通过子窗体刷新父窗体

关键技术

本实例在实现通过子窗体刷新父窗体时，主要通过在自定义事件中执行数据绑定来对主窗体进行刷新，即当子窗体产生更新操作时，通过子窗体的一个方法触发主窗体中对应的处理事件，这个过程主要用到了 EventHandler 事件，下面对该事件进行详细讲解。

EventHandler 事件主要是通过 EventHandler 委托来实现的，该委托表示处理不包含事件数据的事件的方法。语法格式如下：

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public delegate void EventHandler(Object sender,EventArgs e)
```

参数说明

❶ sender：事件源。

❷ e：不包含任何事件数据的 EventArgs 事件数据。

例如，本实例中，通过使用 EventHandler 事件为子窗体添加一个事件处理程序，以便能够刷新父窗体中的

数据。代码如下：

```
public event EventHandler UpdateDataGridView = null;                                    //定义一个处理更新 DataGridView 控件内容的方法
BabyWindow.UpdateDataGridView += new EventHandler(BabyWindow_UpdateDataGridView);//添加事件
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RefreshFormByChildForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 DataGridView 控件，用来显示数据库中的数据。

（3）在该项目中添加一个新的 Windows 窗体，并将其命名为 Frm_Child.cs，在该窗体中添加 4 个 TextBox 控件，分别用来输入编号、姓名、电话和住址；添加一个 ComboBox 控件，用来选择要删除的编号；添加 3 个 Button 控件，分别用来执行添加信息、清空文本和删除信息操作。

（4）程序的主要代码如下：

```
void BabyWindow_UpdateDataGridView(object sender,EventArgs e)
{
    if(Frm_Child.GlobalFlag == false)                                                    //当单击“删除”按钮时
    {
        if(ConnPubs.State == ConnectionState.Closed)                                     //当数据库处于断开状态时
        {
            ConnPubs.Open();                                                             //打开数据库的连接
        }
        string AfreshString = "delete tb_User where userID=" + Frm_Child.DeleteID.Trim(); //定义一个删除数据的字符串
        PersonalInformation = new SqlCommand(AfreshString,ConnPubs);                    //执行删除数据库字段
        PersonalInformation.ExecuteNonQuery();                                           //执行 SQL 语句并返回受影响的行数
        ConnPubs.Close();                                                                //关闭数据库
        DisplayData();                                                                   //显示数据库更新后的内容
        MessageBox.Show("数据删除成功！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
    }
    else
    {
        if(ConnPubs.State == ConnectionState.Closed)                                     //当数据库处于关闭状态时
        {
            ConnPubs.Open();                                                             //打开数据库
        }
        string InsertString = "insert into tb_User values('" + Frm_Child.idContent + "','" + Frm_Child.nameContent + "','" + Frm_Child.phoneContent
+ "','" + Frm_Child.addressContent + "')";                                               //定义一个插入数据的字符串变量
        PersonalInformation = new SqlCommand(InsertString,ConnPubs);                    //执行插入数据库字段
        PersonalInformation.ExecuteNonQuery();                                           //执行 SQL 语句并返回受影响的行数
        ConnPubs.Close();                                                                //关闭数据库
        DisplayData();                                                                   //显示更新后的数据
        MessageBox.Show("数据添加成功！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
    }
}
```

秘笈心法

心法领悟 197：如何执行 SQL 语句。

在 C#中执行 SQL 语句时，可以使用 SqlCommand 类，该类主要用于向 SQL Server 数据库发送 SQL 语句，其位于 System.Data.SqlClient 命名空间中。

实例 198	拖动无边框窗体	中级
	光盘位置：光盘\MR\09\198	趣味指数：★★★★

实例说明

一般情况下，窗体都包含标题栏、菜单栏、工具栏和状态栏等区域，当拖动窗体时直接在标题栏中按住鼠

标左键不放即可实现拖动操作。当做浮动窗体时，如果包含窗体边框，那么界面给使用者的感觉将很不友好，因此浮动窗体没有边框，但对于这种没有边框的窗体，该如何进行拖放操作呢？本实例将带领读者一起来制作一个拖动无边框窗体的程序。实例运行效果如图 9.3 所示。

图 9.3　拖动无边框窗体

关键技术

本实例实现时主要用到了 Windows 的两个 API 函数，即 ReleaseCapture 和 SendMessage，下面分别对它们进行详细讲解。

（1）ReleaseCapture 函数

该函数用来释放被当前线程中某个窗口捕获的光标。语法格式如下：

```
[DllImport("user32.dll")]
public static extern bool ReleaseCapture();
```

说明：程序中使用系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间，下面再遇到类似的情况时将不再提示。

（2）SendMessage 函数

该函数用来向指定的窗体发送 Windows 消息。语法格式如下：

```
[DllImport("user32.dll")]
public static extern bool SendMessage(IntPtr hwdn,int wMsg,int mParam,int lParam);
```

SendMessage 函数语法中的参数及说明如表 9.2 所示。

表 9.2　SendMessage 函数语法中的参数及说明

参　数	描　述
hwdn	表示发送消息的目的窗口的句柄
wMsg	表示被发送的消息
mParam	取决于被发送的消息，表示附加的消息信息
lParam	取决于被发送的消息，表示附加的消息信息
返回值	表示处理是否成功

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DragNoFrameForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将该窗体的 FormBorderStyle 属性设置为 None。

（3）程序的主要代码如下：

```
#region 本程序中用到的 API 函数
[DllImport("user32.dll")]
public static extern bool ReleaseCapture();                    //用来释放被当前线程中某个窗口捕获的光标
//向指定的窗体发送 Windows 消息
[DllImport("user32.dll")]
public static extern bool SendMessage(IntPtr hwdn,int wMsg,int mParam,int lParam);
```

```
#endregion
private void Frm_Main_MouseDown(object sender, MouseEventArgs e)
{
    ReleaseCapture();                                          //用来释放被当前线程中某个窗口捕获的光标
    //向 Windows 发送拖动窗体的消息
    SendMessage(this.Handle, WM_SYSCOMMAND, SC_MOVE + HTCAPTION, 0);
}
```

秘笈心法

心法领悟 198：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 199　禁用窗口上的关闭按钮

光盘位置：光盘\MR\09\199　　初级　　趣味指数：★★★★

实例说明

一般情况下，在窗体的右上角都有最大化、最小化和关闭按钮，但在 MDI 窗体中，有时为了避免重复打开同一个窗口，需要禁用窗口上面的“关闭”按钮，本实例就实现了这样的功能。实例运行效果如图 9.4 所示。

图 9.4　禁用窗口上的关闭按钮

关键技术

本实例主要用到窗口处理方法 WndProc 的重写方法，在该方法内部截获单击关闭窗口的信息，从而实现禁用“关闭”按钮的功能。下面对 WndProc 方法进行详细介绍。

WndProc 方法主要用来处理 Windows 消息。语法格式如下：

```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags = SecurityPermissionFlag.UnmanagedCode)]
[SecurityPermissionAttribute(SecurityAction.InheritanceDemand, Flags = SecurityPermissionFlag.UnmanagedCode)]
protected virtual void WndProc(ref Message m)
```

参数说明

m：要处理的 Windows 消息。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StopCloseButton。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并为该窗体设置指定的背景图片。在 Frm_Main 窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏。

（3）程序的主要代码如下：

```
protected override void WndProc(ref Message m)
{
    const int WM_SYSCOMMAND = 0x0112;                          //定义将要截获的消息类型
    const int SC_CLOSE = 0xF060;                               //定义关闭按钮对应的消息值
    if ((m.Msg == WM_SYSCOMMAND) && ((int)m.WParam == SC_CLOSE))   //当鼠标单击“关闭”按钮时
    {
        return;                                                //直接返回，不进行处理
    }
    base.WndProc(ref m);                                       //传递下一条消息
}
```

秘笈心法

心法领悟 199：窗体界面设计的重要性。

Windows 窗体是可视化编程的基础，窗体界面设计的美观性将直接影响到用户的使用兴趣，因此它在程序的开发过程中是非常重要的。在设计窗体界面时，要遵循简单、大方、美观和易用 4 个基本原则，以便能够更好地吸引用户，并且方便用户的使用。

实例 200　向窗体中拖放图片并显示

光盘位置：光盘\MR\09\200　　初级　　趣味指数：★★★★☆

实例说明

当用户查看图片时，为了节省图片打开或关闭的操作，可以直接将图片拖放到窗体中进行显示，本实例使用 C#语言实现了这样的功能。运行本实例，将图片拖放到窗体中，即可方便地查看图片。实例运行效果如图 9.5 所示。

图 9.5　向窗体中拖放图片并显示

关键技术

本实例实现时主要用到了 DragEventArgs 类和 GetData 方法，下面分别对它们进行详细讲解。

（1）DragEventArgs 类

DragEventArgs 类为 DragDrop、DragEnter 或 DragOver 事件提供数据，其常用的属性及说明如表 9.3 所示。

表 9.3　DragEventArgs 类的常用属性及说明

属　性	说　明
AllowedEffect	获取拖动事件的发起方（或源）所允许的拖放操作
Data	获取 IDataObject，它包含与此事件关联的数据
Effect	获取或设置拖放操作中的目标放置效果
KeyState	获取 Shift、Ctrl 和 Alt 键的当前状态以及鼠标按钮的状态
X	获取鼠标指针在屏幕坐标系中的 x 坐标
Y	获取鼠标指针在屏幕坐标系中的 y 坐标

（2）GetData 方法

该方法检索与指定数据格式相关联的数据，并使用一个布尔值确定是否将数据转换成该格式。语法格式如下：

```
Object GetData (string format, bool autoConvert)
```

参数说明

❶ format：要检索的数据的格式，用 DataFormats 类成员进行设置。DataFormats 类的成员及说明如表 9.4 所示。

❷ autoConvert：将数据转换成指定格式，值为 true；反之，值为 false。

❸ 返回值：与指定格式关联的数据，或为空引用。

表 9.4　DataFormats 类的成员及说明

类 的 成 员	说　明
Bitmap	指定 Windows 位图格式
CommaSeparatedValue	指定以逗号分隔值（CSV）的格式，这是电子表格常用的交换格式
Dib	指定 Windows 与设备无关的位图（DIB）格式

续表

类 的 成 员	说　明
Dif	指定 Windows 数据交换格式（DIF），Windows 窗体不直接使用此格式
EnhancedMetafile	指定 Windows 增强型图元文件格式
FileDrop	指定 Windows 文件放置格式，Windows 窗体不直接使用此格式
MetafilePict	指定 Windows 图元文件格式，Windows 窗体不直接使用此格式
OemText	指定标准 Windows 原始设备制造商（OEM）文本格式

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DragImageToForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
/// <summary>
///在窗体背景中显示被拖放的图片
/// </summary>
/// <param Frm="Form">窗体</param>
/// <param e="DragEventArgs">DragDrop、DragEnter 或 DragOver 事件提供数据</param>
public void SetDragImageToFrm(Form Frm, DragEventArgs e)
{
    if (Var_Style == true)
    {
        e.Effect = DragDropEffects.Copy;                                    //设置拖放操作中的目标放置类型为复制
        String[] str_Drop = (String[])e.Data.GetData(DataFormats.FileDrop, true);//检索数据格式相关联的数据
        string tempstr;
        Bitmap bkImage;                                                     //定义 Bitmap 变量
        tempstr = str_Drop[0];                                              //获取拖放文件的目录
        try
        {
            bkImage = new Bitmap(tempstr);                                  //存储拖放的图片
            //根据图片设置窗体的大小
            Frm.Size = new System.Drawing.Size(bkImage.Width + 6, bkImage.Height + 33);
            Frm.BackgroundImage = bkImage;                                  //在窗体背景中显示图片
        }
        catch { }
    }
}
```

秘笈心法

心法领悟 200：如何设置窗体的背景图片？

设置窗体的背景图片时，需要使用窗体的 BackgroundImage 属性实现，该属性用来获取或设置在窗体中显示的背景图片；另外，也可以通过在“属性”窗口中设置 BackgroundImage 属性实现。

实例 201　使应用程序在进程中消失

光盘位置：光盘\MR\09\201　　初级　　趣味指数：★★★☆

实例说明

开发应用程序时，有的窗体可能需要隐藏，如从登录窗体进入主窗体时，通常都对登录窗体进行隐藏，这样一来，当关闭主窗体时，如果调用 Close 方法，可能就会只关闭主窗体，而应用程序其实并没有退出，本实例将在关闭窗体时彻底地关闭应用程序。实例运行效果如图 9.6 所示。

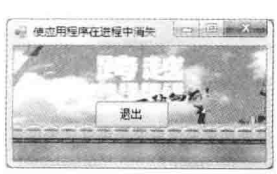


图 9.6　使应用程序在进程中消失

关键技术

本实例实现时主要用到了 Application 类的 Exit 方法，下面对其进行详细讲解。

Application 类提供 static 方法和属性以管理应用程序，例如，启动和停止应用程序、处理 Windows 消息的方法和获取应用程序信息的属性等，其 Exit 方法用来通知所有消息泵必须终止，并且在处理了消息以后关闭所有应用程序窗口，它有两种重载形式，分别如下：

```
public static void Exit()
public static void Exit(CancelEventArgs e)
```

参数说明

e：返回应用程序中的任何 Form 窗体是否已经取消退出。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisposeForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并为该窗体设置指定的背景图片。在 Frm_Main 窗体中添加一个 Button 控件，用来执行退出应用程序操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Application.Exit();                                    //退出应用程序
}
```

秘笈心法

心法领悟 201：什么是进程？

每个正在操作系统上运行的应用程序都是一个进程，一个进程可以包括一个或多个线程，而线程是操作系统分配处理器时间的基本单元，在进程中可以有多个线程同时执行代码。

9.2 其他窗体技术

实例 202	在关闭窗体时弹出确认对话框	初级
	光盘位置：光盘\MR\09\202	趣味指数：★★★☆

实例说明

用户对程序进行操作时，难免会有错误操作的情况，如不小心关闭程序，如果尚有许多资料没有保存，那么损失将非常严重，所以最好使程序具有灵活的交互性。人机交互过程一般都是通过对话框来实现的，对话框中有提示信息，并且提供按钮让用户选择，如“是”或“否”。这样用户就能够对所做的动作进行确认。正如前面所说的不小心关闭程序，如果在关闭程序之前提示用户将要关闭程序，并且提示用户选择是否继续下去，这样就大大减少了误操作现象。本实例就实现了在关闭窗体时弹出确认对话框的功能，实例运行效果如图 9.7 所示。

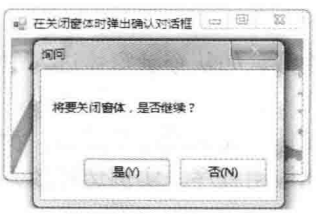


图 9.7 在关闭窗体时弹出确认对话框

关键技术

本实例实现时主要用到了窗体的 FormClosing 事件、FormClosingEventArgs 类的 Cancel 属性以及 MessageBox 参数的返回值，下面对本实例用到的关键技术进行详细讲解。

（1）窗体的 FormClosing 事件

FormClosing 事件在关闭窗体前发生。语法格式如下：

```
public event FormClosingEventHandler FormClosing
```

（2）FormClosingEventArgs 类的 Cancel 属性

FormClosingEventArgs 类用来为 FormClosing 事件提供数据，其 Cancel 属性用来获取或设置指示是否应取消事件的值，该属性的语法格式如下：

```
public bool Cancel { get; set; }
```

参数说明

属性值：如果取消事件，则为 true；否则为 false。

（3）MessageBox 参数的返回值

获取 MessageBox 参数的返回值时需要用到 MessageBox 类的 Show 方法，该方法用来显示消息框，它有多种重载形式，最常用的语法格式如下：

```
public static MessageBoxResult Show(string messageBoxText,string caption,MessageBoxButton button)
```

Show 方法中的参数及说明如表 9.5 所示。

表 9.5　Show 方法中的参数及说明

参　数	描　述
messageBoxText	指定要显示的文本
caption	指定要显示的标题栏标题
button	一个 MessageBoxButton 枚举值，用于指定要显示哪个按钮或哪些按钮
返回值	一个 MessageBoxResult 值，用于指定用户单击了哪个消息框按钮

例如，本实例中判断 MessageBox 参数的返回值是否为“是”的代码如下：

```
if (MessageBox.Show("将要关闭窗体，是否继续？", "询问", MessageBoxButtons.YesNo) == DialogResult.Yes)
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShowDialogByClose。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并为该窗体设置指定的背景图片。

（3）程序的主要代码如下：

```
private void Frm_Main_FormClosing(object sender, FormClosingEventArgs e)          //触发窗体关闭事件
{
    //判断是否单击了“是”按钮
    if (MessageBox.Show("将要关闭窗体，是否继续？", "询问", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        e.Cancel = false;                                                          //关闭窗体
    }
    else
    {
        e.Cancel = true;                                                           //取消事件的执行
    }
}
```

秘笈心法

心法领悟 202：FormClosing 事件的妙用。

开发网络程序或多线程程序时，可以在窗体的 FormClosing 事件中关闭网络连接或多线程，以便释放网络连接或多线程所占用的系统资源。

实例 203 隐藏应用程序在任务栏中的图标

光盘位置：光盘\MR\09\203　　初级　　趣味指数：★★★★☆

实例说明

用过腾讯 QQ 软件的用户大概都知道，腾讯 QQ 的登录窗体不显示在任务栏中，只有登录主窗体之后才会显示，那么使用 C#可以实现这样的功能吗？答案是肯定的。本实例就使用 C#制作了一个隐藏应用程序在任务栏中的图标的程序。实例运行效果如图 9.8 所示。

图 9.8　隐藏应用程序在任务栏中的图标

关键技术

本实例主要是通过设置窗体的 ShowInTaskbar 属性为 false 来实现的，下面对窗体的 ShowInTaskbar 属性进行详细讲解。

ShowInTaskbar 属性用来获取或设置一个值，该值指示是否在 Windows 任务栏中显示窗体。语法格式如下：

```
public bool ShowInTaskbar { get; set; }
```

参数说明

属性值：如果为 true，则运行时在 Windows 任务栏中显示窗体；否则为 false。默认为 true。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 HideIconInTaskBar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并为该窗体设置指定的背景图片。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.ShowInTaskbar = false;                    //隐藏窗体在任务栏中的显示
}
```

秘笈心法

心法领悟 203：通过“属性”窗口更快地设置窗体。

当在任务栏中隐藏窗体时，可以直接在“属性”窗口中进行设置。步骤是，选中窗体，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，弹出“属性”窗口，然后找到 ShowInTaskbar 属性，将其设置为 false 即可。

实例 204 实现动态系统托盘图标

光盘位置：光盘\MR\09\204　　高级　　趣味指数：★★★★★

实例说明

当在 QQ 上收到消息时，任务栏的右下端会有一个图标在不停地闪烁，单击它即可打开信息浏览。本实例模拟信息提示功能，运行本实例，效果如图 9.9 所示。在主窗体中单击“发送消息”按钮，出现图标进行闪烁，

如图 9.10 所示；单击“停止闪动”按钮，即可停止图标的闪烁，如图 9.11 所示。

图 9.9　实现动态系统托盘图标

图 9.10　闪动图标

图 9.11　禁止图标

关键技术

本实例在实现网络数据传输功能时用到了 TcpListener、TcpClient 和 NetworkStream 类，而在控制托盘图标的闪动时用到了 Timer 组件。下面对本实例中用到的关键技术进行详细讲解。

（1）TcpListener 类

该类用来从 TCP 网络客户端侦听连接，其常用方法及说明如表 9.6 所示。

表 9.6　TcpListener 类的常用方法及说明

属　性	说　明
AcceptTcpClient	接受挂起的连接请求
Start	开始侦听传入的连接请求
Stop	关闭侦听器

（2）TcpClient 类

该类用来为 TCP 网络服务提供客户端连接。语法格式如下：

```
public TcpClient(string hostname,int port)
```

参数说明

- hostname：要连接到的远程主机的 DNS 名。
- port：要连接到的远程主机的端口号。

（3）NetworkStream 类

该类提供在阻止模式下通过 Stream 套接字发送和接收数据的方法，其 Read 方法用于从 NetworkStream 流读取数据，该方法的语法格式如下：

```
public override int Read(byte[] buffer,int offset,int size)
```

Read 方法中的参数及说明如表 9.7 所示。

表 9.7　Read 方法中的参数及说明

参　数	描　述
buffer	Byte 类型的数组，它是内存中用于存储从 NetworkStream 读取的数据的位置
offset	buffer 中开始将数据存储到的位置
size	要从 NetworkStream 中读取的字节数
返回值	从 NetworkStream 中读取的字节数

（4）托盘图标闪动

托盘图标闪动主要是在 Timer 组件的 Tick 事件中实现的。在该事件中，如果网络中有数据传输，则通过 notifyIcon1 控件的 Icon 属性更改托盘图标。这里需要注意的是，首先要准备两个图标文件，其中一个图标文件是透明的，这样两个图标文件相互切换时，就出现图标闪动的效果。

说明：关于 Timer 组件的详细讲解请参见第 8 章的实例 188 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DynamicTaskStock。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入对方的主机地址；添加两个 Button 控件，分别用来实现向对方发送消息，以便实现图标闪动和停止图标闪动功能。

（3）程序的主要代码。

在 Frm_Main 窗体中输入对方主机地址后，单击“发送消息”按钮，首先声明要发送的信息，然后根据输入的主机地址和固定端口创建一个 TcpClient 对象，最后将信息写入数据流进行传递。“发送消息”按钮的 Click 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        IPAddress[] ip = Dns.GetHostAddresses(Dns.GetHostName());          //获取本机地址
        string message = "你好兄弟";                                        //传输的内容
        TcpClient client = new TcpClient(txtAdd.Text, 888);                //创建 TcpClient 对象
        NetworkStream netstream = client.GetStream();                      //创建 NetworkStream 对象
        StreamWriter wstream = new StreamWriter(netstream, Encoding.Default);  //创建 StreamWriter 对象
        wstream.Write(message);                                            //将信息写入流
        wstream.Flush();
        wstream.Close();                                                   //关闭流
        client.Close();                                                    //关闭 TcpClient 对象
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

在 Frm_Main 窗体中自定义一个 StartListen 方法，该方法用于监听指定端口，并通过该端口获取发送的信息。StartListen 方法的实现代码如下：

```
private void StartListen()
{
    tcpListener = new TcpListener(888);                                    //创建 TcpListener 对象
    tcpListener.Start();                                                   //开始监听
    while (true)
    {
        TcpClient tclient = tcpListener.AcceptTcpClient();                 //接受连接请求
        NetworkStream nstream = tclient.GetStream();                       //获取数据流
        byte[] mbyte = new byte[1024];                                     //建立缓存
        int i = nstream.Read(mbyte, 0, mbyte.Length);                      //将数据流写入缓存
        message = Encoding.Default.GetString(mbyte, 0, i);                 //获取传输的内容
    }
}
```

Frm_Main 窗体中的 Timer 组件主要控制系统托盘图标的闪动。实现时，首先声明一个 bool 类型的变量 k，用于控制两个图标的切换；然后在 Timer 组件的 Tick 事件中，判断当网络中有消息传递时，开始控制托盘图标的闪动。代码如下：

```
bool k = true;                                                             //一个标记，用于控制图标闪动
private void timer1_Tick(object sender, EventArgs e)
{
    if (message.Length > 0)                                                //如果网络中传输了数据
    {
        if (k)                                                             //k 为 true 时
        {
            notifyIcon1.Icon = Properties.Resources._1;                    //托盘图标为 1
            k = false;                                                     //设 k 为 false
        }
        else                                                               //k 为 false 时
        {
            notifyIcon1.Icon = Properties.Resources._2;                    //托盘图标为 2，透明的图标
            k = true;                                                      //k 为 true
        }
    }
}
```

秘笈心法

心法领悟 204：使用资源文件存储图片。

将图片存入资源文件中的步骤是，首先在“解决方案资源管理器”面板中打开 Properties 中的 Resources.resx 文件；然后在“添加资源”下拉列表框中选择“添加现有文件”选项，弹出“将现有文件添加到资源中”窗体，在该窗体中将指定的图片添加到资源文件中即可。

实例 205	实现气泡提示窗口 光盘位置：光盘\MR\09\205	初级 趣味指数：★★★★

实例说明

气泡提示因为它的美观友好被大多数用户所接受，用户所喜爱的就是程序员要实现的。本实例实现了任务栏气泡提示，运行本实例，效果如图 9.12 所示。在主窗体中单击“提示”按钮，在任务栏中会出现一个托盘图标，在图标的右上方会出现相应的提示，如图 9.13 所示；单击“关闭”按钮，气泡提示和托盘图标同时消失。

图 9.12　实现气泡提示窗口

图 9.13　气泡提示窗口

关键技术

本实例实现时主要用到了 NotifyIcon 控件的 ShowBalloonTip 方法，下面对其进行详细讲解。

NotifyIcon 控件表示在通知区域中创建图标的控件，其 ShowBalloonTip 方法用于在任务栏中持续显示具有指定标题、文本和图标的气球提示指定的时间，该方法的语法格式如下：

```
Void NotifyIcon.ShowBalloonTip(int timeout,string tipTitle,string tipText,ToolTipIcon tipIcon);
```

ShowBalloonTip 方法中的参数及说明如表 9.8 所示。

表 9.8　ShowBalloonTip 方法中的参数及说明

参　数	描　述	参　数	描　述
timeout	表示气球提示显示的时间长度	tipText	表示要在气球提示上显示的文本
tipTitle	表示要在气球提示上显示的标题	tipIcon	表示气球提示的图标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BubbleShowForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来显示气泡提示和关闭气泡提示。

（3）程序的主要代码如下：

```
private void clewButton_Click(object sender,EventArgs e)
{
    this.notifyIcon1.Visible = true;                                    //设置提示控件可见
    //显示气泡提示
    this.notifyIcon1.ShowBalloonTip(1000,"当前时间：",DateTime.Now.ToLocalTime().ToString(),ToolTipIcon.Info);
}
private void closeButton_Click(object sender,EventArgs e)
{
    this.notifyIcon1.Visible = false;                                   //设置提示控件不可见
}
```

秘笈心法

心法领悟 205：任务栏通知区域中图标的作用。

任务栏通知区域中的图标是一些进程的快捷方式，这些进程在计算机后台运行，如防病毒程序或音量控制等；这些进程不会具有自己的用户界面。

实例 206	从桌面右下角显示的 Popup 窗口提醒 光盘位置：光盘\MR\09\206	中级 趣味指数：★★★★★

实例说明

Popup 窗口提醒实际上就是在屏幕右下角弹出的提示窗口，它的应用十分广泛，如程序的升级、提示当天工作内容等，本实例将制作一个动态显示和隐藏的 Popup 窗口提醒。实例运行效果如图 9.14 所示。

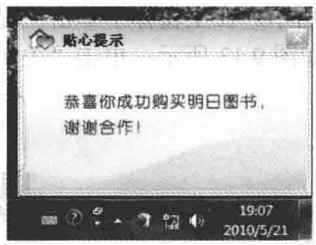


图 9.14　从桌面右下角显示的 Popup 窗口提醒

关键技术

本实例主要是使用 API 函数 AnimateWindow 来动态显示指定的窗体，并用 Control 类的 SetBounds 方法设置窗体的边界。下面对本实例中用到的关键技术进行详细讲解。

（1）AnimateWindow 函数

该函数主要用来实现窗体的动画效果。语法格式如下：

```
[DllImportAttribute("user32.dll")]
private static extern bool AnimateWindow(IntPtr hwnd, int dwTime, int dwFlags);
```

参数说明

❶ hwnd：IntPtr，窗口句柄。

❷ dwTime：动画的持续时间，数值越大动画效果的时间就越长。

❸ dwFlags：动画效果类型选项，其常量值及说明如表 9.9 所示。

表 9.9　dwFlags 参数的常量值及说明

常　量	值	说　明
AW_SLIDE	0x00040000	使用滑动类型。默认则为滚动动画类型。当使用 AW_CENTER 标志时，这个标志就被忽略
AW_ACTIVE	0x00020000	激活窗口。在使用 AW_HIDE 标志后不要使用这个标志
AW_BLEND	0x00080000	使用淡入效果。只有当 hWnd 为顶层窗口时才可以使用此标志
AW_HIDE	0x00010000	隐藏窗口，默认则显示窗口
AW_CENTER	0x00000010	若使用 AW_HIDE 标志，则使窗口向内重叠；若未使用 AW_HIDE 标志，则使窗口向外扩展
AW_HOR_POSITIVE	0x00000001	自左向右显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略
AW_HOR_NEGATIVE	0x00000002	自右向左显示窗口。当使用 AW_CENTER 标志时该标志被忽略

续表

常　量	值	说　明
AW_VER_POSITIVE	0x00000004	自顶向下显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略
AW_VER_NEGATIVE	0x00000008	自下向上显示窗口。该标志可以在滚动动画和滑动动画中使用。当使用 AW_CENTER 标志时，该标志将被忽略

（2）SetBounds 方法

该方法将控件的边界设置为指定位置和大小。语法格式如下：

```
public void SetBounds (int x,int y, int width,int height)
```

SetBounds 方法中的参数及说明如表 9.10 所示。

表 9.10　SetBounds 方法中的参数及说明

参　数	描　述	参　数	描　述
x	控件的新 Left 属性值	width	控件的新 Width 属性值
y	控件的新 Top 属性值	height	控件的新 Height 属性值

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PopupForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来显示和关闭 Popup 提醒窗口。

（3）在项目中添加一个 Windows 窗体，默认名称为 Frm_Info.cs，用来作为提醒窗口，在该窗口中添加一个 PictureBox 控件，用来执行关闭窗体操作；添加一个 ImageList 组件，用来存储程序中用到的图片资源。

（4）程序的主要代码。

在 Frm_Info 窗体中自定义一个 ShowForm 方法，主要用来实现从桌面右下角显示 Popup 提醒窗口的功能。代码如下：

```
public void ShowForm()
{
    switch (this.FormNowState)
    {
        case FormState.Hide:
            if (this.Height <= this.Rect.Height - 192)                          //当窗体没有完全显示时
                this.SetBounds(Rect.X, this.Top - 192, Rect.Width, this.Height + 192);   //使窗体不断上移
            else
            {
                this.SetBounds(Rect.X,Rect.Y,Rect.Width,Rect.Height);           //设置当前窗体的边界
            }
            AnimateWindow(this.Handle, 800, AW_SLIDE + AW_VER_NEGATIVE);       //动态显示窗体
            break;
    }
}
```

在 Frm_Info 窗体中自定义一个 CloseForm 方法，该方法中首先调用 API 函数 AnimateWindow 动画隐藏窗体，然后将窗体的状态设置为隐藏。代码如下：

```
public void CloseForm()
{
    AnimateWindow(this.Handle,800,AW_SLIDE + AW_VER_POSITIVE + AW_HIDE);    //动画隐藏窗体
    this.FormNowState = FormState.Hide;                                      //设定当前窗体状态为隐藏
}
```

秘笈心法

心法领悟 206：使用 Screen 类获取桌面的大小。

Screen 类表示单个系统上的一个或多个显示设备，其 PrimaryScreen 属性用来获取主显示，该属性返回一个 Screen 对象，而调用 Screen 对象的 WorkingArea 属性可以获取显示器的工作区，也就是桌面的大小。

实例 207 设置可执行文件的生成图标

光盘位置：光盘\MR\09\207　　初级　　趣味指数：★★★★☆

实例说明

使用 Visual Studio 2012 开发 Windows 窗体应用程序时，程序会自动使用一个默认的图标作为可执行文件的图标，如图 9.15 所示。但是开发人员可以自行设置该图标，以便使其能够更好地体现软件所代表的意义，本实例将手动设置可执行文件的生成图标。实例运行效果如图 9.16 所示。

图 9.15　可执行文件的默认图标

图 9.16　手动设置可执行文件的图标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetExeIcon。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）在 Visual Studio 2012 开发环境的菜单栏中选择“项目”/“SetExeIcon 属性”命令，如图 9.17 所示。

（4）切换到 SetExeIcon 界面，如图 9.18 所示。在该界面中选中“图标和清单”单选按钮，然后单击“图标”文本框后面的选择按钮，选择指定的 ICON 图标，最后重新应用程序即可。

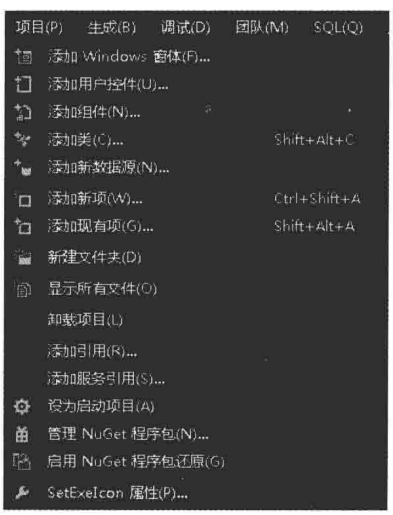


图 9.17　选择“项目”/“SetExeIcon 属性”命令

图 9.18　SetExeIcon 界面

秘笈心法

心法领悟 207：如何更换窗体的图标。

添加一个新的窗体后，窗体的图标是系统默认的图标。如果想更换窗体的图标，可以在“属性”窗口中设置窗体的 Icon 属性。

第10章

MDI 窗体和继承窗体

- MDI 窗体的使用
- 继承窗体的使用

10.1 MDI 窗体的使用

实例 208	设置窗体为父窗体 光盘位置：光盘\MR\10\208	初级 趣味指数：★★★★

实例说明

MDI（Multiple Document Interface）窗体即多文档窗体，主要用于同时显示多个文档，每个文档显示在各自的窗口中。MDI 窗体的应用非常广泛，例如，如果某公司的库存管理系统需要实现自动化，则需要使用窗体来输入客户和货品的数据、发出订单以及跟踪订单等，而且这些窗体必须链接或者从属于一个界面，并且必须能够同时处理多个文件，这时就需要建立 MDI 窗体来满足这些需求。本实例将带领读者一起来学习如何将一个窗体设置为父窗体。实例运行效果如图 10.1 所示。

图 10.1 设置窗体为父窗体

关键技术

本实例实现时主要用到了窗体的 IsMdiContainer 属性和 MdiParent 属性，下面分别对它们进行详细讲解。

（1）IsMdiContainer 属性

该属性用来获取或设置一个值，该值指示窗体是否为多文档界面（MDI）子窗体的容器。语法格式如下：

```
public bool IsMdiContainer { get; set; }
```

参数说明

属性值：如果该窗体是 MDI 子窗体的容器，则为 true；否则为 false。默认为 false。

（2）MdiParent 属性

该属性用来获取或设置此窗体的当前多文档界面（MDI）父窗体。语法格式如下：

```
[BrowsableAttribute(false)]
public Form MdiParent { get; set; }
```

参数说明

属性值：Form 类型，表示 MDI 父窗体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetMDIForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将其 IsMdiContainer 属性设置为 true，以便作为父窗体。在 Frm_Main 窗体中添加一个 Button 控件，用来打开子窗体。

（3）在项目中添加一个 Windows 窗体，并将其命名为 Frm_Child.cs，用来作为要调用的子窗体。

（4）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Frm_Child frm = new Frm_Child();                    //创建子窗体对象
    frm.MdiParent = this;                               //设置子窗体的父窗体为当前窗体
    frm.Show();                                         //显示子窗体
}
```

秘笈心法

心法领悟 208：this 关键字的使用。

this 是 C#中一个特殊的关键字，它有以下 3 种基本的用法：

（1）引用类的当前实例，以限定被相似的名称隐藏的成员。

（2）将对象作为参数传递到其他方法。

（3）声明索引器。

实例 209　使子窗体最大化显示

光盘位置：光盘\MR\10\209　　初级　　趣味指数：★★★★

实例说明

在 MDI 窗体应用程序中打开子窗体时，一般都是以默认大小打开的，但在实际应用中，经常需要使子窗体在第一次打开时就以最大化方式打开，那么如何来实现这样的功能呢？本实例使用 C#实现了这样的功能，实例运行效果如图 10.2 所示。

图 10.2　使子窗体最大化显示

关键技术

本实例实现时主要用到了窗体的 IsMdiContainer、MdiParent 和 WindowState 属性，下面对本实例用到的关键技术进行详细讲解。

说明：（1）关于 IsMdiContainer 属性和 MdiParent 属性的详细讲解请参见实例 208 中的关键技术。

（2）关于 WindowState 属性的详细讲解请参见第 9 章实例 196 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MaxChildForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将其 IsMdiContainer 属性设置为 true，以便作为父窗体。在 Frm_Main 窗体中添加一个 Button 控件，用来以最大化方式打开子窗体。

（3）在项目中添加一个 Windows 窗体，并将其命名为 Frm_Child.cs，用来作为要调用的子窗体。

（4）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Frm_Child frm = new Frm_Child();                          //创建子窗体对象
    frm.MdiParent = this;                                     //设置子窗体的父窗体为当前窗体
    frm.WindowState = FormWindowState.Maximized;              //设置子窗体最大化显示
    frm.Show();                                               //显示子窗体
}
```

秘笈心法

心法领悟 209：什么是 MDI 窗体？

MDI（Multiple Document Interface）窗体即多文档窗体，主要用于同时显示多个文档，每个文档显示在各自的窗口中。MDI 窗体中通常有包含子菜单的窗口菜单，以便在窗口或文档之间进行切换。

实例 210 对子窗体进行水平排列

光盘位置：光盘\MR\10\210

初级

趣味指数：★★★★

实例说明

在一个多文档窗体应用程序中添加完多个子窗体之后，如果一个 MDI 窗体中有多个子窗体同时打开，界面会显得非常混乱，且不容易浏览，这时可以通过使用 C#中的相应方法对多个子窗体进行排列，以便使其看起来更加有序。运行本实例，选择“加载子窗体”菜单项，在 MDI 父窗体中打开 3 个子窗体，效果如图 10.3 所示；选择“水平平铺”菜单项，对打开的 3 个子窗体进行水平排列，效果如图 10.4 所示。

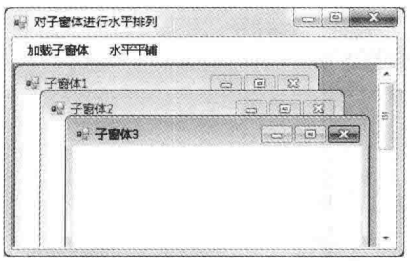


图 10.3 加载所有子窗体

图 10.4 对子窗体进行水平排列

关键技术

本实例主要是通过用带有 MdiLayout 枚举的 LayoutMdi 方法来实现的，下面对该方法进行详细讲解。

LayoutMdi 方法用来在 MDI 父窗体内排列多文档界面（MDI）子窗体。语法格式如下：

```
public void LayoutMdi (MdiLayout value)
```

参数说明

value：MdiLayout 枚举值之一，用来定义 MDI 子窗体的布局。MdiLayout 枚举用于指定 MDI 父窗体中子窗体的布局，其枚举值及说明如表 10.1 所示。

表 10.1 MdiLayout 枚举值及说明

枚 举 值	描 述
Cascade	所有 MDI 子窗体均层叠在 MDI 父窗体的工作区内
TileHorizontal	所有 MDI 子窗体均水平平铺在 MDI 父窗体的工作区内
TileVertical	所有 MDI 子窗体均垂直平铺在 MDI 父窗体的工作区内

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MDIHorizontalSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将其 IsMdiContainer 属性设置为 true，以便作为父窗体。在 Frm_Main 窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏，该菜单栏主要用来执行打开子窗体和对子窗体进行水平排列操作。

（3）在项目中添加 3 个 Windows 窗体，分别命名为 Frm_ChildOne.cs、Frm_ChildTwo.cs 和 Frm_ChildThree.cs，它们分别用来作为 3 个子窗体。

（4）程序的主要代码。

在 Frm_Main 窗体中选择“加载子窗体”菜单项，打开添加的 3 个子窗体。“加载子窗体”菜单项的 Click 事件代码如下：

```
private void 加载子窗体 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    Frm_ChildOne frm2 = new Frm_ChildOne();                    //创建 Frm_ChildOne 对象
    frm2.MdiParent = this;                                     //设置 MdiParent 属性，将当前窗体作为父窗体
    frm2.Show();                                               //使用 Show 方法打开窗体
    Frm_ChildTwo frm3 = new Frm_ChildTwo();                    //创建 Frm_ChildTwo 对象
    frm3.MdiParent = this;                                     //设置 MdiParent 属性，将当前窗体作为父窗体
    frm3.Show();                                               //使用 Show 方法打开窗体
    Frm_ChildThree frm4 = new Frm_ChildThree();                //创建 Frm_ChildThree 对象
    frm4.MdiParent = this;                                     //设置 MdiParent 属性，将当前窗体作为父窗体
    frm4.Show();                                               //使用 Show 方法打开窗体
}
```

选择“水平平铺”菜单项，调用 LayoutMdi 方法对打开的 3 个子窗体进行水平排列。“水平平铺”菜单项的 Click 事件代码如下：

```
private void 水平平铺 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.TileHorizontal);                       //使用 MdiLayout 枚举实现窗体的水平平铺
}
```

秘笈心法

心法领悟 210：父窗体的概念及设置。

在 MDI 窗体中，起到容器作用的窗体被称为“父窗体”，父窗体通过将 IsMdiContainer 属性设置为 true 来实现。

实例 211　对子窗体进行垂直排列

光盘位置：光盘\MR\10\211　　初级　　趣味指数：★★★★

实例说明

本实例主要实现对多个子窗体进行垂直排列的功能，运行本实例，选择“加载子窗体”菜单项，在 MDI 父窗体中打开 3 个子窗体；选择“垂直平铺”菜单项，对打开的 3 个子窗体进行垂直排列，效果如图 10.5 所示。

图 10.5　对子窗体进行垂直排列

关键技术

本实例主要通过 LayoutMdi 方法传递 MdiLayout.TileVertical 枚举值来实现子窗体的垂直排列。

说明：关于 LayoutMdi 方法的详细讲解请参见实例 210 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MDIVerticalSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将其 IsMdiContainer 属性设置为 true，以便作为父窗体。在 Frm_Main 窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏，该菜单栏主要用来执行打开子窗体和对子窗体进行垂直排列操作。

（3）在项目中添加 3 个 Windows 窗体，分别命名为 Frm_ChildOne.cs、Frm_ChildTwo.cs 和 Frm_ChildThree.cs，并分别用来作为 3 个子窗体。

（4）程序的主要代码如下：

```
private void 垂直平铺 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.TileVertical);                    //使用 MdiLayout 枚举实现窗体的垂直平铺
}
```

秘笈心法

心法领悟 211：子窗体的概念及设置。

在 MDI 窗体中，可以放在父窗体中的其他窗体被称为“子窗体”，也称为“MDI 子窗体”，可以通过设置某个窗体的 MdiParent 属性来确定子窗体。

实例 212　对子窗体进行层叠排列

光盘位置：光盘\MR\10\212　　初级　　趣味指数：★★★★

实例说明

本实例主要实现对多个子窗体进行层叠排列的功能，运行本实例，选择“加载子窗体”菜单项，在 MDI 父窗体中打开 3 个子窗体；选择“层叠排列”菜单项，对打开的 3 个子窗体进行层叠排列，效果如图 10.6 所示。

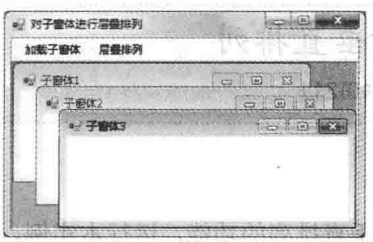


图 10.6　对子窗体进行层叠排列

说明：在 MDI 窗体中加载子窗体时，如果同时加载多个子窗体，默认就是以层叠方式进行排列的。

关键技术

本实例主要通过 LayoutMdi 方法传递 MdiLayout.Cascade 枚举值来实现子窗体的层叠排列。

说明：关于 LayoutMdi 方法的详细讲解请参见实例 210 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MDICascadeSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将其 IsMdiContainer 属性设置为 true，以便作为父窗体。在 Frm_Main 窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏，该菜单栏主要用来执行打开子窗体和对子窗体进行层叠排列操作。

（3）在项目中添加 3 个 Windows 窗体，并将其分别命名为 Frm_ChildOne.cs、Frm_ChildTwo.cs 和 Frm_ChildThree.cs，且分别用来作为 3 个子窗体。

（4）程序的主要代码如下：

```
private void 层叠排列 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    LayoutMdi(MdiLayout.Cascade);                         //使用 MdiLayout 枚举实现窗体的层叠排列
}
```

秘笈心法

心法领悟 212：子窗体与父窗体的关系。

每个应用程序只能有一个父窗体，但一个父窗体可以有多个子窗体，所有的子窗体都在父窗体中显示，并且不能移出父窗体的框架区域。

10.2　继承窗体的使用

实例 213	使用继承选择器创建继承窗体	初级
	光盘位置：光盘\MR\10\213	趣味指数：★★★☆

实例说明

开发 Windows 窗体应用程序时，项目可能需要一个与在以前的项目中创建的窗体类似的窗体，或者希望创建一个基窗体，其中含有随后将在项目中再次使用的控件布局之类的设置，每次重复使用时，都会对该基窗体模板进行修改，这时，就可以使用 C#中的继承窗体来实现。本实例将使用继承选择器来创建一个继承窗体，本实例中的基窗体和继承窗体运行效果分别如图 10.7 和图 10.8 所示。

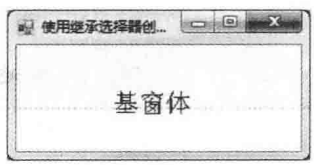


图 10.7　基窗体

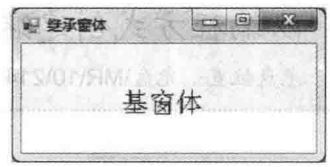


图 10.8　继承窗体

关键技术

本实例的重点是掌握什么是基窗体和继承窗体，下面分别对它们进行详细讲解。

基窗体，实质上相当于面向对象编程中提到的基类，而下面将要讲解到的继承窗体则是子类或者派生类。继承窗体实质上就是派生自基窗体的一个过程。

继承窗体实际上就是根据现有窗体的结构创建一个与其一样的新窗体，而我们将这种从现有窗体继承的过程称为可视化继承。在某种情况下，通过从基窗体继承来创建新 Windows 窗体是重复工作成果的最有效方法。

说明：（1）为了使用“继承选择器”对话框从某个窗体继承，包含该窗体的项目必须已经生成可执行文件或 DLL。

（2）如果要生成项目，可以通过“生成”菜单实现。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InheritFormByDesign。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示文字信息。

（3）在“解决方案资源管理器”面板中选中项目名称，单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建项”命令，打开“添加新项”对话框，如图 10.9 所示。

（4）选择“继承的窗体”选项，并输入窗体名称，单击“添加”按钮，打开“继承选择器”对话框，如图 10.10 所示。

图 10.9 “添加新项”对话框

图 10.10 “继承选择器”对话框

（5）单击“浏览”按钮，选择要继承的基窗体，单击“确定”按钮，完成继承窗体的创建。

秘笈心法

心法领悟 213：使用继承窗体快速提高程序的开发效率。

程序中添加继承窗体时，在“继承选择器”对话框中单击“浏览”按钮之后，可以在打开的对话框中选择其他项目中的窗体作为基窗体，从而快速地提高程序的开发效率。

实例 214 以编程方式创建继承窗体

光盘位置：光盘\MR\10\214　　初级　　趣味指数：★★★★☆

实例说明

本实例通过使用代码编程的方式来创建一个继承窗体，本实例中的基窗体和继承窗体运行效果分别如图 10.11 和图 10.12 所示。

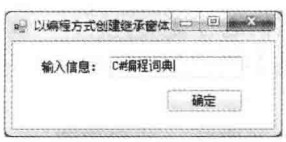


图 10.11 基窗体

图 10.12 继承窗体

关键技术

本实例在实现以编程方式创建继承窗体时，主要是在窗体类中进行定义，并将引用添加到要从其继承的窗体，引用应该包括该窗体的命名空间，后面跟一个句点，然后是基窗体本身的名称，这个过程中主要用到了类的继承机制，下面对其进行详细讲解。

说明：如果继承窗体和基窗体同属于一个命名空间，那么在修改继承窗体的基类引用时，只需要修改为基窗体的类名称即可。

继承是面向对象编程最重要的特性之一，任何类都可以从另外一个类继承，这就是说，这个类拥有它继承的类的所有成员。在面向对象编程中，被继承的类称为父类或基类。C#中提供了类的继承机制，但只支持单继承，而不支持多重继承，即在 C#中一次只允许继承一个类，不能同时继承多个类。

C#中实现继承的语法格式如下：

```
class DerivedClass: BaseClass { }
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InheritFormByCode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件、一个 TextBox 控件和一个 Button 控件，用来修饰 Frm_Main 窗体。

（3）在项目中添加一个 Windows 窗体，并将其命名为 Frm_Inherit.cs，用来作为继承窗体。

（4）修改 Frm_Inherit 窗体代码文件中 Frm_Inherit 类所继承的基类。

Frm_Inherit 窗体的原始代码如下：

```
public partial class Frm_Inherit: Form                    //表示当前窗体继承于 Form 类
{
    public Frm_Inherit ()
    {
        InitializeComponent();
    }
}
```

Frm_Inherit 窗体修改后的代码如下：

```
public partial class Frm_Inherit : Frm_Main               //继承自基窗体
{
    public Frm_Inherit()
    {
        InitializeComponent();
    }
}
```

秘笈心法

心法领悟 214：继承窗体的本质。

继承窗体实质上就是派生自基窗体的一个过程，它其实体现的是面向对象思想中的继承性。继承窗体中的基窗体相当于面向对象编程中提到的基类，而继承窗体则相当于子类或者派生类。

实例 215　修改继承窗体中的控件属性

光盘位置：光盘\MR\10\215　　中级　　趣味指数：★★★★

实例说明

创建继承窗体之后，如果不做任何修改，那么继承窗体的运行结果将与基窗体的运行结果一模一样，但这显然不是开发人员要用到继承窗体的初衷。程序开发人员在用到继承窗体时，不可能全部使用基窗体中原来的功能，而没有新扩展的功能，这时就遇到一个问题：如何扩展基窗体的功能，以便在继承窗体中使用呢？本实例以扩展基窗体中的控件功能为例讲解如何扩展基窗体的功能。在本程序中分别运行基窗体和继承窗体，它们的运行效果分别如图 10.13 和图 10.14 所示。

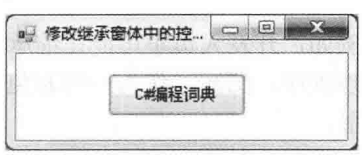


图 10.13　基窗体中的控件文本

图 10.14　继承窗体中的控件文本

关键技术

本实例在修改继承窗体中控件的属性时，主要用到了控件的 Modifiers 属性，下面对该属性进行详细讲解。

Modifiers 属性用来指定控件对象的可见性级别，它是 Visual Studio 2012 开发环境提供的一个属性，该属性的属性值及说明如表 10.2 所示。

表 10.2　Modifiers 属性的属性值及说明

属 性 值	描　述
Private	可访问域限定在类的内部
Protected	只能从其所在类和所在类的子类进行访问
Protected Internal	同一个程序集中的所有类，以及所有程序集中的子类都可以访问
Internal	可访问域限定在类所在的程序内部
Public	访问不受限制，可以在类内和任何类以外的代码中进行访问

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EditInheritForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，设置其 Text 属性为“C#编程词典”，并将其 Modifiers 属性设置为 Public，如图 10.15 所示。

图 10.15　设置 Modifiers 属性

说明：这里也可以将 button1 控件的 Modifiers 属性设置为 Protected 或者 Protected Internal。

（3）按实例 213 中所讲解的步骤添加一个继承窗体，并将其命名为 Frm_Inherit.cs，该窗体继承自 Frm_Main，在继承窗体 Frm_Inherit 中即可修改继承的 button1 控件的属性。例如，这里将 button1 控件的 Text 属性重新设置为“C#范例大全”。

秘笈心法

心法领悟 215：继承窗体的使用场合。

尽管继承窗体的出现非常讨人喜欢，但它并非没有局限性。例如，开发人员不可能在继承窗体中增加或者减少基窗体中的菜单项，也不可能替换掉基窗体中的菜单事件处理程序。然而，作为一种避免复制控件和代码的模板机制，继承窗体在很多时候都值得程序开发人员去考虑使用。

控件应用篇

- 第 11 章　Windows 常用控件的使用
- 第 12 章　Windows 高级控件的使用
- 第 13 章　DataGridView 数据控件
- 第 14 章　自定义用户控件

第11章

Windows 常用控件的使用

- 按钮控件的使用
- TextBox 控件应用
- ComboBox 控件应用
- RichTextBox 控件应用
- ListBox 控件应用
- ListView 控件应用
- 选择类控件应用
- TreeView 控件应用
- 选项卡控件的使用
- 菜单应用实例
- 工具栏设计
- 状态栏设计

11.1　按钮控件的使用

实例 216	轻松打造绚丽按钮 光盘位置：光盘\MR\11\216	初级 趣味指数：★★★★☆

实例说明

程序设计过程中，经常会用到按钮控件，怎样才可以设计出绚丽、友好的按钮呢？在实例中，将会通过设置 Button 控件的一些公共属性来改变 Button 按钮的外观，打造绚丽多彩的按钮。Image 属性可以在按钮上显示图片，ForeColor 属性用于设置按钮上字体的颜色，FlatStyle 用于设置按钮的样式风格，Dock 用于设置停靠布局。实例运行效果如图 11.1 所示。

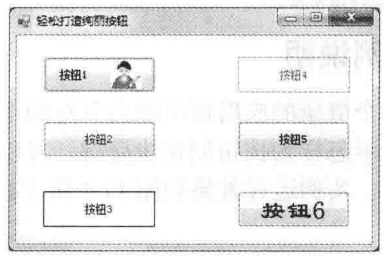


图 11.1　轻松打造绚丽按钮

关键技术

本实例重点介绍怎样使用 Button 按钮的 FlatStyle 属性和 ForeColor 属性方便地设置按钮样式和文字颜色，下面对本实例用到的关键技术进行详细讲解。

（1）FlatStyle 属性

该属性用来获取或设置按钮控件的平面样式外观。语法格式如下：

```
public FlatStyle FlatStyle { get; set; }
```

参数说明

属性值：FlatStyle 枚举值。默认值为 Standard。

（2）ForeColor 属性

该属性用来获取或设置控件的前景色。语法格式如下：

```
public virtual Color ForeColor{ get; set; }
```

参数说明

属性值：Color 结构，用于设置控件的前景色。默认为 DefaultForeColor 属性的值。

技巧：如果 FlatStyle 属性设置为 FlatStyle.System，则分配给 Image 属性的任何图像都不会显示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MenuVestige。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 6 个 Button 按钮，用于演示不同的按钮样式外观。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    btn_One.Image = global::MenuVestige.Properties.Resources.picture;      //设置按钮上的图像
    btn_Tow.ForeColor = Color.Red;                                         //设置按钮前景色为红色
    btn_Three.FlatStyle = FlatStyle.Flat;                                  //设置按钮以平面显示
    btn_Three.ForeColor = Color.Blue;                                      //设置按钮前景色为蓝色
    btn_Four.ForeColor = Color.Green;                                      //设置按钮前景色为绿色
    btn_Four.FlatStyle = FlatStyle.Popup;                                  //得到焦点后按钮为三维样式
    btn_Five.FlatStyle = FlatStyle.Standard;                               //设置按钮以三维样式显示
    btn_six.FlatStyle = FlatStyle.System;                                  //按钮外观由操作系统决定
    btn_six.Font = new Font("隶书", 20);                                   //设置按钮文字字体
}
```

秘笈心法

心法领悟 216：通过设置按钮的 Image 属性为按钮添加图片。

设置按钮对象的 Image 属性可以方便地为按钮添加图片信息，也可以通过按钮的 FlatStyle 属性和 ForeColor 属性为按钮设置样式和前景颜色，打造多彩绚丽的按钮就是这样简单。

实例 217　给 Button 控件创建快捷键

光盘位置：光盘\MR\11\217　　初级　　趣味指数：★★★★☆

实例说明

一个健康的应用程序应当具有如下特点，清晰直观的界面，简洁方便的布局，便捷高效的操作。本实例中将会演示怎样为按钮创建快捷键，可以使用按钮的快捷键来代替鼠标单击按钮，使用户操作起来更加方便，容易上手。实例运行效果如图 11.2 所示。

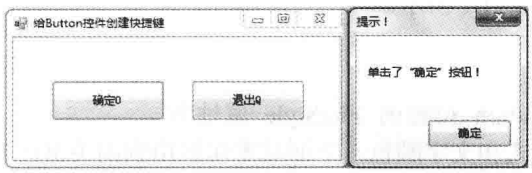


图 11.2　给 Button 控件创建快捷键

关键技术

本实例重点介绍怎样通过按钮的 Text 属性设置按钮快捷键，下面对本实例用到的关键技术进行详细讲解。

为按钮设置快捷键非常简单，设置按钮对象的 Text 属性时，在按钮文本中添加“&”+“字母”即可，使用快捷键时，按下 Alt 键加指定的“字母”便可以激活按钮。

注意：可以通过设置窗体的 AcceptButton 属性方便地为按钮设置快捷键，如果窗体中设置了 AcceptButton 属性为一个指定的按钮，当在窗体中按下 Enter 键时就相当于按下了指定的按钮。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ButtonShortCut。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 Button 按钮，用于演示按钮快捷键。

（3）程序的主要代码如下：

```
private void btn_OK_Click(object sender, EventArgs e)
{
    MessageBox.Show("单击了"确定"按钮！", "提示！");                    //弹出消息对话框
}
private void btn_Quit_Click(object sender, EventArgs e)
{
    MessageBox.Show("单击了"退出"按钮！", "提示！");                    //弹出消息对话框
}
```

秘笈心法

心法领悟 217：使用窗体的 CancelButton 属性为按钮设置快捷键。

可以通过设置窗体的 CancelButton 属性方便地为按钮设置快捷键，如果窗体中设置了 CancelButton 属性为一个指定的按钮，当在窗体中按下 Esc 键时就相当于按下了指定的按钮。

实例 218　使 Button 控件大小自动适应文本长度

光盘位置：光盘\MR\11\218　　初级　　趣味指数：★★★☆

实例说明

在窗体应用程序设计中，经常会用到 Button 按钮控件，而且会经常更改 Button 按钮的文本内容，如果 Button 按钮的文本内容过长，会出现无法正常显示文本内容的情况。本实例中将会演示怎样使 Button 控件大小自动适应文本长度。实例运行效果如图 11.3 所示。

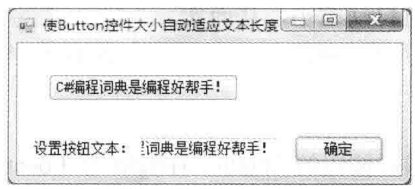


图 11.3　使 Button 控件大小自动适应文本长度

关键技术

本实例重点介绍怎样通过按钮的 Text 属性与 AutoSize 属性设置按钮文本内容，以及使按钮自动适应文本长度。下面对本实例用到的关键技术进行详细讲解。

（1）Text 属性

该属性用来获取或设置与此控件关联的文本内容。语法格式如下：

```
public override string Text { get; set; }
```

参数说明

属性值：属性值为字符串对象。

（2）AutoSize 属性

该属性用来获取或设置一个值，该值指示按钮是否基于其内容调整大小。语法格式如下：

```
public override bool AutoSize { get; set; }
```

参数说明

属性值：如果控件基于其内容自动调整大小则为 true，否则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AutoButton。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 Button 控件，分别用于演示自动适应文本长度和设置按钮文本内容；添加一个 TextBox 控件，用于设置自动适应文本长度按钮的文本内容。

（3）程序的主要代码如下：

```
private void btn_Ok_Click(object sender, EventArgs e)
{
    btn_One.AutoSize = true;                    //设置按钮基于内容自动调整大小
    btn_One.Text = txt_Name.Text;               //设置按钮文本内容
}
```

秘笈心法

心法领悟 218：怎样设置按钮的高度？

通过按钮对象的 Height 属性可以方便地设置按钮的高度，也可以通过按钮的 Location 属性轻松地设置按钮在窗体中的坐标位置。

实例 219 得到鼠标焦点后自动放大的 Button 按钮

光盘位置：光盘\MR\11\219　　初级　　趣味指数：★★★☆

实例说明

程序设计过程中，经常需要使用一些动态的效果，从而增加程序的可操作性，如使用动态图片指示用户下一步的操作。本实例中将会演示如何动态地显示按钮状态，当按钮得到焦点时自动放大，失去焦点时自动缩小，从而实现动态按钮的效果。实例运行效果如图 11.4 所示。

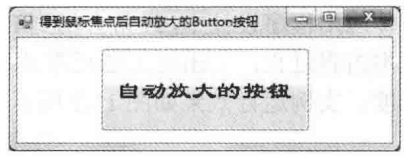


图 11.4　得到鼠标焦点后自动放大的 Button 按钮

关键技术

本实例重点在于向读者介绍怎样通过按钮的 Width、Height 与 Location 属性设置按钮的宽度、高度以及按钮在窗体中的位置，下面对本实例用到的关键技术进行详细讲解。

（1）Width 属性

该属性用来获取或设置控件的宽度。语法格式如下：

```
public int Width { get; set; }
```

参数说明

属性值：整数类型数值，用来设置以像素为单位的按钮的宽度。

（2）Height 属性

该属性用来获取或设置控件的高度。语法格式如下：

```
public int Height { get; set; }
```

参数说明

属性值：整数类型数值，用来设置以像素为单位的按钮的宽度。

（3）Location 属性

该属性用来获取或设置该按钮左上角相对于窗体的左上角的坐标。语法格式如下：

```
public Point Location { get; set; }
```

参数说明

属性值：Point 结构，用来标记按钮在窗体中的坐标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ButtonMagnifier。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 Button 按钮，此按钮用于演示得到焦点后自动放大。

（3）程序的主要代码如下：

```
private void button1_MouseEnter(object sender, EventArgs e)
{
    button1.Font = new Font("隶书", 18);                                    //设置按钮字体样式
    button1.Width = 261;                                                    //设置按钮宽度
    button1.Height = 80;                                                    //设置按钮高度
    button1.Location = new Point((Width - button1.Width) / 2,               //设置按钮位置
        (ClientRectangle.Height - button1.Height) / 2);
}
```

秘笈心法

心法领悟 219：制作在窗体中跑动的按钮。

实例中已经介绍过，通过按钮对象的 Location 属性可以方便地设置按钮在窗体中的坐标位置，现在可以使用 Timer 组件间隔性地更改按钮的 Location 属性，以使按钮在窗体中自由地移动。

实例 220	动态创建多个 Button 控件 光盘位置：光盘\MR\11\220	初级 趣味指数：★★★☆

实例说明

Button 按钮是用户与应用程序交互过程中最常用的控件，通过 Button 按钮的 Click 单击事件，可以轻松地控制应用程序的执行流程。使用 Button 按钮通用的方法是将按钮控件从工具栏拖放到窗体中，然后设置按钮的各种属性并定义按钮事件。本实例中将会演示如何根据用户的选择，在窗体中动态地添加 Button 控件。实例运行效果如图 11.5 所示。

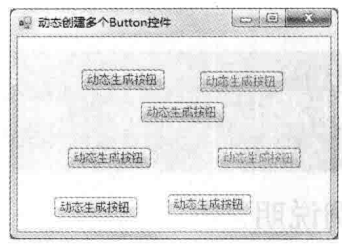


图 11.5　动态创建多个 Button 控件

关键技术

本实例重点介绍怎样使用 Controls 集合的 Add 方法在窗体中动态添加按钮，下面对本实例用到的关键技术进行详细讲解。

（1）Controls 属性

该属性用来获取包含在窗体内的控件的集合。语法格式如下：

```
public Control.ControlCollection Controls { get; }
```

参数说明

属性值：窗体控件的集合。

（2）Controls 集合的 Add 方法

该方法用来将指定的控件添加到窗体控件集合中。语法格式如下：

```
public virtual void Add(Control value)
```

参数说明

value：继承于 Control 类的控件。

说明：由于所有窗体控件都继承于 Control 类，所以，所有窗体控件都可以使用 Controls 集合的 Add 方法动态地将控件添加到窗体中。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GenerateButton。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，再更改 Text 属性为“动态创建多个 Button 控件”。

（3）程序的主要代码如下：

```
private void Frm_Main_MouseClick(object sender, MouseEventArgs e)
{
    Button bt = new Button()                                          //创建按钮对象
    {
        Text = "动态生成按钮",                                          //设置按钮的文本信息
        ForeColor = Color.FromArgb(                                   //设置按钮的前景颜色
        G_Random.Next(0, 255),
        G_Random.Next(0, 255),
        G_Random.Next(0, 255)),
        AutoSize = true,                                              //设置按钮自动调整大小
        Location = e.Location                                         //设置按钮位置
    };
    Controls.Add(bt);                                                 //将按钮加入控件集合
}
```

秘笈心法

心法领悟 220：动态创建多个 TextBox 控件。

动态创建多个 TextBox 控件的方法与动态创建多个 Button 控件的方法基本相同，首先，应当创建 TextBox 对象，并设置 TextBox 相应的属性，然后，通过窗体对象 Controls 集合的 Add 方法将 TextBox 对象添加到窗体控件集合中，此时 TextBox 便可以在窗体中正常显示。

11.2 TextBox 控件应用

实例 221	制作多行文本框 光盘位置：光盘\MR\11\221	初级 趣味指数：★★★★☆

实例说明

程序设计过程中，经常会用到 TextBox 文本框控件，在文本框控件中可以方便地输入文本信息。向窗体中添加文本框控件非常简单，只需要从工具栏中将文本框控件拖入窗体即可。被拖入窗体的文本框控件，只允许输入单行文本内容，那么怎样设置文本框可以输入多行内容呢？本实例中将会演示如何通过更改文本框的相应属性使文本框可以输入多行内容。实例运行效果如图 11.6 所示。

图 11.6 制作多行文本框

关键技术

本实例重点介绍通过设置 TextBox 控件的 Multiline 属性，使 TextBox 可以写入多行文本。下面对本实例用到的关键技术进行详细讲解。

TextBox 控件的 Multiline 属性用于设置 TextBox 控件是否为多行 TextBox 控件。语法格式如下：

```
public override bool Multiline { get; set; }
```

参数说明

属性值：如果该控件是多行 TextBox 控件则值为 true，否则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Multi-LineText。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 文本框控件和一个 Button 控件，分别用于显示多行文本框和将 TextBox 转换为多行文本框。

（3）程序的主要代码如下：

```
private void btn_OK_Click(object sender, EventArgs e)
{
    txt_Change.Location = new Point(0, 0);                          //设置文本框位置
    txt_Change.Multiline = true;                                    //设置文本框显示多行
    txt_Change.Width = this.Width;                                  //设置文本框宽度
    txt_Change.Height = ClientRectangle.Height - btn_OK.Height-15;  //设置文本框高度
}
```

秘笈心法

心法领悟 221：轻松改变 TextBox 控件的背景色。

使用 TextBox 控件的 BackColor 属性可以方便地设置控件的背景色，BackColor 属性接受一个 Color 结构的值。

实例 222　制作密码文本框

光盘位置：光盘\MR\11\222　　初级　　趣味指数：★★★★☆

实例说明

在 TextBox 文本框控件中，可以方便地输入字符串信息，应用程序可以通过用户输入的信息，准确、高效地执行相应的操作。本实例中将会演示如何使用 TextBox 控件制作密码文本框。实例运行效果如图 11.7 所示。

图 11.7　制作密码文本框

关键技术

本实例重点介绍怎样通过 TextBox 控件的 PasswordChar 属性设置密码字符，下面对本实例用到的关键技术进行详细讲解。

TextBox 控件的 PasswordChar 属性用于获取或设置字符，该字符用于屏蔽单行 TextBox 控件中的密码字符。语法格式如下：

```
public char PasswordChar { get; set; }
```

参数说明

属性值：用于屏蔽在单行 TextBox 控件中输入的字符，如果不想让控件在字符输入时将它们屏蔽，请将此属性值设置为“(char)0”。PasswordChar 属性的默认值为“(char)0”。

说明：设置 TextBox 控件的 PasswordChar 属性为‘*’时，TextBox 中的文本内容将会以“密码”的形式显示，如果将 PasswordChar 属性设置为“(char)0”时，TextBox 中的文本内容将会以字符的形式显示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Txt_PassWord。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 文本框控件，用于显示文本框中的密码或文本信息；添加两个 Button 控件，分别用于设置和取消密码的输入。

（3）程序的主要代码如下：

```
private void btn_Str_Click(object sender, EventArgs e)
{
    txt_Change.PasswordChar = (char)0;                          //取消屏蔽
}
private void btn_PassWord_Click(object sender, EventArgs e)
{
    txt_Change.PasswordChar = '*';                              //设置屏蔽字符
}
```

秘笈心法

心法领悟 222：使用 ForeColor 属性设置文本颜色。

使用 TextBox 控件的 ForeColor 属性可以方便地设置控件的文本颜色，ForeColor 属性接受一个 Color 结构的值。

实例 223　在 TextBox 控件中显示回车符

光盘位置：光盘\MR\11\223　　初级　　趣味指数：★★★★☆

实例说明

TextBox 控件可以用来显示和输入文本信息，向 TextBox 控件输入文本信息的过程中，可以使用 Enter（回

车）键方便地为控件中的文本内容换行。如果单击某个按钮后，需要向 TextBox 控件中添加多行文本，那么，多行文本中的换行应当怎样实现呢？本实例中将会演示如何方便地在 TextBox 控件中添加带有换行符的文本内容。实例运行效果如图 11.8 所示。

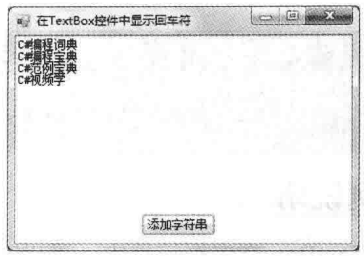


图 11.8　在 TextBox 控件中显示回车符

关键技术

本实例重点介绍怎样通过 Environment 类的 NewLine 属性在 TextBox 控件中显示回车符，下面对本实例用到的关键技术进行详细讲解。

Environment 类的 NewLine 属性用来获取为此环境定义的换行字符串。语法格式如下：

```
public static string NewLine { get; }
```

参数说明

属性值：字符串类型值。包含“\r\n”的字符串，用于非 UNIX 平台或 UNIX 平台。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 textNewLine。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件和一个 Button 控件。TextBox 控件用于显示带有回车符的字符串信息，Button 控件用于向 TextBox 控件中添加字符串。

（3）程序的主要代码如下：

```
private void btn_Add_Click(object sender, EventArgs e)
{
    txt_Str.Text = string.Format(                                    //向字符串中添加换行符
        "C#编程词典{0}C#编程宝典{0}C#范例宝典{0}C#视频学",Environment.NewLine);
}
```

秘笈心法

心法领悟 223：在 TextBox 控件中也可以使用“\r\n”换行符。

实例中介绍了使用 Environment 类的 NewLine 属性在 TextBox 控件中显示换行符，现在，也可以使用“\r\n”在 TextBox 中换行，它与 NewLine 属性提供的换行符效果是一样的。

实例 224　只允许输入数字的 TextBox 控件

光盘位置：光盘\MR\11\224　　初级　　趣味指数：★★★☆

实例说明

可以在 TextBox 控件中轻松地输入文本信息，输入的文本信息可以包括字母、数字、汉字等。如果需要用户在 TextBox 控件中填写年龄信息，那么年龄信息应当只允许为数字，怎样限制用户输入其他信息呢？本实例中将会演示如何只允许用户在 TextBox 控件中输入数字。实例运行效果如图 11.9 所示。

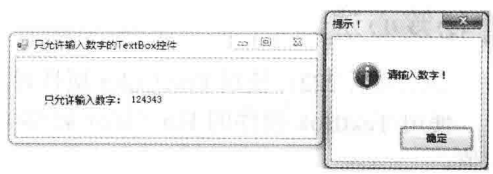


图 11.9　只允许输入数字的 TextBox 控件

关键技术

本实例重点介绍怎样通过 Char 结构的 IsDigit 方法判断用户在 TextBox 控件中输入的字符是否为数字，下面对本实例用到的关键技术进行详细讲解。

TextBox 控件的 KeyPress 事件包含两个参数，即 sender 事件源和包含事件数据的 e。通过 e 的 KeyChar 属性可以获取用户按下键盘的按键，通过 e 的 Handled 属性可以设置是否处理过用户按键的事件。

（1）KeyChar 属性

该属性用于获取或设置按下键对应的字符。语法格式如下：

```
public char KeyChar { get; set; }
```

参数说明

属性值：Char 结构，得到按下键对应的字符。

（2）Handled 属性

该属性用来获取或设置一个值，该值指示是否处理过 System.Windows.Forms.Control.KeyPress 事件。语法格式如下：

```
public bool Handled { get; set; }
```

参数说明

属性值：布尔类型值，指示是否处理过 KeyPress 事件。

使用 Char 结构的 IsDigit 方法可以判断指定字符是否为十进制数字，下面对其进行详细说明。

（3）IsDigit 方法

该方法指示某个字符是否属于十进制数字类别。语法格式如下：

```
public static bool IsDigit(char c)
```

参数说明

❶ c：一个字符。

❷ 返回值：返回布尔值，如果字符 c 是十进制数字则为 true，否则为 false。

说明：通过将 KeyPressEventArgs 的 Handled 设置为 true，可以取消 KeyPress 事件，这样可以取消用户在 TextBox 按键的值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OnlyDigit。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件，用于演示只允许向文本框中输入数字。

（3）程序的主要代码如下：

```
private void txt_Str_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!char.IsDigit(e.KeyChar))                                    //判断是否为数字
    {
        MessageBox.Show("请输入数字！","提示！",                      //弹出消息对话框
            MessageBoxButtons.OK,MessageBoxIcon.Information);
        e.Handled = true;                                            //取消在控件中显示该字符
    }
}
```

秘笈心法

心法领悟 224：可以使用 Char 结构的 IsLetter 方法判断字符是否为字母。

实例中使用了 Char 结构的 IsDigit 方法判断用户输入的字符是否为数字，那么，有什么方法可以有效地判断用户输入的字符是否为字母呢？可以使用 Char 结构的 IsLetter 方法方便地判断用户输入的字符是否为字母，如果为字母方法返回 true，否则返回 false。

实例 225　在 TextBox 控件底端显示下划线

初级

光盘位置：光盘\MR\11\225

趣味指数：★★★☆

实例说明

TextBox 控件可以方便地显示文本内容，而且还可以通过更改控件相应的属性，从而控制其内部显示文本

的字体样式。本实例中将会演示如何为控件中的所有文本内容添加下划线。实例运行效果如图 11.10 所示。

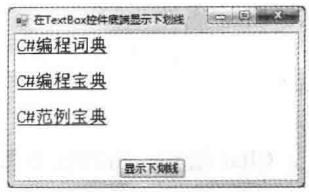


图 11.10　在 TextBox 控件底端显示下划线

关键技术

本实例重点介绍怎样通过 TextBox 控件的 Font 属性为控件中的文本内容添加下划线，下面对本实例用到的关键技术进行详细讲解。

TextBox 控件的 Font 属性用来获取或设置控件显示的文字的字体。语法格式如下：

```
public virtual Font Font { get; set; }
```

参数说明

属性值：Font 类型的对象，主要用于获取或设置控件中文本的字体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UnderLine。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 控件和一个 Button 控件，TextBox 控件用于显示带有下划线的文字，Button 控件用于改变 TextBox 中的文本样式。

（3）程序的主要代码如下：

```
private void btn_DisplayUnderLine_Click(object sender, EventArgs e)
{
    txt_Str.Font = new Font(new Font("宋体",15), FontStyle.Underline);          //设置文本框中文字的字体
}
```

秘笈心法

心法领悟 225：通过 TextBox 的 Width 属性设置文本框宽度。

实例中已经介绍了怎样使用 TextBox 控件的 Font 属性方便地设置控件中文本内容的字体。现在可以通过控件的 Width 属性轻松地设置控件的宽度，Width 属性接收了一个整数数值，它会以像素为单位设置 TextBox 控件的宽度。

实例 226　屏蔽 TextBox 控件上的粘贴功能

光盘位置：光盘\MR\11\226　　初级　　趣味指数：★★★★☆

实例说明

Ctrl+C 键和 Ctrl+V 键是常用的系统快捷键，这两个快捷键用于复制和粘贴内容，使用这两个快捷键可以方便地向 TextBox 控件中粘贴文本内容，但是，在某些特殊情况下，不希望使用此快捷键的功能，例如，复制和粘贴密码内容是很危险的。本实例中将会演示如何屏蔽 TextBox 控件的粘贴功能。实例运行效果如图 11.11 所示。

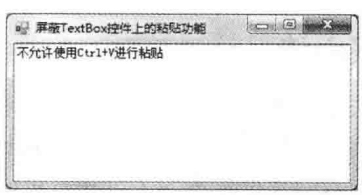


图 11.11　屏蔽 TextBox 控件上的粘贴功能

关键技术

本实例重点介绍怎样通过重写 TextBox 类的 WndProc 方法有效地控制控件中的粘贴功能，下面对本实例用到的关键技术进行详细讲解。

使用 TextBox 类的 WndProc 方法可以方便地屏蔽控件中的粘贴功能。语法格式如下：

```
protected override void WndProc(ref Message m)
```

参数说明

m：一个 Windows 消息对象，用来传递消息信息。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShieldPaste。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，再更改 Text 属性为“屏蔽 TextBox 控件上的粘贴功能”。

（3）程序的主要代码如下：

```
class TextBoxx : TextBox
{
    public const int WM_PASTE = 0x0302;                          //粘贴消息信息
    protected override void WndProc(ref Message m)               //重写处理消息方法
    {
        if (m.Msg != WM_PASTE)                                   //屏蔽粘贴消息信息
        {
            base.WndProc(ref m);                                 //调用基类消息处理方法
        }
    }
}
```

秘笈心法

心法领悟 226：通过 TextBox 的 Height 属性设置文本框高度。

要设置 TextBox 文本框的高度，首先要设置控件的 Multiline 属性为 true，允许多行文本输入，然后通过 TextBox 文本框的 Height 属性设置文本框高度，Height 属性接收了一个整数数值，它会以像素为单位设置控件的高度。

实例 227　屏蔽 TextBox 控件上默认的右键菜单

光盘位置：光盘\MR\11\227　　初级　　趣味指数：★★★☆

实例说明

可以方便地向 TextBox 控件中输入文本内容，也可以使用 TextBox 控件中的右键菜单，便捷地操作文本内容。但是，如果应用程序不需要 TextBox 控件的右键菜单，要怎样才可以禁止使用它呢？本实例中将会演示如何屏蔽 TextBox 控件中的右键菜单。实例运行效果如图 11.12 所示。

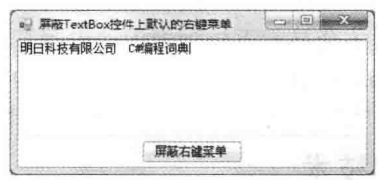


图 11.12　屏蔽 TextBox 控件上默认的右键菜单

关键技术

本实例重点介绍通过设置 TextBox 文本框的 ShortcutsEnabled 属性来禁用文本框中的右键菜单，下面对本实例用到的关键技术进行详细讲解。

TextBox 文本框的 ShortcutsEnabled 属性用来获取或设置一个值，该值指示是否启用定义的快捷方式。语法格式如下：

```
public virtual bool ShortcutsEnabled { get; set; }
```

参数说明

属性值：布尔类型值，如果启用快捷方式，属性值为 true，否则为 false。

说明：将 TextBox 文本框控件的 ShortcutsEnabled 属性设置为 false，可以禁用文本框中的右键菜单，同时也可以有效地禁止使用 Ctrl+V 快捷键向文本框中粘贴内容。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShieldMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TextBox 文本框控件和一个 Button 控件，TextBox 文本框控件用于演示被屏蔽的右键菜单，Button 控件用于屏蔽 TextBox 控件中的右键菜单。

（3）程序的主要代码如下：

```
private void btn_Shield_Click(object sender, EventArgs e)
{
    txt_Str.ShortcutsEnabled = false;                                    //停用文本框的快捷方式
}
```

秘笈心法

心法领悟 227：禁用文本框中的复制功能。

设置文本框的 ShortcutsEnabled 属性为 false，可以有效地禁用控件中鼠标右键菜单和 Ctrl+V 快捷键的粘贴功能，同时还可以禁用 Ctrl+C 快捷键，禁止复制文本框中的文本内容。

11.3 ComboBox 控件应用

实例 228	设置 ComboBox 控件的默认选项为第一项	初级
	光盘位置：光盘\MR\11\228	趣味指数：★★★☆

实例说明

在窗体应用程序开发中，经常会用到 ComboBox 控件，ComboBox 控件中的下拉列表中可以显示多项数据。可以使用 ComboBox 控件中 Items 集合的 Add 方法向控件中添加数据项。然而，ComboBox 控件默认显示的数据项一直是空项，没有显示任何内容，只有单击此控件后才会显示下拉列表中的内容，怎样才可以使 ComboBox 控件默认显示第一项数据内容呢？这正是本实例要实现的功能。实例运行效果如图 11.13 所示。

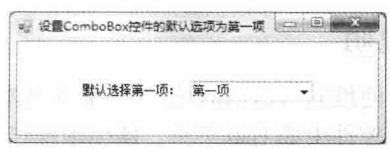


图 11.13 设置 ComboBox 控件的默认选项为第一项

关键技术

本实例重点介绍怎样通过 ComboBox 控件中 Items 集合的 Add 方法向 ComboBox 控件中添加数据，并设置 SelectedIndex 属性，使 ComboBox 控件的默认选项为第一项。下面对本实例用到的关键技术进行详细讲解。

（1）Add 方法

使用 ComboBox 控件中 Items 集合的 Add 方法可以方便地向 ComboBox 控件中添加数据项。语法格式如下：

```
public int Add(object item);
```

参数说明

item：object 类型的对象。

（2）SelectedIndex 属性

使用 ComboBox 控件的 SelectedIndex 属性可以轻松地设置控件的选项，该属性用来获取或设置指定当前选定项的索引。语法格式如下：

```
public override int SelectedIndex { get; set; }
```

参数说明

属性值：整型数值，表示当前选定项的从零开始的索引，如果未选定任何项则返回值为-1。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectDefault。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 控件，用于演示默认选择控件中的第一项。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    cbox_Select.SelectedIndex = 0;                                    //默认选择第一项
}
```

秘笈心法

心法领悟 228：在 ComboBox 控件的 Items 集合的 Add 方法中可以添加任何对象。

实例中已经详细地讲解了 Items 集合的 Add 方法，该方法接收一个 object 类型的对象作为参数，并可以将 object 对象放入数据项集合中，由于 C#中的所有对象都继承于 object 类，所以任何对象都可以放入 ComboBox 的数据集合中，读者可能有些不理解，如果向 ComboBox 控件中添加非字符串对象后，ComboBox 应当怎样显示呢？ComboBox 控件首先会调用 Items 集合中每一个数据项对象的 ToString 方法，得到数据项对象的字符串信息并显示此信息。

实例 229　将数据表中的字段添加到 ComboBox 控件中

光盘位置：光盘\MR\11\229　　初级　　趣味指数：★★★☆

实例说明

可以使用 ComboBox 控件中 Items 集合的 Add 方法方便地向控件中添加数据项。而且 ComboBox 控件还提供了更加强大的功能，可以将数据集合中的数据轻松地绑定到 ComboBox 控件中。本实例中将会演示如何将数据表中的字段加到 ComboBox 控件中，首先读取数据库表中的数据到数据集合，再将 ComboBox 绑定到数据集合，然后 ComboBox 控件会显示数据集合中的数据。实例运行效果如图 11.14 所示。

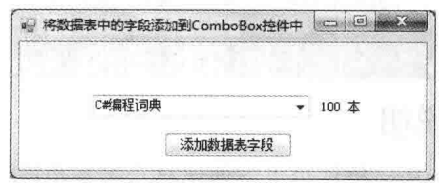


图 11.14　将数据表中的字段添加到 ComboBox 控件中

关键技术

本实例重点介绍怎样使用 ComboBox 控件的 DataSource 属性设置数据源，并通过 DisplayMember 和 ValueMember 属性设置数据项的数值。下面对本实例用到的关键技术进行详细讲解。

（1）DataSource 属性

该属性用来获取或设置 ComboBox 的数据源。语法格式如下：

```
public Object DataSource { get; set; }
```

参数说明

属性值：任何实现 IList 接口的对象，如 DataSet、泛型集合或数组等。

（2）DisplayMember 属性

该属性用来获取或设置要为此列表控件显示的属性。语法格式如下：

```
public string DisplayMember { get; set; }
```

参数说明

属性值：字符串对象，指定 DataSource 属性的集合中包含属性的名称，默认值为 string.Empty。

（3）ValueMember 属性

该属性用来获取或设置一个属性，此属性将用作列表控件中数据项的实际值。语法格式如下：

```
public string ValueMember { get; set; }
```

参数说明

属性值：字符串对象，指定 DataSource 属性的集合中包含属性的名称，默认值为 string.Empty。

说明：在 ComboBox 控件中使用 SelectedText 属性，可以方便地得到 ValueMember 属性指定的成员属性的值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BindToComboBox。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 控件和一个 Button 控件，ComboBox 控件用于显示数据表中的数据，Button 控件用于将数据表中的数据绑定到 ComboBox 控件。

（3）程序的主要代码如下：

```
private void btn_Add_Click(object sender, EventArgs e)
{
    cbox_Display.DataSource = new DataTier().GetMessage();          //绑定到数据表中的数据
    cbox_Display.DisplayMember = "book";                            //设置显示属性
    cbox_Display.ValueMember = "count";                             //设置实际值
}
```

秘笈心法

心法领悟 229：轻松地使用 SelectedText 属性。

通过对实例的认真学习，读者可以使用 ComboBox 控件的 Items 集合的 Add 方法向控件中添加数据项，也可以使用 DataSource 属性设置数据源的方式向控件中添加数据项，然后，可以通过 ComboBox 控件的 SelectedText 属性，方便地得到控件中被选择的数据项的文本内容。

实例 230　在 ComboBox 下拉列表中显示图片

光盘位置：光盘\MR\11\230　　初级　　趣味指数：★★★★☆

实例说明

ComboBox 控件可以方便地显示多条数据信息，可以使用 Items 集合的 Add 方法向控件中添加数据项，也可以使用数据绑定的方法将指定数据集合中的数据绑定到 ComboBox 控件。本实例中将会演示如何在 ComboBox 控件中显示图片信息。实例运行效果如图 11.15 所示。

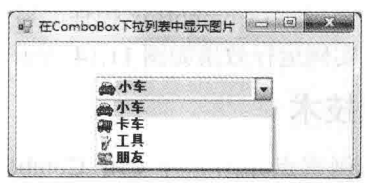


图 11.15　在 ComboBox 下拉列表中显示图片

关键技术

本实例重点介绍怎样在 ComboBox 控件的 DrawItem 事件中使用 ImageList 的 Draw 方法在下拉列表中绘制图片，下面对本实例用到的关键技术进行详细讲解。

使用 ImageList 的 Draw 方法可以在指定的绘图上下文中绘制图像。语法格式如下：

```
public void Draw(Graphics g,int x,int y,int index)
```

Draw 方法中的参数说明如表 11.1 所示。

表 11.1　Draw 方法中的参数说明

参　数	描　述	参　数	描　述
g	要进行绘制的 Graphics 对象	y	图像绘制处的垂直位置
x	图像绘制处的水平位置	index	要绘制的 ImageList 中图像的索引

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PicturesInComboBox。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 控件和一个 Button 控件，ComboBox 控件用于演示带有图片信息的下拉列表，Button 控件用于向下拉列表中添加数据。

（3）程序的主要代码如下：

```
private void cbox_DisplayPictures_DrawItem(object sender, DrawItemEventArgs e)
{
    if (G_ImageList != null)                                          //判断 ImageList 是否为空
    {
        Graphics g = e.Graphics;                                      //得到绘图对象
        Rectangle r = e.Bounds;                                       //得到绘图范围
        Size imageSize = G_ImageList.ImageSize;                       //获取图像大小
        if (e.Index >= 0)                                             //判断是否有绘制项
        {
            Font fn = new Font("宋体", 10, FontStyle.Bold);           //创建字体对象
            string s = cbox_DisplayPictures.Items[e.Index].ToString(); //得到绘制项的字符串
            DrawItemState dis = e.State;
            if (e.State == (DrawItemState.NoAccelerator | DrawItemState.NoFocusRect))
            {
                e.Graphics.FillRectangle(new SolidBrush(Color.LightYellow), r);  //画条目背景
                G_ImageList.Draw(e.Graphics, r.Left, r.Top, e.Index);            //绘制图像
                e.Graphics.DrawString(s, fn, new SolidBrush(Color.Black),        //显示字符串
                    r.Left + imageSize.Width, r.Top);
                e.DrawFocusRectangle();                                          //显示取得焦点时的虚线框
            }
            else
            {
                e.Graphics.FillRectangle(new SolidBrush(Color.LightGreen), r);   //画条目背景
                G_ImageList.Draw(e.Graphics, r.Left, r.Top, e.Index);            //绘制图像
                e.Graphics.DrawString(s, fn, new SolidBrush(Color.Black),        //显示字符串
                    r.Left + imageSize.Width, r.Top);
                e.DrawFocusRectangle();                                          //显示取得焦点时的虚线框
            }
        }
    }
}
```

秘笈心法

心法领悟 230：使用 DrawString 方法轻松绘制文字。

实例中已经详细地介绍了怎样使用 ImageList 的 Draw 方法绘制图像信息，现在，也可以使用 Graphics 对象的 DrawString 方法方便地在指定的位置绘制文本信息，DrawString 方法提供了多个重载，在绘制文本内容时可以设置文本内容的字体、大小、颜色等信息。

实例 231　用 ComboBox 控件制作浏览器网址输入框

光盘位置：光盘\MR\11\231　　初级　　趣味指数：★★★★

实例说明

随着信息技术的不断发展，互联网已经融入了我们的工作与生活。可以在 IE 或其他浏览器的地址栏中输入指定的网址浏览网页信息。在地址栏中输入网址信息时，细心的读者可能会发现，地址栏带有一定的智能提示功能。在下面的实例中将会使用 ComboBox 控件制作浏览器网址输入框。实例运行效果如图 11.16 所示。

图 11.16　用 ComboBox 控件制作浏览器网址输入框

关键技术

本实例重点介绍 ComboBox 控件的 FindString 方法和 Select 方法的使用，下面对本实例用到的关键技术进行详细讲解。

（1）FindString 方法

使用 ComboBox 控件的 FindString 方法可以方便地查找数据项集合中指定数据项的索引。语法格式如下：

```
public int FindString(string s)
```

参数说明

❶ s：将要搜索的字符串对象。

❷ 返回值：返回整型数值，找到的第一个项从零开始的索引，如果未找到匹配项则返回-1。

（2）Select 方法

使用 ComboBox 控件的 Select 方法可以选择 ComboBox 可编辑部分中的文本范围。语法格式如下：

```
public void Select(int start,int length)
```

参数说明

❶ start：整型数值，文本框中当前选定文本的第一个字符的位置。

❷ length：整型数值，要选择的字符数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ResembleBrowser。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 下拉列表控件，用于演示网址输入框。

（3）程序的主要代码如下：

```
private void cbox_Url_TextChanged(object sender, EventArgs e)
{
    if (State)                                                    //当变量的值为真时
    {
        string importText = cbox_Url.Text;                        //获得输入的文本
        int index = cbox_Url.FindString(importText);              //在 ComboBox 集合中查找匹配的文本
        if (index >= 0)                                           //当有查找结果时
        {
            State = false;                                        //关闭编辑状态
            cbox_Url.SelectedIndex = index;                       //找到对应项
            State = true;                                         //打开编辑状态
            cbox_Url.Select(importText.Length, cbox_Url.Text.Length);  //设定文本的选择长度
        }
    }
}
```

秘笈心法

心法领悟 231：设置下拉列表中选中的内容为只读。

在程序运行中，如果不希望用户更改 ComboBox 下拉列表中选中的内容，可以设置 DropDownStyle 属性为 DropDownList，这样可以使用户选择的项为只读。

实例 232　实现带查询功能的 ComboBox 控件

初级

光盘位置：光盘\MR\11\232

趣味指数：★★★☆

实例说明

ComboBox 控件可以方便地显示多项数据内容，通过设置 ComboBox 控件的 AutoCompleteSource 属性和 AutoCompleteMode 属性，可以实现从 ComboBox 控件中查询已存在的项，自动完成控件内容的输入。当用户在

ComboBox 控件中输入一个字符时，ComboBox 控件会自动列出最有可能与之匹配的选项，如果符合用户的要求，则直接确认，从而加快用户输入。实例运行效果如图 11.17 所示。

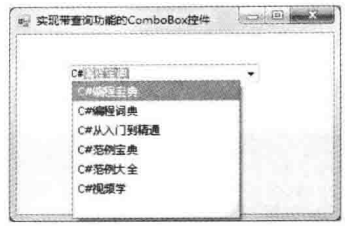


图 11.17　实现带查询功能的 ComboBox 控件

关键技术

本实例重点介绍怎样使用 ComboBox 控件的 AutoCompleteMode 属性和 AutoCompleteSource 属性实现带有查询功能的 ComboBox 控件，下面对本实例用到的关键技术进行详细讲解。

（1）AutoCompleteMode 属性

该属性用来获取或设置控制自动完成的模式。语法格式如下：

```
public AutoCompleteMode AutoCompleteMode { get; set; }
```

参数说明

属性值：AutoCompleteMode 枚举值。这些值分别为 AutoAppend、AutoSuggest、AutoSuggestAppend 和 None，默认为 None。

（2）AutoCompleteSource 属性

该属性用来获取或设置一个枚举值，这个枚举值用于设置自动完成字符串的源。语法格式如下：

```
public AutoCompleteSource AutoCompleteSource { get; set; }
```

参数说明

属性值：AutoCompleteSource 枚举值。这些值分别为 AllSystemSources、AllUrl、FileSystem、HistoryList、RecentlyUsedList、CustomSource 和 None，默认为 None。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ComboBoxFind。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ComboBox 下拉列表控件和一个 Button 控件，ComboBox 控件用于演示查询功能，Button 控件用于设置 ComboBox 控件的查询功能。

（3）程序的主要代码如下。

向 ComboBox 控件中添加元素的代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    cbox_Find.Items.Clear();                          //清空 ComboBox 集合
    cbox_Find.Items.Add("C#编程词典");                 //向 ComboBox 集合添加元素
    cbox_Find.Items.Add("C#编程宝典");                 //向 ComboBox 集合添加元素
    cbox_Find.Items.Add("C#视频学");                   //向 ComboBox 集合添加元素
    cbox_Find.Items.Add("C#范例宝典");                 //向 ComboBox 集合添加元素
    cbox_Find.Items.Add("C#从入门到精通");             //向 ComboBox 集合添加元素
    cbox_Find.Items.Add("C#范例大全");                 //向 ComboBox 集合添加元素
}
```

设置 ComboBox 控件查询功能的代码如下：

```
private void btn_Begin_Click(object sender, EventArgs e)
{
    cbox_Find.AutoCompleteMode =                      //设置自动完成的模式
        AutoCompleteMode.SuggestAppend;
    cbox_Find.AutoCompleteSource =                    //设置自动完成字符串的源
        AutoCompleteSource.ListItems;
}
```

秘笈心法

心法领悟 232：设置 ComboBox 控件中数据项文本的字体。

通过 ComboBox 控件的 Font 属性，可以方便地设置控件中文本内容的字体、大小等信息，也可以通过控件的 ForeColor 属性方便地设置控件内文本内容的文字颜色。

11.4 RichTextBox 控件应用

实例 233	在 RichTextBox 控件中添加超链接文字	中级
	光盘位置：光盘\MR\11\233	趣味指数：★★★☆

实例说明

程序设计过程中，经常会用到 RichTextBox 控件，该控件的功能非常强大，主要用来输入输出和编辑文本信息。本实例中将会演示如何向 RichTextBox 控件中添加超链接文字信息。实例运行效果如图 11.18 所示。

图 11.18 在 RichTextBox 控件中添加超链接文字

关键技术

本实例重点介绍怎样使用 RichTextBox 控件的 AppendText 方法，向控件中添加超链接文本，以及使用 Process 类的 Start 方法来启动一个进程资源。下面对本实例用到的关键技术进行详细讲解。

（1）AppendText 方法

使用 RichTextBox 控件的 AppendText 方法可以方便地向控件中添加文本内容。语法格式如下：

```
public void AppendText(string textData)
```

参数说明

textData：字符串对象，将指定字符串追加到 RichTextBox 控件。

（2）Start 方法

使用 Process 类的 Start 方法可以用来启动一个进程资源。语法格式如下：

```
public static Process Start(string fileName,string arguments)
```

参数说明

❶ fileName：字符串对象，要在该进程中运行的应用程序文件的名称。

❷ arguments：字符串对象，启动该进程时传递的命令行参数。

说明：Process 类的 Start 方法提供了多个重载，例如，可以使用“Process.Start("iexplore.exe");”方便地打开 IE 浏览器应用程序。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddHyperLink。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件和一个 Button 控件，RichTextBox 控件用于显示带有超链接的文本信息，Button 控件用于向 RichTextBox 控件中添加文本信息。

（3）程序的主要代码如下：

```
private void btn_Add_Click(object sender, EventArgs e)
{
    rtbox_HyperLink.AppendText(                                    //向控件中添加文本信息
        @"明日科技：http://www.mingrisoft.com/
谷歌：http://www.google.cn/
网易：http://www.163.com/
百度：http://www.baidu.com/
CSDN：http://www.csdn.net/
腾讯：http://www.qq.com/
```

```
QQ 书签：http://shuqian.qq.com/
QQ 空间：http://qzone.qq.com/
校内网：http://www.xiaonei.com/");
}
```

秘笈心法

心法领悟 233：向 RichTextBox 控件中添加文本信息的另一种方法。

实例中使用了 RichTextBox 控件的 AppendText 方法向控件中添加文本内容，使用控件的 Text 属性也可以实现同样的功能，Text 属性接收一个字符串对象，如果向 RichTextBox 控件中添加文本内容，只要设置其 Text 属性即可。

实例 234	在 RichTextBox 控件中插入图片 光盘位置：光盘\MR\11\234	中级 趣味指数：★★★★☆

实例说明

可以使用 RichTextBox 控件方便地显示文本信息，也可以使用 Font 属性更改控件中文字字体的样式。本实例中将会介绍一个有趣的功能，通过系统剪贴板，轻松地向 RichTextBox 控件中插入图片信息。实例运行效果如图 11.19 所示。

图 11.19　在 RichTextBox 控件中插入图片

关键技术

本实例重点介绍怎样使用 Clipboard 类的 SetDataObject 方法，将数据放入系统剪贴板，以及使用 RichTextBox 控件的 Paste 方法，将系统剪贴板的内容粘贴到控件中。下面对本实例用到的关键技术进行详细讲解。

（1）SetDataObject 方法

使用 Clipboard 控件的 SetDataObject 方法可以在系统剪贴板上存储指定的数据对象。语法格式如下：

```
public static void SetDataObject(Object data,bool copy)
```

参数说明

❶ data：object 类型的对象，要放置在系统剪贴板上的数据对象。

❷ copy：布尔类型数值，如果应用程序退出时要将数据保留在系统剪贴板中，参数值为 true；如果应用程序退出时要将数据从系统剪贴板中清除，参数值则为 false。

（2）Paste 方法

使用 RichTextBox 控件的 Paste 方法，可以将剪贴板的内容粘贴到 RichTextBox 控件中的当前选定内容上。语法格式如下：

```
public void Paste()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InsertImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件和一个 Button 控件，RichTextBox 控件用于显示被插入的图片信息，Button 控件用于向 RichTextBox 控件中插入图片信息。

（3）程序的主要代码如下：

```
private void btn_InsertImage_Click(object sender, EventArgs e)
{
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                    //创建打开文件对话框对象
    P_OpenFileDialog.Filter = "*.jpg|*.jpg|*.bmp|*.bmp";                       //筛选文件
    DialogResult P_DialogResult = P_OpenFileDialog.ShowDialog();               //弹出打开文件对话框
    if (P_DialogResult==DialogResult.OK)                                       //判断是否选中文件
    {
        Clipboard.SetDataObject(Image.FromFile(P_OpenFileDialog.FileName), false);   //将图像放入剪贴板
        if (rtbox_Display.CanPaste(DataFormats.GetFormat(DataFormats.Bitmap)))       //判断剪贴板内是否是图像
        {
            rtbox_Display.Paste();                                             //粘贴剪贴板的内容
        }
    }
}
```

秘笈心法

心法领悟 234：方便地使用 Cut 方法。

实例中已经详细地介绍了怎样使用 Copy 方法将控件中的内容复制到系统剪贴板，以及使用 Paste 方法将系统剪贴板的内容粘贴到控件中。同样地，也可以使用 Cut 方法从 RichTextBox 控件中删除当前选定内容，并将其复制到系统剪贴板，实现剪切功能。

实例 235　在 RichTextBox 控件中显示 RTF 格式的文件

光盘位置：光盘\MR\11\235　　中级　　趣味指数：★★★☆

实例说明

Windows 操作系统自带的“写字板”应用程序，具有很强的文本编辑能力，可以方便地向写字板文档中写入文本信息、编辑文本样式、插入图片信息等，写字板文档保存的格式为 RTF。本实例中将会演示如何使用 RichTextBox 控件方便地打开并编辑 RTF 文件。实例运行效果如图 11.20 所示。

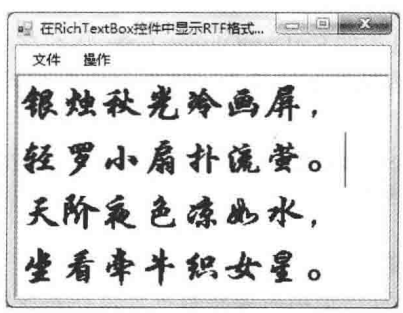


图 11.20　在 RichTextBox 控件中显示 RTF 格式的文件

关键技术

本实例主要用到了 RichTextBox 控件的 LoadFile 方法和 Clear 方法，以及 OpenFileDialog 对象的 ShowDialog 方法，下面分别对它们进行详细讲解。

（1）LoadFile 方法

使用 RichTextBox 控件的 LoadFile 方法，可以将文件的内容加载到 RichTextBox 控件中。语法格式如下：

```
public void LoadFile(string path,RichTextBoxStreamType fileType)
```

参数说明

❶ path：字符串对象，要加载到控件中的文件的名称和位置。

❷ fileType：RichTextBoxStreamType 枚举值，用于选择打开文件的类型。

（2）Clear 方法

使用 RichTextBox 控件的 Clear 方法，可以清除 RichTextBox 控件中的所有文本。语法格式如下：

```
public void Clear()
```

（3）OpenFileDialog 对象的 ShowDialog 方法

使用 OpenFileDialog 对象的 ShowDialog 方法，可以用来弹出打开文件对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举值，如果用户单击“确定”按钮，正确选择文件，则 DialogResult 的值为 DialogResult.OK，否则为 DialogResult.Cancel。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayRTFFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件和一个 MenuStrip 菜单控件，RichTextBox 控件用于显示 RTF 文件内容，MenuStrip 菜单中的功能项用于打开和保存 RTF 文件。

（3）打开 RTF 文档的代码如下：

```
private void 打开 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    ((ToolStripMenuItem)Menu_Main.Items["文件 ToolStripMenuItem"])
        DropDownItems["保存 ToolStripMenuItem"].Enabled = false;                    //停用保存功能
    G_OpenFileDialog.Filter = "RTF 文件(*.RTF)|*.RTF";                              //设置打开文件的过滤参数
    if (G_OpenFileDialog.ShowDialog() == DialogResult.OK && G_OpenFileDialog.FileName.Length > 0)
    {
        fileName = G_OpenFileDialog.FileName;                                       //保存打开文件的文件名
        this.richTextBox1.LoadFile(fileName, RichTextBoxStreamType.RichText);       //从指定位置加载 RTF 文件
    }
}
```

（4）保存 RTF 文档的代码如下：

```
private void 保存 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (File.Exists(fileName))                                                      //如果存在该文件
    {
        richTextBox1.SaveFile(fileName, RichTextBoxStreamType.RichNoOleObjs);       //在指定路径下保存
        MessageBox.Show("保存成功！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        richTextBox1.Clear();                                                       //清空 RichTextBox 控件中的内容
    }
    else                                                                            //当不存在该文件时
    {
        G_SaveFileDialog.Filter = "RTF 文件(*.RTF)|*.RTF";                          //设置保存文件的保存格式
        if (G_SaveFileDialog.ShowDialog() == DialogResult.OK && G_SaveFileDialog.FileName.Length > 0)
        {
            richTextBox1.SaveFile(G_SaveFileDialog.FileName + ".RTF");              //在指定位置下保存 RTF 文件
        }
    }
}
```

秘笈心法

心法领悟 235：使控件得到焦点。

RichTextBox 控件提供了很多的方法、属性和事件，对于知识的掌握一定要活学活用，可以使用 LoadFile 方法打开文件，使用 Clear 方法清空控件内的文本内容，那么，怎样才可以使 RichTextBox 控件得到焦点呢？可

以通过调用 RichTextBox 控件的 Focus 方法来实现。

实例 236　使用 RichTextBox 控件保存文件

光盘位置：光盘\MR\11\236　　中级　　趣味指数：★★★☆

实例说明

RichTextBox 控件有着强大的文本编辑功能，可以使用 Select 方法方便地选中控件中的文本，并使用 SelectedText 属性读取或更改选中的文本，也可以使用 SelectionColor 属性方便地设置指定文本内容的文字颜色。本实例中将会演示如何将 RichTextBox 控件中的文本保存为 RTF 文件或 txt 文件。实例运行效果如图 11.21 所示。

图 11.21　使用 RichTextBox 控件保存文件

关键技术

本实例主要用到了 RichTextBox 控件的 SaveFile 方法，以及 SaveFileDialog 对象的 ShowDialog 方法，下面分别对它们进行详细讲解。

（1）SaveFile 方法

使用 RichTextBox 控件的 SaveFile 方法，可以将 RichTextBox 控件中的内容保存到文件中。语法格式如下：

```
public void SaveFile(string path,RichTextBoxStreamType fileType)
```

参数说明

❶ path：字符串对象，要加载到控件中的文件的名称和位置。

❷ fileType：RichTextBoxStreamType 枚举值，用于选择保存文件的类型。

（2）SaveFileDialog 对象的 ShowDialog 方法

使用 SaveFileDialog 对象的 ShowDialog 方法，可以用来弹出保存文件对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举值，如果用户单击“确定”按钮，正确保存文件，则 DialogResult 的值为 DialogResult.OK，否则为 DialogResult.Cancel。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SaveFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件和一个 MenuStrip 菜单控件，RichTextBox 控件用于显示 RTF 文件内容，MenuStrip 菜单中的功能项用于打开 RTF 文件和保存 txt 文件。

（3）打开 RTF 文档的代码如下：

```
private void 打开 RTFToolStripMenuItem_Click(object sender, EventArgs e)
{
    G_OpenFileDialog.Filter = "text.rtf|*.rtf*";                    //筛选文件信息
    if (this.G_OpenFileDialog.ShowDialog() == DialogResult.OK)      //判断是否打开文件
    {
        rtbox_Display.LoadFile(                                     //载入 rtf 文件
            G_OpenFileDialog.FileName, RichTextBoxStreamType.RichText);
    }
}
```

（4）保存为 txt 文件的代码如下：

```
private void 保存成 TXT 文件 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (rtbox_Display.Text != "")                                   //判断控件中是否有文本内容
    {
```

```
        G_SaveFileDialog.DefaultExt = "*.txt";                          //设置文件默认扩展名
        G_SaveFileDialog.Filter = "Txt Files|*.txt";                    //筛选文件信息
        if (this.G_SaveFileDialog.ShowDialog() == DialogResult.OK)      //判断是否确认保存文件
        {
            rtbox_Display.SaveFile(                                     //保存文件
                this.G_SaveFileDialog.FileName, RichTextBoxStreamType.PlainText);
            MessageBox.Show("保存成功", "信息提示",                      //弹出消息对话框
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    else
    {
        MessageBox.Show("请打开文件", "信息提示",                        //弹出消息对话框
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 236：设置默认文件扩展名。

实例中已经介绍了使用 SaveFileDialog 对象的 ShowDialog 方法可以方便地弹出保存文件对话框，在该对话框中可以填写文件名称与文件扩展名，单击“确定”按钮后保存文件；也可以通过设置 SaveFileDialog 对象的 DefaultExt 属性，添加默认文件扩展名。

实例 237　为 RichTextBox 控件添加自定义滚动条

光盘位置：光盘\MR\11\237　　中级　　趣味指数：★★★☆

实例说明

如果向 RichTextBox 控件中添加过多的文本内容，那么一部分内容将会被控件隐藏，此时，可以通过设置 RichTextBox 控件的 ScrollBars 属性，为 RichTextBox 控件添加滚动条，用户可以拖动滚动条方便地浏览控件中的文本内容。本实例中将会演示如何为 RichTextBox 控件添加自定义滚动条。实例运行效果如图 11.22 所示。

图 11.22　为 RichTextBox 控件添加自定义滚动条

关键技术

本实例主要用到了 RichTextBox 控件的 GetFirstCharIndexFromLine 方法、ScrollToCaret 方法以及 SelectionStart 属性，下面分别对它们进行详细讲解。

（1）GetFirstCharIndexFromLine 方法

RichTextBox 控件的 GetFirstCharIndexFromLine 方法用于检索给定行第一个字符的索引。语法格式如下：

```
public int GetFirstCharIndexFromLine(int lineNumber)
```

参数说明

❶ lineNumber：整型数值，将要获取其第一个字符的索引的行数。

❷ 返回值：返回整型数值，位于指定行中从零开始的字符索引。

（2）ScrollToCaret 方法

使用 RichTextBox 控件的 ScrollToCaret 方法，可以将控件的内容滚动到当前插入符号位置。语法格式如下：

```
public void ScrollToCaret()
```

（3）SelectionStart 属性

使用 RichTextBox 控件的 SelectionStart 属性，用来获取或设置文本框中选定的文本起始点。语法格式如下：

```
public int SelectionStart { get; set; }
```

参数说明

属性值：整型数值，获取或选定文本框中选定的文本的起始位置。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddScroll。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件，用于显示文本内容；添加两个 Button 控件，用于滚动 RichTextBox 中的文本内容。

（3）程序的主要代码如下：

```
private void btn_Up_Click(object sender, EventArgs e)
{
    i = --i > -1 ? i : ++i;                                    //计算滚动的行数
    start = rtbox_Display.GetFirstCharIndexFromLine(i);        //得到行首第一个字符索引
    rtbox_Display.SelectionStart = start;                      //设置文本框选定的起始点
    rtbox_Display.ScrollToCaret();                             //滚动到起始点位置
}
private void btn_Down_Click(object sender, EventArgs e)
{
    i =++i < rtbox_Display.Lines.Length ? i : --i;             //计算滚动的行数
    start = rtbox_Display.GetFirstCharIndexFromLine(i);        //得到行首第一个字符索引
    rtbox_Display.SelectionStart = start;                      //设置文本框选定的起始点
    rtbox_Display.ScrollToCaret();                             //滚动到起始点位置
}
```

秘笈心法

心法领悟 237：适当地使用 ScrollBar 属性。

实例中详细介绍了使用 RichTextBox 控件的 GetFirstCharIndexFromLine 方法、ScrollToCaret 方法以及 SelectionStart 属性实现自定义滚动条，当然，也可以通过 RichTextBox 控件的 ScrollBar 属性来设置和使用控件自带的滚动条。

实例 238　在 RichTextBox 控件中实现关键字描红

光盘位置：光盘\MR\11\238　　中级　　趣味指数：★★★☆

实例说明

在字符串操作中，可以使用 IndexOf 方法与 LastIndexOf 方法方便地在字符串中查找指定的字符串信息。同样地，也可以使用此方法查找 RichTextBox 控件中指定的文本信息，找到文本内容后并将文本内容的文本颜色设置为红色。实例运行效果如图 11.23 所示。

图 11.23　在 RichTextBox 控件中实现关键字描红

关键技术

本实例主要用到了 RichTextBox 控件的 Select 方法和 SelectionColor 属性，下面分别对它们进行详细讲解。

（1）Select 方法

RichTextBox 控件的 Select 方法，用于在 RichTextBox 控件中选择一个文本范围。语法格式如下：

```
public void Select(int start,int length)
```

参数说明

❶ start：整型数值，表示文本框中当前选定文本的第一个字符的位置。

❷ length：整型数值，表示要选择的字符数。

（2）SelectionColor 属性

RichTextBox 控件的 SelectionColor 属性，用于获取或设置当前选定文本或插入点的文本颜色。语法格式如下：

```
public Color SelectionColor { get; set; }
```

参数说明

属性值：Color 结构，表示应用到当前选定文本的颜色。

说明：可以使用 SelectionFont 属性方便地设置当前选定文本或插入点的字体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FindKey。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件，用于显示文本信息和被描红的文本信息；添加一个 TextBox 控件，用于添加查找字符串；添加两个 Button 控件，分别用于查找字符串和打开文本文件。

（3）程序的主要代码如下：

```
private void btn_query_Click(object sender, EventArgs e)
{
    int i = 0;                                                    //定义计数器
label1:                                                           //定义标签
    if (G_str_array[i].Contains(txt_query.Text))                  //判断是否找到图书
    {
        lbox_str.SelectedIndex = i;                               //选中查找到的结果
        MessageBox.Show(txt_query.Text + " 已经找到！", "提示！");  //提示找到信息
        return;
    }
    i++;
    if (i < G_str_array.Length) goto label1;                      //条件满足则跳转到标签
    MessageBox.Show(txt_query.Text + " 没有找到！", "提示！");     //提示未找到信息
}
```

秘笈心法

心法领悟 238：怎样获取控件中被选中文本的字符数量？

在 RichTextBox 控件中，可以使用 Text 属性得到控件中的文本内容，文本内容是一个字符串对象，通过字符串对象的 Length 属性，可以方便地计算出控件中包含的字符数量。现在问题出现了，要怎样才可以得到 RichTextBox 控件中被选中文本的字符数量？可以使用控件的 SelectionLength 属性轻松地实现。

实例 239　在 RichTextBox 控件中替换文本文字

光盘位置：光盘\MR\11\239　　中级　　趣味指数：★★★☆

实例说明

可以在 RichTextBox 控件中方便地操作文本内容，设置指定文本的字体、大小、颜色等。本实例中将会演示如何替换 RichTextBox 控件中的文本内容，首先选中将要被替换的文本内容，然后单击“开始替换”按钮，此时会将 RichTextBox 控件中选中的文本替换为指定的文本内容。实例运行效果如图 11.24 所示。

图 11.24　在 RichTextBox 控件中替换文本文字

关键技术

本实例主要用到了 RichTextBox 控件的 SelectedText 属性，下面对其进行详细讲解。

RichTextBox 控件的 SelectedText 属性用来获取或设置 RichTextBox 控件中选定的文本内容。语法格式如下：

```
public override string SelectedText { get; set; }
```

参数说明

属性值：字符串对象，表示控件中选定文本的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReplaceTxt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件，用于显示文本信息；添加一个 TextBox 控件，用于输入替换信息；添加一个 Button 控件，用于替换 RichTextBox 中的文本信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (this.richTextBox1.SelectedText.ToString() == "")
    {
        MessageBox.Show(                                                    //弹出提示信息
            "请选重要替换的文本", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        this.richTextBox1.SelectedText = txt_Content.Text;                  //替换选中的文本
    }
}
```

秘笈心法

心法领悟 239：方便地更改文本字体。

使用 RichTextBox 控件的 SelectionFont 属性可以更改选中文本的字体，要怎样才可以设置控件中所有文本的字体样式呢？可以通过设置 RichTextBox 控件的 Font 属性来实现。

实例 240　在 RichTextBox 控件中实现项目符号功能

光盘位置：光盘\MR\11\240　　中级　　趣味指数：★★★★☆

实例说明

在对大量数据进行统计时，Word 应用程序为用户提供了众多方法，如项目编号和项目符号等。也可以使用 RichTextBox 控件实现此功能。本实例将在 RichTextBox 控件中为文本信息添加项目符号和项目编号。实例运行效果如图 11.25 所示。

图 11.25　在 RichTextBox 控件中实现项目符号功能

关键技术

本实例主要用到了 RichTextBox 控件的 SelectionBullet 属性和 File 类的 Exists 方法，下面分别对它们进行详

细讲解。

（1）SelectionBullet 属性

RichTextBox 控件的 SelectionBullet 属性，用来指示项目符号样式是否应用到当前选定内容或插入点。语法格式如下：

```
public bool SelectionBullet { get; set; }
```

参数说明

属性值：布尔值，如果当前选定内容或插入点应用了项目符号样式，值为 true，否则值为 false。

（2）File 类的 Exists 方法

File 类的 Exists 方法，用于确定指定的文件是否存在。语法格式如下：

```
public static bool Exists(string path)
```

参数说明

❶ path：字符串对象，检查是否存在的文件。

❷ 返回值：返回布尔值，确定文件是否存在，如果存在为 true，否则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayNumber。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 4 个 Button 按钮，分别用于打开 RTF 文件，保存 RTF 文件，为 RichTextBox 控件中的文本添加项目符号，为 RichTextBox 控件中的文本添加数字编号。

（3）程序的主要代码如下：

```
[DllImport("user32.dll", CharSet = System.Runtime.InteropServices.CharSet.Auto)]
private static extern IntPtr SendMessage(HandleRef hWnd, int msg, int wParam,
    [In,Out,MarshalAs(UnmanagedType.LPStruct)] PARAFORMAT2 lParam);          //定义一个发送消息的 API 函数
private AdvRichTextBulletType _BulletType = AdvRichTextBulletType.Number;    //设定项目编号的起始类型
private AdvRichTextBulletStyle _BulletStyle = AdvRichTextBulletStyle.Period; //设定项目编号的起始样式
private short _BulletNumberStart = 1;                                        //设定项目编号的起始数字为 1
public AdvRichTextBulletType BulletType
{
    get { return _BulletType; }                                              //返回项目符号的类型
    set
    {
        _BulletType = value;                                                 //为项目符号的类型赋值
        NumberedBullet(true);                                                //设定新实例的各个属性
    }
}
public AdvRichTextBulletStyle BulletStyle
{
    get { return _BulletStyle; }                                             //返回项目符号的样式
    set
    {
        _BulletStyle = value;                                                //为项目符号样式设定值
        NumberedBullet(true);                                                //设定新实例的各个属性
    }
}
public void NumberedBullet(bool TurnOn)
{
    PARAFORMAT2 paraformat1 = new PARAFORMAT2();                             //初始化指定类的一个新实例
    paraformat1.dwMask = (int)(PFM_NUMBERING | PFM_OFFSET | PFM_NUMBERINGSTART |
        PFM_NUMBERINGSTYLE | PFM_NUMBERINGTAB);                              //设置实例的 dwMask 属性
    if(!TurnOn)                                                              //当和 TurnOn 的初始值相反时
    {
        paraformat1.wNumbering = 0;                                          //设置 wNumbering 属性为 0
        paraformat1.dxOffset = 0;                                            //设置 dxOffset 属性为 0
    }
    Else                                                                     //当和 TurnOn 的初始值相同时
    {
        paraformat1.wNumbering = (short)_BulletType;                         //设置 wNumbering 的值
        paraformat1.dxOffset = this.BulletIndent;                            //设置 dxOffset 的值
        paraformat1.wNumberingStyle = (short)_BulletStyle;                   //设置项目编号的样式
```

```
        paraformat1.wNumberingStart = _BulletNumberStart;                    //设置项目编号的起始位置
        paraformat1.wNumberingTab = 500;                                     //设置按 Tab 键文本移动的距离
    }
    SendMessage(new System.Runtime.InteropServices.HandleRef(this, this.Handle),
        0x447,0,paraformat1);                                                //发送指定的消息
}
```

秘笈心法

心法领悟 240：轻松操作文件。

实例中使用了 File 类的 Exists 静态方法判断指定的文件是否存在，File 类提供了多种操作文件的方法，可以使用 Create 方法创建文件，使用 Delete 方法删除文件，还可以使用 Move 方法移动文件。

实例 241 设置 RichTextBox 控件中的文本对齐方式

光盘位置：光盘\MR\11\241　　中级　　趣味指数：★★★☆

实例说明

在 Word 文档中可以方便地设置文本内容的对齐方式，可以将文本设置为左对齐、右对齐、居中对齐方式。而在 RichTextBox 控件中也可以实现此功能，本实例中将会演示怎样在 RichTextBox 控件中设置文本内容的对齐方式。实例运行效果如图 11.26 所示。

图 11.26　设置 RichTextBox 控件中的文本对齐方式

关键技术

本实例主要用到了 RichTextBox 控件的 SelectionAlignment 属性，下面对其进行详细讲解。

RichTextBox 控件的 SelectionAlignment 属性用来获取或设置应用到当前选定内容或插入点的对齐方式。语法格式如下：

```
public HorizontalAlignment SelectionAlignment { get; set; }
```

参数说明

属性值：HorizontalAlignment 枚举值，用来设置对齐方式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TxtAlignment。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 RichTextBox 控件，用于显示文本信息；添加 5 个 Button 控件，分别用于打开 RTF 文件、保存 RTF 文件和设置 RichTextBox 控件中文本内容的对齐方式。

（3）程序的主要代码如下：

```
private void justifyCenter_Click(object sender,EventArgs e)
{
    this.richTextBox1.SelectionAlignment = HorizontalAlignment.Center;       //设置选定的文本为居中对齐
}
private void justifyLeft_Click(object sender,EventArgs e)
{
    this.richTextBox1.SelectionAlignment = HorizontalAlignment.Left;         //设置选定的文本为左对齐
}
private void justifyRight_Click(object sender,EventArgs e)
{
    this.richTextBox1.SelectionAlignment = HorizontalAlignment.Right;        //设置选定的文本为右对齐
}
```

秘笈心法

心法领悟 241：轻松设置被选择文本的颜色。

实例中使用了 RichTextBox 控件的 SelectionAlignment 属性设置被选中文本的对齐方式，RichTextBox 控件提供了很多操作控件中文本的方法，可以使用 SelectionColor 方法方便地设置被选择文本的颜色。

11.5　ListBox 控件应用

实例 242	在 ListBox 控件中实现选择项功能 光盘位置：光盘\MR\11\242	中级 趣味指数：★★★☆

实例说明

程序设计过程中，经常会用到 ListBox 控件，该控件中可以显示多条数据，可以使用 ListBox 控件中 Items 集合的 Add 方法向控件中添加数据信息。也可以通过 SelectedIndex 属性得到被选择数据项的索引。本实例中将会演示怎样设置 ListBox 控件选择项。实例运行效果如图 11.27 所示。

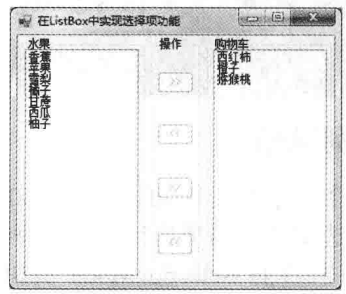


图 11.27　在 ListBox 控件中实现选择项功能

关键技术

本实例主要用到了 ListBox 控件中 Items 集合的 Add 方法和 Remove 方法，下面分别对它们进行详细讲解。

（1）Add 方法

ListBox 控件的 Add 方法，用于向 ListBox 的项列表添加数据项。语法格式如下：

```
public int Add(Object item)
```

参数说明

❶ item：object 对象，表示要添加到集合中的数据项。

❷ 返回值：返回整型数值，表示添加到集合中的数据项在集合中的索引。

（2）Remove 方法

使用 ListBox 控件的 Remove 方法，可以从集合中移除指定的数据项。语法格式如下：

```
public void Remove(Object value)
```

参数说明

value：object 对象，表示要从集合中移除的数据项。

说明：ListBox 控件的 Items 集合，实现了 Ilist 接口，可以方便地使用 Add 方法向集合中添加数据项，使用 Remove 方法删除集合中的数据项，也可以使用 Insert 方法向集合中插入数据项。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ListBoxItem。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 ListBox 控件，用于演示在两个 ListBox 控件间移动数据；添加 4 个 Button 控件，用于移动两个 ListBox 控件中的数据。

（3）程序的主要代码如下：

```
private void ListBoxItem_Load(object sender,EventArgs e)
{
    listBox1.Items.Add("香蕉");                                    //向 listBox1 控件中添加“香蕉”
```

```
    listBox1.Items.Add("苹果");                    //向 listBox1 控件中添加“苹果”
    listBox1.Items.Add("雪梨");                    //向 listBox1 控件中添加“雪梨”
    listBox1.Items.Add("西红柿");                  //向 listBox1 控件中添加“西红柿”
    listBox1.Items.Add("橘子");                    //向 listBox1 控件中添加“橘子”
    listBox1.Items.Add("甘蔗");                    //向 listBox1 控件中添加“甘蔗”
    listBox1.Items.Add("西瓜");                    //向 listBox1 控件中添加“西瓜”
    listBox1.Items.Add("橙子");                    //向 listBox1 控件中添加“橙子”
    listBox1.Items.Add("柚子");                    //向 listBox1 控件中添加“柚子”
    listBox1.Items.Add("猕猴桃");                  //向 listBox1 控件中添加“猕猴桃”
    DecideTrueOrFalse();                           //如果不存在选项停用按钮
}
```

秘笈心法

心法领悟 242：得到控件中被选择的数据项。

实例中使用了 ListBox 控件中 Items 集合的 Add 方法和 Remove 方法，方便地向 ListBox 控件中添加和删除数据项，那么，要怎样才可以得到控件中被选择的数据项呢？可以使用 ListBox 控件的 SelectedItem 属性来实现。

实例 243	选中 ListBox 控件中全部条目	中级
	光盘位置：光盘\MR\11\243	趣味指数：★★★★☆

实例说明

程序运行过程中，可以选中 ListBox 控件中的数据项，并根据选中的数据项执行相应的操作。也可以通过设置 ListBox 控件的 SelectionMode 属性更改选择数据项的方式。本实例中将会演示怎样通过代码的方式选中 ListBox 控件中所有的数据项。实例运行效果如图 11.28 所示。

图 11.28　选中 ListBox 控件中全部条目

关键技术

本实例主要用到了 ListBox 控件的 SelectedItems 属性和 SetSelected 方法，下面分别对它们进行详细讲解。

（1）SelectedItems 属性

ListBox 控件的 SelectedItems 属性，可以用来获取 ListBox 控件中被选中数据项的集合。语法格式如下：

```
public ListBox.SelectedObjectCollection SelectedItems { get; }
```

参数说明

属性值：ListBox.SelectedObjectCollection 数据项集合，ListBox 控件中被选中数据项的集合。

（2）SetSelected 方法

ListBox 控件的 SetSelected 方法，用于选择或清除 ListBox 控件中选定的数据项。语法格式如下：

```
public void SetSelected(int index,bool value)
```

参数说明

❶ index：整型数值，ListBox 控件中要选择或清除对其选定的项的从零开始的索引。

❷ value：布尔值，如果选择指定的项则值为 true，否则值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectAll。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示多个数据项；添加两个 Button 控件，分别用于选中 ListBox 控件中的所有项和显示选中项的数量。

（3）程序的主要代码如下：

```
private void bntList_Click(object sender, EventArgs e)
{
```

```
    this.listBox1.SelectionMode = SelectionMode.MultiExtended;                    //可能选多项
    for (int i = 0; i < listBox1.Items.Count; i++)                                //遍历数据项集合
    {
        this.listBox1.SetSelected(i, true);                                       //选定数据项
    }
}
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(listBox1.SelectedItems.Count.ToString()+"项被选中","提示！ "); //弹出消息对话框
}
```

秘笈心法

心法领悟 243：使用索引的方式设置控件中被选中的数据项。

使用 ListBox 控件的 SelectIndex 属性，可以按数据项索引的方式，轻松地获取或设置被选中的数据项索引。SelectIndex 属性与实例中的 SetSelected 方法都可以选中控件中的数据项，但是 SetSelected 方法可以选择多个数据项。

实例 244　对 ListBox 控件中的数据进行排序

光盘位置：光盘\MR\11\244　　中级　　趣味指数：★★★☆

实例说明

ListBox 控件的 Items 集合中保存着多个数据项，可以使用 for 或 foreach 方便地遍历 Items 集合中的每一个数据项，也可以使用 Items 集合的 Add 或 Remove 方法方便地向集合中添加和删除数据项。本实例中将会演示如何对 ListBox 控件中的数据项进行排序。实例运行效果如图 11.29 所示。

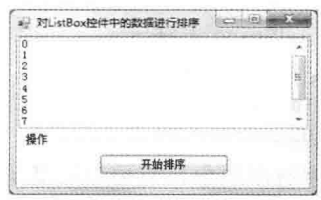


图 11.29　对 ListBox 控件中的数据进行排序

关键技术

本实例主要用到了 ListBox 控件的 Sorted 属性，下面对其进行详细讲解。

ListBox 控件的 Sorted 属性用来指示 ListBox 控件中的数据项是否按字母顺序排序。语法格式如下：

```
public bool Sorted { get; set; }
```

参数说明

属性值：布尔值，如果按字母排序 ListBox 控件中的数据项，则此属性值为 true，否则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DataSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示多个数据项；添加一个 Button 控件，用于对 ListBox 控件中的数据项排序。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Sorted = true;                                                       //启用排序
}
```

秘笈心法

心法领悟 244：设置 ListBox 控件中文本的字体。

ListBox 控件的 Font 属性接收一个 Font 对象，可以通过此 Font 属性设置 ListBox 控件中文本的字体、大小等信息。

实例 245 在 ListBox 控件中查找指定项

光盘位置：光盘\MR\11\245　　中级　　趣味指数：★★★★☆

实例说明

ListBox 控件中包含着多个数据项的集合，可以方便地添加、删除数据项，也可以简单地对数据项进行排序。本实例中可以使用 FindString 方法方便地查找指定字符串在 ListBox 控件数据项集合中的索引。实例运行效果如图 11.30 所示。

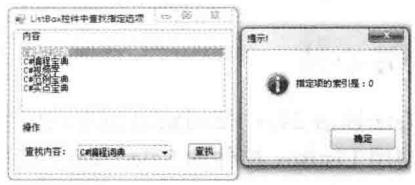


图 11.30　在 ListBox 控件中查找指定项

关键技术

本实例主要用到了 ComboBox 控件的 SelectedIndex 属性和 ListBox 控件的 FindString 方法，下面分别对它们进行详细讲解。

（1）ComboBox 控件的 SelectedIndex 属性

ComboBox 控件的 SelectedIndex 属性，用来获取或设置指定当前选定的数据项的索引。语法格式如下：

```
public override int SelectedIndex { get; set; }
```

参数说明

属性值：整型数值，当前选定项的从零开始的索引。

（2）FindString 方法

ListBox 控件的 FindString 方法，用于查找 ListBox 中以指定字符串开头的第一个项。语法格式如下：

```
public int FindString(string s)
```

参数说明

❶ s：字符串对象，将要获取其第一个字符的索引的行数。

❷ 返回值：返回整型数值，找到与指定字符串匹配的第一个数据项的从零开始的索引。

说明：如果在 ListBox 控件中使用 FindString 方法，没有找到指定数据项，则方法返回 ListBox.NoMatches 整型常量数值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FindThis。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示多个数据项信息；添加一个 ComboBox 控件，用于选择查询信息；添加一个 Button 控件，用于查找 ListBox 控件中的指定数据项。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int intIndex = this.listBox1.FindString(cbox_Select.Text);                    //查找指定项的索引
    if (intIndex != ListBox.NoMatches)
    {
        listBox1.SelectedIndex = intIndex;                                        //选中指定项
        MessageBox.Show("指定项的索引是：" + intIndex.ToString(),                  //弹出消息对话框
            "提示!", MessageBoxButtons.OK,MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("没有找到相关的选项",                                      //弹出消息对话框
            "信息提示", MessageBoxButtons.OK,MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 245：设置 ListBox 控件中数据项文本的颜色。

使用 ListBox 控件的 ForeColor 属性，可以方便地设置 ListBox 控件中文本的颜色。

实例 246　将数据库数据添加到组合框中

光盘位置：光盘\MR\11\246　　中级　　趣味指数：★★★☆

实例说明

数据库中存储着大量数据，可以方便地读取数据库中的数据，并对数据进行相应的处理。本实例中将会演示怎样读取 Access 数据库中的数据，并将读取到的数据添加到 ListBox 数据项中。实例运行效果如图 11.31 所示。

图 11.31　将数据库数据添加到组合框中

关键技术

本实例主要用到了 ListBox 控件中 Items 集合的 Add 方法，下面对其进行详细讲解。

ListBox 控件的 Add 方法用于向 ListBox 的项列表添加数据项。语法格式如下：

```
public int Add(Object item)
```

参数说明

❶ item：object 对象，表示要添加到集合中的数据项。

❷ 返回值：返回整型数值，表示添加到集合中的数据项在集合中的索引。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddToListBox。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示数据库中的数据信息；添加一个 Button 控件，用于查询数据库信息并将数据信息放入 ListBox 控件。

（3）程序的主要代码如下：

```
private void btn_Add_Click(object sender, EventArgs e)
{
    try
    {
        string P_Connection = string.Format(                                    //创建数据库连接字符串
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
        OleDbConnection P_OLEDBConnection =                                     //创建连接对象
            new OleDbConnection(P_Connection);
        P_OLEDBConnection.Open();                                               //连接到数据库
        OleDbCommand P_OLEDBCommand = new OleDbCommand(                         //创建命令对象
            "select * from [message]",P_OLEDBConnection);
        OleDbDataReader P_Reader = P_OLEDBCommand.ExecuteReader();              //得到数据读取器
        while (P_Reader.Read())                                                 //读取数据
        {
            lb_Str.Items.Add(P_Reader[0]);                                      //将数据放入集合
        }
        MessageBox.Show("成功读取数据！", "提示！");                              //弹出消息对话框
    }
    catch (Exception ex)
    {
        MessageBox.Show("数据读取失败！\r\n"+ex.Message,"错误！");                //弹出消息对话框
    }
}
```

秘笈心法

心法领悟 246：不要轻易使用 goto 语句。

从本实例中读者可以看到，使用 goto 语句可以很方便地跳转到定义标签所在的位置，但是在这里建议大家，不要过多使用 goto 语句，由于 goto 语句过于灵活，或多或少地会干扰程序的执行逻辑。

实例 247　在 ListBox 控件间交换数据

光盘位置：光盘\MR\11\247　　中级　　趣味指数：★★★☆

实例说明

通过 ListBox 控件的 SelectedItems 属性可以方便地得到控件中选中数据项的集合，然后，可以使用 for 或 foreach 遍历此集合中的每一个数据项，得到数据项中的数据信息。也可以使用控件的 SelectedItem 属性得到选中的单个数据项，并得到数据项中的信息。本实例中将会演示在两个 ListBox 控件间，交换选中的数据项。实例运行效果如图 11.32 所示。

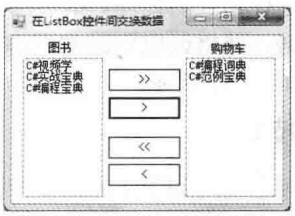


图 11.32　在 ListBox 控件间交换数据

关键技术

本实例主要用到了 ListBox 控件的 SelectedItem 属性和 Items 集合的 Clear 方法，下面分别对它们进行详细讲解。

（1）SelectedItem 属性

ListBox 控件的 SelectedItem 属性，用来获取或设置 ListBox 控件中当前选定的数据项。语法格式如下：

```
public Object SelectedItem { get; set; }
```

参数说明

属性值：object 对象，表示 ListBox 控件中被选中的数据项。

（2）Items 集合的 Clear 方法

ListBox 控件中 Items 集合的 Clear 方法，用于从数据项集合中移除所有的数据项。语法格式如下：

```
public virtual void Clear()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExchangeItem。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 ListBox 控件，用于演示在两个 ListBox 控件间移动数据；添加 4 个 Button 控件，用于移动两个 ListBox 控件中的数据。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)                    //全部添加到选择的项中
{
    for (int i = 0; i < lb_Source.Items.Count; i++)
    {
        lb_Source.SelectedIndex = i;                                       //按索引选中项
        lb_Choose.Items.Add(   lb_Source.SelectedItem.ToString());         //添加新项
    }
    lb_Source.Items.Clear();                                               //清空项
}
private void button3_Click(object sender, EventArgs e)                    //全部添加到数据源中
{
    for (int i = 0; i < lb_Choose.Items.Count; i++)
    {
        lb_Choose.SelectedIndex = i;                                       //按索引选中项
        lb_Source.Items.Add(lb_Choose.SelectedItem.ToString());            //添加项
```

```
    }
    lb_Choose.Items.Clear();                                    //清空项
}
```

秘笈心法

心法领悟 247：使用 for 循环遍历 ListBox 控件的数据项集合。

ListBox 控件的 Items 集合实现了 Ilist 接口，所以，可以使用索引器的方式访问 Items 集合中的每一个数据项，还可以使用 for 或 foreach 方便地遍历 Items 集合中的每一个元素。

实例 248　借助绑定控件实现数据选择录入

光盘位置：光盘\MR\11\248　　中级　　趣味指数：★★★★☆

实例说明

ListBox 控件的数据项集合中存放的是 object 类型的对象，这意味着所有对象都可以放入 ListBox 数据项集合中，当 ListBox 数据项显示对象信息时，将会调用对象的 ToString 方法，得到 ListBox 控件中数据项的显示信息。本实例中将会利用 ListBox 控件的 SelectedIndexChanged 事件获取被选中数据项的字符串信息，然后将此字符串信息赋值给 TextBox 控件并显示。实例运行效果如图 11.33 所示。

图 11.33　借助绑定控件实现数据选择录入

关键技术

本实例主要用到了 TextBox 控件的 Text 属性和 ListBox 控件的 SelectItem 属性，下面分别对它们进行详细讲解。

（1）TextBox 控件的 Text 属性

TextBox 控件的 Text 属性，用来获取或设置 TextBox 控件中的文本内容。语法格式如下：

```
public virtual string Text { get; set; }
```

参数说明

属性值：字符串对象，TextBox 控件中显示的字符串。

（2）ListBox 控件的 SelectedItem 属性

ListBox 控件的 SelectedItem 属性，用来获取或设置 ListBox 控件中当前选定的数据项。语法格式如下：

```
public Object SelectedItem { get; set; }
```

参数说明

属性值：object 对象，表示 ListBox 控件中被选中的数据项。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectInsert。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示图书信息；添加一个 TextBox 控件，用于显示购买图书的信息；添加两个 Button 控件，用于购买和取消购买图书。

（3）程序的主要代码如下：

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lb_str.SelectedItem.ToString() != null)
    {
        txt_Name.Text = lb_str.SelectedItem.ToString();          //得到 ListBox 控件中选择项的字符串
    }
}
```

秘笈心法

心法领悟 248：设置 ListBox 的背景颜色。

使用 ListBox 控件的 BackColor 属性可以方便地设置 ListBox 控件的背景颜色。

11.6 ListView 控件应用

实例 249	ListView 控件间的数据移动 光盘位置：光盘\MR\11\249	中级 趣味指数：★★★☆

实例说明

程序设计过程中，经常会用到 ListView 控件，ListView 控件中可以显示多条数据，可以使用 ListView 控件中 Items 集合的 Add 方法向控件中添加数据信息。本实例中将会演示怎样在两个 ListView 控件间移动数据。实例运行效果如图 11.34 所示。

图 11.34 ListView 控件间的数据移动

关键技术

本实例主要用到了 ListView 控件中 Items 集合的 Add 方法和 Remove 方法，下面分别对它们进行详细讲解。

（1）Items 集合的 Add 方法

ListView 控件中 Items 集合的 Add 方法，用于将数据项添加到 ListView 数据项集合中。语法格式如下：

```
public virtual ListViewItem Add(string text)
```

参数说明

❶ text：字符串对象，表示在 ListView 控件中数据项的文本。

❷ 返回值：返回已经添加到 ListView 控件的 ListViewItem 数据项的引用。

（2）Items 集合的 Remove 方法

ListView 控件中 Items 集合的 Remove 方法，从 ListView 控件的数据集合中移除指定的数据项。语法格式如下：

```
public virtual void Remove(ListViewItem item)
```

item：ListViewItem 数据项对象，表示要从 ListView 控件的数据集合中移除的数据项。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MoveData。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加两个 ListView 控件，用于演示在两个 ListView 控件间移动数据；添加 4 个 Button 控件，用于移动两个 ListView 控件中的数据。

（3）程序的主要代码如下：

```
private void listView1_SelectedIndexChanged(object sender,EventArgs e)
{
    if(listView1.SelectedItems.Count == 0)                    //当 listView1 中的选择项为 0 时
    {
        allRight.Enabled = false;                             //设置团购按钮为不可用状态
        allLeft.Enabled = false;                              //设置团退按钮为不可用状态
        right.Enabled = false;                                //设置单购按钮为不可用状态
        left.Enabled = false;                                 //设置单退按钮为不可用状态
    }
    else if(listView1.SelectedItems.Count == 1)               //当 listView1 中的选择项为 1 时
    {
```

```
            allRight.Enabled = false;                                    //设置团购按钮为不可用状态
            allLeft.Enabled = false;                                     //设置团退按钮为不可用状态
            right.Enabled = true;                                        //设置单购按钮为可用状态
            left.Enabled = false;                                        //设置单退按钮为不可用状态
        }
        else if(listView1.SelectedItems.Count > 1)                       //listView1 中的选择项大于 1 时
        {
            right.Enabled = false;                                       //设置单购按钮为不可用状态
            left.Enabled = false;                                        //设置单退按钮为不可用状态
            allLeft.Enabled = false;                                     //设置团退按钮为不可用状态
            allRight.Enabled = true;                                     //设置团购按钮为可用状态
        }
    }
    private void listView2_SelectedIndexChanged(object sender,EventArgs e)
    {
        if(listView2.SelectedIndices.Count == 0)                         //listView2 中的选择项为 0 时
        {
            allRight.Enabled = false;                                    //设置团购按钮为不可用状态
            allLeft.Enabled = false;                                     //设置团退按钮为不可用状态
            right.Enabled = false;                                       //设置单购按钮为不可用状态
            left.Enabled = false;                                        //设置单退按钮为不可用状态
        }
        else if(listView2.SelectedItems.Count == 1)                      //当 listView2 中的选择项为 1 时
        {
            allRight.Enabled = false;                                    //设置团购按钮为不可用状态
            allLeft.Enabled = false;                                     //设置团退按钮为不可用状态
            right.Enabled = false;                                       //设置单购按钮为不可用状态
            left.Enabled = true;                                         //设置单退按钮为可用状态
        }
        else if(listView2.SelectedItems.Count > 1)                       //当 listView2 中的选择项大于 1 时
        {
            right.Enabled = false;                                       //设置单购按钮为不可用状态
            left.Enabled = false;                                        //设置单退按钮为不可用状态
            allLeft.Enabled = true;                                      //设置团退按钮为可用状态
            allRight.Enabled = false;                                    //设置团购按钮为不可用状态
        }
    }
```

秘笈心法

心法领悟 249：停用 Button 按钮。

在窗体应用程序开发过程中，可以通过设置按钮的 Enable 属性来启用和停用按钮，当 Button 按钮的 Enable 属性为 true 时，启用按钮；当 Button 按钮的 Enable 属性为 false 时，停用按钮。

实例 250　将数据库数据添加到 ListView 控件

光盘位置：光盘\MR\11\250　　中级　　趣味指数：★★★☆

实例说明

数据库中存储着大量数据，可以方便地读取数据库中的数据，并对数据进行相应的处理。本实例中将会演示怎样读取数据库中的数据信息，并将数据信息添加到 ListView 控件的数据集合中。实例运行效果如图 11.35 所示。

图 11.35　将数据库数据添加到 ListView 控件

关键技术

本实例主要用到了 ListView 控件中 Items 集合的 Clear 方法，下面对其进行详细讲解。

Items 集合的 Clear 方法用于从 ListView 控件的数据项集合中移除所有的数据项。语法格式如下：

```
public virtual void Clear()
```

说明：当使用 Clear 方法移除 ListView 控件中所有数据项时，会丢失控件中所有的数据项。如果要从集合中移除单个数据项，可以使用 Remove 或 RemoveAt 方法方便地实现。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InsertToListView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListView 控件，用于显示数据库中的数据信息；添加两个 Button 控件，用于添加和删除 ListView 控件中的数据。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        string P_Connection = string.Format(                                    //创建数据库连接字符串
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
        OleDbConnection P_OLEDBConnection =                                     //创建连接对象
            new OleDbConnection(P_Connection);
        P_OLEDBConnection.Open();                                               //连接到数据库
        OleDbCommand P_OLEDBCommand = new OleDbCommand(                         //创建命令对象
            "select * from [book]",P_OLEDBConnection);
        OleDbDataReader P_Reader = P_OLEDBCommand.ExecuteReader();              //得到数据读取器
        while (P_Reader.Read())                                                 //读取数据
        {
            ListViewItem lv = new ListViewItem(P_Reader[0].ToString());
            lv.SubItems.Add(P_Reader[1].ToString());
            lv.SubItems.Add(P_Reader[2].ToString());
            listView1.Items.Add(lv);
        }
        P_OLEDBConnection.Close();                                              //关闭数据库连接
    }
    catch (Exception ex)
    {
        MessageBox.Show(                                                        //弹出消息对话框
            "数据读取失败！\r\n" + ex.Message, "错误！");
    }
}
```

秘笈心法

心法领悟 250：设置 ListView 控件的背景颜色。

使用 ListView 控件的 BackColor 属性，可以方便地设置 ListView 控件的背景颜色。

实例 251 在 ListView 控件中实现修改功能

光盘位置：光盘\MR\11\251　　中级　　趣味指数：★★★☆

实例说明

程序设计过程中，可以通过设置 ListView 控件的 LabelEdit 属性为 true，从而允许用户手动更改 ListView 控件中数据项的标签。本实例中将会演示怎样读取数据库的内容到 ListView 控件，并更改 ListView 控件中数据项的标签，然后将 ListView 控件中数据的更改同步到数据库中。实例运行效果如图 11.36 所示。

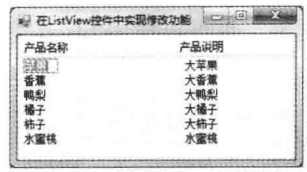


图 11.36　在 ListView 控件中实现修改功能

关键技术

本实例主要用到了 ListView 控件的 LabelEdit 属性和 Columns 集合的 Add 方法，下面分别对它们进行详细讲解。

（1）LabelEdit 属性

ListView 控件的 LabelEdit 属性，用来指示用户是否可以编辑控件中数据项的标签。语法格式如下：

```
public bool LabelEdit { get; set; }
```

参数说明

属性值：布尔值，如果用户可以在运行时编辑数据项的标签，则属性值为 true，否则值为 false。默认值为 false。

（2）Columns 集合的 Add 方法

ListView 控件中 Columns 集合的 Add 方法，用于将列标题添加至数据列的集合。语法格式如下：

```
public virtual ColumnHeader Add(string text,int width,HorizontalAlignment textAlign)
```

参数说明

❶ text：字符串对象，表示将要在列标题中显示的文本。

❷ width：整数数值，表示列标题的初始宽度。

❸ textAlign：HorizontalAlignment 枚举值，用于设置对齐方式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ModificationTxt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在 ListView 控件中实现修改功能”，向窗体中添加一个 ListView 控件，用于显示和修改 ListView 控件与数据库中的数据。

（3）程序的主要代码如下：

```
private void listView1_AfterLabelEdit(object sender,LabelEditEventArgs e)
{
    WidgetConnection = new OleDbConnection(ConnectString);                    //初始化一个数据库连接
    if(WidgetConnection.State == ConnectionState.Closed)                      //当数据库连接处于关闭状态时
    {
        WidgetConnection.Open();                                              //打开数据库连接
    }
    if(e.Label != null && e.Label != "")                                      //当选定项的文本内容存在且不为空时
    {
        string RefreshString = "update tb_WidgetApply set 产品名称='"        //定义更新数据库字符串
            + e.Label + "' where 产品编号=" + (e.Item+1).ToString();
        OleDbCommand WidgetCommand = new OleDbCommand(                        //声明一个执行 SQL 语句的对象
            RefreshString,WidgetConnection);
        WidgetCommand.ExecuteNonQuery();                                      //执行 SQL 语句
        WidgetConnection.Close();                                             //关闭数据库连接
        MessageBox.Show("数据修改成功！","提示信息",                          //弹出信息提示
            MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
    }
}
```

秘笈心法

心法领悟 251：向 ListView 控件中插入数据项。

ListView 控件的 Items 集合实现了 Ilist 接口，可以使用 Add 方法向数据项集合中添加数据项，使用 Remove

方法删除数据项集合中指定的数据项。同样地，也可以使用 Insert 方法方便地向数据项集合中插入数据项。

实例 252 在 ListView 控件中对数据排序

光盘位置：光盘\MR\11\252　　中级　　趣味指数：★★★☆

实例说明

ListView 控件的 Items 集合中保存着大量的 ListViewItem 对象，而每一个 ListViewItem 对象都包含着多个数据项。本实例中将会演示怎样将数据库中的数据保存到 ListView 控件中，并根据数据库的查询字符串巧妙地实现 ListView 控件对数据排序的功能。实例运行效果如图 11.37 所示。

图 11.37　在 ListView 控件中对数据排序

关键技术

本实例主要用到了 ListViewItem 数据项中 SubItems 集合的 Add 方法，下面对其进行详细讲解。

ListViewItem 数据项中 SubItems 集合的 Add 方法用于将子数据项添加到子数据项的集合中。语法格式如下：

```
public ListViewItem.ListViewSubItem Add(string text)
```

参数说明

❶ text：字符串对象，子数据项显示的文本。

❷ 返回值：返回已经添加到子数据项集合中的子数据项对象的引用。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SortOrStatistics。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListView 控件，用于显示数据库中的数据信息；添加两个 Button 控件，用于排序 ListView 控件中的数据信息。

（3）程序的主要代码如下：

```
public void getScoure(string strName)
{
    try
    {
        string P_Connection = string.Format(                                    //创建数据库连接字符串
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
        OleDbConnection P_OLEDBConnection =                                     //创建连接对象
            new OleDbConnection(P_Connection);
        P_OLEDBConnection.Open();                                               //连接到数据库
        OleDbCommand P_OLEDBCommand = new OleDbCommand(                         //创建命令对象
            strName,P_OLEDBConnection);
        OleDbDataReader P_Reader = P_OLEDBCommand.ExecuteReader();              //得到数据读取器
        listView1.View = View.Details;                                          //设置控件显示方式
        listView1.GridLines = true;                                             //显示网格线
        listView1.FullRowSelect = true;                                         //是否连带选中子项
        listView1.Items.Clear();                                                //清空元素
        while (P_Reader.Read())                                                 //读取数据
        {
            ListViewItem lv =  new ListViewItem(P_Reader[0].ToString());        //创建项
            lv.SubItems.Add(P_Reader[1].ToString());                            //创建项
            lv.SubItems.Add(P_Reader[2].ToString());                            //创建项
            listView1.Items.Add(lv);                                            //向 ListView 控件中添加项
        }
        P_OLEDBConnection.Close();                                              //关闭连接
    }
    catch (Exception ex)
```

```
    {
        MessageBox.Show("数据读取失败！\r\n" + ex.Message, "错误！");                    //弹出消息对话框
    }
}
```

秘笈心法

心法领悟 252：遍历数据项集合。

由于 ListView 控件中的 Items 集合实现了 Ilist 接口，所以可以使用索引器的方式访问 Items 集合中的数据项。同样地，也可以使用 for 或 foreach 方便地遍历 ListView 控件中的数据项集合。

实例 253	在 ListView 控件中绘制底纹	中级
	光盘位置：光盘\MR\11\253	趣味指数：★★★☆

实例说明

使用 ListView 控件时，默认其背景颜色为白色，如果在其中显示图标，背景就会显得有些不和谐，这时，如果把 ListView 控件的背景设置为一个底纹图片，界面就会变得协调统一。本实例将会演示怎样为 ListView 控件绘制底纹。实例运行效果如图 11.38 所示。

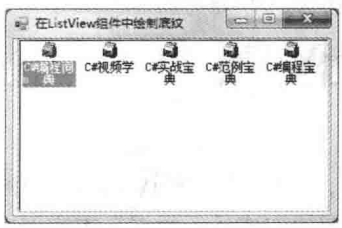


图 11.38　在 ListView 控件中绘制底纹

关键技术

本实例主要用到了 ListView 控件的 BackgroundImage 属性，下面对其进行详细讲解。

ListView 控件的 BackgroundImage 属性，用于获取或设置在控件中显示的背景图像。语法格式如下：

```
public virtual Image BackgroundImage { get; set; }
```

参数说明

属性值：Image 对象，表示在 ListView 控件的背景中显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawBackGround。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在 ListView 组件中绘制底纹”，向窗体中添加一个 ListView 控件，用于显示带有底纹的数据信息。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    try
    {
        string P_Connection = string.Format(                                        //创建数据库连接字符串
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
        OleDbConnection P_OLEDBConnection =                                         //创建连接对象
            new OleDbConnection(P_Connection);
        P_OLEDBConnection.Open();                                                   //连接到数据库
        OleDbCommand P_OLEDBCommand = new OleDbCommand(                             //创建命令对象
            "select * from [message]",P_OLEDBConnection);
        OleDbDataReader P_Reader = P_OLEDBCommand.ExecuteReader();                  //得到数据读取器
```

```
            listView1.BackgroundImageTiled = true;                          //设置平铺背景图像
            listView1.View = View.LargeIcon;                                //设置控件的显示方式
            listView1.LargeImageList = imageList1;
            while (P_Reader.Read())                                         //读取数据
            {
                ListViewItem lv = new ListViewItem(P_Reader[0].ToString(), 0);   //创建项
                listView1.Items.Add(lv);                                    //向控件中添加项
            }
            P_OLEDBConnection.Close();                                      //关闭数据库连接
        }
        catch (Exception ex)
        {
            MessageBox.Show("数据读取失败！\r\n" + ex.Message, "错误！");          //弹出消息对话框
        }
    }
```

秘笈心法

心法领悟 253：设置数据项文本颜色。

ListView 控件的 Items 集合中包含了多个数据项，每一个数据项都是一个 ListViewItem 对象，通过设置 ListViewItem 对象的 ForeColor 属性，可以方便地设置数据项文本的颜色。

实例 254　在列表视图中拖动视图项

光盘位置：光盘\MR\11\254　　中级　　趣味指数：★★★☆

实例说明

通过设置 ListView 控件的 View 属性，可以更改 ListView 控件中数据项的显示方式。如果将 View 属性设置为 LargeIcon，则 ListView 控件会以视图的方式显示数据。本实例中将会讲解如何在 ListView 控件中拖动视图项、移动视图项的位置。实例运行效果如图 11.39 所示。

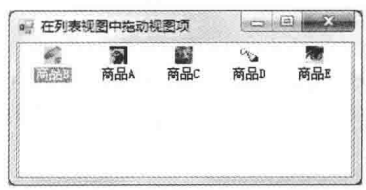


图 11.39　在列表视图中拖动视图项

关键技术

本实例主要用到了 ListView 控件的 View 和 AllowDrop 属性，下面分别对它们进行详细讲解。

（1）View 属性

ListView 控件的 View 属性，用来获取或设置数据项在控件中的显示方式。语法格式如下：

```
public View View { get; set; }
```

参数说明

属性值：View 枚举值，默认值为 LargeIcon。

（2）AllowDrop 属性

ListView 控件的 AllowDrop 属性，用来指示 ListView 控件是否可以接收用户拖放到它上面的数据。语法格式如下：

```
public virtual bool AllowDrop { get; set; }
```

参数说明

属性值：布尔值，如果控件允许拖放操作，则值为 true，否则值为 false。默认值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DragViewTerm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在列表视图中拖动视图项”，向窗体中添加一个 ListView 控件，用于显示可拖动的数据信息。

（3）程序的主要代码如下：

```
private void listView1_DragDrop(object sender, DragEventArgs e)
{
    if (listView1.SelectedItems.Count == 0)                          //判断是否选择拖放的项
    {
        return;                                                      //退出方法
    }
    Point cp = listView1.PointToClient(new Point(e.X, e.Y));         //定义项的坐标点
    ListViewItem dragToItem = listView1.GetItemAt(cp.X, cp.Y);       //得到指定位置的项
    if (dragToItem == null)                                          //判断是否为空
    {
        return;                                                      //退出方法
    }
    List<ListViewItem> ls = new List<ListViewItem>();                //创建项集合
    foreach (ListViewItem lvi in listView1.SelectedItems)            //遍历选中的项
    {
        ls.Add(lvi);                                                 //将选中项添加到集合
    }
    for (int i = 0; i < ls.Count; i++)
    {
        listView1.Items.Remove(ls[i]);                               //移除数据项
    }
    for (int i = 0; i < ls.Count; i++)
    {
        listView1.Items.Insert(dragToItem.Index, ls[i]);             //添加数据项
    }
    ls.Clear();                                                      //清空数据集合
    for (int i = 0; i < listView1.Items.Count; i++)
    {
        ls.Add(listView1.Items[i]);                                  //向数据集合添加数据
    }
    listView1.Items.Clear();                                         //清空 ListView 数据项
    for (int i = 0; i < ls.Count; i++)
    {
        listView1.Items.Add(ls[i]);                                  //向 ListView 中添加数据项
    }
}
```

秘笈心法

心法领悟 254：设置 ListView 控件中数据项的字体。

使用 ListView 控件的 Font 属性，可以轻松地设置 ListView 控件中所有数据项中文本内容的字体，当 ListView 控件的 Font 属性设置为大文字后，ListView 控件会自动调整数据项的单元格，使数据项中的文本内容可以正常显示。

实例 255　使 ListView 控件中的选择项高亮显示

光盘位置：光盘\MR\11\255　　中级　　趣味指数：★★★★☆

实例说明

ListView 控件的 Items 集合中包含大量的 ListViewItem 对象，每一个 ListViewItem 对象都是控件中的一个记录项，通过设置 ListViewItem 对象的 BackColor 方法，可以方便地设置每一个记录项的背景颜色。本实例中将会演示怎样高亮显示 ListView 控件中的选择项。实例运行效果如图 11.40 所示。

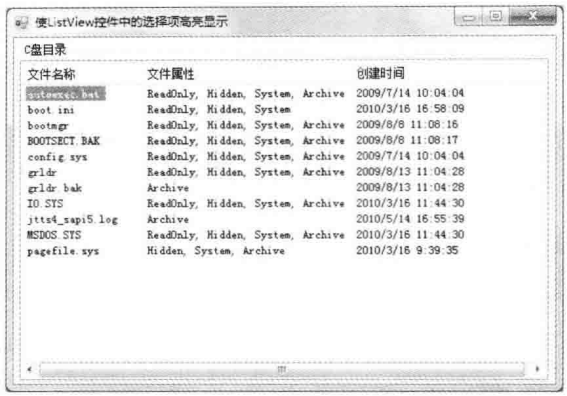


图 11.40　使 ListView 控件中的选择项高亮显示

关键技术

本实例主要用到了 ListView 控件的 SelectedItems 属性和 ListViewItem 数据项的 BackColor 属性，下面分别对它们进行详细讲解。

（1）SelectedItems 属性

ListView 控件的 SelectedItems 属性，用来获取在 ListView 控件中被选中数据项的集合。语法格式如下：

```
public ListView.SelectedListViewItemCollection SelectedItems { get; }
```

参数说明

属性值：SelectedListViewItemCollection 被选中的数据项的集合，如果当前没有选中任何数据项，则返回空。

说明：SelectedListViewItemCollection 集合实现了 Ilist 接口，所以可以使用 for 或 foreach 方便地遍历集合。

（2）BackColor 属性

ListViewItem 数据项的 BackColor 属性，用来获取或设置该数据项的背景色。语法格式如下：

```
public Color BackColor { get; set; }
```

参数说明

属性值：Color 结构，表示数据项的背景色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectFlash。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“使 ListView 控件中的选择项高亮显示”，向窗体中添加一个 ListView 控件，用于演示高亮显示数据项。

（3）程序的主要代码如下：

```
private void SelectFlash_Load(object sender, EventArgs e)
{
    listView1.GridLines = true;                                          //是否在控件中显示网格线
    listView1.Dock = DockStyle.Fill;                                     //设置在其父容器中的停靠方式
    listView1.Columns.Add("文件名称", 120, HorizontalAlignment.Left);      //添加“文件名称”列
    listView1.Columns.Add("文件属性", 275, HorizontalAlignment.Left);      //添加“文件属性”列
    listView1.Columns.Add("创建时间", 261, HorizontalAlignment.Left);      //添加“创建时间”列
    foreach (String fileName in Directory.GetFiles("C:\\"))              //循环遍历 C 盘目录空间
    {
        FileInfo file = new FileInfo(fileName);                          //声明一个操作文件的实例
        ListViewItem OptionItem = new ListViewItem(file.Name);           //创建选择项对象
        OptionItem.SubItems.Add(file.Attributes.ToString());             //添加文件属性列
        OptionItem.SubItems.Add(file.CreationTime.ToString());           //创建时间列
        listView1.Items.Add(OptionItem);                                 //追加新添加的各列
    }
    listView1.HideSelection = true;                                      //设置控件的高亮显示
}
```

秘笈心法

心法领悟 255：遍历 SelectedItems 集合。

ListView 控件的 SelectedItems 属性，用来获取在 ListView 控件中被选中数据项的集合，由于 SelectedItems 集合实现了 Ilist 接口，所以可以方便地使用 for 和 foreach 遍历集合中的每一个被选中的数据项。

实例 256　制作带复选框的 ListView 控件

光盘位置：光盘\MR\11\256　　中级　　趣味指数：★★★★☆

实例说明

ListView 控件中可以包含大量的数据项，如何从众多的数据项中挑选出用户需要的和不需要的数据，这直接决定该用户的工作效率。通过对本实例的学习，用户可以选定自己不需要的数据进行标记，也可以选定自己需要的进行标记，以达到对选定数据的区分。实例运行效果如图 11.41 所示。

图 11.41　制作带复选框的 ListView 控件

关键技术

本实例主要用到了 ListView 控件的 GridLines 和 CheckBoxes 属性，以及 ListViewItem 数据项的 Checked 属性，下面分别对它们进行详细讲解。

（1）GridLines 属性

ListView 控件的 GridLines 属性，用来设置是否在 ListView 控件中显示网格线。语法格式如下：

```
public bool GridLines { get; set; }
```

参数说明

属性值：布尔值，如果显示网格线值为 true，不显示网格线则值为 false。默认值为 false。

（2）CheckBoxes 属性

ListView 控件的 CheckBoxes 属性，用来设置 ListView 控件中各数据项是否显示复选框。语法格式如下：

```
public bool CheckBoxes { get; set; }
```

参数说明

属性值：布尔值，如果在 ListView 控件的数据项中显示复选框则值为 true，不显示复选框则值为 false。默认值为 false。

（3）Checked 属性

ListViewItem 数据项的 Checked 属性，用来判断是否选中此数据项中的复选框。语法格式如下：

```
public bool Checked { get; set; }
```

参数说明

属性值：布尔值，如果选中数据项中的复选框则值为 true，否则值为 false。默认值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CheckBoxInListView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示带有复选框的文件信息；添加两个 Button 控件，分别用于全选、取消选择 ListView 中每一个数据项前的复选框。

（3）程序的主要代码如下：

```
private void CheckBoxInListView_Load(object sender,EventArgs e)
{
    listView1.CheckBoxes = true;                                    //设置 listView1 的复选框属性为真
```

```
    listView1.View = View.Details;                                    //设置 listView1 的视图方式
    listView1.GridLines = true;                                       //设置 listView1 显示网格线
    listView1.Columns.Add("文件名称",150,HorizontalAlignment.Left);    //向 listView1 中添加“文件名称”列
    listView1.Columns.Add("创建时间",261,HorizontalAlignment.Left);    //向 listView1 中添加“创建时间”列
    foreach(String fileName in Directory.GetFiles("C:\\"))            //循环遍历 C 盘的内容
    {
        FileInfo file = new FileInfo(fileName);                       //定义一个操作文件的实例
        ListViewItem OptionItem = new ListViewItem(file.Name);        //定义一个 listView 选择项的实例
        OptionItem.SubItems.Add(file.CreationTime.ToString());        //添加文件创建时间列
        listView1.Items.Add(OptionItem);                              //执行添加操作
    }
}
```

秘笈心法

心法领悟 256：获取 ListView 控件中数据项的数量。

由于 ListView 控件的 Items 集合实现了 Ilist 接口，所以可以使用 Items 集合的 Count 属性方便地得到集合中数据项的数量。

11.7 选择类控件应用

实例 257 利用选择控件实现权限设置　　中级

光盘位置：光盘\MR\11\257　　趣味指数：★★★☆

实例说明

注册用户时，应当分配给用户一些相应的权限，这样能更好地管理用户数据，防止非法用户或没有相关权限的用户登录系统查看或修改相关数据。实例运行过程中，可以根据用户的职责，选中相应模块前的复选框，如果取消选中相应模块前的复选框，则取消该用户操作模块的权限。实例运行效果如图 11.42 所示。

图 11.42　利用选择控件实现权限设置

关键技术

本实例主要用到了 CheckedListBox 控件的 Items 属性，下面对其进行详细讲解。

CheckedListBox 控件的 Items 属性，用来获取 CheckedListBox 控件中数据项的集合。语法格式如下：

```
public CheckedListBox.ObjectCollection Items { get; }
```

参数说明

属性值：ObjectCollection 数据项集合。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Selected。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示数组中的图书信息；添加 5 个 TextBox 控件，用于输入用户信息；添加 4 个 CheckedListBox 控件，用于选择用户操作权限。

（3）程序的主要代码如下：

```
private void ckShop_CheckedChanged(object sender, EventArgs e)
{
```

```
    if (ckShop.Checked == true)                                    //判断是否选中进货管理
    {
        cklShop.Visible = true;                                    //显示进货管理信息
        CheckAll(cklShop);                                         //选中所有进货管理
    }
    else
    {
        cklShop.Visible = false;                                   //隐藏进货管理信息
        CheckAllEsce(cklShop);                                     //取消选中所有进货管理
    }
}
private void ckSell_CheckedChanged(object sender, EventArgs e)
{
    if (ckSell.Checked == true)                                    //判断是否选中销售管理
    {
        cklSell.Visible = true;                                    //显示销售管理信息
        CheckAll(cklSell);                                         //选中所有销售管理
    }
    else
    {
        cklSell.Visible = false;                                   //隐藏销售管理信息
        CheckAllEsce(cklSell);                                     //取消选中所有销售管理
    }
}
private void ckMange_CheckedChanged(object sender, EventArgs e)
{
    if (ckMange.Checked == true)                                   //判断是否选中库存管理
    {
        cklMange.Visible = true;                                   //显示库存管理
        CheckAll(cklMange);                                        //选中所有库存管理
    }
    else
    {
        cklMange.Visible = false;                                  //隐藏库存管理
        CheckAllEsce(cklMange);                                    //取消选中所有库存管理
    }
}
```

秘笈心法

心法领悟 257：显示和隐藏 CheckedListBox 控件。

通过设置 CheckedListBox 控件的 Visible 属性，可以方便地显示和隐藏 CheckedListBox 控件，Visible 属性为布尔值，当属性为 true 时，显示控件；当属性为 false 时，隐藏控件。

实例 258　利用选择控件实现复杂查询

光盘位置：光盘\MR\11\258　　中级　　趣味指数：★★★★☆

实例说明

CheckBox 控件可以设置和判断某种条件是真还是假，通常用于为用户提供 true 或 false 选项。如果 CheckBox 控件被选中，则控件的 Checked 属性为 true；如果未被选中，则控件的 Checked 属性为 false。实例中实现了对学生信息的查询，并根据相应的 CheckBox 控件的选择状态判断查询内容。实例运行效果如图 11.43 所示。

图 11.43　利用选择控件实现复杂查询

关键技术

本实例主要用到了 CheckBox 控件的 Checked 属性，下面对其进行详细讲解。

CheckBox 控件的 Checked 属性，用来指示 CheckBox 控件是否处于选中状态。语法格式如下：

```
public bool Checked { get; set; }
```

参数说明

属性值：布尔值，如果 CheckBox 控件处于选定状态，值为 true，否则值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CheckedListBoxForSelect。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 3 个 TextBox 控件，用于输入用户查询信息；添加一个 DataGridView 控件，用于显示数据库中的数据信息；添加两个 Button 控件，用于搜索和取消搜索数据库中的数据。

（3）程序的主要代码如下：

```
private void bntSearch_Click(object sender, EventArgs e)
{
    intCount = 0;
    if (txtstuId.Text != "")                                    //判断输入编号是否为空
    {
        strId = "学生编号   like '%" + txtstuId.Text + "%'"  ;   //定义 SQL 字符串
        strScoure[intCount]=strId;                              //向字符串集合添加 SQL 字符串
        intCount++;                                             //开始计数
    }
    if (txtName.Text != "")
    {
        strName = "学生姓名  like '%" + txtName.Text + "%' ";    //定义 SQL 字符串
        strScoure[intCount]= strName;                           //向字符串集合添加 SQL 字符串
        intCount++;                                             //开始计数
    }
    if (txtClass.Text != "")
    {
        strClass = "所在年级  like '%" + txtClass.Text + "%'";   //定义 SQL 字符串
        strScoure[intCount]= strClass;                          //向字符串集合添加 SQL 字符串
        intCount++;                                             //开始计数
    }
    if (rdbMan.Checked == true)
    {
        strSex = "学生性别  like '%" + rdbMan.Text + "%'";       //定义 SQL 字符串
        strScoure[intCount]= strSex;                            //向字符串集合添加 SQL 字符串
        intCount++;                                             //开始计数
    }
    if (rdbWoMan.Checked==true)
    {
        strSex = "学生性别  like '%" + rdbWoMan.Text + "%'";     //定义 SQL 字符串
        strScoure[intCount]= strSex;                            //向字符串集合添加 SQL 字符串
        intCount++;                                             //开始计数
    }
    for (int i = 0; i < strScoure.Length; i++)                  //遍历字符串集合
    {
        if (strScoure[i] != null)                               //判断字符是否为空
        {
            strSql += strScoure[i];                             //组合查询字符串
            intAdd++;                                           //开始计数
        }
    }
    switch (intAdd)                                             //使用多路选择语句组合查询语句
    {
        case 0:
            strSql = "select * from tb_Student";
            break;
        case 1:
            strSql = "select * from tb_Student where " + strScoure[0];
            break;
```

```
            case 2:
                strSql = "select * from tb_Student where " + strScoure[0] + " and " + strScoure[1];
                break;
            case 3:
                strSql = "select * from tb_Student where " + strScoure[0] +
                    " and " + strScoure[1] + " and " + strScoure[2];
                break;
            case 4:
                strSql = "select * from tb_Student where " + strScoure[0] +
                    " and " + strScoure[1] + " and " + strScoure[2] + " and " + strScoure[3];
                break;
        }
        GetScoure(strSql);                                  //查询数据库中的数据
        intAdd = 0;                                         //记数器置 0
        intCount = 0;                                       //记数器置 0
        strSql = "";                                        //重置 SQL 语句
        for (int i = 0; i < strScoure.Length; i++)
        {
            if (strScoure[i] != null)
            {
                strScoure[i] = null;                        //清空字符串数组内容
            }
        }
    }
```

秘笈心法

心法领悟 258：巧妙地使用 SQL 语句。

实例中根据用户的选择，动态生成相应的 SQL 语句，并根据此 SQL 语句查询数据库中的数据，从而实现复杂查询。

11.8　TreeView 控件应用

实例 259	在 TreeView 控件节点中显示图片	中级
	光盘位置：光盘\MR\11\259	趣味指数：★★★☆

实例说明

程序设计过程中，经常会用到 TreeView 控件，该控件提供了树形的数据显示，使用树形视图可以方便直观地显示数据。本实例中使用 TreeView 控件显示多条数据，并在每一个数据节点中添加图片信息。实例运行效果如图 11.44 所示。

图 11.44　在 TreeView 控件节点中显示图片

关键技术

本实例主要用到了 TreeView 控件中 Nodes 集合的 Add 方法以及 TreeView 控件的 ExpandAll 方法，下面分别对它们进行详细讲解。

（1）Add 方法

TreeView 控件中 Nodes 集合的 Add 方法，用于创建节点并将节点放入集合中。语法格式如下：

```
public virtual TreeNode Add(string key,string text,int imageIndex,int selectedImageIndex)
```

Add 方法中的参数说明如表 11.2 所示。

表 11.2 Add 方法中的参数说明

参　数	描　述
key	字符串对象，树节点的名称
text	字符串对象，在树节点中显示的文本
imageIndex	整型数值，将要在树节点中显示的图像的索引
selectedImageIndex	整型数值，处于选定状态时要在树节点中显示的图像的索引
TreeNode	返回已添加到集合中的树节点对象的引用

（2）ExpandAll 方法

TreeView 控件的 ExpandAll 方法，可以展开 TreeView 控件中的所有树节点。语法格式如下：

```
public void ExpandAll()
```

说明：调用 TreeView 控件的 CollapseAll 方法，可以折叠 TreeView 控件中的所有树节点。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PicturesInTreeView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 TreeView 控件，用于显示带有图片的节点信息；添加一个 Button 按钮，用于向 TreeView 控件中添加数据。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    treeView1.ImageList = imageList1;                                    //绑定 ImageList 控件
    TreeNode tr = new TreeNode("公司职员", 0, 1);
    tr.Nodes.Add("", "小张", 0, 1);
    tr.Nodes.Add("", "小王", 0, 1);
    tr.Nodes.Add("", "小李", 0, 1);
    tr.Nodes.Add("", "小刘", 0, 1);
    tr.Nodes.Add("", "小孙", 0, 1);
    tr.Nodes.Add("", "小赵", 0, 1);
    tr.Nodes.Add("", "小懂", 0, 1);
    tr.Nodes.Add("", "小高", 0, 1);
    treeView1.Nodes.Add(tr);                                             //添加节点
    treeView1.ExpandAll();                                               //展开所有节点
}
```

秘笈心法

心法领悟 259：设置 TreeView 控件的背景色。

使用 TreeView 控件的 BackColor 属性，可以方便地更改 TreeView 控件的背景颜色。

实例 260　实现带复选框的 TreeView 控件

光盘位置：光盘\MR\11\260　　中级　　趣味指数：★★★☆

实例说明

通过 TreeView 控件，可以方便地为用户显示节点层次结构，树视图中的节点中可以包含其他子节点，而子节点中也可以包含子节点。本实例中将会演示怎样创建带有复选框的 TreeView 控件。实例运行效果如图 11.45 所示。

图 11.45　实现带复选框的 TreeView 控件

关键技术

本实例主要用到了 TreeView 控件的 CheckBoxes 属性，下面对其进行详细讲解。

TreeView 控件的 CheckBoxes 属性，用来指示是否在树视图控件中的树节点中显示复选框。语法格式如下：

```
public bool CheckBoxes { get; set; }
```

参数说明

属性值：布尔值，如果在树视图控件中的树节点中显示复选框则值为 true，否则值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CheckBoxInTreeView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“实现带复选框的 TreeView 控件”，向窗体中添加一个 TreeView 控件，用于显示带有复选框的数据信息。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    treeView1.CheckBoxes = true;                                    //显示复选框
}
```

秘笈心法

心法领悟 260：设置节点的文本颜色。

TreeView 控件的 Nodes 集合中包含着多个子节点，而每一个子节点中也可以包含多个节点，通过节点对象的 ForeColor 属性可以方便地设置节点中文本的颜色。

实例 261　将数据库数据显示到树视图中

光盘位置：光盘\MR\11\261　　中级　　趣味指数：★★★☆

实例说明

程序设计过程中，经常使用 TreeView 控件显示多项数据，TreeView 控件显示数据的最大优点是层次清晰。例如，在 Windows 资源管理器中，可以通过 TreeView 控件显示目录层次。本实例将会从数据库中读取数据信息，并将数据信息放入 TreeView 控件中。实例运行效果如图 11.46 所示。

图 11.46　将数据库数据显示到树视图中

关键技术

本实例主要用到了 TreeView 控件中 Nodes 集合的 Add 方法，下面对其进行详细讲解。

说明：关于 Add 方法的详细讲解请参见实例 259 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DataBaseToTreeView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将数据库数据显示到树视图中”，向窗体中添加一个 TreeView 控件，用于显示数据库中的数据信息。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    treeView1.ShowLines = true;                                          //设置绘制连线
    treeView1.ImageList = imageList1;                                    //设置 ImageList 属性
    string P_Connection = string.Format(                                 //创建数据库连接字符串
     "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
    OleDbConnection P_OLEDBConnection = new OleDbConnection(P_Connection);  //创建连接对象
    P_OLEDBConnection.Open();                                            //连接到数据库
    OleDbCommand P_OLEDBCommand = new OleDbCommand(                      //创建命令对象
        "select * from [Ware]",P_OLEDBConnection);
    OleDbDataReader P_Reader = P_OLEDBCommand.ExecuteReader();           //得到数据读取器
    TreeNode newNode1 = treeView1.Nodes.Add("A","商品信息",1,2);          //一级节点
    while (P_Reader.Read())
    {
        TreeNode newNode12 = new TreeNode(                               //二级节点
            "商品编号" + P_Reader[1].ToString(), 3, 4);
        newNode12.Nodes.Add("A", "商品名称：" + P_Reader[0].ToString(), 5, 6);
        newNode12.Nodes.Add("A", "商品数量：" + P_Reader[3].ToString(), 7, 8);
        newNode12.Nodes.Add("A", "商品价格：" + P_Reader[2].ToString(), 9, 10);
        newNode1.Nodes.Add(newNode12);                                   //添加节点
    }
    P_OLEDBConnection.Close();                                           //关闭数据库连接
    treeView1.ExpandAll();                                               //展开所有节点
}
```

秘笈心法

心法领悟 261：设置节点背景颜色。

TreeView 控件的 Nodes 集合中包含着多个子节点，而每一个子节点中也可以包含多个节点，通过节点对象的 BackColor 属性可以方便地设置节点的背景颜色。

实例 262 用树形列表动态显示菜单

光盘位置：光盘\MR\11\262　　中级　　趣味指数：★★★☆

实例说明

程序设计过程中，窗体界面的设计是至关重要的，一个良好的窗体布局，可以增强应用程序的可操作性。例如，如何让用户更直观、更快速地了解本程序的相关功能及操作，如何在主窗体中显示当前用户的权限等。本实例中演示将菜单中的内容动态添加到树形列表中，并根据菜单中的用户权限，对树形列表中的相应项进行设置。实例运行效果如图 11.47 所示。

图 11.47　用树形列表动态显示菜单

关键技术

本实例主要用到了 TreeNode 对象的 ForeColor 属性，下面对其进行详细讲解。

TreeNode 对象的 ForeColor 属性，用来设置节点的文本颜色。语法格式如下：

```
public override Color ForeColor { get; set; }
```

参数说明

属性值：Color 结构，节点的文本颜色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于在窗体中显示菜单项；添加一个 TreeView 控件，用于显示菜单项的内容。

（3）程序的主要代码如下：

```
public void GetCavernMenu(TreeNode newNodeA, ToolStripDropDownItem newmenuA, bool BL)
{
    bool Var_Bool = true;                                                    //设置布尔值
    if (newmenuA.HasDropDownItems && newmenuA.DropDownItems.Count > 0)
        for (int j = 0; j < newmenuA.DropDownItems.Count; j++)               //遍历二级菜单项
        {
            //将二级菜单名称添加到 TreeView 组件的子节点 newNode1 中，并设置当前节点的子节点 newNode2
            TreeNode newNodeB = newNodeA.Nodes.Add(newmenuA.DropDownItems[j].Text);
            Var_Bool = true;
            if (BL == false)                                                 //判断一级命令是否可用
            {
                newNodeB.ForeColor = Color.Silver;                           //设置命令项的字体颜色
                newNodeB.Tag = 0;                                            //标识，不显示相应的窗体
                Var_Bool = false;
            }
            else
            {
                if (newmenuA.DropDownItems[j].Enabled == false)              //判断命令项是否为可用
                {
                    newNodeB.ForeColor = Color.Silver;                       //设置命令项的字体颜色
                    newNodeB.Tag = 0;                                        //标识，不显示相应的窗体
                    Var_Bool = false;
                }
                else
                {
                    newNodeA.ForeColor = Color.Black;                        //设置命令项的字体颜色
                    newNodeB.Tag =                                           //标识，显示相应的窗体
                      int.Parse(newmenuA.DropDownItems[j].Tag.ToString());
                }
            }
            //将当前菜单项的所有相关信息存入到 ToolStripDropDownItem 对象中
```

```
            ToolStripDropDownItem newmenuB = (ToolStripDropDownItem)newmenuA.DropDownItems[j];
            if (newmenuB.HasDropDownItems                                    //如果当前命令项有子项
                && newmenuA.DropDownItems.Count > 0)
            {
                newNodeB.Tag = 0;                                            //标识，有子项的命令项
                GetCavernMenu(newNodeB, newmenuB, Var_Bool);                 //调用递归方法
            }
        }
    }
```

秘笈心法

心法领悟 262：为窗体添加背景图片。

在窗体应用程序开发中，为了让窗体更加美观，经常会使用窗体的 Image 属性，方便地为窗体添加背景图片。

实例 263　用 TreeView 控件遍历磁盘目录

光盘位置：光盘\MR\11\263　　中级　　趣味指数：★★★★

实例说明

TreeView 控件通常用于显示树形视图，使用 TreeView 控件显示磁盘目录是一个很好的想法，实例中将利用 TreeView 控件为用户显示磁盘目录信息，就像在 Windows 操作系统的资源管理器的左窗格中显示文件和文件夹一样。实例运行效果如图 11.48 所示。

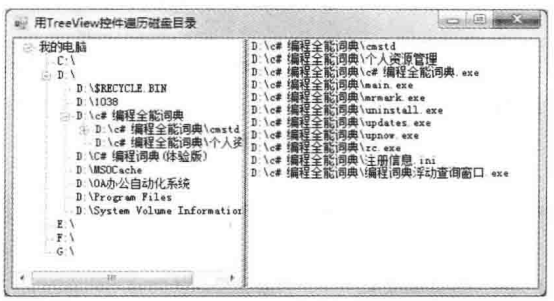


图 11.48　用 TreeView 控件遍历磁盘目录

关键技术

本实例主要用到了 ListView 控件中 Items 集合的 Add 方法和 TreeView 控件中 Nodes 集合的 Add 方法，下面分别对它们进行详细讲解。

ListView 控件中 Items 集合的 Add 方法，用于将数据项添加到 ListView 数据项集合中。语法格式如下：

```
public virtual ListViewItem Add(string text)
```

参数说明

❶ text：字符串对象，表示在 ListView 控件中数据项的文本。

❷ 返回值：返回已经添加到 ListView 控件的 ListViewItem 数据项的引用。

说明：关于 TreeView 控件中 Nodes 集合的 Add 方法的详细讲解请参见实例 259 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Getdirectory。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 SplitContainer 面板，用于装载 TreeView 和 ListView 控件；添加一个 TreeView 控件，用于显示磁盘与文件夹信息；添加一个 ListView 控件，用于显示文件信息。

（3）程序的主要代码如下：

```
private void ListViewShow(TreeNode NodeDir)                                    //初始化 ListView 控件
{
    ListViewFile.Clear();
    if (NodeDir.Parent == null)                                                //是否添加分区名称
    {
        foreach (string DrvName in Directory.GetLogicalDrives())               //获得硬盘分区名
        {
            ListViewItem ItemList = new ListViewItem(DrvName);
            ListViewFile.Items.Add(ItemList);                                  //添加信息
        }
    }
    else
    {
        foreach (string DirName in Directory.GetDirectories((string)NodeDir.Tag))  //遍历所有目录
        {
            ListViewItem ItemList = new ListViewItem(DirName);                 //创建数据项
            ListViewFile.Items.Add(ItemList);                                  //添加目录信息
        }
        foreach (string FileName in Directory.GetFiles((string)NodeDir.Tag))   //遍历所有目录的文件
        {
            ListViewItem ItemList = new ListViewItem(FileName);                //创建数据项
            ListViewFile.Items.Add(ItemList);                                  //添加文件信息
        }
    }
}
```

秘笈心法

心法领悟 263：得到磁盘驱动器信息。

文件操作过程中，可以使用 File 类的多个静态方法方便地操作磁盘文件；也可以使用 Directory 类的多个静态方法，对文件夹进行操作，通过 Directory 类的 GetLogicalDrives 静态方法可以轻松地得到磁盘驱动器信息。

实例 264　修改 TreeView 控件的节点文本

光盘位置：光盘\MR\11\264　　中级　　趣味指数：★★★★☆

实例说明

通过 TreeView 控件中 Nodes 集合的 Add 方法，可以方便地向 TreeView 控件中添加 TreeNode 节点。每个 TreeNode 自身还有一个用来存储其子节点的 Nodes 集合，可以使用相同的 Add 方法向节点中添加节点。本实例中将会演示怎样查询数据库的信息并添加到 TreeView 控件中，同时，将用户修改的 TreeView 文本信息同步到数据库。实例运行效果如图 11.49 所示。

图 11.49　修改 TreeView 控件的节点文本

关键技术

本实例主要用到了 TreeView 控件的 LabelEdit 属性和 Nodes 集合的 Clear 方法，下面分别对它们进行详细讲解。

（1）LabelEdit 属性

TreeView 控件的 LabelEdit 属性，用来指示是否可以编辑树节点的标签文本。语法格式如下：

```
public bool LabelEdit { get; set; }
```

参数说明

属性值：布尔值，如果可以编辑树节点的标签文本则值为 true，否则值为 false。

（2）Clear 方法

TreeView 控件中 Nodes 集合的 Clear 方法，用于从集合中删除所有树节点。语法格式如下：

```
public virtual void Clear()
```

说明：TreeView 控件中 Nodes 集合实现了 Ilist 接口，所以可以使用 Add、Remove 和 Clear 等方法，方便地向节点集合中添加子节点。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ModifiedNode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“修改 TreeView 控件的节点文本”，向窗体中添加一个 TreeView 控件，用于显示与修改节点中的文本，并将更改后的信息同步到数据库。

（3）程序的主要代码如下：

```
private void treeView1_AfterLabelEdit(object sender,NodeLabelEditEventArgs e)
{
    if(e.Label != null && e.Label != "")                          //如果选定内容不为空
    {
        NexusConnection.Open();                                   //打开数据库连接
        string RefreshString = "update Ware set  产品名称='" +    //定义一个数据库连接字段
            e.Label + "' where  产品编号=" + (e.Node.Index + 1).ToString();
        NexusCommand = new OleDbCommand(RefreshString,NexusConnection);  //定义一个执行 SQL 语句的对象
        NexusCommand.ExecuteNonQuery();                           //执行 SQL 语句
        NexusConnection.Close();                                  //关闭数据库连接
        MessageBox.Show("修改成功！","提示信息",                   //弹出修改成功的提示信息
                MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
    }
}
```

秘笈心法

心法领悟 264：将节点中更新的内容同步到数据库。

TreeView 控件的 AfterLabelEdit 事件，在用户编辑节点后发生。可以在该事件中写入同步数据的代码，将节点中文本代码的更改同步到数据库。

实例 265 将 XML 文件节点绑定到 TreeView 控件中

光盘位置：光盘\MR\11\265　　中级　　趣味指数：★★★☆

实例说明

程序设计过程中，经常用到 XML（eXtensible Markup Language，可扩展标记语言），XML 是一种严谨的描述数据的语言，在 C#中可以使用 XmlReader 和 XmlDocument 对象方便地读取 XML 文件的内容。本实例中将会使用 XmlDocument 读取指定的 XML 文档内容，并将内容添加到 TreeView 控件中。实例运行效果如图 11.50 所示。

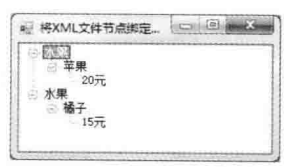


图 11.50　将 XML 文件节点绑定到 TreeView 控件中

关键技术

本实例主要用到了 XmlDocument 对象的 Load 方法，下面对其进行详细讲解。

XmlDocument 对象的 Load 方法，用来从指定的位置加载 XML 文档。语法格式如下：

```
public virtual void Load(string filename)
```

参数说明

filename：字符串对象，XML 文档的位置信息。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BindingXML。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 XML 文件节点绑定到 TreeView 控件中”，向窗体中添加一个 TreeView 控件，用于显示 XML 文档中的数据内容。

（3）程序的主要代码如下：

```
private void RecursionTreeControl(XmlNode xmlNode,TreeNodeCollection nodes)
{
    foreach(XmlNode node in xmlNode.ChildNodes)                    //循环遍历当前元素的子元素集合
    {
        string temp = (node.Value != null ? node.Value :           //表示 TreeNode 节点的文本内容
            (node.Attributes != null && node.Attributes.Count > 0) ? node.Attributes[0].Value : node.Name);
        TreeNode new_child = new TreeNode(temp);                   //定义一个 TreeNode 节点对象
        nodes.Add(new_child);                                      //向当前集合中添加当前节点
        RecursionTreeControl(node,new_child.Nodes);                //调用本方法进行递归
    }
}
```

秘笈心法

心法领悟 265：使用递归方法遍历 XML 文档。

实例中使用了递归方法得到了 XML 文档中的节点信息，得到节点信息后，判断得到的节点信息是否拥有子节点，如果拥有子节点，则再次使用递归的方式调用自身，得到子节点信息。

11.9　选项卡控件的使用

实例 266	更改选项卡的显示样式	初级
	光盘位置：光盘\MR\11\266	趣味指数：★★★★☆

实例说明

本实例通过 C#代码实现了更改选项卡显示样式的功能，运行本实例，可以看到选项卡呈现三维样式，并且在每一个选项卡中都显示用户自定义的图标。实例运行效果如图 11.51 所示。

图 11.51　更改选项卡的显示样式

关键技术

本实例实现时主要用到了 TabControl 控件的 ImageList 属性和 Appearance 属性，下面分别对它们进行详细讲解。

（1）ImageList 属性

TabControl 控件用来显示多个选项卡，这些选项卡类似于笔记本中的分隔卡和档案柜文件夹中的标签，其 ImageList 属性用来获取或设置在控件的选项卡上显示的图像。语法格式如下：

```
public ImageList ImageList { get; set; }
```

参数说明

属性值：ImageList 对象，指定在选项卡上显示的图像。

（2）Appearance 属性

该属性用来获取或设置控件选项卡的可视外观。语法格式如下：

```
public TabAppearance Appearance { get; set; }
```

参数说明

属性值：TabAppearance 枚举值之一，默认为 Normal。

TabAppearance 枚举值及说明如表 11.3 所示。

表 11.3　TabAppearance 枚举值及说明

枚 举 值	说　明
Normal	选项卡具有选项卡的标准外观
Buttons	选项卡具有三维按钮的外观
FlatButtons	选项卡具有平面按钮的外观

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TCtrlStyle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TabControl 控件，用来作为选项卡控件；添加一个 ImageList 组件，用来存储选项卡中用到的图标。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    tabControl1.ImageList = imageList1;                    //设置控件的 ImageList 属性为 imageList1
    tabPage1.ImageIndex = 0;                               //第一个选项卡的图标是 imageList1 中索引为 0 的图标
    tabPage1.Text = "C#编程词典——全能版";                  //设置第一个选项卡标题
    tabPage2.ImageIndex = 0;                               //第二个选项卡的图标是 imageList1 中索引为 0 的图标
    tabPage2.Text = "C#编程词典——珍藏版";                  //设置第二个选项卡标题
    //将控件的 Appearance 属性设置为 Buttons，使选项卡具有三维按钮的外观
    tabControl1.Appearance = TabAppearance.Buttons;
}
```

秘笈心法

心法领悟 266：TabControl 控件的使用。

TabControl 控件，又称为选项卡控件，它可以添加多个选项卡，然后可以在选项卡上添加子控件，这样就可以把窗体设计成多页，并且使窗体的功能划分为多个部分。选项卡控件的选项卡中可以包含图片或其他控件。

实例 267　添加多个选项卡

光盘位置：光盘\MR\11\267　　初级　　趣味指数：★★★★☆

实例说明

使用选项卡控件时，可以直接在 Visual Studio 2012 开发环境的“属性”窗口中添加选项卡，但有时选项卡需要根据程序的运行结果动态增加，这时该如何实现呢？本实例使用 C#实现了手动增加选项卡的功能。实例运行效果如图 11.52 所示。

图 11.52　添加多个选项卡

关键技术

本实例实现时主要用到了 TabControl 控件的 TabPages 属性的 Add 方法，下面对其进行详细讲解。

TabControl 控件的 TabPages 属性，用来获取选项卡控件中选项卡页的集合。语法格式如下：

```
public TabControl.TabPageCollection TabPages { get; }
```

参数说明

属性值：TabPageCollection 对象，它包含 TabControl 控件中的 TabPage 对象。

TabPages 属性的属性值是一个 TabPageCollection 对象，该对象包含 TabPage 对象的集合，其 Add 方法用来将选项卡页添加到集合，该方法是一个可重载方法，本实例中用到的它的重载形式如下：

```
public void Add(TabPage value)
```

参数说明

value：要添加的 TabPage 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddTabPages。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TabControl 控件，用来作为选项卡控件；添加一个 Button 控件，用来执行添加选项卡操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string Title = "新增选项卡 " + (tabControl1.TabCount + 1).ToString();        //声明一个字符串变量，用于生成新增选项卡的名称
    TabPage MyTabPage = new TabPage(Title);                                      //创建 TabPage 对象
    //使用 TabControl 控件的 TabPages 属性的 Add 方法添加新的选项卡
    tabControl1.TabPages.Add(MyTabPage);
    MessageBox.Show("现有" + tabControl1.TabCount + "个选项卡");                  //获取选项卡个数
}
```

秘笈心法

心法领悟 267：如何获取选项卡控件中的选项卡个数？

获取选项卡控件中的选项卡个数时，需要用到 TabControl 控件的 TabCount 属性，该属性用来获取选项卡条中选项卡的数目。例如，本实例在弹出的对话框中显示选项卡个数的实现代码如下：

```
MessageBox.Show("现有" + tabControl1.TabCount + "个选项卡");                  //获取选项卡个数
```

实例 268　获取选中的选项卡名称

光盘位置：光盘\MR\11\268　　初级　　趣味指数：★★★★

实例说明

程序中使用多个选项卡时，每一个选项卡所表示的意义都是不同的，那么如何来根据不同的选项卡执行不同的操作呢？第一步首先需要知道选择的是哪个选项卡，接下来才需要根据不同的选项卡执行不同的操作。本实例实现了获取选中选项卡名称的功能。实例运行效果如图 11.53 所示。

图 11.53　获取选中的选项卡名称

关键技术

本实例实现时主要用到了 TabControl 控件的 SelectedTab 属性，下面对其进行详细讲解。

SelectedTab 属性用来获取或设置当前选定的选项卡。语法格式如下：

```
[BrowsableAttribute(false)]
public TabPage SelectedTab { get; set; }
```

参数说明

属性值：TabPage 对象，表示选定的选项卡。如果未选定任何选项卡，则值为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetSelectTabPage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TabControl 控件，用来作为选项卡控件。

（3）程序的主要代码如下：

```
private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
{
    string P_str_TabName = tabControl1.SelectedTab.Text;                    //获取选择的选项卡名称
    MessageBox.Show("您选择的选项卡为：" + P_str_TabName, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);//弹出信息提示
}
```

秘笈心法

心法领悟 268：使用 MessageBox 显示消息对话框。

向用户显示信息和消息的对话框称为消息对话框，MessageBox 是一个预定义对话框，主要用于向用户显示与应用程序相关的信息及请求来自用户的信息，其 Show 方法用来显示消息对话框。

实例 269　删除指定的选项卡

光盘位置：光盘\MR\11\269　　初级　　趣味指数：★★★★

实例说明

本实例主要实现删除指定选项卡的功能，运行本实例，选中指定的选项卡，单击“移除选项卡”按钮，即可删除选择的选项卡。实例运行效果如图 11.54 所示。

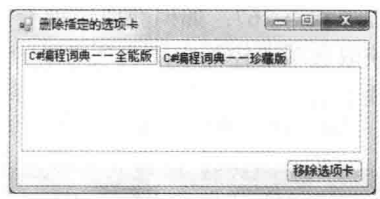


图 11.54　删除指定的选项卡

关键技术

本实例实现时主要用到了 TabControl 控件的 TabPages 属性的 Remove 方法，下面对其进行详细讲解。

TabControl 控件的 TabPages 属性用来获取选项卡控件中选项卡的集合。语法格式如下：

```
public TabControl.TabPageCollection TabPages { get; }
```

参数说明

属性值：TabPageCollection 对象，它包含 TabControl 控件中的 TabPage 对象。

TabPages 属性的属性值是一个 TabPageCollection 对象，该对象包含 TabPage 对象的集合，其 Remove 方法用来从集合中移除 TabPage 对象。语法格式如下：

```
public void Remove(TabPage value)
```

参数说明

value：要移除的 TabPage 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RemoveTabPage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TabControl 控件，用来作为选项卡控件；添加一个 Button 控件，用来执行删除选项卡操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (tabControl1.SelectedIndex == 0)                    //判断是否选择了要移除的选项卡
    {
```

```
        MessageBox.Show("请选择要移除的选项卡");                    //如果没有选择，弹出提示
    }
    else
    {
        //使用 TabControl 控件的 TabPages 属性的 Remove 方法移除指定的选项卡
        tabControl1.TabPages.Remove(tabControl1.SelectedTab);
    }
}
```

秘笈心法

心法领悟 269：如何判断是否选择了选项卡？

判断是否选择了选项卡时，需要用到 TabControl 控件的 SelectedIndex 属性，该属性用来获取或设置当前选定的选项卡的索引。例如，本实例判断是否选择了要移除选项卡的实现代码如下：

```
if (tabControl1.SelectedIndex == 0)                                //判断是否选择了要移除的选项卡
{
    MessageBox.Show("请选择要移除的选项卡");                        //如果没有选择，弹出提示
}
```

11.10　菜单应用实例

实例 270　为菜单添加快捷键

光盘位置：光盘\MR\11\270　　中级　　趣味指数：★★★☆

实例说明

程序设计过程中，经常会使用 MenuStrip 控件，MenuStrip 控件通常被称作弹出菜单，使用弹出菜单可以让用户更加清晰直观地了解应用程序所提供的功能。本实例中将会讲解如何为弹出菜单添加快捷键，使用户在不使用鼠标的情况下，也可以正常地使用菜单操作。实例运行效果如图 11.55 所示。

图 11.55　为菜单添加快捷键

关键技术

本实例重点介绍怎样通过 ToolStripMenuItem 菜单项的 Text 属性设置菜单快捷键，下面对本实例用到的关键技术进行详细讲解。

为菜单项设置快捷键非常简单，设置菜单项的 Text 属性时，在菜单项文本中添加“&”+“字母”即可，使用快捷键时，按下 Alt 键加指定的“字母”便可以使用快捷键激活菜单项。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShortCutInMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“为菜单添加快捷键”，向窗体中添加一个 MenuStrip 控件，用于演示为菜单添加快捷键。

（3）程序的主要代码如下：

```
private void 退出 QToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("退出应用程序","提示！ ");                      //弹出消息对话框
    Close();                                                       //退出应用程序
}
private void 打开 OToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    MessageBox.Show("已经单击\"打开\"菜单项", "提示！");                    //弹出消息对话框
}
```

秘笈心法

心法领悟 270：消息对话框的妙用。

程序运行过程中，针对用户的操作，应用程序经常需要做出不同的反应，使用 MessageBox 类的 Show 方法可以方便地弹出消息对话框，以增强用户与应用程序之间的交互。MessageBox 类的 Show 方法会根据用户的选择返回一个 DialogResult 枚举对象，通过此枚举对象判断用户的选择并做出相应的操作。

实例 271　设置菜单项是否可用

光盘位置：光盘\MR\11\271　　中级　　趣味指数：★★★☆

实例说明

MenuStrip 弹出菜单是窗体应用程序中常用的控件，菜单中可以添加多个菜单项，而菜单项中还可以包含多个子菜单项，在窗体应用程序执行过程中，如果需要禁用菜单中一部分子菜单项要怎样实现呢？本实例中将会演示怎样停用 MenuStrip 控件中的菜单项。实例运行效果如图 11.56 所示。

图 11.56　设置菜单项是否可用

关键技术

本实例主要用到了 ToolStripMenuItem 菜单项的 Enabled 属性，下面对其进行详细讲解。

ToolStripMenuItem 菜单项的 Enabled 属性，用来指示菜单项是否可用。语法格式如下：

```
public override bool Enabled { get; set; }
```

参数说明

属性值：布尔值，如果已启用菜单则值为 true，否则值为 false。

说明：ToolStripMenuItem 菜单项的 Enabled 属性默认值为 true，所以在默认情况下，菜单项为可用状态。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisableMenuItem。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于演示启用和停用菜单项；添加两个 Button 控件，分别用于启用和停用菜单项。

（3）程序的主要代码如下：

```
private void btn_Enable_Click(object sender, EventArgs e)
{
    ToolStripMenuItem P_ToolStripMenuItem = (ToolStripMenuItem)menus_Main.Items[0];   //得到菜单项
    foreach (ToolStripMenuItem item in P_ToolStripMenuItem.DropDownItems)             //遍历菜单项集合
    {
        item.Enabled = true;                                                          //启用菜单项
    }
}
private void btn_Disable_Click(object sender, EventArgs e)
{
    ToolStripMenuItem P_ToolStripMenuItem = (ToolStripMenuItem)menus_Main.Items[0];   //得到菜单项
    foreach (ToolStripMenuItem item in P_ToolStripMenuItem.DropDownItems)             //遍历菜单项集合
    {
        item.Enabled = false;                                                         //停用菜单项
    }
}
```

秘笈心法

心法领悟 271：遍历菜单项集合。

ToolStripMenuItem 菜单项的 DropDownItems 集合包含了多个菜单项，由于 DropDownItems 集合实现了 Ilist 接口，所以可以使用 for 或 foreach 方便地遍历集合中的菜单项。

实例 272	将菜单项的字体设置为粗体	中级
	光盘位置：光盘\MR\11\272	趣味指数：★★★☆

实例说明

MenuStrip 弹出菜单中可以包含多个菜单项，在应用程序执行过程中，可能需要着重地显示某些菜单项，那么，将这些菜单项的字体设置为粗体是一个不错的选择。本实例中将会介绍怎样将菜单中的某些菜单项设置为粗体。实例运行效果如图 11.57 所示。

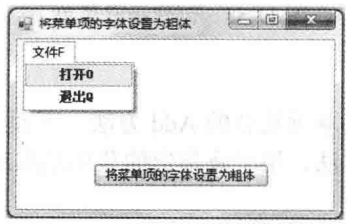


图 11.57　将菜单项的字体设置为粗体

关键技术

本实例主要用到了 ToolStripMenuItem 菜单项的 Font 属性，下面对其进行详细讲解。

ToolStripMenuItem 菜单项的 Font 属性，用来设置菜单项文本内容的字体。语法格式如下：

```
public virtual Font Font { get; set; }
```

参数说明

属性值：Font 字体对象，菜单项文本的字体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BoldItem。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于显示加粗的菜单项；添加一个 Button 控件，用于设置菜单项中的字体为粗体。

（3）程序的主要代码如下：

```
private void btn_Bold_Click(object sender, EventArgs e)
{
    ToolStripMenuItem P_ToolStripMenuItem =(ToolStripMenuItem)menus_Bold.Items[0];    //得到菜单项
    foreach (ToolStripItem item in P_ToolStripMenuItem.DropDownItems)                 //遍历菜单项集合
    {
        item.Font = new Font(new Font("宋体",9), FontStyle.Bold);                     //设置菜单项字体
    }
}
```

秘笈心法

心法领悟 272：设置菜单项的背景颜色。

每一个菜单项都是一个 ToolStripMenuItem 对象，通过设置 ToolStripMenuItem 对象的 BackColor 属性可以方便地更改该菜单项的背景颜色。

实例 273 创建级联菜单

光盘位置：光盘\MR\11\273

中级

趣味指数：★★★☆

实例说明

如果应用程序提供了非常多的功能，那么，使用 MenuStrip 弹出菜单会是一个不错的选择，可以将应用程序所提供的功能分类放入不同的菜单项中，现在问题出现了，怎样在菜单中添加多个菜单项呢？本实例中将会介绍如何创建多级菜单项。实例运行效果如图 11.58 所示。

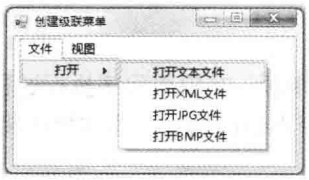


图 11.58 创建级联菜单

关键技术

本实例主要用到了 DropDownItems 菜单项集合的 Add 方法，下面对其进行详细讲解。

DropDownItems 菜单项集合的 Add 方法，用于将指定的菜单项添加到菜单项集合。语法格式如下：

```
public int Add(ToolStripItem value)
```

参数说明

❶ value：ToolStripItem 对象，ToolStripItem 为 ToolStripMenuItem 的抽象基类，所以可以向菜单项集合中添加 ToolStripMenuItem 对象。

❷ 返回值：返回整型数值，表示菜单项集合中新添加菜单项的索引。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“创建级联菜单”，向窗体中添加一个 MenuStrip 控件，用于演示创建级联菜单。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    ToolStripMenuItem P_ts = (ToolStripMenuItem)menuStrip1.Items[0];                    //得到文件菜单项
    ToolStripMenuItem ts1 = new ToolStripMenuItem("打开文本文件");                       //创建菜单项
    ToolStripMenuItem ts2 = new ToolStripMenuItem("打开 XML 文件");                      //创建菜单项
    ToolStripMenuItem ts3 = new ToolStripMenuItem("打开 JPG 文件");                      //创建菜单项
    ToolStripMenuItem ts4 = new ToolStripMenuItem("打开 BMP 文件");                      //创建菜单项
    ToolStripMenuItem P_ts2 = (ToolStripMenuItem)P_ts.DropDownItems[0];                  //得到子菜单项
    P_ts2.DropDownItems.Add(ts1);                                                        //添加菜单项
    P_ts2.DropDownItems.Add(ts2);                                                        //添加菜单项
    P_ts2.DropDownItems.Add(ts3);                                                        //添加菜单项
    P_ts2.DropDownItems.Add(ts4);                                                        //添加菜单项
}
```

秘笈心法

心法领悟 273：创建级联菜单。

每一个菜单项都是一个 ToolStripMenuItem 对象，而每一个 ToolStripMenuItem 对象都有 DropDownItems 集合，该集合中可以存放 ToolStripMenuItem 对象，此集合表示菜单项的子菜单项，所以每一个菜单项都可以通过 DropDownItems 集合添加子菜单项。

实例 274　级联菜单的动态合并

光盘位置：光盘\MR\11\274　　中级　　趣味指数：★★★★☆

实例说明

程序设计过程中，经常会使用 MenuStrip 弹出菜单，并且一个窗体中可以存在多个弹出菜单。在 MDI 应用程序中，当 MDI 子窗体最大化时，子窗体和主窗体的菜单能够自动地合并。这是如何实现的呢？这正是本实例将要介绍的内容，实例中将两个弹出菜单动态地合并成一个弹出菜单，运行效果如图 11.59 所示。

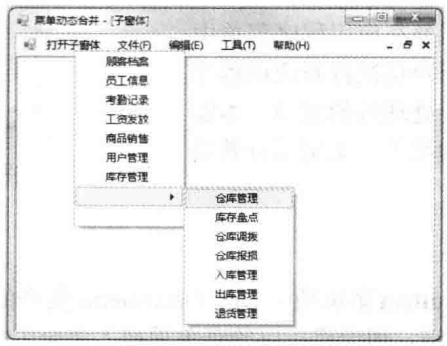


图 11.59　级联菜单的动态合并

关键技术

本实例主要用到了 ContextMenuStrip 控件中 Items 集合的 AddRange 方法，下面对其进行详细讲解。

ContextMenuStrip 控件中 Items 集合的 AddRange 方法，用于将 ToolStripItem 控件的数组添加到菜单集合中。语法格式如下：

```
public void AddRange(ToolStripItem[] toolStripItems)
```

参数说明

toolStripItems：ToolStripItem 对象的控件数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AmalgamateMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，设置窗体的 IsMdiContainer 属性为 true，将窗体设置为多文档窗体；向窗体中添加一个 MenuStrip 控件和一个 ContextMenuStrip 控件，分别用于显示菜单和窗体右键菜单。

（3）程序的主要代码如下：

```
void f_Resize(object sender, EventArgs e)
{
    Form2 f = (Form2)sender;                                          //得到窗体对象
    ToolStripMenuItem item = new ToolStripMenuItem();                 //创建菜单项
    for (int i = 0; i < f.contextMenuStrip2.Items.Count; )            //遍历窗体菜单项集合
    {
        item.DropDownItems.Add(f.contextMenuStrip2.Items[i]);         //添加菜单项
    }
    this.contextMenuStrip1.Items.AddRange(                            //向主窗体中添加菜单项集合
        new System.Windows.Forms.ToolStripItem[] {item});
}
```

秘笈心法

心法领悟 274：更改菜单项的文字颜色。

每一个菜单项都是一个 ToolStripMenuItem 对象，通过设置 ToolStripMenuItem 对象的 ForeColor 属性可以方便地更改该菜单项的文字颜色。

实例 275　带历史信息的菜单

光盘位置：光盘\MR\11\275　　中级　　趣味指数：★★★☆

实例说明

在应用程序开发过程中，经常需要在应用程序菜单中记录最近打开文档的历史信息，当用户再次打开应用程序时可以方便地从历史信息中找到曾经处理过的文档。本实例中将会介绍怎样创建带历史信息的菜单。实例运行效果如图 11.60 所示。

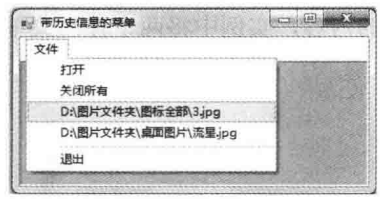


图 11.60　带历史信息的菜单

关键技术

本实例主要用到了 ToolStripMenuItem 菜单项中 DropDownItems 集合的 Insert 方法，下面对其进行详细讲解。

DropDownItems 集合的 Insert 方法，用于将指定的菜单项插入集合中的指定索引处。语法格式如下：

```
public void Insert(int index,ToolStripItem value)
```

参数说明

❶ index：整型数值，插入菜单项的位置。

❷ value：ToolStripItem 对象，要插入的菜单项。

提示：Insert 方法中 Index 参数的数值不能超出 DropDownItems 集合的索引范围，否则会抛出异常。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 HistoryMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于打开图像文件，并记录打开文件的历史信息。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    StreamReader sr = new StreamReader(address + "\\History.ini");          //创建流读取器对象
    int i =文件 ToolStripMenuItem.DropDownItems.Count-2;                    //得到菜单项索引
    while (sr.Peek()>=0)                                                    //循环读取流中的文本
    {
        ToolStripMenuItem menuitem =      new ToolStripMenuItem(sr.ReadLine());   //创建菜单项对象
        this.文件 ToolStripMenuItem.DropDownItems.Insert(i, menuitem);        //向菜单中添加新项
        i++;                                                                //向菜单中插入索引的位置
        menuitem.Click += new EventHandler(menuitem_Click);                 //添加单击事件
    }
    sr.Close();                                                             //关闭流
}
```

秘笈心法

心法领悟 275：向菜单项中添加菜单。

每一个菜单项都是一个 ToolStripMenuItem 对象，而每一个 ToolStripMenuItem 对象都有 DropDownItems 集合，该集合中存放着多个子菜单项，由于 DropDownItems 集合实现了 Ilist 接口，所以可以使用集合的 Add 方法，方便地向菜单项中添加多个子菜单项。

实例 276　像开始菜单一样漂亮的菜单

光盘位置：光盘\MR\11\276　　中级　　趣味指数：★★★★☆

实例说明

Windows 是我们常用的操作系统，它的开始菜单设计得很独特，可以适当地在 MenuStrip 弹出菜单中添加图片信息，使菜单看起来更加整洁、清晰。本实例中将会在窗体中设计一个像开始菜单一样漂亮的菜单。实例运行效果如图 11.61 所示。

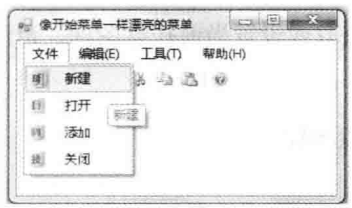


图 11.61　像开始菜单一样漂亮的菜单

关键技术

本实例主要用到了 ToolStripMenuItem 菜单项的 Image 属性，下面对其进行详细讲解。

ToolStripMenuItem 菜单项的 Image 属性，用来获取或设置菜单项上的图像。语法格式如下：

```
public virtual Image Image { get; set; }
```

参数说明

属性值：Image 对象，菜单项上显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StartMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“像开始菜单一样漂亮的菜单”。

（3）向窗体中添加一个 MenuStrip 控件，单击 MenuStrip 中的菜单项新建菜单，通过设置新建菜单项的 Text 属性确定菜单项的文本，通过设置菜单项的 Image 属性确定菜单项中显示的图片。

（4）向窗体中添加一个 ToolStrip 控件，用于显示工具栏按钮。

秘笈心法

心法领悟 276：为菜单项添加图片。

每一个菜单项都是一个 ToolStripMenuItem 对象，可以通过设置 ToolStripMenuItem 对象的 Image 属性，方便地为菜单项添加图片。

实例 277　制作任务栏托盘菜单

光盘位置：光盘\MR\11\277　　中级　　趣味指数：★★★★☆

实例说明

窗体应用程序开发过程中，可以通过 NotifyIcon 组件方便地为应用程序添加托盘图标，设置 NotifyIcon 组件的 ContextMenuStrip 属性可以为托盘添加菜单项。本实例中利用 NotifyIcon 组件制作了一个任务栏托盘菜单。实例运行效果如图 11.62 所示。

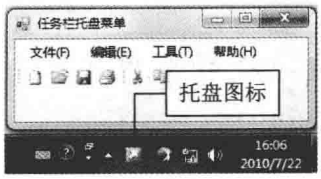


图 11.62　制作任务栏托盘菜单

关键技术

本实例主要用到了 NotifyIcon 组件的 Icon 和 ContextMenu 属性，下面分别对它们进行详细讲解。

（1）Icon 属性

NotifyIcon 组件的 Icon 属性，用来获取或设置当前托盘图标。语法格式如下：

```
public Icon Icon { get; set; }
```

参数说明

属性值：Icon 对象，用于显示的托盘图标。

（2）ContextMenu 属性

NotifyIcon 组件的 ContextMenu 属性，获取或设置托盘图标的快捷菜单。语法格式如下：

```
public ContextMenu ContextMenu { get; set; }
```

参数说明

属性值：ContextMenu 对象，托盘图标的 ContextMenu 菜单。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetNotifyIcon。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“任务栏托盘菜单”，向窗体中添加一个 MenuStrip 控件，用于显示窗体菜单项。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    NotifyIcon P_NotifyIcon = new NotifyIcon();                          //创建托盘图标对象
    P_NotifyIcon.Visible = true;                                         //设置托盘图标可见
    Icon ic = new Icon("2.ico");                                         //创建 Icon 对象
    P_NotifyIcon.Icon = ic;                                              //设置托盘使用的图标
    MenuItem P_mi = new MenuItem("最大化");                              //创建菜单项
    P_mi.Click += new EventHandler(P_mi_Click);                          //设置单击事件
    MenuItem P_mi2 = new MenuItem("退出");                               //创建菜单项
    P_mi2.Click += new EventHandler(P_mi2_Click);                        //设置单击事件
    P_NotifyIcon.ContextMenu = new ContextMenu(new MenuItem[] { P_mi, P_mi2 });  //为托盘添加菜单
}
```

秘笈心法

心法领悟 277：隐藏菜单项。

每一个菜单项都是一个 ToolStripMenuItem 对象，可以通过设置 ToolStripMenuItem 对象的 Visible 属性为 false 隐藏菜单项，也可以设置 Visible 属性为 true 显示菜单项。

实例 278	可以拉伸的菜单	中级
	光盘位置：光盘\MR\11\278	趣味指数：★★★☆

实例说明

如果应用程序分类中的菜单项过多，而用户只使用一些常用菜单，这时，可以将主菜单项下的不常用菜单

隐藏起来。此种显示方式类似于对菜单进行拉伸。使用时，只需单击展开菜单，即可显示相应菜单功能。本实例中将会向读者演示怎样制作可以拉伸的菜单。实例运行效果如图 11.63 所示。

图 11.63　可以拉伸的菜单

关键技术

本实例主要用到了 ToolStripMenuItem 菜单项的 Visible 属性，下面对其进行详细讲解。

ToolStripMenuItem 菜单项的 Visible 属性用来设置是否显示菜单项。语法格式如下：

```
public bool Visible { get; set; }
```

参数说明

属性值：布尔值，如果显示菜单项则值为 true，否则值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 StretchMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于演示可以拉伸的菜单。

（3）程序的主要代码如下：

```
private void toolStripMenuItem1_Click(object sender, EventArgs e)
{
    switch (G_bl)
    {
        case false:
            this.设置密码 ToolStripMenuItem.Visible = false;                    //隐藏菜单项
            this.添加用户 ToolStripMenuItem.Visible = false;                    //隐藏菜单项
            this.忘记密码 ToolStripMenuItem.Visible = false;                    //隐藏菜单项
            this.修改密码 ToolStripMenuItem.Visible = false;                    //隐藏菜单项
            this.员工录入 ToolStripMenuItem.Visible = false;                    //隐藏菜单项
            G_bl = true;                                                        //设置布尔值
            操作 ToolStripMenuItem.ShowDropDown();                              //显示菜单项
            break;
        case true:
            this.设置密码 ToolStripMenuItem.Visible = true;                     //显示菜单项
            this.添加用户 ToolStripMenuItem.Visible = true;                     //显示菜单项
            this.忘记密码 ToolStripMenuItem.Visible = true;                     //显示菜单项
            this.修改密码 ToolStripMenuItem.Visible = true;                     //显示菜单项
            this.员工录入 ToolStripMenuItem.Visible = true;                     //显示菜单项
            G_bl = false;                                                       //设置布尔值
            this.操作 ToolStripMenuItem.ShowDropDown();                         //显示菜单项
            break;
    }
}
```

秘笈心法

心法领悟 278：停用菜单项。

每一个菜单项都是一个 ToolStripMenuItem 对象，可以通过设置 ToolStripMenuItem 对象的 Enable 属性为 false

停用菜单项，也可以设置 Enable 属性为 true 启用菜单项。

实例 279　仿 XP 系统的任务栏菜单

光盘位置：光盘\MR\11\279　　中级　　趣味指数：★★★☆

实例说明

在窗体应用程序开发过程中，为了使窗体更加美观、实用，可以模仿 XP 系统任务栏菜单的效果。那么，怎样设计任务栏菜单呢？这正是本实例所演示的内容。实例运行效果如图 11.64 所示。

图 11.64　仿 XP 系统的任务栏菜单

关键技术

本实例主要用到了 PictureBox 控件的 Image 属性和 Panel 控件的 Visible 属性，下面分别对它们进行详细讲解。

（1）Image 属性

PictureBox 控件的 Image 属性，用来获取或设置 PictureBox 控件中显示的图像。语法格式如下：

```
public Image Image { get; set; }
```

参数说明

属性值：Image 对象，在 PictureBox 控件中显示的图像。

（2）Visible 属性

Panel 控件的 Visible 属性，用来设置是否显示 Panel 控件及 Panel 控件中包含的控件。语法格式如下：

```
public bool Visible { get; set; }
```

参数说明

属性值：布尔值，如果显示 Panel 控件及 Panel 控件中包含的控件则值为 true，否则值为 false。

说明：如果设置 Panel 控件的 Visible 属性为 false，Panel 面板及其中的控件都会被隐藏。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LikesXP。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“仿 XP 系统的任务栏菜单”，向窗体中添加 4 个 Panel 控件，用于建立仿 XP 系统的任务栏菜单。

（3）程序的主要代码如下：

```
private void pictureBox_1_Click(object sender, EventArgs e)
{
    Var_i = Convert.ToInt16(((PictureBox)sender).Tag.ToString());          //得到控件中的数据
    switch (Var_i)
    {
        case 1:
            {
```

```
                Var_Panel = panel_Gut_1;                                    //得到面板对象引用
                Var_Pict = pictureBox_1;                                    //得到 PictureBox 对象引用
                break;
            }
        case 2:
            {
                Var_Panel = panel_Gut_2;                                    //得到面板对象引用
                Var_Pict = pictureBox_2;                                    //得到 PictureBox 对象引用
                break;
            }
        case 3:
            {
                Var_Panel = panel_Gut_3;                                    //得到面板对象引用
                Var_Pict = pictureBox_3;                                    //得到 PictureBox 对象引用
                break;
            }
    }
    if (Convert.ToInt16(Var_Panel.Tag.ToString()) == 0 || Convert.ToInt16(Var_Panel.Tag.ToString()) == 2)
    {
        Var_Panel.Tag = 1;                                                  //设置为隐藏标识
        Var_Pict.Image = Properties.Resources.朝下按钮;                      //设置图像属性
        Var_Panel.Visible = false;                                          //隐藏面板
    }
    else
    {
        if (Convert.ToInt16(Var_Panel.Tag.ToString()) == 1)
        {
            Var_Panel.Tag = 2;                                              //设置为显示标识
            Var_Pict.Image = Properties.Resources.朝上按钮;                  //设置图像属性
            Var_Panel.Visible = true;                                       //显示面板
        }
    }
}
```

秘笈心法

心法领悟 279：动画效果。

可以使用 Timer 控件或线程，间隔性地更改 PictureBox 控件的 Image 属性，使 PictureBox 控件显示不同的图像，以达到动画效果。

11.11　工具栏设计

实例 280	带图标的工具栏	中级
	光盘位置：光盘\MR\11\280	趣味指数：★★★☆

实例说明

程序设计过程中，经常会用到 ToolStrip 工具栏，在该工具栏中可以添加按钮、标签、菜单等信息。本实例中将会向读者演示怎样向 ToolStrip 工具栏中添加带图标的按钮控件。实例运行效果如图 11.65 所示。

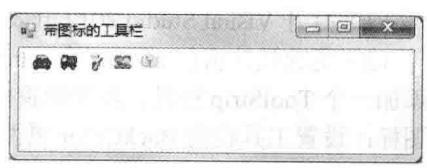


图 11.65　带图标的工具栏

关键技术

本实例主要用到了 ToolStripButton 对象的 Image 属性，下面对其进行详细讲解。

ToolStripButton 对象的 Image 属性，用来获取或设置在 ToolStripButton 中显示的图像。语法格式如下：

```
public virtual Image Image { get; set; }
```

参数说明

属性值：Image 对象，在工具栏按钮中显示的图像。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PicturesInTool。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“带图标的工具栏”，向窗体中添加一个 ToolStrip 控件，用于向窗体中添加工具栏按钮，设置工具栏按钮的 Image 属性可以指定工具栏按钮的图标。

秘笈心法

心法领悟 280：为工具栏按钮添加图标。

ToolStrip 工具栏中可以包含多个 ToolStripButton 对象，可以通过设置 ToolStripButton 对象的 Image 属性，方便地在工具栏的按钮中显示图标。

实例 281 带背景图片的工具栏

光盘位置：光盘\MR\11\281　　中级　　趣味指数：★★★☆

实例说明

使用 ToolStrip 工具栏可以方便、友好地为用户提供应用程序功能，在工具栏中可以通过设置工具栏按钮项的 BackColor 属性设置按钮的背景颜色，本实例中将会向读者演示创建带有背景色的工具栏。实例运行效果如图 11.66 所示。

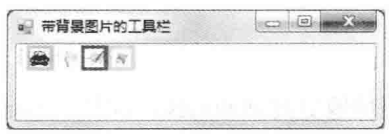


图 11.66　带背景图片的工具栏

关键技术

本实例主要用到了 ToolStripButton 对象的 BackColor 属性，下面对其进行详细讲解。

ToolStripButton 对象的 BackColor 属性，用来获取或设置工具栏按钮的背景色。语法格式如下：

```
public virtual Color BackColor { get; set; }
```

参数说明

属性值：Color 结构，在工具栏按钮中显示的背景色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BackGround。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“带背景图片的工具栏”，向窗体中添加一个 ToolStrip 控件，用于向窗体中添加工具栏按钮，设置工具栏按钮的 Image 属性可以指定工具栏按钮的图标；设置工具栏的 BackColor 属性可以指定工具栏按钮的背景图颜色。

秘笈心法

心法领悟 281：为工具栏按钮添加背景色。

ToolStrip 工具栏中可以包含多个 ToolStripButton 对象，可以使用 ToolStripButton 对象的 BackColor 属性，方便地为工具栏的按钮设置背景颜色。

实例 282　设计浮动工具栏

光盘位置：光盘\MR\11\282　　中级　　趣味指数：★★★★☆

实例说明

在使用 Word 应用程序时会发现，Word 应用程序上部的工具栏是可以被拖动的，这样做有利于用户对 Word 文档的操作。通过对本实例的学习，读者也可以在窗体中设计一个可以被拖动的工具栏。实例运行效果如图 11.67 所示。

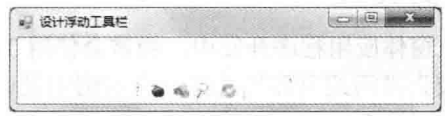


图 11.67　设计浮动工具栏

关键技术

本实例主要用到了 ToolStripPanel 控件的 Dock 属性和 ToolStripButton 对象的 Join 方法，下面分别对它们进行详细讲解。

（1）Dock 属性

ToolStripPanel 控件的 Dock 属性，用来获取或设置控件的停靠方式。语法格式如下：

```
public override DockStyle Dock { get; set; }
```

参数说明

属性值：DockStyle 枚举值，默认为 None。

（2）Join 方法

ToolStripButton 对象的 Join 方法，用于将指定的 ToolStrip 添加到 ToolStripPanel。语法格式如下：

```
public void Join(ToolStrip toolStripToDrag)
```

参数说明

toolStripToDrag：ToolStrip 工具栏控件，要添加到 ToolStripPanel 的 ToolStrip 工具栏。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DriftTool。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“设计浮动工具栏”，向窗体中添加一个 ToolStrip 控件，用于演示浮动工具栏。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    ToolStripPanel tsp_Top = new ToolStripPanel();                    //创建 ToolStripPanel 对象
    ToolStripPanel tsp_Bottom = new ToolStripPanel();                 //创建 ToolStripPanel 对象
    ToolStripPanel tsp_Left = new ToolStripPanel();                   //创建 ToolStripPanel 对象
    ToolStripPanel tsp_right = new ToolStripPanel();                  //创建 ToolStripPanel 对象
    tsp_Top.Dock = DockStyle.Top;                                     //设置停靠方式
    tsp_Bottom.Dock = DockStyle.Bottom;                               //设置停靠方式
    tsp_Left.Dock = DockStyle.Left;                                   //设置停靠方式
    tsp_right.Dock = DockStyle.Right;                                 //设置停靠方式
    Controls.Add(tsp_Top);                                            //添加到控件集合
    Controls.Add(tsp_Bottom);                                         //添加到控件集合
    Controls.Add(tsp_Left);                                           //添加到控件集合
    Controls.Add(tsp_right);                                          //添加到控件集合
    tsp_Bottom.Join(toolStrip1);                                      //将指定工具栏添加到面板
}
```

秘笈心法

心法领悟 282：更改控件在窗体中的位置。

由于所有控件都继承于 Control 类，所以，所有控件都有 Location 属性，Location 属性用于标记控件在窗体

中的坐标位置，可以通过设置该属性来更改控件在窗体中的位置。

实例 283　带下拉菜单的工具栏

光盘位置：光盘\MR\11\283　　中级　　趣味指数：★★★☆

实例说明

在窗体应用程序开发中，经常会使用 ToolStrip 工具栏控件来完成一些较为常用或重要的功能。本实例中设计了一个带有下拉菜单的工具栏，这样可以使用户的操作更加方便。实例运行效果如图 11.68 所示。

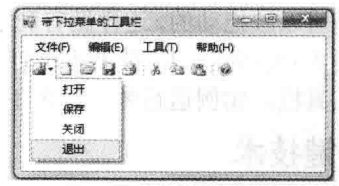


图 11.68　带下拉菜单的工具栏

关键技术

本实例主要用到了 ToolStripDropDownButton 的 DropDownItems 集合的 Add 方法，下面对其进行详细讲解。

ToolStripDropDownButton 的 DropDownItems 集合的 Add 方法，用于向集合中添加项。语法格式如下：

```
public int Add(ToolStripItem value)
```

参数说明

❶ value：ToolStripItem 对象。

❷ 返回值：返回整型数值，表示向集合中添加的新项的索引。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DropDownMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“带下拉菜单的工具栏”，向窗体中添加一个 MenuStrip 控件，用于显示文件菜单；添加一个 ToolStrip 控件，用于演示带有下拉菜单的工具栏。

（3）添加带有下拉菜单的工具栏非常简单，单击工具栏控件中的新建图标，选择新建 DropDownButton 即可。

秘笈心法

心法领悟 283：ToolStrip 的 Items 集合。

由于 ToolStrip 工具栏的 Items 集合实现了 Ilist 接口，所以可以使用 Items 集合的 Add 方法，方便地向 ToolStrip 工具栏中添加项。

实例 284　具有提示功能的工具栏

光盘位置：光盘\MR\11\284　　中级　　趣味指数：★★★☆

实例说明

在使用 Word 应用程序时会发现，Word 应用程序上部的工具栏是可以被拖动的，而且当鼠标在工具栏按钮上停留片刻，会出现一个提示框，提示框内描述了工具栏按钮所提供的功能，本实例使用工具栏的 ToolTipText 属性实现了上述功能。实例运行效果如图 11.69 所示。

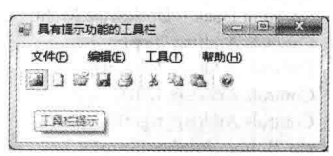


图 11.69　具有提示功能的工具栏

关键技术

本实例主要用到了 ToolStripButton 对象的 ToolTipText 属性，下面对其进行详细讲解。

ToolStripButton 对象的 ToolTipText 属性，用来获取或设置作为控件的 ToolTip 显示的文本。语法格式如下：

```
public string ToolTipText { get; set; }
```

参数说明

属性值：字符串对象，表示工具栏提示文本的字符串。

说明：只有将 ToolStrip 控件的 ShowItemToolTips 设置为 true 时，ToolStrip 控件中的 ToolStripButton 对象的 ToolTipText 属性才会生效。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MessageTool。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“具有提示功能的工具栏”，向窗体中添加一个 MenuStrip 控件，用于显示菜单信息；添加一个 ToolStrip 控件，用于显示带有提示功能的工具栏。

（3）为工具栏中的按钮添加提示信息非常简单，只需要设置工具栏按钮的 ToolTipText 属性即可。

秘笈心法

心法领悟 284：显示提示信息。

使用 ToolStripButton 对象的 ToolTipText 属性，可以方便地为 ToolStrip 工具栏中的按钮设置提示信息，要注意，只有将 ToolStrip 控件的 ShowItemToolTips 设置为 true 时，ToolStrip 控件中的 ToolStripButton 对象的 ToolTipText 属性才可以正常显示提示信息。

11.12　状态栏设计

实例 285	在状态栏中显示复选框	中级
	光盘位置：光盘\MR\11\285	趣味指数：★★★★☆

实例说明

设计程序界面时，为了规范界面，可以将一些控件放置在状态栏中，这样既能起到控制程序的作用，又能使界面和谐、美观。本实例实现了在状态栏中显示复选框的功能。实例运行效果如图 11.70 所示。

图 11.70　在状态栏中显示复选框

关键技术

本实例在状态栏中显示复选框时，只需要将 CheckBox 控件从工具箱中拖入状态拦即可。下面对 CheckBox 控件进行详细讲解。

CheckBox 控件，又称为复选框控件，它主要用来表示是否选取了某个选项条件，常用于为用户提供具有是/否或真/假值的选项，该控件的常用属性及说明如表 11.4 所示。

表 11.4　CheckBox 控件的常用属性及说明

属　性	说　明
Appearance	获取或设置确定 CheckBox 控件外观的值
AutoCheck	获取或设置一个值，该值指示当单击某一 CheckBox 时，Checked 或 CheckState 的值以及该 CheckBox 的外观是否自动改变
AutoEllipsis	获取或设置一个值，指示是否要在控件的右边缘显示省略号（…），以表示控件文本超出指定的控件长度
CheckAlign	获取或设置 CheckBox 控件上的复选框的水平和垂直对齐方式

续表

属　性	说　明
Checked	获取或设置一个值，该值指示 CheckBox 是否处于选中状态
CheckState	获取或设置 CheckBox 的状态
Text	获取或设置与此控件关联的文本
TextAlign	获取或设置 CheckBox 控件上的文本对齐方式
ThreeState	获取或设置一个值，该值指示此 CheckBox 是否允许 3 种复选状态而不是两种

CheckBox 控件的常用事件及说明如表 11.5 所示。

表 11.5　CheckBox 控件的常用事件及说明

事　件	说　明
CheckedChanged	当 Checked 属性的值更改时发生
CheckStateChanged	当 CheckState 属性的值更改时发生

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CheckBoxInStatusBar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 ToolStrip 控件，用来作为窗体的工具栏；添加一个 StatusStrip 控件，用来作为窗体的状态栏；添加一个 CheckBox 控件，用来在状态栏中进行显示。

（3）程序的主要代码如下：

```
private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
    if (this.checkBox2.Checked)                                    //当复选框 checkBox2 处于选中状态时
    {
        statusStrip1.Items[1].Text = "日期:" + DateTime.Now.ToString();   //在控件 statusStrip1 中显示系统当前日期
    }
    else                                                           //当复选框 checkBox2 处于未选中状态时
    {
        statusStrip1.Items[1].Text = "";                           //控件 statusStrip1 的内容设置为空
    }
}
```

秘笈心法

心法领悟 285：如何使窗体中的所有 CheckBox 控件处于取消选中状态？

使窗体中的所有 CheckBox 控件处于取消选中状态时，可以通过 foreach 遍历来实现。代码如下：

```
foreach (Control ctl in this.Controls)                             //遍历窗体中的控件
{
    if (ctl.GetType().Name == "CheckBox")                          //判断控件类型是否是 CheckBox
    {
        CheckBox cbox = (CheckBox)ctl;                             //将遍历到的控件强制转换为 CheckBox 类型
        cbox.Checked = false;                                      //设置复选框不选中
    }
}
```

实例 286　在状态栏中显示当前系统操作员

光盘位置：光盘\MR\11\286　　中级　　趣味指数：★★★☆

实例说明

在窗体应用程序开发中，经常会用到 StatusStrip 状态栏控件，在状态栏中可以显示当前应用程序的执行状

态。例如，Word 文档中的状态栏一般用于显示文档页数、焦点所在行列等信息。本实例中将演示在应用程序状态栏中显示当前系统操作员。实例运行效果如图 11.71 所示。

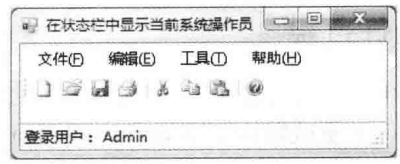


图 11.71　在状态栏中显示当前系统操作员

关键技术

本实例主要用到了 ToolStripStatusLabel 控件的 Text 属性，下面对其进行详细讲解。

ToolStripStatusLabel 控件的 Text 属性，用来获取或设置要显示在标签项上的文本。语法格式如下：

```
public virtual string Text { get; set; }
```

参数说明

属性值：字符串对象，表示标签项文本的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayUser。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于显示菜单信息；添加一个 ToolStrip 控件，用于显示工具栏信息；添加一个 StatusStrip 控件，用于显示当前系统操作员信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    Login P_l = new Login();                                    //创建窗体对象
    P_l.Owner = this;                                           //设置 Owner 属性
    P_l.ShowDialog();                                           //显示窗体
    toolStripStatusLabel1.Text = "登录用户：  "+user;            //设置用户登录信息
}
```

秘笈心法

心法领悟 286：设置状态栏文本颜色。

ToolStripStatusLabel 控件是状态栏的一个文本标签，使用 ToolStripStatusLabel 控件的 ForeColor 属性，可以轻松地设置状态栏中文本的颜色。

实例 287　在状态栏中实时显示当前系统时间

光盘位置：光盘\MR\11\287　　中级　　趣味指数：★★★★

实例说明

状态栏用于显示应用程序当前的状态信息或用户操作信息。本实例中将演示怎样在状态栏中显示当前系统时间，首先需要使用 Timer 组件间隔性地获取系统时间或创建一条线程获取系统时间，得到系统时间后在状态栏中显示。实例运行效果如图 11.72 所示。

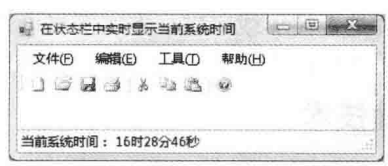


图 11.72　在状态栏中实时显示当前系统时间

关键技术

本实例主要用到了 DateTime 结构的 Now 属性，下面对其进行详细讲解。

DateTime 结构的 Now 属性，用来获取计算机上的当前日期和时间。语法格式如下：

```
public static DateTime Now { get; }
```

参数说明

属性值：DateTime 结构，当前系统日期和时间的 DateTime 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于显示菜单信息；添加一个 ToolStrip 控件，用于显示工具栏信息；添加一个 StatusStrip 控件，用于动态显示系统时间。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    Thread P_th = new Thread(                                          //创建线程对象
        () =>                                                          //使用 Lambda 表达式
        {
            while (true)                                               //无限循环
            {
                Invoke(                                                //调用窗体线程
                    (MethodInvoker)(() =>                              //使用 Lambda 表达式
                    {
                        toolStripStatusLabel1.Text =                   //设置状态栏文本内容
                            "当前系统时间：  "+ DateTime.Now.ToString("HH 时 mm 分 s 秒");
                    }));
                Thread.Sleep(1000);                                    //线程挂起 1 秒
            }
        });
    P_th.IsBackground = true;                                          //设置线程为后台线程
    P_th.Start();                                                      //线程开始
}
```

秘笈心法

心法领悟 287：在状态栏中动态显示系统时间。

在状态栏中动态显示系统时间的实现方法很简单，首先使用 Timer 组件或线程间隔性地得到系统时间，然后调用时间对象的 ToString 方法，得到系统时间的字符串表示，最后设置 ToolStripStatusLabel 控件的 Text 属性，在状态栏中显示系统时间。

实例 288　带进度条的状态栏

光盘位置：光盘\MR\11\288　　中级　　趣味指数：★★★☆

实例说明

我们经常使用 IE 浏览器浏览网页信息，在打开网页的过程中，浏览器下方会显示网页加载的进度，通过观察这个进度条，便可以了解网页是否被加载完成。本实例将设计一个带进度条的状态栏，并且在程序运行中动态显示进度。实例运行效果如图 11.73 所示。

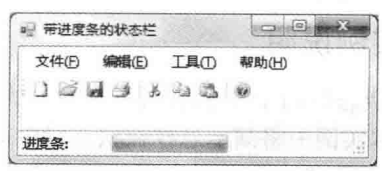


图 11.73　带进度条的状态栏

关键技术

本实例主要用到了 ToolStripProgressBar 控件的 Step 属性和 PerformStep 方法，下面分别对它们进行详细讲解。

（1）Step 属性

ToolStripProgressBar 控件的 Step 属性，用来获取或设置调用 PerformStep 方法时 ToolStripProgressBar 的当前值的增加量。语法格式如下：

```
public int Step { get; set; }
```

参数说明

属性值：整型数值，表示增加量的整数值。默认值为 10。

（2）PerformStep 方法

ToolStripProgressBar 控件的 PerformStep 方法，用于按照 Step 属性的数量增加进度栏的当前位置。语法格式如下：

```
public void PerformStep()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ProgressStatus。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“带进度条的状态栏”，向窗体中添加一个 MenuStrip 控件，用于显示菜单信息；添加一个 ToolStrip 控件，用于显示工具栏信息；添加一个 StatusStrip 控件，用于显示操作进度条。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    while (toolStripProgressBar1.Value < toolStripProgressBar1.Maximum)
    {
        this.toolStripProgressBar1.PerformStep();                    //增加进度条进度
    }
}
```

秘笈心法

心法领悟 288：操作进度条的另一种方法。

使用 ToolStripProgressBar 控件的 Value 属性，可以方便地设置进度条的位置。

实例 289　制作动画效果的状态栏

光盘位置：光盘\MR\11\289　　中级　　趣味指数：★★★☆

实例说明

在窗体应用程序开发中，可以使用 Timer 组件或线程在窗体中制作动画效果。本实例中将讲解如何在状态栏中显示动画效果。实例运行效果如图 11.74 所示。

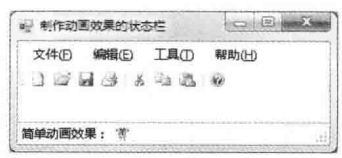


图 11.74　制作动画效果的状态栏

关键技术

本实例主要用到了 ToolStripStatusLabel 控件的 Image 属性，下面对其进行详细讲解。

ToolStripStatusLabel 控件的 Image 属性，用来获取或设置显示在 ToolStripStatusLabel 控件上的图像。语法格式如下：

```
public virtual Image Image { get; set; }
```

参数说明

属性值：Image 对象，控件中显示的 Image 图像。

说明：可以使用 Timer 组件或线程，间隔性地更改 ToolStripStatusLabel 控件的 Image 属性，使 ToolStripStatusLabel 控件中的图像具有动态效果。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DynamicStatus。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 MenuStrip 控件，用于显示菜单信息；添加一个 ToolStrip 控件，用于显示工具栏信息；添加一个 StatusStrip 控件，用于显示动画效果。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    int i=1;
    Thread P_th = new Thread(                                          //创建线程对象
        () =>                                                          //使用 Lambda 表达式
        {
            while (true)                                               //无限循环
            {
                Invoke(                                                //调用窗体线程
                    (MethodInvoker)(() =>                              //使用 Lambda 表达式
                    {
                        toolStripStatusLabel1.Image =                  //设置图像信息
                            Image.FromFile((++i > 8 ? (i=1) : i).ToString() + ".bmp");
                    }));
                Thread.Sleep(50);                                      //线程挂起 50 毫秒
            }
        });
    P_th.IsBackground = true;                                          //设置线程为后台线程
    P_th.Start();                                                      //线程开始
}
```

秘笈心法

心法领悟 289：适当地使用线程。

在窗体应用程序开发过程中，应当尽量避免使用窗体线程做高强度的运算或 I/O 操作，如果窗体线程参与了过多的运算，会导致用户的操作不能及时分配到资源，用户界面会出现卡或无响应情况，所以线程是一个不错的选择。

第12章

Windows 高级控件的使用

- 对话框组件的使用
- Timer 组件的使用
- EventLog 组件
- 其他常用控件的典型应用
- 常用控件控制技术
- 控件焦点变换

12.1 对话框组件的使用

实例 290 获得弹出对话框的相关返回值

光盘位置：光盘\MR\12\290　　中级　　趣味指数：★★★★

实例说明

在程序的开发过程中，为了使应用程序与用户充分交流，经常会使用消息对话框，在程序中的适当位置弹出消息对话框，可以帮助用户更好地了解应用程序的执行流程，使应用程序更加稳定地工作。实例运行效果如图 12.1 所示。

图 12.1 获得弹出对话框的相关返回值

关键技术

本实例主要用到了 MessageBox 类的 Show 方法和 Application 类的 Exit 方法，下面分别对它们进行详细讲解。

（1）Show 方法

MessageBox 类的 Show 方法，用于在窗体中弹出消息对话框，并根据用户的选择返回结果。语法格式如下：

```
public static DialogResult Show(IWin32Window owner,string text)
```

参数说明

❶ owner：窗体对象，表示弹出消息框的所有者窗体。

❷ text：字符串对象，用于指定在消息对话框中显示的文本内容。

❸ DialogResult：返回 DialogResult 枚举值，此枚举值用来判断用户选择的操作。

（2）Exit 方法

Application 类的 Exit 方法，用于关闭所有应用程序窗口。语法格式如下：

```
public static void Exit()
```

说明：使用 Application 类的 Exit 静态方法，可以方便地退出应用程序，与 Form 类的 Close 方法不同，Close 方法是关闭当前窗体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetResult。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于弹出消息对话框。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    DialogResult result = MessageBox.Show(this,                                    //弹出消息对话框
        "真的要关闭该窗口吗?", "关闭提示",MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
    if (result == DialogResult.OK)                                                 //判断是否单击“确定”按钮
    {
        Application.Exit();                                                        //关闭窗口
    }
}
```

秘笈心法

心法领悟 290：Show 方法的多个重载。

MessageBox 类的 Show 方法提供了多个重载，可以使用多个重载的方法方便地弹出带有标题、内容、按钮和图标等信息的消息对话框。

实例 291　使用 OpenFileDialog 组件打开文件

光盘位置：光盘\MR\12\291　　初级　　趣味指数：★★★★☆

实例说明

窗体应用程序运行中，经常需要用户手动选择文件，并在后面的工作中处理该文件。使用 OpenFileDialog 组件的 ShowDialog 方法，可以方便地弹出打开文件对话框，用户可以在该对话框中选择将要打开的文件。实例运行效果如图 12.2 所示。

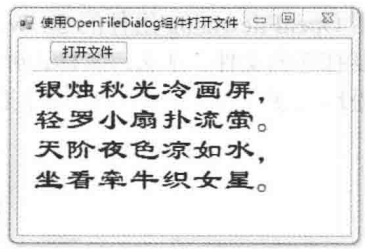


图 12.2　使用 OpenFileDialog 组件打开文件

关键技术

本实例主要用到了 OpenFileDialog 组件的 ShowDialog 方法和 FileName 属性，下面分别对它们进行详细讲解。

（1）ShowDialog 方法

OpenFileDialog 组件的 ShowDialog 方法，用于弹出打开文件对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举值，如果用户在对话框中单击“确定”按钮，则返回 DialogResult.OK，否则返回 DialogResult.Cancel。

（2）FileName 属性

OpenFileDialog 组件的 FileName 属性，用于获取被弹出的打开文件对话框中被选中的文件名的字符串。语法格式如下：

```
public string FileName { get; set; }
```

参数说明

属性值：字符串对象，文件对话框中选择的文件名，默认值为空字符串 string.Empty。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OpenFile。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于打开文件；添加一个 TextBox 控件，用于显示文件信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "txt 文件(*.txt)|*.txt";                          //筛选文件
    if (openFileDialog1.ShowDialog() == DialogResult.OK)                       //弹出打开文件对话框
    {
        System.IO.StreamReader sr = new                                        //创建文件读取器对象
            System.IO.StreamReader(openFileDialog1.FileName,Encoding.Default);
        textBox1.Text = sr.ReadToEnd();                                        //显示文本文件内容
        sr.Close();                                                            //关闭流
    }
}
```

秘笈心法

心法领悟 291：打开并读取文本文件所有内容。

使用 OpenFileDialog 组件的 ShowDialog 方法可以方便地弹出打开文件对话框，如果打开一个文本文件，那么，怎样读取文本文件的内容呢？可以使用 File 类的 ReadAllText 方法轻松地实现。

实例 292 设置 OpenFileDialog 组件中只能选择图片文件

初级

光盘位置：光盘\MR\12\292

趣味指数：★★★☆

实例说明

使用 OpenFileDialog 组件的 ShowDialog 方法，可以方便地弹出打开文件对话框，用户可以在文件对话框中选择将要打开的文件，那么，怎样让弹出的文件对话框只能选择图片文件呢？可以通过设置文件对话框的 Filter 属性轻松地实现此功能。实例运行效果如图 12.3 所示。

图 12.3　设置 OpenFileDialog 组件中只能选择图片文件

关键技术

本实例主要用到了 OpenFileDialog 组件的 Filter 属性和 Image 类的 FromFile 方法，下面分别对它们进行详细讲解。

（1）Filter 属性

OpenFileDialog 组件的 Filter 属性，用于设置筛选字符串，该字符串确定在弹出的打开文件对话框中显示的文件类型。语法格式如下：

```
public string Filter { get; set; }
```

参数说明

属性值：字符串对象，默认值为 string.Empty，表示不筛选任何文件类型。

（2）FromFile 方法

Image 类的 FromFile 方法，用于从指定的文件创建 Image 对象。语法格式如下：

```
public static Image FromFile(string filename)
```

参数说明

❶ filename：字符串对象，表示将要创建的 Image 对象的文件的名称。

❷ Image：Image 对象，表示此方法创建的 Image 对象。

说明：使用 Image 类的 FromFile 方法可以得到 Image 对象，同样，也可以使用 BitMap 构造方法得到 Image 对象，如 Image P_image = new Bitmap("图像文件位置")，由于 BitMap 类继承于 Image 类，所以 Bitmap 对象可以隐式转换为 Image 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OnlyPictures。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择图像文件；添加一个 PictureBox 控件，用于显示图像信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "*.jpg|*.jpg|*.bmp|*.bmp";                    //设置筛选字符串
    if (openFileDialog1.ShowDialog() == DialogResult.OK)                    //弹出打开文件对话框
    {
        pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);       //显示图像
    }
}
```

秘笈心法

心法领悟 292：筛选多种文件。

OpenFileDialog 组件的 Filter 属性可以用来筛选文件。例如，Filter 属性等于“*.txt|*.txt|*.jpg|*.jpg”字符串，使用 OpenFileDialog 组件打开文件时，将会显示扩展名为.txt 和.jpg 的文件。

实例 293　使用 OpenFileDialog 组件打开多个文件

光盘位置：光盘\MR\12\293　　中级　　趣味指数：★★★★

实例说明

使用 OpenFileDialog 组件的 ShowDialog 方法，可以方便地弹出打开文件对话框，用户可以在该对话框中选择将要打开的文件，那么，怎样才可以在弹出的打开文件对话框中选择多个文件呢？可以通过设置打开文件对话框的 Multiselect 属性轻松地实现此功能。实例运行效果如图 12.4 所示。

图 12.4　使用 OpenFileDialog 组件打开多个文件

关键技术

本实例主要用到了 OpenFileDialog 组件的 Multiselect 属性和 FileNames 属性，下面分别对它们进行详细讲解。

（1）Multiselect 属性

OpenFileDialog 组件的 Multiselect 属性，用于设置当前的打开文件对话框是否可以选择多个文件。语法格式如下：

```
public bool Multiselect { get; set; }
```

参数说明

属性值：布尔值，如果对话框允许同时选定多个文件，则值为 true；如果只允许选定单个文件，则值为 false。

（2）FileNames 属性

OpenFileDialog 组件的 FileNames 属性，用于获取打开文件对话框中所有选定文件的文件名。语法格式如下：

```
public string[] FileNames { get; }
```

参数说明

属性值：字符串数组，包含打开文件对话框中所有选定文件的文件名的字符串类型的数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OpenMulti。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择多个文件；添加一个 TextBox 控件，用于显示多个文件信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "txt 文件(*.txt)|*.txt";                    //筛选文件
    openFileDialog1.Multiselect = true;                                  //允许选中多个文件
    if (openFileDialog1.ShowDialog() == DialogResult.OK)                 //判断是否选中文件
    {
        string Content = string.Empty;
        foreach (string filename in openFileDialog1.FileNames)
        {
            System.IO.StreamReader sr = new                              //创建流读取器对象
                System.IO.StreamReader(filename,Encoding.Default);
            Content += sr.ReadToEnd();                                   //读取文件内容
            sr.Close();                                                  //关闭流
        }
        textBox1.Text = Content;                                         //显示文件内容
    }
}
```

秘笈心法

心法领悟 293：怎样打开多个文件？

通过将 OpenFileDialog 组件的 Multiselect 属性设置为 true，可以在弹出的打开文件对话框中选择多个文件，在 OpenFileDialog 组件的 FileNames 属性中保存了多个被选择文件路径的字符串数组，可以使用 for 或 foreach 遍历数组中的每一个字符串，从而得到被选中文件的路径信息，最后使用 FileStream 打开并处理每一个文件。

实例 294　使用 SaveFileDialog 组件保存文件

光盘位置：光盘\MR\12\294　　初级　　趣味指数：★★★★☆

实例说明

在使用 Word 应用程序时，可以单击工具栏中的“新建空白文档”按钮，新建一个 Word 文档，当单击“保存”或“另存为”按钮时，会弹出保存文件对话框，用户可以使用此对话框方便地保存文件，在 C#中可以使用 SaveFileDialog 组件的 ShowDialog 方法弹出保存文件对话框。实例运行效果如图 12.5 所示。

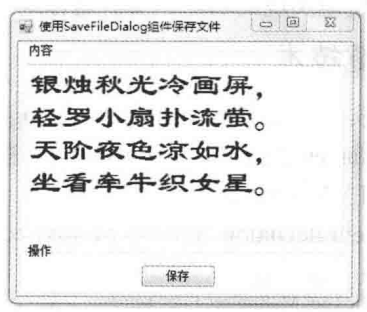


图 12.5　使用 SaveFileDialog 组件保存文件

关键技术

本实例主要用到了 SaveFileDialog 组件的 ShowDialog 方法和 Form 类的 Close 方法，下面分别对它们进行详细讲解。

（1）ShowDialog 方法

SaveFileDialog 组件的 ShowDialog 方法，用于弹出保存文件对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：该方法返回 DialogResult 枚举，如果用户确定保存文件则值为 DialogResult.OK，否则为 DialogResult.Cancel。

（2）Close 方法

Form 类的 Close 方法，用于关闭窗体。语法格式如下：

```
public void Close()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SaveToFile。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于保存文件；添加一个 TextBox 控件，用于用户输入文本信息。

（3）程序的主要代码如下：

```
private void btn_Save_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "*.txt|*.txt";                          //保存文件类型
    if (saveFileDialog1.ShowDialog()==DialogResult.OK)               //判断是否选定文件
    {
        StreamWriter sw = new StreamWriter( saveFileDialog1.FileName);  //创建文件写入器
        sw.Write(txt_Message.Text);                                  //向文件中写入文本信息
        sw.Close();                                                  //关闭文件流
    }
}
```

秘笈心法

心法领悟 294：向现有文本文件中追加文本内容。

StreamWriter 实例构造器中提供了多个重载，可以使用 StreamWriter(string path,bool append)构造器创建流写入器对象，并将构造方法中的 append 参数设置为 true，在创建对象后，可以使用该对象的 Write 方法方便地向文本文件中追加文本信息。

实例 295　使用 FolderBrowserDialog 组件选择文件夹

初级

光盘位置：光盘\MR\12\295　　趣味指数：★★★☆

实例说明

在程序的执行过程中，经常需要对文件夹或文件夹内的文件进行操作，怎样才可以方便地选中指定的文件夹呢？使用 FolderBrowserDialog 组件的 ShowDialog 方法，可以方便地弹出浏览文件夹对话框，用户可以在该对话框中选择指定的文件夹。实例运行效果如图 12.6 所示。

图 12.6　使用 FolderBrowserDialog 组件选择文件夹

关键技术

本实例主要用到了 FolderBrowserDialog 组件的 ShowDialog 方法和 SelectedPath 属性，下面分别对它们进行详细讲解。

（1）ShowDialog 方法

FolderBrowserDialog 组件的 ShowDialog 方法，用于弹出选择路径对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举，如果用户选择了路径信息，则值为 DialogResult.OK，否则为 DialogResult.Cancel。

（2）SelectedPath 属性

FolderBrowserDialog 控件的 SelectedPath 属性，用于获取或设置用户选定的路径。语法格式如下：

```
public string SelectedPath { get; set; }
```

参数说明

属性值：字符串对象，对话框中选定文件夹的路径。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SelectFolder。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择文件夹；添加一个 TextBox 控件，用于显示文件夹路径信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)          //确定是否已经选择文件夹
    {
        textBox1.Text = folderBrowserDialog1.SelectedPath;             //显示文件夹路径
    }
}
```

秘笈心法

心法领悟 295：判断是否已经选中文件夹。

FolderBrowserDialog 组件的 ShowDialog 方法会返回一个 DialogResult 枚举值，如果用户在弹出的路径对话框中选择了路径信息，则该值为 DialogResult.OK，否则该值为 DialogResult.Cancel。

实例 296　设置 FolderBrowserDialog 组件的默认路径

光盘位置：光盘\MR\12\296　　中级　　趣味指数：★★★★

实例说明

使用 FolderBrowserDialog 组件的 ShowDialog 方法，可以方便地弹出浏览文件夹对话框，用户可以在该对话框中选择指定的文件夹。本实例中使用了 FolderBrowserDialog 组件的 SelectedPath 属性设置该组件打开文件夹的默认路径。实例运行效果如图 12.7 所示。

图 12.7　设置 FolderBrowserDialog 组件的默认路径

关键技术

本实例主要用到了 FolderBrowserDialog 组件的 SelectedPath 属性，下面对其进行详细讲解。

FolderBrowserDialog 组件的 SelectedPath 属性，用于获取或设置用户选定的路径。语法格式如下：

```
public string SelectedPath { get; set; }
```

参数说明

属性值：字符串对象，对话框中选定文件夹的路径。

说明：SelectedPath 属性一般用于获取或设置用户选定的路径，当 SelectedPath 属性被赋值后，再一次弹出选择路径对话框时，默认路径会从 SelectedPath 属性中的路径打开，所以设置了 FolderBrowserDialog 组件的 SelectedPath 属性就等于设置了 FolderBrowserDialog 组件的默认路径。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DefaultSelectFolder。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择文件夹；

添加一个 TextBox 控件，用于显示文件夹路径信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    folderBrowserDialog1.SelectedPath = @"D:\";                    //设置选定的路径
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)      //确定是否已经选择文件夹
    {
        textBox1.Text = folderBrowserDialog1.SelectedPath;         //显示文件夹路径
    }
}
```

秘笈心法

心法领悟 296：怎样操作文件夹？

实例中已经详细地介绍了怎样使用 FolderBrowserDialog 组件的 SelectedPath 方法方便地选择路径信息，那么，怎样操作文件夹呢？Directory 类提供的多个静态方法可以满足上述需求，可以分别使用 Directory 类中的 CreateDirectory 方法创建文件夹，使用 Delete 方法删除文件夹，使用 Exists 方法判断指定的文件夹是否存在。

实例 297　使用 FontDialog 组件设置字体

光盘位置：光盘\MR\12\297　　中级　　趣味指数：★★★★

实例说明

在 Windows 窗体编程中，经常需要更改文本框中或控件中文本的字体，怎样友好地实现更改字体呢？本实例中使用了 FontDialog 组件的 ShowDialog 方法，此方法执行后，会弹出字体选择对话框，用户可以在此对话框中方便地设置文本字体。实例运行效果如图 12.8 所示。

图 12.8　使用 FontDialog 组件设置字体

关键技术

本实例主要用到了 FontDialog 组件的 ShowDialog 方法和 Font 属性，下面分别对它们进行详细讲解。

（1）ShowDialog 方法

FontDialog 组件的 ShowDialog 方法，用于弹出字体选择对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举，如果用户选择了字体信息，则值为 DialogResult.OK，否则为 DialogResult.Cancel。

（2）Font 属性

FontDialog 组件的 Font 属性，用于获取或设置选定的字体。语法格式如下：

```
public Font Font { get; set; }
```

参数说明

属性值：Font 对象，表示选定的字体。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetFont。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择字体信息；添加一个 TextBox 控件，用于显示文本信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.fontDialog1.ShowDialog();                                    //弹出字体选择对话框
    this.textBox1.Font = this.fontDialog1.Font;                       //设置文字字体
}
```

秘笈心法

心法领悟 297：更改文本框中文本的字体。

通过文本框的 Font 属性可以设置文本框中文本的字体，Font 属性值是一个 Font 对象，可以使用 FontDialog 组件弹出字体选择对话框，按照用户的选择设置 Font 对象，最后将此 Font 对象交给文本框的 Font 属性。

实例 298　使用 ColorDialog 组件设置字颜色

光盘位置：光盘\MR\12\298　　初级　　趣味指数：★★★★

实例说明

窗体应用程序执行过程中，会经常更改文本框内文本的颜色或控件中文本的颜色，怎样友好地实现此功能呢？本实例中使用了 ColorDialog 组件的 ShowDialog 方法，此方法执行后，会弹出颜色选择对话框，用户可以在此对话框中方便地设置颜色信息。实例运行效果如图 12.9 所示。

图 12.9　使用 ColorDialog 组件设置字颜色

关键技术

本实例主要用到了 ColorDialog 组件的 ShowDialog 方法和 Color 属性，下面分别对它们进行详细讲解。

（1）ShowDialog 方法

ColorDialog 组件的 ShowDialog 方法，用于弹出颜色选择对话框。语法格式如下：

```
public DialogResult ShowDialog()
```

参数说明

DialogResult：返回 DialogResult 枚举，如果用户选择了颜色信息，则值为 DialogResult.OK，否则为 DialogResult.Cancel。

（2）Color 属性

ColorDialog 组件的 Color 属性，用于获取或设置用户选定的颜色。语法格式如下：

```
public Color Color { get; set; }
```

参数说明

属性值：Color 结构，表示用户选定的颜色，如果没有选定颜色，则默认值为黑色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetColor。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于选择颜色信息；添加一个 TextBox 控件，用于显示文本信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    colorDialog1.ShowDialog();                                    //弹出颜色选择对话框
    textBox1.ForeColor = colorDialog1.Color;                      //设置文字颜色
}
```

秘笈心法

心法领悟 298：更改文本框中文本的颜色。

通过文本框的 ForeColor 属性可以设置文本框中文本的颜色，ForeColor 属性值是一个 Color 结构，可以使用 ColorDialog 组件的 ShowDialog 方法弹出颜色选择对话框，可在其中按照用户的选择设置颜色信息，最后将此 Color 交给文本框的 ForeColor 属性。

12.2　Timer 组件的使用

实例 299	使用 Timer 组件制作计时器	中级
	光盘位置：光盘\MR\12\299	趣味指数：★★★★

实例说明

使用 Timer 组件，可以按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，本实例中使用 Timer 组件制作了一个简单的计时器。实例运行效果如图 12.10 所示。

图 12.10　使用 Timer 组件制作计时器

关键技术

本实例主要用到了 Timer 组件的 Interval 属性和 NumericUpDown 控件的 Value 属性，下面分别对它们进行详细讲解。

（1）Interval 属性

Timer 组件的 Interval 属性，用于获取或设置 Timer 组件 Tick 事件发生的时间间隔。语法格式如下：

```
public int Interval { get; set; }
```

参数说明

属性值：数值类型，表示时间间隔毫秒数。

提示：Timer 组件的 Interval 属性值不能小于 1。

（2）Value 属性

NumericUpDown 控件的 Value 属性，用于获取或设置数字显示框的值。语法格式如下：

```
public decimal Value { get; set; }
```

参数说明

属性值：Decimal 数值，表示 NumericUpDown 控件中的数值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Calculagraph。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加 3 个 NumericUpDown 控件，用于设置定时信息；添加两个 Button 控件，分别用于确定和取消定时信息。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    DateTime get_time1 = DateTime.Now;                                          //得到系统当前时间
    DateTime sta_ontime1 = Convert.ToDateTime(                                  //获取定时信息
        Convert.ToDateTime(textBox2.Text.Trim().ToString()));
    long dat = DateAndTime.DateDiff(                                            //计算两个时间间隔的秒数
        "s", get_time1, sta_ontime1,FirstDayOfWeek.Sunday,FirstWeekOfYear.FirstFourDays);
    if (dat > 0)                                                                //如果时间间隔大于 0 秒
    {
        if (timer2.Enabled != true)
        {
            timer2.Enabled = true;                                              //开始计时
            label4.Text = "闹钟已启动";                                          //显示操作信息
            label1.Text = "剩余" + dat.ToString() + "秒";                        //显示剩余时间
        }
        else
        {
            MessageBox.Show("时钟已经启动，请取消后，再启动");                    //弹出消息对话框
        }
    }
    else
    {
        long hour = 24 * 3600 + dat;                                            //计算到下一天时间的秒数
        timer2.Enabled = true;                                                  //开始计时
        label4.Text = "闹钟已启动";                                              //显示操作信息
        label1.Text = "剩余" + hour.ToString() + "秒";                           //显示剩余时间
    }
}
```

秘笈心法

心法领悟 299：启用计时器。

由于 Timer 组件的 Enable 属性默认值为 false，所以计时器在默认情况下是未启动的，可以通过在代码中更改 Enable 属性值为 true 或调用 Timer 组件的 Start 方法来启动计时器。

实例 300　使用 Timer 组件显示当前系统时间

光盘位置：光盘\MR\12\300　　中级　　趣味指数：★★★★☆

实例说明

使用 DateTime 结构的 Now 属性，可以得到系统的时间对象，调用时间对象的 ToString 方法会返回系统时间的字符串表示，本实例中使用了 Timer 组件，按照指定的时间间隔获取系统时间，并在窗体中动态显示此时间。实例运行效果如图 12.11 所示。

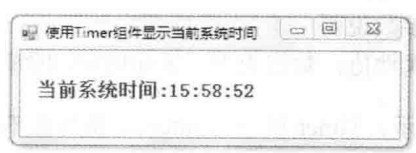


图 12.11　使用 Timer 组件显示当前系统时间

关键技术

本实例主要用到了 Label 控件的 Text 属性和 DateTime 结构的 ToLongTimeString 方法，下面分别对它们进行详细讲解。

（1）Text 属性

Label 控件的 Text 属性，用于获取或设置标签中的文本。语法格式如下：

```
public override string Text { get; set; }
```

参数说明

属性值：字符串对象，表示标签中的文本。

（2）ToLongTimeString 方法

DateTime 结构的 ToLongTimeString 方法，用于将时间对象的值转换为其等效的长时间字符串表示形式。语法格式如下：

```
public string ToLongTimeString()
```

参数说明

string：返回字符串对象，表示时间对象的长时间字符串表示形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayTime。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Label 标签，用于显示当前系统时间；添加一个 Timer 组件，用于间隔性地获取并显示系统时间。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = "当前系统时间:" + DateTime.Now.ToLongTimeString();                    //显示系统时间
}
```

秘笈心法

心法领悟 300：怎样获取系统时间？

本实例中详细地介绍了怎样在窗体中显示系统时间，可以通过 DateTime 结构的 Now 方法方便地获取系统时间，得到系统时间后，使用 ToLongTimeString 方法将时间转换为字符串对象，最后使用 Label 标签在窗体中显示系统时间信息。

实例 301　使用 Timer 组件制作左右飘动的窗体

光盘位置：光盘\MR\12\301　　中级　　趣味指数：★★★★☆

实例说明

通过窗体的 Left 和 Top 属性，可以方便地设置窗体在桌面上的位置，Left 属性用于标记窗体在桌面的横向坐标，Top 属性用于标记窗体在桌面的纵向坐标。本实例中使用了 Timer 组件，按照指定的时间间隔更改窗体的 Left 和 Top 属性，实现了在桌面中左右飘动的窗体。实例运行效果如图 12.12 所示。

图 12.12　使用 Timer 组件制作左右飘动的窗体

关键技术

本实例主要用到了 Form 类的 Left 和 Top 属性，下面分别对它们进行详细讲解。

（1）Left 属性

Form 类的 Left 属性，用于获取或设置窗体左边缘与桌面左边缘的距离。语法格式如下：

```
public int Left { get; set; }
```

参数说明

属性值：整型数值，表示窗体左边缘与桌面左边缘的距离。

（2）Top 属性

Form 类的 Top 属性，用于获取或设置窗体上边缘与桌面上边缘之间的距离。语法格式如下：

```
public int Top { get; Seti }
```

参数说明

属性值：整型数值，表示窗体上边缘与桌面上边缘的距离。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MoveForm。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加两个 Timer 控件，分别用于控制窗体横纵坐标的移动。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    Rectangle rect = Screen.GetWorkingArea(this);                    //获取屏幕大小
    if (this.Left != (rect.Width - this.Width))
    {
        this.Left++;                                                 //窗体向右移动
        this.Top += 1;                                               //窗体向下移动
    }
    else
    {
        timer1.Enabled = false;                                      //停用 Timer 组件
        timer2.Enabled = true;                                       //启用 Timer 组件
    }
}
private void timer2_Tick(object sender, EventArgs e)
{
    Rectangle rect = Screen.GetWorkingArea(this);                    //获取屏幕大小
    if (this.Left == 0)
    {
        timer2.Enabled = false;                                      //停用 Timer 组件
        timer1.Enabled = true;                                       //启用 Timer 组件
    }
    else
    {
        this.Left--;                                                 //窗体向左移动
        this.Top -= 1;                                               //窗体向上移动
    }
}
```

秘笈心法

心法领悟 301：设置窗体在桌面的位置。

通过窗体对象的 Left 和 Top 属性可以方便地设置窗体在桌面上的位置，Left 属性用于标记窗体在桌面的横向坐标，Top 属性用于标记窗体在桌面的纵向坐标。

实例 302　使用 Timer 组件实现世界杯倒计时

高级

光盘位置：光盘\MR\12\302

趣味指数：★★★★★

实例说明

使用 Timer 组件，可以按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，本实例中使用 Timer 组件实现了世界杯倒计时功能。实例运行效果如图 12.13 所示。

图 12.13　使用 Timer 组件实现世界杯倒计时

关键技术

本实例主要用到了 Timer 组件的 Enabled 属性，下面对其进行详细讲解。

Timer 组件的 Enabled 属性，用于获取或设置计时器是否正在运行。语法格式如下：

```
public Virtual bool Enabled { get; set; }
```

参数说明

属性值：布尔值，如果计时器当前处于启用状态则 Enable 值为 true，否则值为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TimeNow。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加 8 个 TextBox 文本框控件，分别用于显示世界杯倒计时信息；添加一个 Timer 组件，用于间隔性地计算倒计时信息。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    DateTime get_time1 = DateTime.Now;                                          //得到当前系统时间
    DateTime sta_ontime1 = Convert.ToDateTime(                                  //得到世界杯开幕时间
        Convert.ToDateTime("2014-10-13 00:00:00"));
    txtYear.Text = DateAndTime.DateDiff(                                        //计算相隔年数
        "yyyy", get_time1, sta_ontime1,FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    txtMonth.Text = DateAndTime.DateDiff(                                       //计算相隔月数
        "m", get_time1, sta_ontime1, FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    textday.Text = DateAndTime.DateDiff(                                        //计算相隔天数
        "d", get_time1, sta_ontime1, FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    txtHour.Text = DateAndTime.DateDiff(                                        //计算相隔小时数
        "h", get_time1, sta_ontime1, FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    txtmintue.Text = DateAndTime.DateDiff(                                      //计算相隔分钟数
        "n", get_time1, sta_ontime1, FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    txtsecon.Text = DateAndTime.DateDiff(                                       //计算相隔秒数
        "s", get_time1, sta_ontime1, FirstDayOfWeek.Sunday, FirstWeekOfYear.FirstFourDays).ToString();
    textBox1.Text = DateTime.Now.ToString();
}
```

秘笈心法

心法领悟 302：使用 DateAndTime 类的 DateDiff 方法计算时间间隔。

由于 DateAndTime 类定义在 Visual Basic 的程序集中，所以，在使用 DateAndTime 类前，首先要引用 Visual Basic 程序集，引用 Visual Basic 程序集并添加 Microsoft.VisualBasic 命名空间后，即可方便地使用 DateAndTime 类的 DateDiff 方法。

实例 303　使用 Timer 组件实现人物动画效果

高级

光盘位置：光盘\MR\12\303

趣味指数：★★★★★

实例说明

在窗体中实现动画效果是一个非常不错的想法，使用 Timer 组件可以按用户定义的时间间隔来引发事件，

引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，本实例中使用 Timer 组件实现了人物动画效果。实例运行效果如图 12.14 所示。

图 12.14　使用 Timer 组件实现人物动画效果

关键技术

本实例主要用到了 Graphics 类的 DrawImage 方法，下面对其进行详细讲解。

Graphics 类的 DrawImage 方法用于在指定位置，并且按原始大小绘制指定的 Image 对象。语法格式如下：

```
public void DrawImage(Image image,Point point)
```

参数说明

❶ image：Image 对象，表示将要绘制的 Image 对象。

❷ point：Point 结构，表示所绘制图像的左上角的位置。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseTimer。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Timer 组件，用于间隔性地向窗体中绘制图片信息。

（3）程序的主要代码如下：

```
private void tmr_Action_Tick(object sender, EventArgs e)
{
    CreateGraphics().DrawImage(                                                    //在窗体中绘制图片信息
        Image.FromFile( (index++ > 7 ? (index = 1) : index).ToString() + ".jpg"), new Point(0, 0));
}
```

秘笈心法

心法领悟 303：在窗体中使用动画效果。

在窗体中使用动画效果非常简单，通过 Graphics 对象的 DrawImage 方法可以在特定的上下文内容中绘制图像信息。使用 Timer 组件的 Tick 事件间隔性地调用 DrawImage 方法，可以在窗体中绘制不同的图片，从而轻松地实现动画效果。

12.3　EventLog 组件

实例 304	使用 EventLog 组件读写事件日志	高级
	光盘位置：光盘\MR\12\304	趣味指数：★★★★★

实例说明

本实例中使用 EventLog 组件，创建一个日志并向日志中写入内容和读取日志中的内容。首先，使用 EventLog 类的 SourceExists 静态方法，判断本机是否存在指定的事件源，然后，使用 CreateEventSource 静态方法创建日

志信息，最后，使用 EventLog 组件的 WriteEntry 方法写入日志信息。实例运行效果如图 12.15 所示。

图 12.15　使用 EventLog 组件读写事件日志

关键技术

本实例主要用到了 EventLog 类的 SourceExists 方法和 DeleteEventSource 方法，下面分别对它们进行详细讲解。

（1）SourceExists 方法

EventLog 类的 SourceExists 方法，用于确定事件源是否已在本地计算机上注册。语法格式如下：

```
public static bool SourceExists(string source)
```

参数说明

❶ source：字符串对象，表示事件源的名称。

❷ bool：返回布尔值，如果事件源已在本地计算机上注册，则值为 true，否则值为 false。

（2）DeleteEventSource 方法

EventLog 类的 DeleteEventSource 方法，用于从事件日志中移除应用程序的事件源注册。语法格式如下：

```
public static void DeleteEventSource(string source)
```

参数说明

source：字符串对象，应用程序在事件日志系统中用来注册的名称。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetEventLog。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加两个 Button 控件，用于读取和写入日志信息；添加一个 TextBox 文本框控件，用于用户输入日志信息；添加一个 ListBox 控件，用于显示日志信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    if (System.Diagnostics.EventLog.SourceExists("ZhyScoure"))                    //判断是否存在事件源
    {
        System.Diagnostics.EventLog.DeleteEventSource("ZhyScoure");               //删除事件源注册
    }
    System.Diagnostics.EventLog.CreateEventSource("ZhyScoure", "NewLog1");        //创建日志信息
    eventLog1.Log = "NewLog1";                                                    //设置日志名称
    eventLog1.Source = "ZhyScoure";                                               //事件源名称
    this.eventLog1.MachineName = ".";                                             //表示本机
}
private void btn_Write_Click(object sender, EventArgs e)
{
    if (System.Diagnostics.EventLog.Exists("NewLog1"))                            //判断日志是否存在
    {
        if (textBox1.Text != "")                                                  //如果文本框为空
        {
            eventLog1.WriteEntry(textBox1.Text.ToString());                       //写入日志
            MessageBox.Show("日志写入成功");                                      //弹出消息对话框
            textBox1.Text = "";                                                   //清空文本框信息
        }
        else
        {
            MessageBox.Show("日志内容不能为空");                                  //弹出消息对话框
        }
    }
    else
    {
        MessageBox.Show("日志不存在");                                            //弹出消息对话框
    }
}
```

秘笈心法

心法领悟 304：适当地使用 MessageBox 类的 Show 方法。

实例中使用了 WriteEntry 方法向指定的日志中写入信息，当写入信息后，会调用 MessageBox 类的 Show 方法弹出消息对话框，提示用户日志写入成功。在程序中的适当位置弹出消息对话框，可以帮助用户更好地了解应用程序的执行流程，使应用程序更加稳定地工作。

实例 305　使用 EventLog 组件保存 Windows 系统日志

高级

光盘位置：光盘\MR\12\305

趣味指数：★★★★★

实例说明

程序运行过程中，如果出现了未知的错误，程序将会抛出异常，可以使用 try catch 捕获异常，然后对异常信息进行处理，在处理异常信息的同时，建议将异常信息保存到自定义的日志文件中。本实例中使用 EventLog 组件，查找系统日志中的错误信息并把这些信息显示出来。实例运行效果如图 12.16 所示。

图 12.16　使用 EventLog 组件保存 Windows 系统日志

关键技术

本实例主要用到了 EventLog 类的 CreateEventSource 方法和 EventLog 组件的 Log 属性和 Source 属性，下面分别对它们进行详细讲解。

（1）CreateEventSource 方法

EventLog 类的 CreateEventSource 方法，用于建立一个应用程序，使用指定的 Source 作为向本地计算机上的日志中写入日志项的有效事件源。该方法还可在本地计算机上创建一个新的自定义日志。语法格式如下：

```
public static void CreateEventSource(string source,string logName)
```

参数说明

❶ source：字符串对象，表示应用程序在本地计算机上注册时所采用的源名称。

❷ logName：字符串对象，表示源的项写入的日志名。

（2）Log 属性

EventLog 组件的 Log 属性，用于获取或设置读取或写入的日志名称。语法格式如下：

```
public string Log { get; set; }
```

参数说明

属性值：字符串对象，表示日志的名称。

（3）Source 属性

EventLog 组件的 Source 属性，用于获取或设置在写入事件日志时要注册和使用的源名称。语法格式如下：

```
public string Source { get; set; }
```

参数说明

属性值：字符串对象，表示在事件日志中注册为源的名称。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SaveLog。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 ListBox 控件，用于显示错误日志信息；添加一个 Button 控件，用于查找并保存系统错误日志信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    if (System.Diagnostics.EventLog.SourceExists("ErrEventLog"))                    //判断是否存在事件源
    {
        System.Diagnostics.EventLog.DeleteEventSource("ErrEventLog");               //删除事件源注册
    }
    System.Diagnostics.EventLog.CreateEventSource("ErrEventLog", "Application");    //创建日志信息
    eventLog2.Log = "Application";                                                   //设置日志名称
    eventLog2.Source = "ErrEventLog";                                                //事件源名称
    this.eventLog1.MachineName = ".";                                                //表示本机
}
private void btn_Find_Click(object sender, EventArgs e)
{
    if (eventLog1.Entries.Count > 0)                                                 //判断是否存在系统日志
    {
        foreach (System.Diagnostics.EventLogEntry entry in eventLog1.Entries)       //遍历日志信息
        {
            if (entry.EntryType == System.Diagnostics.EventLogEntryType.Error)      //判断是否为错误日志
            {
                listBox1.Items.Add(entry.Message);                                   //向控件中添加数据项
                eventLog2.WriteEntry(entry.Message,                                  //写入日志信息
                    System.Diagnostics.EventLogEntryType.Error);
            }
        }
    }
    else
    {
        MessageBox.Show("系统没有错误日志.");                                        //弹出消息对话框
    }
}
```

秘笈心法

心法领悟 305：向 ListBox 控件中添加数据。

本实例中使用了 ListBox 控件显示错误日志信息，通过 ListBox 控件中 Items 集合的 Add 方法，可以方便地向 ListBox 中添加数据，Add 方法接收一个 object 对象作为参数，由于所有类型都继承于 object，这意味着所有对象都可以放入 ListBox 控件的数据项集合中，当 ListBox 显示数据项的信息时，将会调用每一个数据项的 ToString 方法，获取并显示每一个数据项的字符串。

实例 306　使用 EventLog 组件向本机现有日志中添加条目

光盘位置：光盘\MR\12\306　　高级　　趣味指数：★★★★★

实例说明

实例中使用了 EventLog 组件，向本机现有日志中添加信息。通过 EventLog 组件的 Log 属性设置写入的日志名称，通过 EventLog 组件的 WriteEntry 方法向日志中写入信息。实例运行效果如图 12.17 所示。

图 12.17　使用 EventLog 组件向本机现有日志中添加条目

关键技术

本实例主要用到了 EventLog 组件的 MachineName 属性和 WriteEntry 方法，下面分别对它们进行详细讲解。

（1）MachineName 属性

EventLog 组件的 MachineName 属性，用于获取或设置在其上读取或写入事件的计算机名称。语法格式如下：

```
public string MachineName { get; set; }
```

参数说明

属性值：字符串对象，表示事件日志驻留的服务器名称，默认为本地计算机。

（2）WriteEntry 方法

EventLog 组件的 WriteEntry 方法，用于将信息类型项与给定的消息文本一起写入事件日志。语法格式如下：

```
public void WriteEntry(string message)
```

参数说明

message：字符串对象，表示要写入事件日志的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddLog。

（2）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，向窗体中添加一个 Button 控件，用于添加日志信息；添加一个 ComboBox 控件，用于选择日志源名称；添加一个 TextBox 控件，用于用户输入日志信息；添加一个 ListView 控件，用于显示日志信息。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    if (eventLog1.Entries.Count > 0)
    {
        foreach (System.Diagnostics.EventLogEntry entry in eventLog1.Entries)      //遍历所有日志
        {
            if (comboBox1.Items.Count == 0)                                         //判断是否为第一个日志
            {
                comboBox1.Items.Add(entry.Source.ToString());                       //添加日志信息
            }
            else
            {
                if (!comboBox1.Items.Contains(entry.Source.ToString()))             //判断应用程序是否重复
                {
                    comboBox1.Items.Add(entry.Source.ToString());                   //添加日志信息
                }
            }
        }
    }
}
private void btn_Add_Click(object sender, EventArgs e)
{
    if (comboBox1.SelectedItem == null)                                             //如果没有选择应用程序
    {
        MessageBox.Show("请选择日志名称");                                          //弹出消息对话框
        return;
    }
    if (textBox1.Text == "")                                                        //如果没有填写日志内容
    {
        MessageBox.Show("请填写日志内容");                                          //弹出消息对话框
        textBox1.Focus();                                                           //控件得到焦点
        return;                                                                     //退出方法
    }
    eventLog1.Log = "System";                                                       //设置读写日志的名称
    eventLog1.Source = comboBox1.SelectedItem.ToString();                           //设置日志源名称
    eventLog1.MachineName = ".";                                                    //设置写入日志的计算机名称
    eventLog1.WriteEntry(textBox1.Text);
    MessageBox.Show("添加成功");                                                    //弹出提示信息
    if (eventLog1.Entries.Count > 0)                                                //如果日志中有内容
    {
        foreach (System.Diagnostics.EventLogEntry entry in eventLog1.Entries)      //遍历日志内容
        {
            listView1.Items.Add(entry.Message);                                     //在控件中显示日志内容
        }
    }
}
```

秘笈心法

心法领悟 306：将系统中的异常信息写入日志。

程序执行过程中，不可避免地会出现异常，可以使用 try catch 块捕获并处理异常信息，在正确处理异常信息后，应当将异常信息写入日志，可以建立一个 Log 文件保存日志信息，也可以使用实例中介绍的 EventLog 组件将异常信息写入日志中。

12.4　其他常用控件的典型应用

实例 307	使用 MaskedTextBox 控件实现输入验证	中级
	光盘位置：光盘\MR\12\307	趣味指数：★★★★

实例说明

在应用软件开发中，根据实际业务的规定，往往需要对输入的数据进行严格的验证，这样才能够提高录入数据的准确性。本实例使用 MaskedTextBox 控件来验证用户输入的身份证号、邮政编码、出生日期等。实例运行效果如图 12.18 所示。

图 12.18　使用 MaskedTextBox 控件实现输入验证

关键技术

通过设置 MaskedTextBox 控件的 Mask 属性可以验证用户在文本框中输入数据的格式。MaskedTextBox 控件是一个增强型的 TextBox 控件，它支持用于接受或拒绝用户输入的声明性语法。当 MaskedTextBox 控件在运行中显示时，会将掩码表示为一系列提示字符和可选的原义字符，表示一个必须或可选输入的每个可编辑掩码位置都显示为单个提示字符。例如，数字符号（#）通常用作数字字符输入的占位符。当用户在掩码文本框中输入内容时，有效的输入字符将按顺序替换各自的提示字符。如果用户输入无效的字符，将不会发生替换。在这种情况下，如果 BeepOnError 属性设置为 true，将发出警告声，并引发 MaskInputRejected 事件，这样就可以通过处理 MaskInputRejected 事件来提供自定义错误的处理逻辑。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TxtValidate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加 4 个 MaskedTextBox 控件，分别用来输入 18 位身份证号、15 位身份证号、邮政编码和出生日期。

（3）程序的主要代码如下。

在窗体的 Load 事件中，为 4 个 MaskedTextBox 控件设置掩码，即设置控件的 Mask 属性值。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    this.maskedTextBox1.Mask = "000000-00000000-000A";                    //身份证号码 18 位
    this.maskedTextBox2.Mask = "000000-000000-000";                       //身份证号码 15 位
    this.maskedTextBox3.Mask = "000000";                                  //邮政编码
    this.maskedTextBox4.Mask = "0000 年 90 月 90 日";                     //出生日期
}
```

当输入字符或输入的字符不符合掩码规范时，将触发 MaskedTextBox 控件的 MaskInputRejected 事件。代码如下：

```
private void maskedTextBox1_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)          //输入 18 位身份证号
{
    MessageBox.Show("请输入 18 位身份证号",                                                          //弹出消息对话框
        "信息提示", MessageBoxButtons.OK,MessageBoxIcon.Information);
}
private void maskedTextBox2_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)          //输入 15 位身份证号
{
    MessageBox.Show("请输入 15 位身份证号",                                                          //弹出消息对话框
        "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
private void maskedTextBox3_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)          //输入邮政编码
{
    MessageBox.Show("请输入正确邮政编号",                                                            //弹出消息对话框
        "信息提示", MessageBoxButtons.OK,MessageBoxIcon.Information);
}
private void maskedTextBox4_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)          //输入出生日期
{
    MessageBox.Show("请输入正确出生日期格式",                                                        //弹出消息对话框
        "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 307：总结 MaskedTextBox 控件的应用。

若使用 MaskedTextBox 控件实现输入验证，则需要组合使用该控件的 Mask 属性与 MaskInputRejected 事件。Mask 属性用于设置验证掩码，MaskInputRejected 事件用于提供自定义错误的处理逻辑。

实例 308　制作日历计划任务

光盘位置：光盘\MR\12\308　　高级　　趣味指数：★★★★★

实例说明

随着社会的进步，人们日常生活的节奏也逐渐加快，每天的日程都被安排得很满，并且都要有提前预约。本实例可以记载本月份已经预约的日期，帮助用户有条理地安排工作计划。运行程序，在“任务内容”文本框中填写任务信息，单击日历控件，这时程序会弹出“是否添加任务”对话框，单击“确定”按钮会在任务列表中添加任务。实例运行效果如图 12.19 所示。

图 12.19　制作日历计划任务

关键技术

本实例在实现时主要用到了 DateTime 类型的 ToShortDateString 方法，该方法将此实例的值转换为其等效的短日期字符串表示形式。语法格式如下：

```
public string ToShortDateString ( )
```

参数说明

返回值：一个字符串，包含与此实例的日期值等效的数字月份、该月中的数字日期和年份。

例如，将系统日期转换为短日期格式的代码如下：

```
DateTime.Now.ToShortDateString( );
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TimeTask。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，向窗体中添加一个 ListView 控件，用来显示任务列表；添加一个 MonthCalendar 控件，用来选择任务日期；添加一个 TextBox 控件，用来输入任务信息；添加一个 Button 控件，用来删除任务列表中的某项任务。

（3）程序的主要代码如下。

在窗体的 Load 事件中，从数据库中取出任务信息，并显示在列表视图中，同时开始监视任务。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    GetMessage();                                                    //显示任务信息
    BeginTask();                                                     //开始监视任务
}
```

Load 事件中的 GetMessage 方法实现显示任务信息。代码如下：

```
private void GetMessage()                                            //显示任务信息
{
    lv_Task.Items.Clear();                                           //清空控件项
    G_Task = new DataTier().Select();                                //得到任务集合
    foreach (task t in G_Task)                                       //遍历任务集合
    {
        lv_Task.Items.Add(new ListViewItem(new string[]              //向控件中加入任务项
        {
            t.Date.ToShortDateString(),t.Task
        }));
    }
}
```

Load 事件中的 BeginTask 方法实现开始监视任务。代码如下：

```
private void BeginTask()                                             //开始监视任务
{
    Thread th = new Thread(                                          //建立线程
        (() =>                                                       //使用 Lambda 表达式
        {
            while (true)                                             //无限循环
            {
                for (int i = 0; i < G_Task.Count; i++)               //循环任务集合
                {
                    if (DateTime.Now.ToShortDateString() ==          //判断是否执行任务
                        G_Task[i].Date.ToShortDateString())
                    {
                        this.Invoke(                                 //调用窗体线程
                            ((MethodInvoker)(() =>                   //使用 Lambda 表达式
                            {
                                Form P_Form = new Form();            //创建窗体对象
                                Label lb_txt = new Label();          //创建 Label 标签
                                lb_txt.Text = G_Task[i].Task;        //设置标签文本
                                lb_txt.Font = new Font("隶书", 30);  //设置标签字体
                                lb_txt.AutoSize = true;              //设置标签自动调整大小
                                lb_txt.ForeColor = Color.Blue;       //设置文字颜色
                                P_Form.Controls.Add(lb_txt);         //向控件集合添加标签
                                P_Form.Width = 500;                  //设置窗体宽度
                                P_Form.Height = 100;                 //设置窗体高度
                                P_Form.StartPosition =               //设置窗体开始位置
                                    FormStartPosition.CenterScreen;
                                P_Form.Text = "任务提示！ ";          //设置窗体标题文本
                                P_Form.Show();                       //显示窗体
                            })));
```

```
                    new DataTier().Delete(                                    //从数据库中删除数据
                        G_Task[i].Date.ToShortDateString(),G_Task[i].Task);
                    Thread.Sleep(2000);                                       //线程挂起 2 秒
                    G_Task.RemoveAt(i);                                       //删除任务
                    this.Invoke(                                              //调用窗体线程
                        ((MethodInvoker)(() =>                                //使用 Lambda 表达式
                        {
                            GetMessage();                                     //显示任务信息
                        })));
                    break;                                                    //退出循环
                }
            }
            Thread.Sleep(5000);                                               //线程挂起 5 秒
        }
    }));
    th.IsBackground = true;                                                   //设置线程为后台线程
    th.Start();                                                               //开始执行线程
}
```

秘笈心法

心法领悟 308：使用 Lambda 表达式实现委托。

在应用程序开发中，有些方法的参数被定义成委托，可以根据委托参数的实际情况，恰当地使用 Lambda 表达式来实现委托约定的方法。

实例 309　在 ProgressBar 控件中显示进度百分比

高级

光盘位置：光盘\MR\12\309　　趣味指数：★★★★★

实例说明

当进行软件安装和文件复制时，在安装和复制的过程中，都会有一个进度条显示当前操作的进度。通过对本实例的学习，读者可以掌握如何在窗体中使用 ProgressBar 控件来显示进度条。实例运行效果如图 12.20 所示。

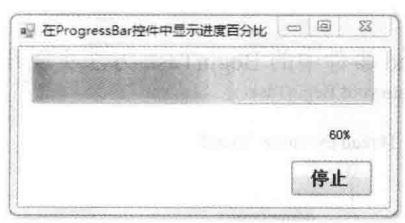


图 12.20　在 ProgressBar 控件中显示进度百分比

关键技术

本实例主要用到 ProgressBar 控件的 Value 属性、Maximum 属性、Minimum 属性以及 PerformStep 方法，下面介绍本实例中用到的技术。

（1）Value 属性

该属性表示 ProgressBar 控件的当前值介于最大值和最小值之间。

（2）Maximum 属性

该属性表示 ProgressBar 控件正在使用的范围的上限。

（3）Minimum 属性

该属性表示 ProgressBar 控件正在使用的范围的下限。语法格式如下：

```
public int Minimum { get; set; }
```

（4）PerformStep 方法

该方法表示按照 Step 属性的数量增加进度栏的当前位置。语法格式如下：

```
public void PerformStep()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ProgressBar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，该窗体上主要用到的控件及说明如表 12.1 所示。

表 12.1　窗体上主要用到的控件及说明

控 件 类 型	控 件 名 称	说　　明
ProgressBar	progressBar1	用于显示进度
Label	label1	显示当前进度条的百分比
Timer	timer1	设置 Enabled 属性为 true，设置 Interval 属性为 1000
Button	StartOrStop	用于控制进度条的开启与关闭状态

（3）程序的主要代码如下：

```
//Timer 控件的 Tick 事件，在本实例中，该事件每当经过 1 秒钟就被触发一次
private void timer1_Tick(object sender,EventArgs e)
{
    if(this.progressBar1.Value == this.progressBar1.Maximum)          //当进度条的当前值等于最大值时
    {
        this.progressBar1.Value = this.progressBar1.Minimum;          //设置进度条的当前值为最小值
    }
    else                                                              //当进度条的当前值小于最大值时
    {
        this.progressBar1.PerformStep();                              //按指定的增量增加进度条中的进度块
    }
    int percentValue = 100 *                                          //将当前进度转化为百分比的形式
        (this.progressBar1.Value - this.progressBar1.Minimum)
        / (this.progressBar1.Maximum - this.progressBar1.Minimum);
    label1.Text = percentValue.ToString() + "%";                      //在 Label 中显示百分比的值
}
private void StartOrStop_Click(object sender,EventArgs e)            //单击按钮实现停止或启动进度条
{
    if(timer1.Enabled)                                                //当 Timer 处于可用状态时
    {
        timer1.Enabled = false;                                       //设置 Timer 为不可用状态
        StartOrStop.Text = "开始";                                    //设置“开始”按钮上的文本为“开始”
    }
    else                                                              //当 Timer 处于不可用状态时
    {
        timer1.Enabled = true;                                        //设置 Timer 为可用状态
        StartOrStop.Text = "停止";                                    //设置“停止”按钮上的文本为“停止”
    }
}
```

秘笈心法

心法领悟 309：Timer 控件的 Enabled 属性。

如果设置 Timer 控件的 Enabled 属性值为 true，则启用计时器；如果设置 Timer 控件的 Enabled 属性值为 false，则禁用计时器。

实例 310　使用 BackgroundWorker 组件执行异步操作

高级

光盘位置：光盘\MR\12\310

趣味指数：★★★★★

实例说明

运行程序时，有些操作可能需要执行很长时间，例如，图像下载、Web 服务调用和文件下载、上传等，这些操作可能导致用户界面在操作运行时挂起，如果此时需要用户界面进行响应，可以使用 BackgroundWorker 组件在不同于应用程序的主用户界面线程的另一线程上异步（“在后台”）执行耗时操作。本实例实现使用 BackgroundWorker 组件异步执行耗时操作的功能，运行程序，单击“开始”按钮，将执行异步操作。实例运行效果如图 12.21 所示。

图 12.21　使用 BackgroundWorker 组件执行异步操作

关键技术

实现本实例的功能主要用到 BackgroundWorker 组件的 CancelAsync 方法、RunWorkerAsync 方法、ProgressChanged 事件和 DoWork 事件。下面分别进行介绍。

（1）CancelAsync 方法

该方法请求取消挂起的后台操作。语法格式如下：

```
public void CancelAsync()
```

（2）RunWorkerAsync 方法

该方法开始执行后台操作。语法格式如下：

```
public void RunWorkerAsync()
```

（3）ProgressChanged 事件

该事件在调用 ReportProgress 方法时发生。语法格式如下：

```
public event ProgressChangedEventHandler ProgressChanged
```

（4）DoWork 事件

该事件在调用 RunWorkerAsync 方法时发生。语法格式如下：

```
public event DoWorkEventHandler DoWork
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseBackgroundWorker。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，该窗体上主要用到的控件及说明如表 12.2 所示。

表 12.2　窗体上主要用到的控件及说明

控件类型	控件名称	说　明
BackgroundWorker	BackgroundWorker1	用于异步操作
Label	resultLabel	显示运行结果
NumericUpDown	numericUpDown1	用于设置执行异步操作的值
ProgressBar	progressBar1	显示异步操作的进度
Button	startAsyncButton	执行异步操作
	cancelAsyncButton	取消异步操作

（3）程序的主要代码如下。

在另一个线程上运行事件处理的实现代码如下：

```
//定义整型变量 numberToCompute
private int numberToCompute = 0;
//定义整型变量 highestPercentageReached
private int highestPercentageReached = 0;
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)          //运行 BackGroundWorker 组件
{

    e.Result = ComputeFibonacci((int)e.Argument, this.backgroundWorker1, e);          //为组件的运行结果赋值
}
//组件的进程改变事件
private void backgroundWorker1_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    this.progressBar1.Value = e.ProgressPercentage;                                   //为进度条赋值
}
```

```
//当辅助线程完成（无论是成功、失败还是取消）时引发
private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Error != null)                                          //当有错误出现时
    {
        MessageBox.Show(e.Error.Message);                         //显示错误信息
    }
    else if (e.Cancelled)                                         //当取消时
    {
        resultLabel.Text = "Canceled";                            //在 Label 标签中显示 Canceled
    }
    else                                                          //当为其他情况时
    {
        resultLabel.Text = e.Result.ToString( );                  //在 Label 标签中显示对应的结果
    }
    this.numericUpDown1.Enabled = true;                           //使 NumericUpDown 变为可用状态
    startAsyncButton.Enabled = true;                              //使控件变为可用状态
    cancelAsyncButton.Enabled = false;                            //使控件变为不可用状态
}
```

启动异步操作的实现代码如下：

```
private void startAsyncButton_Click(object sender, EventArgs e)
{
    resultLabel.Text = String.Empty;                              //将结果窗口设为空
    this.numericUpDown1.Enabled = false;                          //使 NumericUpDown 变为不可用状态
    this.startAsyncButton.Enabled = false;                        //使 startAsyncButton 变为不可用状态
    this.cancelAsyncButton.Enabled = true;                        //使 cancelAsyncButton 变为可用状态
    numberToCompute = (int)numericUpDown1.Value;                  //把 NumericUpDown 值赋给整型变量
    highestPercentageReached = 0;                                 //初始化 highestPercentageReached
    backgroundWorker1.RunWorkerAsync(numberToCompute);            //开始执行后台操作
}
```

自定义方法 ComputeFibonacci 用来实现异步操作。代码如下：

```
long ComputeFibonacci(int n, BackgroundWorker worker, DoWorkEventArgs e)
{
    if ((n < 0) || (n > 91))                                      //若 int 型变量 n 不在区间（0，91）
    {
        throw new ArgumentException("value must be >= 0 and <= 91", "n");   //抛出异常
    }
    long result = 0;                                              //定义 long 型变量 result
    if (worker.CancellationPending)                               //当应用程序请求取消操作时
    {
        e.Cancel = true;                                          //取消事件的值
    }
    else                                                          //当应用程序没有请求取消操作时
    {
        if (n < 2)                                                //当 int 型变量小于 2 时
        {
            result = 1;                                           //设定 long 型变量 result 的值为 1
        }
        else                                                      //当 int 型变量大于等于 2 时
        {
            result = ComputeFibonacci(n - 1, worker, e) +ComputeFibonacci(n - 2, worker, e);//递归调用自定义方法 ComputeFibonacci
        }
        int percentComplete =(int)((float)n / (float)numberToCompute * 100);   //将进程转化为百分比
        if (percentComplete > highestPercentageReached)           //当完成的进程百分比大于最大值时
        {
            highestPercentageReached = percentComplete;           //为 highestPercentageReached 赋值
            worker.ReportProgress(percentComplete);               //引发 ProgressChanged 事件
        }
    }
    return result;                                                //返回 result 的值
}
private void cancelAsyncButton_Click(object sender, EventArgs e)
{
    this.backgroundWorker1.CancelAsync( );                        //调用组件的请求取消挂起的后台操作
    cancelAsyncButton.Enabled = false;                            //设定 cancelAsyncButton 按钮不可用
}
```

秘笈心法

心法领悟 310：自定义异常输出信息。

Exception 对象的 Message 属性用来提供异常信息，可以通过在创建异常对象时向构造器中传入自定义信息来改变 Message 属性的默认值。

实例 311 使用 ErrorProvider 组件验证文本框输入

光盘位置：光盘\MR\12\311　　高级　　趣味指数：★★★★★

实例说明

ErrorProvider 组件可以用来对窗体或控件上的用户输入进行验证，当验证用户在窗体中的输入或显示数据集内的错误时，一般要用到该控件。相对于在消息框中显示错误信息，ErrorProvider 组件是更好的选择。ErrorProvider 组件在相关控件（如文本框）右侧显示一个错误图标，当用户将鼠标指针放在该错误图标上时，将出现错误信息的提示工具。实例运行效果如图 12.22 所示。

图 12.22　使用 ErrorProvider 组件验证文本框输入

关键技术

实现本实例的功能主要用到了 ErrorProvider 组件的 BlinkStyle 属性、BlinkRate 属性、SetError 方法和 TextBox 文本框的 Validating 事件。下面分别进行介绍。

（1）BlinkStyle 属性

该属性用于获取或设置一个值，该值指示错误图标的闪烁时间。语法格式如下：

```
public ErrorBlinkStyle BlinkStyle { get; set; }
```

参数说明

属性值：ErrorBlinkStyle 枚举值。默认为 BlinkIfDifferentError。ErrorBlinkStylem 的枚举值及说明如表 12.3 所示。

表 12.3　ErrorBlinkStylem 的枚举值及说明

枚 举 值	说　明
AlwaysBlink	错误图标第一次显示时，或者当为控件设置了错误描述字符串并且错误图标已经显示时，总是闪烁
BlinkIfDifferentError	当图标已经显示并且为控件设置了新的错误字符串时闪烁
NeverBlink	错误图标从不闪烁

（2）BlinkRate 属性

该属性用于获取或设置错误图标的闪烁速率。语法格式如下：

```
public int BlinkRate { get; set; }
```

参数说明

属性值：错误图标的闪烁速率（以毫秒为单位），默认为 315 毫秒。

（3）SetError 方法

该方法将 Error 属性设置为指定的 TextWriter 对象。语法格式如下：

```
public static void SetError (TextWriter newError)
```

参数说明

newError：一个 TextWriter 流，是新的标准错误输出。

（4）Validating 事件

该事件在控件正在验证时发生。语法格式如下：

```
public event CancelEventHandler Validating
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InputText。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体上添加两个 ErrorProvider 组件，将其 BlinkRate 属性设置为 100，BlinkStyle 属性值设置为 AlwaysBlink，该控件用于验证输入是否正确；添加两个 TextBox 控件，用于输入文本；添加两个 Button 控件，用于引发 ErrorProvider 组件验证。

（3）程序的主要代码如下：

```
string strA = null;                                          //定义字符串字段
string strB = null;                                          //定义字符串字段
private void txtPasword_Validating(object sender, CancelEventArgs e)   //验证密码
{
    if (txtPasword.Text != "mrscoft")                        //当密码输入不正确时
    {
        errPassword.SetError(txtPasword, "密码确误");          //显示错误提示信息
    }
    else                                                     //当密码输入正确时
    {
        errPassword.SetError(txtPasword, "");                //不显示任何内容
        strA = txtPasword.Text;                              //得到密码字符串
    }
}
private void txtUser_Validating(object sender, CancelEventArgs e)      //验证登录名
{
    if (txtUser.Text != "mr")                                //当登录名输入错误时
    {
        errUser.SetError(txtUser, "登录名错误");               //显示错误的提示信息
    }
    else                                                     //当登录名输入正确时
    {
        errUser.SetError(txtUser, "");                       //不显示任何内容
        strB = txtUser.Text;                                 //得到用户名字符串
    }
}
```

秘笈心法

心法领悟 311：让错误图标不闪烁。

当发生错误验证时，ErrorProvider 组件的错误图标会按指定速率闪烁。将 BlinkStyle 设置为 NeverBlink 时，表示闪烁速率为 0，即不闪烁。

实例 312　使用 FileSystemWatcher 组件监视日志文件

光盘位置：光盘\MR\12\312　　高级　　趣味指数：★★★★★

实例说明

利用 FileSystemWatcher 组件可以监视文件系统，并在文件系统所包含的目录和文件发生更改时做出响应，该组件可以在特定的文件和目录中被创建、修改或删除，快速方便地启动业务处理。本实例实现使用 FileSystemWatcher 组件监视系统日志文件是否被更改（包括文件被删除、修改和创建等情况）。实例运行效果如图 12.23 所示。

图 12.23　使用 FileSystemWatcher 组件监视日志文件

关键技术

实现本实例时主要用到 FileSystemWatcher 组件的 Path 属性、Filter 属性、EndInit 方法、Created 事件、Changed 事件和 Deleted 事件。下面分别进行介绍。

（1）Path 属性

该属性用于获取或设置要监视的目录的路径。语法格式如下：

```
public string Path { get; set; }
```

参数说明

属性值：要监视的路径。默认值为空字符串（""）。

（2）Filter 属性

该属性用于获取或设置筛选字符串，用于确定在目录中监视哪些文件。语法格式如下：

```
public string Filter { get; set; }
```

参数说明

属性值：筛选器字符串。默认值为“*.*”（监视所有文件）。

（3）EndInit 方法

该方法结束在窗体上使用或由另一个组件使用的 FileSystemWatcher 的初始化，初始化发生在运行时。语法格式如下：

```
public void EndInit()
```

（4）Created 事件

该事件当在指定 Path 中创建文件和目录时发生。语法格式如下：

```
public event FileSystemEventHandle Created
```

（5）Changed 事件

该事件当更改指定 Path 中的文件和目录时发生。语法格式如下：

```
public eventFileSystemEventHandler Changed
```

（6）Deleted 事件

该事件删除指定 Path 中的文件或目录时发生。语法格式如下：

```
public event FileSystemEventHandler Deleted
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileMonitor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体上添加一个 FileSystemWatcher 组件，用来监视系统日志文件；添加一个 Button 控件，用来监视日志文件操作；添加一个 TextBox 控件，用来显示本机系统日志文件的路径；添加一个 ListBox 控件，用来显示日志提示信息。

（3）程序的主要代码如下。

加载文件指定系统日志文件的路径的实现代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    textBox1.Text = Environment.SystemDirectory + "\\config";          //设置文件路径
    textBox1.Enabled = false;                                          //设置 textBox1 为不可用状态
}
```

启动 FileSystemWatcher 组件开始操作的实现代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    fileSystemWatcher1.Path = textBox1.Text;                           //设置要监视的文件的路径
    this.fileSystemWatcher1.Filter = "*.Evt";                          //该属性获取或设置筛选字符串
    this.fileSystemWatcher1.EndInit( );                                //结束 fileSystemWatcher1 的初始化过程
}
```

创建日志文件时给出提示的实现代码如下：

```
private void fileSystemWatcher1_Created(object sender, System.IO.FileSystemEventArgs e)
{
    listBox1.Items.Add("日志文件:" + e.FullPath+"被创建");              //向 listBox1 中添加创建的内容
}
```

修改日志文件时给出提示的实现代码如下：

```
private void fileSystemWatcher1_Changed(object sender, System.IO.FileSystemEventArgs e)
{
    listBox1.Items.Add("日志文件:" + e.FullPath + "被更改");                    //向 listBox1 中添加修改信息
}
```

删除日志文件时给出提示的实现代码如下：

```
private void fileSystemWatcher1_Deleted(object sender, System.IO.FileSystemEventArgs e)
{
    listBox1.Items.Add("日志文件:" + e.FullPath + "被删除");                    //向 listBox1 中添加删除信息
}
```

秘笈心法

心法领悟 312：获取当前环境信息。

Environment 类提供有关当前环境和平台的信息以及操作它们的操作方法。例如，通过该类的 NewLine 属性可以获取为此环境定义的换行字符串，通过该类的 SetEnvironmentVariable 方法可实现创建、修改或删除当前进程中存储的环境变量等。

实例 313　使用 HelpProvider 组件调用帮助文件

高级

光盘位置：光盘\MR\12\313

趣味指数：★★★★★

实例说明

HelpProvider 组件用于将 HTML 帮助、1.x 帮助文件（.htm 文件或由 HTML Help Workshop 产生的 chm 文件）与 Windows 应用程序相关联。程序中可以通过多种方式提供帮助，例如，为 Windows 窗体中的控件提供区分上下文的帮助。本实例实现使用 HelpProvider 组件调用帮助文件的功能。运行程序，按 F1 键调用财务管理系统帮助文件，实例运行效果如图 12.24 所示。

图 12.24　使用 HelpProvider 组件调用帮助文件

关键技术

实现本实例时主要用到了 HelpProvider 组件的 HelpNamespace 属性。

HelpNamespace 属性用来获取或设置一个值，该值指定与此 HelpProvider 对象关联的帮助文件名。语法格式如下：

```
public virtual string HelpNamespace { get; set; }
```

参数说明

属性值：帮助文件的名称，形式可以是 C:\path\sample.chm 或\folder\file.htm。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InvokHelp。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中主要添加一个 HelpProvider 组件，用来调用帮助文件。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
```

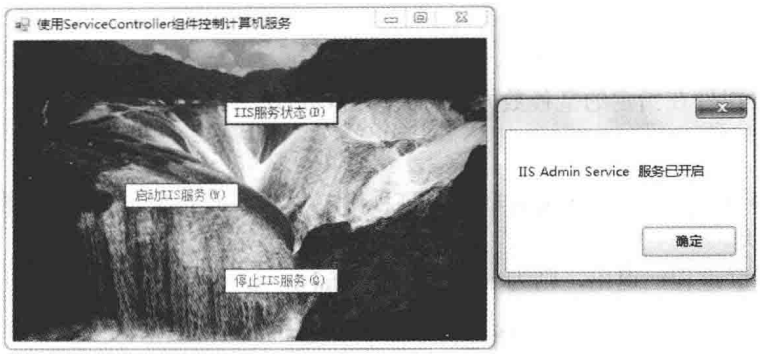


图 12.26　使用 ServiceController 组件控制计算机服务

关键技术

本实例实现时主要用到了 ServiceController 组件的 MachineName 属性、ServiceName 属性、Status 属性、DisplayName 属性、CanStop 属性、Start 方法和 Stop 方法。下面介绍本实例中用到的技术。

（1）MachineName 属性

该属性用于获取或设置此服务所驻留的计算机的名称。语法格式如下：

```
public string MachineName { get; set; }
```

参数说明

属性值：正在运行与此 ServiceController 实例关联的服务的计算机名称。默认为本地计算机（""）。

（2）ServiceName 属性

该属性用于获取或设置对此实例引用的服务进行标识的名称。语法格式如下：

```
public string ServiceName { get; set; }
```

参数说明

属性值：对此 ServiceController 实例引用的服务进行标识的名称。默认值为空字符串（""）。

（3）Status 属性

该属性用于获取由此实例引用的服务的状态。语法格式如下：

```
public ServiceControllerStatus Status { get; }
```

参数说明

属性值：ServiceControllerStatus 值之一，指示服务是正在运行、已停止还是已暂停，或者启动、停止、暂停或继续命令被挂起。

（4）DisplayName 属性

该属性用于获取或设置服务的友好名称。语法格式如下：

```
public string DisplayName { get; set; }
```

参数说明

属性值：服务的友好名称，可用于标识该服务。

（5）CanStop 属性

该属性用于获取一个值，该值指示服务在启动后是否可以停止。语法格式如下：

```
public bool CanStop { get; }
```

参数说明

属性值：如果服务可以停止并调用了 OnStop 方法，则为 true；否则为 false。

（6）Start 方法

该方法启动服务，不传递任何参数。语法格式如下：

```
public void Start ( )
```

（7）Stop 方法

该方法停止该服务以及任何依赖于该服务的服务。语法格式如下：

```
public void Stop ( )
```

示模式对话框），则为 false。

（5）WaitForExit 方法

该方法指示 Process 组件在指定的毫秒数内等待关联进程退出。语法格式如下：

```
public bool WaitForExit (int milliseconds)
```

参数说明

❶ milliseconds：等待关联进程退出的时间（以毫秒为单位）。最大值为 32 位整数的最大可能值，这对于操作系统而言表示无限大。

❷ 返回值：如果关联进程已退出，则为 true；否则为 false。

📖说明：使用 Process 类必须引用 System.Diagnostics 命名空间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetProcess。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体上添加一个 Process 组件，用来调用记事本进程；添加两个 Button 控件，用来执行启动和关闭记事本进程操作。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)                    //设置要启动的应用程序
{
    process1.StartInfo.FileName = "notepad.exe";                       //设置要启动的应用程序
}
private void button1_Click(object sender, EventArgs e)
{
    process1.Start( );                                                 //启动记事本进程
}
private void button2_Click(object sender, EventArgs e)                 //关闭记事本进程
{
     System.Diagnostics.Process[] myProcesses;                         //声明一个进程数组
     myProcesses =  System.Diagnostics.Process.                        //获取当前的所有记事本进程
           GetProcessesByName("Notepad");
       foreach (System.Diagnostics.Process instance in myProcesses)    //循环遍历数组中的每一个元素
       {
           instance.CloseMainWindow( );                                //关闭拥有用户界面的进程
           instance.WaitForExit(3000);                                 //在指定的时间内等待关联进程的退出
           instance.Close( );                                          //释放与此组件关联的所有资源
       }
}
```

秘笈心法

心法领悟 314：释放与 Process 组件关联的所有资源。

在使用 Process 组件的 CloseMainWindow 方法关闭拥有用户界面的进程后，还需要调用 Process 组件的 Close 方法来释放与此组件关联的所有资源，这样可以减少不必要的系统资源浪费。

实例 315　使用 ServiceController 组件控制计算机服务

光盘位置：光盘\MR\12\315　　高级　　趣味指数：★★★★★

实例说明

ServiceController 组件用来将操作请求传递给服务控制管理器，而不是传递给服务本身，然后利用服务控制管理器停止、启动或暂停服务，或者执行其他指定的操作。本实例实现使用 ServiceController 组件控制 IIS 服务的功能。运行程序，单击“IIS 服务状态”按钮，提示当前 IIS 服务状态，如果 IIS 服务是关闭状态，“启动 IIS 服务”按钮可用；IIS 服务是开启状态，“停止 IIS 服务”按钮可用。实例运行效果如图 12.26 所示。

```
    string strpath =                                                    //得到路径信息
        Application.StartupPath.Substring(0, Application.StartupPath.Substring(0,
        Application.StartupPath.LastIndexOf("\\")).LastIndexOf("\\"));
    strpath += @"\mrHelp.chm";                                          //得到帮助文件信息
    helpProvider1.HelpNamespace=strpath;                                //关联帮助文件
}
```

秘笈心法

心法领悟 313：窗体中添加了 HelpProvider 组件。

在 Windows 窗体中添加 HelpProvider 组件可以使该窗体中的其他控件公开 HelpProvider 组件的 Help 属性，这样就可以提供有关 Windows 窗体控件的帮助。

实例 314 使用 Process 组件访问本地进程

光盘位置：光盘\MR\12\314　　中级　　趣味指数：★★★★★

实例说明

Process 组件提供对本地和远程进程的访问，并使用户能够启动和停止本地系统进程。本实例使用 Process 组件实现开启和结束记事本进程。实例运行效果如图 12.25 所示。

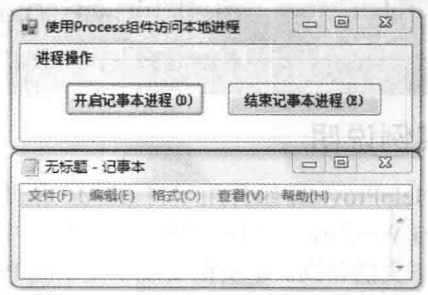


图 12.25　使用 Process 组件访问本地进程

关键技术

本实例实现时主要用到了 Process 组件的 StartInfo 属性、Start 方法，Process 类的 GetProcessesByName、CloseMainWindow 和 WaitForExit 方法，下面介绍本实例中用到的技术。

（1）StartInfo 属性

该属性用于获取或设置要传递给 Process 的 Start 方法的属性。语法格式如下：

```
public ProcessStartInfo StartInfo { get; set; }
```

参数说明

属性值：ProcessStartInfo 表示启动该进程时要使用的数据。

（2）Start 方法

该方法启动（或重用）此 Process 组件的 StartInfo 属性指定的进程资源，并将其与该组件关联。语法格式如下：

```
public bool Start ( )
```

参数说明

返回值：如果启动进程资源，则为 true；如果没有启动新的进程资源（例如，重用现有进程），则为 false。

（3）GetProcessesByName 方法

该方法创建新的 Process 组件的数组，并将其与本地计算机上共享指定的进程名称的所有进程资源关联。语法格式如下：

```
public static Process[] GetProcessesByName (string processName)
```

参数说明

❶ processName：该进程的友好名称。

❷ 返回值：Process 类型的数组，表示运行指定应用程序或文件的进程资源。

（4）CloseMainWindow 方法

该方法通过向进程的主窗口发送关闭消息来关闭拥有用户界面的进程。语法格式如下：

```
public bool CloseMainWindow ( )
```

参数说明

返回值：如果成功发送关闭消息，则为 true；如果关联进程没有主窗口或禁用主窗口（例如，如果当前显

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetService。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中主要添加一个 ServiceController 组件，用来控制 IIS 服务状态；添加 3 个 Button 控件，用来执行开启、停止和判断 IIS 服务状态操作。

（3）程序的主要代码如下。

启动 IIS 服务的实现代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    serviceController1.MachineName = ".";                                    //设置此服务所在的计算机名称
    serviceController1.ServiceName = "IISAdmin";                             //设置服务名称
     if (serviceController1.Status == ServiceControllerStatus.Running)       //判断服务状态
    {
        MessageBox.Show(                                                     //弹出消息对话框
            serviceController1.DisplayName + "  服务正在运行");
        Application.Exit();                                                  //退出应用程序
    }
    else
    {
        serviceController1.Start();                                          //启动服务
        MessageBox.Show(serviceController1.DisplayName + "  服务已开启");    //弹出消息对话框
        Application.Exit();                                                  //退出应用程序
    }
}
```

判断 IIS 服务状态的实现代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        serviceController1.MachineName = ".";                                //设置此服务所在的计算机名称
        serviceController1.ServiceName = "IISAdmin";                         //设置服务名称
        if (serviceController1.Status == ServiceControllerStatus.Running)    //判断服务状态
        {
            MessageBox.Show(serviceController1.DisplayName + "  服务已开启"); //弹出消息对话框
            btn_Stop.Enabled = true;                                         //启用停止服务按钮
            btn_Status.Enabled = false;                                      //停用状态按钮
        }
        else
        {
            MessageBox.Show(serviceController1.DisplayName + "服务已停止");  //弹出消息对话框
            btn_Status.Enabled = false;                                      //停用状态按钮
            btn_Start.Enabled = true;                                        //启用开始服务按钮
        }
    }
    catch (Exception ee)                                                     //捕获异常
    { MessageBox.Show(ee.Message); }                                         //弹出消息对话框
}
```

停止 IIS 服务的实现代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        serviceController1.MachineName = ".";                                //设置此服务所在的计算机名称
        serviceController1.ServiceName = "IISAdmin";                         //设置服务名称
        if (serviceController1.CanStop)                                      //判断是否可以停止服务
        {
            serviceController1.Stop();                                       //停止服务
            MessageBox.Show(                                                 //弹出消息对话框
                serviceController1.DisplayName + "服务已停止");
            Application.Exit();                                              //退出应用程序
         }
        else
```

```
        {
            MessageBox.Show(                                    //弹出消息对话框
                serviceController1.DisplayName + "不可以停止");
            Application.Exit();                                 //退出应用程序
        }
    }
    catch (Exception ee)                                        //捕获异常
        { MessageBox.Show(ee.Message); }                        //弹出消息对话框
}
```

秘笈心法

心法领悟 315：关于应用 ServiceController 组件的总结。

ServiceController 组件实际上将它的操作请求传递给服务控制管理器，而不是传递给服务本身。然后服务控制管理器停止、启动或暂停服务，或者执行其他指定的操作。ServiceController 将请求传递给服务控制管理器后返回，而不必等待服务控制管理器将请求传递给服务。

实例 316　使用 ImageList 组件制作动画图片

光盘位置：光盘\MR\12\316　　中级　　趣味指数：★★★★★

实例说明

ImageList 组件用于存储图像，这些图像可由控件显示。图像列表使用户能够为一致的单个图像目录编写代码。本实例使用 ImageList 组件制作了一张动画图片，运行程序，窗体中的图片会不停地变化，从而实现了动画效果。实例运行效果如图 12.27 所示。

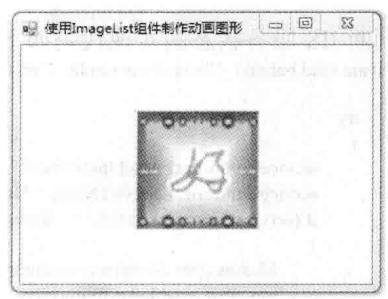


图 12.27　使用 ImageList 组件制作动画图片

关键技术

实现本实例的功能主要用到了 Timer 组件的 Enabled 属性、Tick 事件，PictureBox 控件的 Image 属性和 ImageList 组件的 Images 属性。下面介绍本实例中用到的技术。

（1）Enabled 属性

该属性用于获取或设置计时器是否正在运行。语法格式如下：

```
public virtual bool Enabled { get; set; }
```

参数说明

属性值：如果计时器当前处于启用状态，则为 true；否则为 false。默认为 false。

（2）Tick 事件

该事件当指定的计时器间隔已过去而且计时器处于启用状态时（即 Timer 组件的 Enabled 属性值为 true）发生。

（3）Image 属性

该属性用于获取或设置 PictureBox 显示的图像。语法格式如下：

```
public Image Image { get; set; }
```

参数说明

属性值：要显示的 Image。

（4）Images 属性

该属性用于获取此图像列表的 ImageList.ImageCollection。语法格式如下：

```
public ImageCollection Images { get; }
```

参数说明

属性值：图像集合。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetAnimation。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ImageList 组件，用来添加图片集合；添加一个 Timer 控件，用来控制图片的变化时间；添加一个 PictureBox 控件，用于显示图片。

（3）程序的主要代码如下：

```
public int intImage =-1;                                                    //控件图片索引
private void timer1_Tick(object sender, EventArgs e)                        //触发 Timer 组件的 Tick 事件
{
    pictureBox1.Image = imageList1.Images[intImage = intImage++ > 9 ? 0 : intImage];   //更改图片索引

}
private void Form2_Load(object sender, EventArgs e)                         //加载窗体时启动计时器
{
    timer1.Enabled = true;                                                  //启动计时器
}
```

秘笈心法

心法领悟 316：使用条件运算符。

根据代码的实际情况，恰当地使用条件运算符代替 if…else 结构的计算，可以使代码更加简洁和优雅。

实例 317　使用 DirectoryEntry 组件建立虚拟目录

中级

光盘位置：光盘\MR\12\317

趣味指数：★★★★☆

实例说明

DirectoryEntry 组件公开用于创建、移动、枚举目录和子目录的实例方法。本实例实现了使用 DirectoryEntry 组件建立虚拟目录的功能。运行程序，单击“选择虚拟目录”按钮，接着输入虚拟目录名称，然后单击“建立虚拟目录”按钮，虚拟目录即可设置成功。实例运行结果如图 12.28 所示。

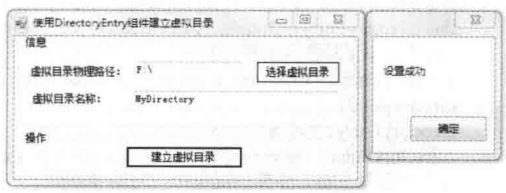


图 12.28　使用 DirectoryEntry 组件建立虚拟目录

关键技术

实现本实例的功能主要用到了 DirectoryEntry 组件的 Path 属性、Children 属性、Properties 属性、SchemaClassName 属性和 FolderBrowserDialog 控件的 SelectedPath 属性。下面介绍本实例中用到的属性。

（1）Path 属性

该属性用于获取或设置 DirectoryEntry 的路径。语法格式如下：

```
public string Path { get; set; }
```

参数说明

属性值：DirectoryEntry 对象的路径。默认值为空字符串（""）。

（2）Children 属性

该属性用于获取 Active Directory 层次结构中节点的子项。语法格式如下：

```
public DirectoryEntries Children { get; }
```

参数说明

属性值：一个 DirectoryEntries 对象，包含 Active Directory 层次结构中此节点的子项。

（3）Properties 属性

该属性用于获取此 DirectoryEntry 对象的 Active Directory 属性。语法格式如下：

```
public PropertyCollection Properties { get; }
```

参数说明

属性值：一个 PropertyCollection 对象，包含此项所设属性。

（4）SchemaClassName 属性

该属性用于获取此 DirectoryEntry 对象的架构类的名称。语法格式如下：

```
public string SchemaClassName { get; }
```

参数说明

属性值：用于此 DirectoryEntry 对象的架构类的名称。

（5）SelectedPath 属性

该属性用于获取或设置用户选定的路径。语法格式如下：

```
public string SelectedPath { get; set; }
```

参数说明

属性值：对话框中选定的第一个文件夹或用户选定的最后一个文件夹的路径。默认值为空字符串（""）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateDirectory。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DirectoryEntry 组件，用于设置虚拟目录；添加一个 FolderBrowserDialog 控件，用来选择虚拟目录的路径；添加两个 TextBox 控件，分别用来显示虚拟目录的路径和名称；添加两个 Button 控件，分别用来执行选择虚拟目录和建立虚拟目录操作。

（3）程序的主要代码如下。

自定义方法 getFilter 用来建立虚拟目录，代码如下：

```
public void getFilter(string strPath, string strName)
{
    string virtualDirName = strName;                                          //虚拟目录名称
    string physicalPath = strPath;                                            //虚拟目录实际路径
    this.directoryEntry1.Path = "IIS://localhost/W3SVC/1/ROOT";               //获取设置文件的路径
    this.directoryEntry1.Children.Add(virtualDirName, directoryEntry1.SchemaClassName);
    //设置虚拟目录指向的物理路径
    directoryEntry1.Properties["Path"][0] = physicalPath;
    directoryEntry1.Invoke("AppCreate", true);                                //设置读取权限
    directoryEntry1.Properties["AccessRead"][0] = false;                      //设置读取权限
     directoryEntry1.Properties["ContentIndexed"][0] = true;
     //设置默认文档，多值情况下中间用逗号分割
    directoryEntry1.Properties["DefaultDoc"][0] = "index.asp,Default.aspx";
    directoryEntry1.Properties["AppFriendlyName"][0] = virtualDirName;        //应用程序名称
    directoryEntry1.Properties["AccessScript"][0] = true;                     //执行权限
     directoryEntry1.Properties["DontLog"][0] = true;
    //设置目录的安全性，0 表示不允许匿名访问，1 为允许，3 为基本身份验证，7 为 Windows 继承身份验证
    directoryEntry1.Properties["AuthFlags"][0] = 0;
    directoryEntry1.Properties["AuthFlags"][0] = 1;
    directoryEntry1.CommitChanges();                                          //将目录保存到 IIS 中
}
```

单击“选择虚拟目录”按钮实现设置虚拟目录物理路径，代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = folderBrowserDialog1.SelectedPath;                    //设置虚拟目录路径
    }
}
```

单击“建立虚拟目录”按钮实现按照虚拟目录的路径和名称创建虚拟目录，代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.SelectedPath.ToString() != "" && textBox2.Text != "")
    {
        getFilter(folderBrowserDialog1.SelectedPath, textBox2.Text.TrimEnd());  //建立虚拟目录
        MessageBox.Show("设置成功");
```

```
    }
    else
    {
        MessageBox.Show("请选择虚拟目录的物理路径或输入虚拟目录名称", "信息提示");
    }
}
```

秘笈心法

心法领悟 317：保存虚拟目录信息到 IIS 中。

在设置完成虚拟目录的相关信息（如读写权限、默认页面等）之后，需要调用 DirectoryEntry 组件的 CommitChanges 方法把设置的虚拟目录信息保存到 IIS 中。

12.5　常用控件控制技术

实例 318	程序运行时智能增减控件	中级
	光盘位置：光盘\MR\12\318	趣味指数：★★★★★

实例说明

设计程序时，为了更好地保护信息安全，通常会为操作员设置用户权限。使用人员只能根据自己的权限来操作相应的功能模块。在本实例中，设计一个系统登录模块，用户在登录时会根据其权限显示相应的功能模块。运行程序，进入系统登录界面。首先以操作员身份登录，登录名为 Admin，密码为 Admin，进入图 12.29 所示的操作界面。然后以系统管理员身份登录，登录名为 Mr，密码为 Mrscoft，进入图 12.29 所示的操作界面。

图 12.29　程序运行时智能增减控件

关键技术

实现本实例时主要用到了 Button 控件的 Visible 属性、Form 窗体的 Show 方法和 Hide 方法。下面分别进行介绍。

（1）Visible 属性

该属性用于获取或设置一个值，该值指示是否显示该控件。语法格式如下：

```
public bool Visible { get; set; }
```

参数说明

属性值：如果显示该控件，则为 true，否则为 false。默认为 true。

（2）Show 方法

该方法显示窗体。语法格式如下：

```
public void Show ( )
```

（3）Hide 方法

该方法隐藏窗体。语法格式如下：

```
public void Hide ( )
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddAndRemoveControl。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体上添加 6 个 Button 控件，把 FlatStyle 属性

设置为 Flat，TextImageRelation 属性设置为 ImageAboveText，用来控制程序操作；添加一个 Imagelist 控件，用来为 Button 控件提供背景图片。

（3）新建一个窗体，并将其命名为 Frm_Login，在该窗体中添加两个 TextBox 控件，用来输入用户名和密码；添加两个 Button 按件，用来执行登录和退出登录操作。

（4）程序的主要代码如下：

```
Frm_Main frm = new Frm_Main();                                          //创建窗体对象
private void btnOK_Click(object sender, System.EventArgs e)             //确定
{
    if (txtUser.Text == "")                                             //如果用户名为空
    {
        MessageBox.Show("请输入用户名");                                 //弹出消息对话框
        return;                                                         //退出方法
    } else if (txtPasword.Text=="")                                     //如果密码为空
    {
        MessageBox.Show("请输入用户密码");                               //弹出消息对话框
        return;                                                         //退出方法
    }
    else if (txtUser.Text == "Admin" && txtPasword.Text == "Admin")     //如果输入的用户名和密码正确
    {
        frm.Show();                                                     //显示窗体
        frm.button1.Visible = false;                                    //隐藏 Button 按钮
        frm.button4.Visible = false;                                    //隐藏 Button 按钮
        frm.Text = frm.Text + "      " +"操作员:" + txtUser.Text;        //显示窗体标题
        this.Hide();                                                    //隐藏登录窗体
    }
    else if (txtUser.Text == "Mr" && txtPasword.Text == "Mrsoft")       //如果输入的用户名和密码正确
    {
        frm.Show();                                                     //显示窗体
        frm.Text = frm.Text + "      " + "系统管理员:" + txtPasword.Text;//显示窗体标题
        this.Hide();                                                    //隐藏登录窗体
    }
    else
    {
        MessageBox.Show("用户名或密码错误");                             //弹出消息对话框
        txtUser.Text = "";                                              //清空用户名
        txtPasword.Text = "";                                           //清空密码
        txtUser.Focus();                                                //控件得到焦点
    }
}
```

秘笈心法

心法领悟 318：设置用户权限。

通过对本实例的学习得出这样的总结，若要在程序中设置操作用户的权限，首先要判断登录用户的身份或级别，然后再根据登录用户的身份或级别来设置哪些模块可以使用。

实例 319　多控件焦点循环移动

光盘位置：光盘\MR\12\319　　中级　　趣味指数：★★★★☆

实例说明

一般情况下，移动控件间的焦点都是通过 Tab 键来实现的。本实例在原有的基础上增加了按 Enter 键实现焦点的移动功能。另外，当焦点移动至最后一个控件时自动跳转到第一个控件，也就是多控件的焦点循环移动。实例运行效果如图 12.30 所示。

图 12.30　多控件焦点循环移动

关键技术

本实例主要用到各个 TextBox 控件的 Enter 事件、Leave 事件以及 KeyDown 事件和自定义方法 Clear_Control。下面介绍本实例中用到的技术。

（1）Enter 事件

该事件表示当该控件处于当前窗体的活动控件时发生。语法格式如下：

```
public event EventHandler Enter
```

（2）Leave 事件

该事件表示当该控件不是当前窗体的活动控件时发生。语法格式如下：

```
public event EventHandler Leave
```

（3）KeyDown 事件

该事件表示在控件有焦点的情况下按下键时发生。语法格式如下：

```
public event KeyEventHandler KeyDown
```

（4）Clear_Control 方法

该方法是个自定义方法，用来遍历指定的控件。语法格式如下：

```
public void Clear_Control(Control.ControlCollection Con,int n,int m)
```

Clear_Control 方法中的参数说明如表 12.4 所示。

表 12.4　Clear_Control 方法中的参数说明

参　数	说　明
Con	表示当前的可视化控件
n	表示控件标识
m	表示控件的最大标识
返回值	该方法无返回值

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetFocus。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中主要添加 6 个文本框控件，用来显示焦点。

（3）程序的主要代码如下。

当前控件变为该窗体的活动控件时，使文本框的背景颜色变为蓝色。代码如下：

```
private void AllControl_Enter(object sender,EventArgs e)
{
    ((TextBox)sender).BackColor = Color.CornflowerBlue;                    //当前控件成为活动控件时设置它的背景颜色为蓝色
}
```

当前控件变为该窗体的非活动控件时，使文本框的背景颜色变为白色。代码如下：

```
private void AllControl_Leave(object sender,EventArgs e)
{
    ((TextBox)sender).BackColor = Color.White;                             //当前控件成为不活动控件时设置它的背景颜色为白色
}
```

当焦点处于文本框中，若有键被按下，则触发控件的 KeyDown 事件。本实例通过按键值判断是否按下 Enter 键。代码如下：

```
private void AllControl_KeyDown(object sender,KeyEventArgs e)
{
    if(e.KeyValue == 13)                                                   //当按下 Enter 键时
    {
        int n = Convert.ToInt32(((TextBox)sender).Tag.ToString());         //获取控件标识
        Clear_Control(groupBox1.Controls, n, 6);                           //进入下一个控件
    }
}
```

上面的代码中用到了 Clear_Control 方法，该方法用来遍历该窗体中指定类型的所有控件。代码如下：

```
/// <param Con="ControlCollection">可视化控件</param>
/// <param n="int">控件标识</param>
```

```
/// <param m="int">最大标识</param>
public void Clear_Control(Control.ControlCollection Con,int n,int m)
{
    int tem_n = 0;                                          //初始化一个 int 型变量
    foreach(Control C in Con)                               //遍历可视化组件中的所有控件
    {
        if(C.GetType().Name == "TextBox")                   //判断是否为 TextBox 控件
        {
            if(n == m)                                      //当循环至最后一个控件时
                tem_n = 1;                                  //设置控件标识的值为 1
            else                                            //当没有循环到最后一个控件时
                tem_n = n + 1;                              //使控件的标识值递增 1
            if(Convert.ToInt32(((TextBox)C).Tag.ToString()) == tem_n)
                                                            //若与当前控件关联的数据对象为下一个控件
                ((TextBox)C).Focus();                       //为当前控件设置焦点
        }
    }
}
```

秘笈心法

心法领悟 319：获取容器控件内的控件集合。

当遍历窗体上所有的或指定类型的控件时，需要注意这些控件是否被添加到某个容器控件中，若被添加到某个容器控件中，则需要遍历该容器控件内的控件集合。

实例 320　使用控件的 Tag 属性传递信息

初级

光盘位置：光盘\MR\12\320

趣味指数：★★★★☆

实例说明

用控件的 Tag 属性传值，只要在控件的属性窗口内给 Tag 属性设置值或通过编写代码给 Tag 属性赋值即可。本实例通过编写代码给 Button 控件的 Tag 属性赋值，当单击 Button 控件时，程序会弹出显示 Button 控件的 Tag 属性值的对话框。实例运行效果如图 12.31 所示。

图 12.31　使用控件的 Tag 属性传递信息

关键技术

本实例实现时主要用到了 Control 类的 Tag 属性，该属性用来获取或设置包含有关控件的数据的对象。语法格式如下：

```
public object Tag { get; set; }
```

参数说明

属性值：一个 Object 类型的对象，它包含有关控件的数据，默认为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetTag。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来演示通过 Tag 属性传递信息。

（3）程序的主要代码如下。

在窗体的 Load 事件中，给 Button 控件的 Tag 属性赋值。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    btn_Tag.Tag = "本技巧是 Tag 属性应用";                    //为按钮的数据对象赋值
}
```

单击 Button 控件，程序弹出显示 Button 控件的 Tag 属性值的对话框。代码如下：

```
private void btn_Tag_Click(object sender, EventArgs e)
{
    MessageBox.Show(                                          //弹出消息对话框，并显示 Button 控件的 Tag 属性值
        this.btn_Tag.Tag.ToString(),"提示！ ");
}
```

秘笈心法

心法领悟 320：使用 Form 窗体的 Tag 属性。

与普通控件一样，Form 窗体也继承自 Control 类，自然窗体也有 Tag 属性，这样使用窗体的 Tag 属性就可以实现在不同窗体间传递数据。

实例 321　为控件设置快捷键

光盘位置：光盘\MR\12\321　　初级　　趣味指数：★★★★☆

实例说明

在计算机操作中，快捷键的应用比较广泛。例如，在 Windows 操作系统的桌面上按 F1 键，系统会打开帮助文档。本实例为 Button 控件的单击操作设置快捷键为 Alt+D，运行程序，当同时按 Alt+D 快捷键时，程序会弹出显示“快捷键”3 个字的对话框。实例运行效果如图 12.32 所示。

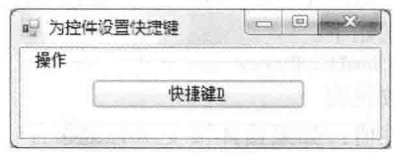


图 12.32　为控件设置快捷键

关键技术

本实例的关键技术是为控件设置快捷键的操作步骤，具体如下：

（1）单击要设置快捷键的控件，在控件的属性窗口中，首先设置 Text 属性要显示的基本信息。

（2）把输入法设置为英文输入状态，按 Shift+9 快捷键显示左括号，按 Shift+0 快捷键显示右括号。

（3）按 Shift+7 快捷键显示&特殊符号，最后在&后面输入快捷键的主体字母。例如，“(&D)”表示快捷键是 Alt+D。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShortCutMenu。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来测试快捷键。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("快捷键");                                //按 Alt+D 快捷键也可以弹出消息对话框
}
```

秘笈心法

心法领悟 321：关于快捷键。

控件的快捷键主体是&+字母，甚至可以省略括号，只是省略括号后的显示效果不够完美。

12.6 控件焦点变换

实例 322	按回车键变换控件焦点	中级
	光盘位置：光盘\MR\12\322	趣味指数：★★★★★

实例说明

开发软件时，在数据录入窗口中可能包含许多控件，用户录入数据时，如果利用鼠标定位光标位置，这样操作起来非常麻烦。本实例实现当用户输入数据时，按回车键控制鼠标焦点移动的功能，十分方便用户的操作。实例运行效果如图 12.33 所示。

图 12.33 按回车键变换控件焦点

关键技术

本实例实现时主要用到了窗体的 KeyPreview 属性、SelectNextControl 方法和 OnKeyPress 方法。下面分别进行介绍。

（1）KeyPreview 属性

该属性用于获取或设置一个值，该值指示在将键事件传递到具有焦点的控件前，窗体是否接受此键事件。语法格式如下：

```
public bool KeyPreview { get; set; }
```

参数说明

属性值：如果窗体接受所有键事件，则为 true；如果窗体上当前选定控件接受键事件，则为 false。默认为 false。

注意：当设置此属性为 true 时，窗体将接受所有 KeyPress、KeyDown 和 KeyUp 事件。在窗体的事件处理程序处理完该击键后，将该击键分配给具有焦点的控件。

（2）SelectNextControl 方法

该方法激活下一个控件。语法格式如下：

```
public bool SelectNextControl (Control ctl,bool forward,bool tabStopOnly,bool nested, bool wrap)
```

SelectNextControl 方法的参数说明如表 12.5 所示。

表 12.5 SelectNextControl 方法的参数说明

参　数	说　明
ctl	从其上开始搜索的 Control
forward	如果为 true，则在 Tab 键顺序中前移；如果为 false，则在 Tab 键顺序中后移
tabStopOnly	true 表示忽略 TabStop 属性设置为 false 的控件，false 表示不忽略
nested	true 表示包括嵌套子控件（子控件的子级），false 表示不包括
wrap	true 表示在到达最后一个控件之后 Tab 键顺序中第一个控件开始继续搜索，false 表示不继续搜索
返回值	如果控件已激活，则为 true；否则为 false

（3）OnKeyPress 方法

该方法触发 KeyPress 事件。语法格式如下：

```
protected override void OnKeyPress (KeyPressEventArgs e)
```

参数说明

e：所引发的 KeyPressEventArgs。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChangeFocus。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 6 个 TextBox 控件，用来显示焦点的移动。

（3）程序的主要代码如下：

```
protected override void OnKeyPress(KeyPressEventArgs e)
{
    if (e.KeyChar == 13)                                                    //判断是否按下 Enter 键
    {
        this.SelectNextControl(this.ActiveControl, true, true, true, true);  //激活下一个控件
    }
    base.OnKeyPress(e);                                                     //调用基类的 OnKeyPress 方法
}
```

秘笈心法

心法领悟 322：重写基类的虚方法。

在重写基类的虚方法时，在当前程序允许的情况下，可以调用基类的虚方法，这样能够保证基类中该虚方法的某些功能得到执行，如本实例中的 base.OnKeyPress(e)语句。

实例 323　程序运行时拖动控件

光盘位置：光盘\MR\12\323　　高级　　趣味指数：★★★★☆

实例说明

一般的软件在运行时，控件的位置都是固定的，例如按钮，用户不能通过鼠标拖动这些控件，但在一些程序中恰恰需要实现这种功能，例如拼图游戏。本实例将介绍如何实现在程序运行时拖动控件的功能，运行本实例，用鼠标拖动按钮，按钮的位置将会随着鼠标位置的移动而变化。实例运行效果如图 12.34 所示。

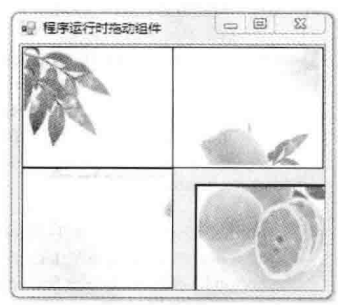


图 12.34　程序运行时拖动控件

关键技术

本实例实现时主要用到了窗体的 AllowDrop 属性、DoDragDrop 方法、DragDrop 事件、DragEnter 事件和 Button 控件的 MouseDown 事件。下面介绍本实例中用到的技术。

（1）AllowDrop 属性

该属性用于获取或设置一个值，该值指示控件是否可以接收用户拖放到它上面的数据。语法格式如下：

```
public virtual bool AllowDrop { get; set; }
```

参数说明

属性值：如果控件允许拖放操作，则为 true；否则为 false。默认为 false。

（2）DoDragDrop 方法

该方法表示开始拖放操作。语法格式如下：

```
public DragDropEffects DoDragDrop(object data, DragDropEffects allowedEffects)
```

参数说明

❶ data：要拖动的数据。

❷ allowedEffects：DragDropEffects 枚举值。

❸ 返回值：DragDropEffects 枚举的值，它表示在拖放操作期间执行的最终效果。

（3）DragDrop 事件

该事件在拖放操作完成时发生。语法格式如下：

```
public event DragEventHandler DragDrop
```

（4）DragEnter 事件

该事件在用鼠标将某项拖动到该控件的工作区时发生。语法格式如下：

```
public event DragEventHandler DragEnter
```

（5）MouseDown 事件

该事件当鼠标指针在组件上方并按下鼠标按钮时发生。语法格式如下：

```
public event MouseEventHandler MouseDown
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DragControl。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 Button 控件，用来演示在运行状态下的拖放操作。

（3）程序的主要代码如下。

开始移动控件的实现代码如下（这里仅以 button1 按钮为例）：

```
private void button1_MouseDown(object sender, MouseEventArgs e)
{
    button1.Tag = e.Location;                                    //得到按钮中鼠标的坐标
    DoDragDrop(button1, DragDropEffects.Move);                   //开始拖放操作
}
```

判断对哪个按钮进行操作和计算按钮移动后位置的代码如下：

```
private void Form1_DragDrop(object sender, DragEventArgs e)
{
    //判断对哪个按钮操作
    object data = e.Data.GetData(typeof(Button));
    if (data == button1)
    {
        button1.Top = this.PointToClient(                        //设置 button1 距容器上边缘之间的距离
            new Point(e.X, e.Y)).Y - ((Point)button1.Tag).Y;
        button1.Left = this.PointToClient(                       //设置 button1 距容器左边缘之间的距离
            new Point(e.X, e.Y)).X - ((Point)button1.Tag).X;
    }
    if (data == button2)
    {
        button2.Top = this.PointToClient(                        //设置 button2 距容器上边缘之间的距离
            new Point(e.X, e.Y)).Y - ((Point)button2.Tag).Y;
        button2.Left = this.PointToClient(                       //设置 button2 距容器左边缘之间的距离
            new Point(e.X, e.Y)).X - ((Point)button2.Tag).X;
    }
    if (data == button3)
    {
        button3.Top = this.PointToClient(                        //设置 button3 距容器上边缘之间的距离
            new Point(e.X, e.Y)).Y - ((Point)button3.Tag).Y;
        button3.Left = this.PointToClient(                       //设置 button3 距容器左边缘之间的距离
            new Point(e.X, e.Y)).X - ((Point)button3.Tag).X;
    }
    if (data == button4)
    {
        button4.Top = this.PointToClient(                        //设置 button4 距容器上边缘之间的距离
            new Point(e.X, e.Y)).Y - ((Point)button4.Tag).Y;
        button4.Left = this.PointToClient(                       //设置 button4 距容器左边缘之间的距离
            new Point(e.X, e.Y)).X - ((Point)button4.Tag).X;
    }
}
```

设置以何种方式移动的代码如下：

```
private void Form1_DragEnter(object sender, DragEventArgs e)
{
    e.Effect = DragDropEffects.Move;                             //将拖动源的数据移动到放置目标
}
```

秘笈心法

心法领悟 323：设置控件的位置。

在设置控件的位置时，可以使用控件公共属性 Location 的 X 和 Y 属性，也可以使用控件的 Top、Bottom、Left 和 Right 属性。

实例 324　控件得到焦点时变色

光盘位置：光盘\MR\12\324　　中级　　趣味指数：★★★★☆

实例说明

本实例设计了一个得到焦点时文本框变色的程序，从而达到了标记和提醒用户输入的目的。实例运行效果如图 12.35 所示。

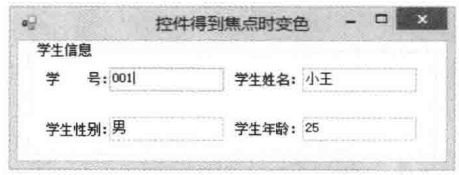


图 12.35　控件得到焦点时变色

关键技术

本实例实现时主要用到了 TextBox 控件的 Enter 事件、Leave 事件和 BackColor 属性。下面介绍本实例中用到的技术。

（1）Enter 事件

该事件在鼠标进入控件时发生。语法格式如下：

```
public event EventHandler Enter
```

（2）Leave 事件

该事件在输入焦点离开控件时发生。语法格式如下：

```
public event EventHandler Leave
```

（3）BackColor 属性

该属性用于获取或设置控件的背景色。语法格式如下：

```
public override Color BackColor { get; set; }
```

参数说明

属性值：表示控件背景色的 Color。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中主要添加 4 个文本框控件，用来演示得到焦点变色的效果。

（3）程序的主要代码如下。

当窗体的文本框控件得到焦点，则背景色变为米色；失去焦点，则背景色变为白色，代码如下：

```
private void textBox1_Enter(object sender, EventArgs e)
{
    textBox1.BackColor = Color.Beige;                    //设置控件背景颜色为米色
}
private void textBox1_Leave(object sender, EventArgs e)
{
    textBox1.BackColor = Color.White;                    //设置控件背景颜色为白色
}
```

秘笈心法

心法领悟 324：关于文本框的总结。

若文本框控件得到焦点，则将触发 Enter 事件；若文本框失去焦点，则将触发 Leave 事件。

第13章

DataGridView 数据控件

- DataGridView 控件基本设置
- DataGridView 控件实用开发

13.1　DataGridView 控件基本设置

实例 325　设置 DataGridView 控件中网格线的样式

光盘位置：光盘\MR\13\325　　初级　　趣味指数：★★★☆

实例说明

本实例实现把 DataGridView 控件绑定到数据集合，并通过设置控件的 GridColor 属性将控件的网格线设置为蓝色。实例运行效果如图 13.1 所示。

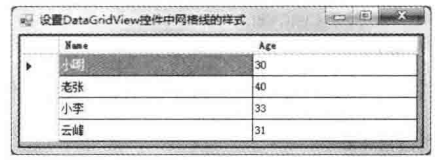


图 13.1　设置 DataGridView 控件中网格线的样式

关键技术

本实例主要通过设置 DataGridView 控件的 GridColor 属性值来设置网格线的外观样式。GridColor 属性用来获取和设置网格线的颜色。语法格式如下：

```
public Color GridColor { get; set; }
```

参数说明

属性值：表示网格线颜色的 Color 对象或 SystemColors 对象，默认为 ControlDarkDark 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GridStyle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.GridColor = Color.Blue;                    //设置网格颜色
    dgv_Message.DataSource = new List<Student>() {         //绑定到数据集合
    new Student(){Name="小明",Age=30},
    new Student(){Name="老张",Age=40},
    new Student(){Name="小李",Age=33},
    new Student(){Name="云峰",Age=31}};
    dgv_Message.Columns[0].Width = 200;                    //设置列宽
    dgv_Message.Columns[1].Width = 170;                    //设置列宽
}
```

秘笈心法

心法领悟 325：使用集合初始化器。

在 C# 3.0 中，引入了集合初始化器的新概念。在实例化集合的同时，即可初始化集合元素。

实例 326　设置 DataGridView 控件中字体的样式

光盘位置：光盘\MR\13\326　　初级　　趣味指数：★★★☆

实例说明

本实例实现把 DataGridView 控件绑定到数据集合，并通过设置该控件公共属性 DefaultCellStyle 的 Font 属性来设置 DataGridView 控件中字体的样式。实例运行效果如图 13.2 所示。

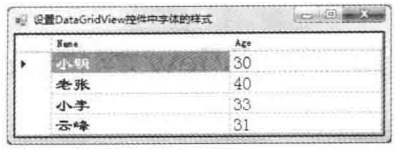


图 13.2　设置 DataGridView 控件中字体的样式

关键技术

本实例实现时主要用到了 DataGridView 控件公共属性 DefaultCellStyle 的 Font 属性，该属性实现获取或设置应用于 DataGridView 单元格的文本内容的字体。语法格式如下：

```
public Font Font { get; set; }
```

参数说明

属性值：应用于单元格文本的 Font 实例，默认为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FontStyle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Student>() {              //绑定到数据集合
    new Student(){Name="小明",Age=30},
    new Student(){Name="老张",Age=40},
    new Student(){Name="小李",Age=33},
    new Student(){Name="云峰",Age=31}};
    dgv_Message.Columns[0].Width = 200;                         //设置列宽
    dgv_Message.Columns[1].Width = 170;                         //设置列宽
    dgv_Message.DefaultCellStyle.Font = new Font("隶书",15);    //设置网格中字体的样式

}
```

秘笈心法

心法领悟 326：使用对象和集合初始化器。

在 C# 3.0 中，引入了对象初始化器的新概念。在实例化对象的同时，即可初始化对象的公共成员（如属性、公共字段等）。

实例 327　在 DataGridView 控件中设置数据显示格式

初级

光盘位置：光盘\MR\13\327

趣味指数：★★★★

实例说明

本实例实现把 DataGridView 控件绑定到数据集合，并通过设置该控件的列的 DefaultCellStyle.Format 属性来设置该列单元格的数据显示格式。实例运行效果如图 13.3 所示。

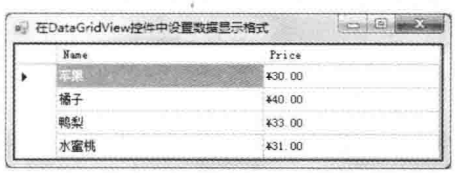


图 13.3　在 DataGridView 控件中设置数据显示格式

关键技术

本实例主要用到了 DataGridViewCellStyle 类的 Format 属性，该属性用来获取或设置应用于 DataGridView 单元格的文本内容的格式字符串。语法格式如下：

```
public string Format { get; set; }
```

参数说明

属性值：指示单元格的值的格式的字符串。默认为 System.String.Empty。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DataFormat。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.Columns[1].DefaultCellStyle.Format = "c";           //设置内容格式
}
```

秘笈心法

心法领悟 327：把 DataGridView 控件绑定到数据集合。

通常情况下，把 DataGridView 控件绑定到数据库中的数据表，但也可以把控件绑定到数据集合。例如，本实例把 DataGridView 控件绑定到 List 泛型集合。

实例 328　设置 DataGridView 单元格的文本对齐方式

初级

光盘位置：光盘\MR\13\328

趣味指数：★★★☆

实例说明

本实例实现把 DataGridView 控件绑定到数据集合，并通过设置控件的列的 DefaultCellStyle.Alignment 属性来设置 DataGridView 单元格的文本对齐方式，本实例设置为居中对齐。实例运行效果如图 13.4 所示。

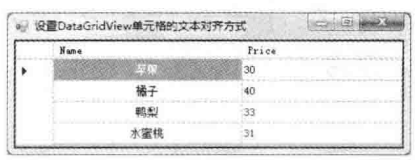


图 13.4　设置 DataGridView 单元格的文本对齐方式

关键技术

本实例主要用到了 DataGridViewCellStyle 类的 Alignment 属性，该属性用来获取或设置一个值，该值指示 DataGridView 单元格内的单元格内容的位置。语法格式如下：

```
public DataGridViewContentAlignment Alignment { get; set; }
```

参数说明

属性值：DataGridViewContentAlignment 类型的枚举值之一。该类型的枚举值及说明如表 13.1 所示。

表 13.1　DataGridViewContentAlignment 枚举值及说明

枚 举 值	说　明
NotSet	未设定对齐方式
TopLeft	内容与单元格的顶部垂直对齐，与单元格的左侧水平对齐
TopCenter	内容与单元格的顶部垂直对齐，与单元格的中间水平对齐
TopRight	内容与单元格的顶部垂直对齐，与单元格的右侧水平对齐
MiddleLeft	内容与单元格的中间垂直对齐，与单元格的左侧水平对齐
MiddleCenter	内容与单元格的垂直和水平中心对齐
MiddleRight	内容与单元格的中间垂直对齐，与单元格的右侧水平对齐
BottomLeft	内容与单元格的底部垂直对齐，与单元格的左侧水平对齐
BottomCenter	内容与单元格的底部垂直对齐，与单元格的中间水平对齐
BottomRight	内容与单元格的底部垂直对齐，与单元格的右侧水平对齐

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CellAlignment。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.Columns[0].DefaultCellStyle.Alignment =             //设置对齐方式
        DataGridViewContentAlignment.MiddleCenter;
}
```

秘笈心法

心法领悟 328：动态设置列宽。

通常情况下，在 DataGridView 控件的属性窗口中设置各列的宽度，但有时需要根据实际的数据长度来设置列宽，这可以通过编写代码动态设置 DataGridView 控件的列的 Width 属性值来解决。

实例 329　在 DataGridView 控件中验证数据输入

光盘位置：光盘\MR\13\329　　初级　　趣味指数：★★★★

实例说明

在 DataGridView 控件的单元格中可以编辑数据，但有时需要控制单元格输入的数据类型。本实例中，在 DataGridView 控件的第二列的单元格中只允许输入数字，否则系统的验证将无法通过。实例运行效果如图 13.5 所示。

图 13.5　在 DataGridView 控件中验证数据输入

关键技术

实现本实例的功能主要是利用 DataGridView 控件的公共

事件 CellValidating 和 CellEndEdit 来处理数据。CellValidating 事件在单元格失去输入焦点时发生，并启用内容验证功能。CellEndEdit 事件在为当前选定的单元格停止编辑模式时发生。在 CellValidating 事件中进行数据验证，如果验证失败，将 System.Windows.Forms.DataGridViewCellValidatingEventArgs 类的 Cancel 属性设置为 true，这将导致 DataGridView 控件阻止光标离开该单元格。另外，在 CellValidating 事件中，还需要将该行的 ErrorText 属性设置为解释性字符串，这将显示错误图标，其工具提示将包含此错误文本。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CellValidate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据和编辑单元格的数据。

（3）程序的主要代码如下：

```
private void dataGridView1_CellValidating(object sender, DataGridViewCellValidatingEventArgs e)
{
    if (e.ColumnIndex == 1)                                          //验证指定列
    {
        float result = 0;                                            //定义值类型变量并赋值
        if (!float.TryParse(e.FormattedValue.ToString(), out result))  //判断数据是否为数值类型
        {
            dgv_Message.Rows[e.RowIndex].ErrorText ="内容必须为数值类型";  //提示错误信息
            e.Cancel = true;                                         //取消事件的值
        }
    }
}
```

秘笈心法

心法领悟 329：使用 TryParse 方法转换并验证类型。

在应用程序开发中，常遇到将字符串转换为对应的值类型或验证字符串是否可转为某种值类型的情况，这时可以考虑使用该种值类型的 TryParse 方法，并且该方法是静态的。

实例 330　在 DataGridView 的单元格中启用换行

初级

光盘位置：光盘\MR\13\330

趣味指数：★★★★☆

实例说明

当 DataGridView 控件的某个单元格中显示了大量的文本数据时，可考虑把文本数据换行。本实例实现在 DataGridView 控件基于文本的单元格中启用换行的功能。实例运行效果如图 13.6 所示。

图 13.6　在 DataGridView 的单元格中启用换行

关键技术

本实例主要用到了 DataGridView 控件公共属性 DefaultCellStyle 的 WrapMode 属性，该属性用来获取或设置一个值，该值指示当 DataGridView 单元格中的文本内容太长而不能放在单行中时，是将它换到下一行还是将它截断。语法格式如下：

```
public DataGridViewTriState WrapMode { get; set; }
```

参数说明

属性值：DataGridViewTriState 类型的枚举值之一。该类型的枚举值及说明如表 13.2 所示。

表 13.2　DataGridViewTriState 枚举值及说明

枚 举 值	说　明
NotSet	该属性尚未设置，将采用默认方式
true	该属性的状态为 true，表示换行
false	该属性的状态为 false，表示不换行

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CellNewLine。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.Columns[0].DefaultCellStyle.Alignment =             //设置对齐方式
        DataGridViewContentAlignment.MiddleCenter;
    dgv_Message.DefaultCellStyle.WrapMode = DataGridViewTriState.True;  //换行显示过长的文本内容
    dgv_Message.Rows[3].Height = 30;                                //设置行高度
}
```

秘笈心法

心法领悟 330：动态设置行高。

在本实例窗体中，最后一行数据的第一单元格由于数据量大而进行了换行设置，这样 DataGridView 控件初始的行高就无法满足换行后的需求，可以通过编写代码动态设置 DataGridView 控件的行的 Height 属性值来解决。

实例 331　禁止 DataGridView 控件中添加和删除行

初级

光盘位置：光盘\MR\13\331

趣味指数：★★★☆

实例说明

本实例通过设置 DataGridView 控件的相关属性，实现禁止在 DataGridView 控件中添加行、删除行及编辑单元格。实例运行效果如图 13.7 所示。

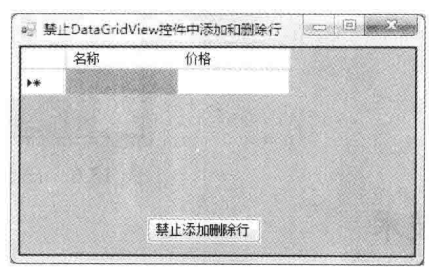


图 13.7　禁止在 DataGridView 控件中添加和删除行

关键技术

本实例实现时主要用到了 DataGridView 控件的 AllowUserToAddRows、AllowUserToDeleteRows 和 ReadOnly 属性，下面分别对这些属性进行介绍。

（1）AllowUserToAddRows 属性

该属性用于获取或设置一个值，该值指示是否向用户显示添加行的选项。语法格式如下：

```
public bool AllowUserToAddRows { get; set; }
```

参数说明

属性值：如果向用户显示“添加行”选项，为 true；否则为 false。默认为 true。

（2）AllowUserToDeleteRows 属性

该属性用于获取或设置一个值，该值指示是否允许用户从 DataGridView 中删除行。语法格式如下：

```
public bool AllowUserToDeleteRows { get; set; }
```

参数说明

属性值：如果用户可以删除行，为 true；否则为 false。默认为 true。

（3）ReadOnly 属性

该属性用来获取一个值，该值指示用户是否可以编辑 DataGridView 控件的单元格。语法格式如下：

```
public bool ReadOnly { get; set; }
```

参数说明

属性值：如果用户不能编辑 DataGridView 控件的单元格，为 true；否则为 false。默认为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NoAddAndRemoveLine。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据；添加一个 Button 控件，用来设置 DataGridView 控件的相关属性。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)          //为 DataGridView 控件动态添加列
{
    dgv_Message.Columns.Add("Name", "名称");                    //向控件中添加列
    dgv_Message.Columns.Add("Price", "价格");                   //向控件中添加列
}
private void btn_No_Click(object sender, EventArgs e)           //设置 DataGridView 相关属性
{
    dgv_Message.AllowUserToAddRows = false;                     //禁止添加行
    dgv_Message.AllowUserToDeleteRows = false;                  //禁止删除行
    dgv_Message.ReadOnly = true;                                //设置单元格为不可编辑
}
```

秘笈心法

心法领悟 331：在 DataGridView 控件中动态添加列。

在应用程序开发中，有时需要向 DataGridView 控件中动态添加列，这可以通过调用 DataGridView 控件的公共属性 Column 的 Add 方法来实现。

实例 332　禁用 DataGridView 控件列表头自动排序功能

初级

光盘位置：光盘\MR\13\332

趣味指数：★★★★☆

实例说明

本实例通过设置 DataGridView 控件的列的相关属性，实现禁用 DataGridView 控件列表头自动排序的功能。实例运行效果如图 13.8 所示。

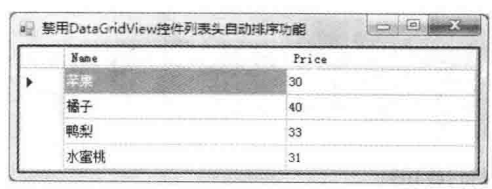


图 13.8　禁用 DataGridView 控件列表头自动排序功能

关键技术

本实例实现时主要用到了 DataGridView 控件的列的 SortMode 属性，该属性用于获取或设置列的排序模式。语法格式如下：

```
public DataGridViewColumnSortMode SortMode { get; set; }
```

参数说明

属性值：DataGridViewColumnSortMode 类型的枚举值之一。该类型的枚举值及说明如表 13.3 所示。

表 13.3 DataGridViewColumnSortMode 枚举值及说明

枚 举 值	说 明
NotSortable	此列仅能以编程方式进行排序，但由于它原本并不打算排序，所以列表头将不包含排序标志符号的空间
Automatic	除非列表头用于进行选择，否则用户可以通过单击列表头对列进行排序，排序标志符号将自动显示出来
Programmatic	此列仅能以编程方式进行排序，并且列表头将包含排序标志符号的空间

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NoSort。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
//自定义类，用于描述水果的属性
class Fruit
{
    public string Name { get; set; }                                    //水果的名称
    public float Price { get; set; }                                    //水果的单价
}
//窗体的 Load 事件
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                        //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                                 //设置列宽度
    dgv_Message.Columns[1].Width = 170;                                 //设置列宽度
    dgv_Message.Columns[0].SortMode = DataGridViewColumnSortMode.NotSortable;   //设置列排序模式
    dgv_Message.Columns[1].SortMode = DataGridViewColumnSortMode.NotSortable;   //设置列排序模式
}
```

秘笈心法

心法领悟 332：关于自动实现属性。

自动实现属性是 C# 3.0 引入的新概念，这种新特性减少了重复性的工作量，极大地提高了开发效率。

13.2 DataGridView 控件实用开发

实例 333 对 DataGridView 控件进行数据绑定 中级

光盘位置：光盘\MR\13\333 趣味指数：★★★☆

实例说明

DataGridView 控件既可以绑定到数据表，也可以绑定到数据集合。本实例实现绑定 DataGridView 控件到数据集合。实例运行效果如图 13.9 所示。

图 13.9　对 DataGridView 控件进行数据绑定

关键技术

本实例实现时主要用到了 DataGridView 控件的 DataSource 属性，该属性用于获取或设置 DataGridView 控件所显示数据的数据源。语法格式如下：

```
public object DataSource { get; set; }
```

参数说明

属性值：包含 DataGridView 控件要显示的数据的对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GridBind。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定到数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.Columns[0].DefaultCellStyle.Alignment =             //设置对齐方式
        DataGridViewContentAlignment.MiddleCenter;
}
```

秘笈心法

心法领悟 333：DataGridView 控件可以绑定多种数据源。

DataGridView 控件可以绑定多种数据源，例如，DataTable 实例、DataView 实例、BindingSource 组件、数据集合等。

实例 334　DataGridView 选中单元格时整个行背景变色　初级

光盘位置：光盘\MR\13\334　趣味指数：★★★★☆

实例说明

用 DataGridView 控件显示数据记录，当用鼠标选中某一行时，若当前行的背景发生颜色变换，则非常利于使用者查看当前行的数据，本实例即实现了这种功能。实例运行效果如图 13.10 所示。

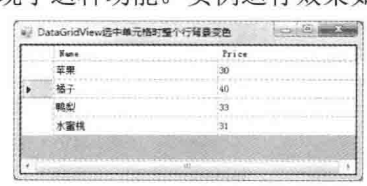


图 13.10　DataGridView 选中单元格时整个行背景变色

关键技术

本实例实现时主要用到了 DataGridView 控件的 SelectionMode 属性、DataGridView 控件公共属性 DefaultCellStyle 的 SelectionForeColor 属性及 SelectionBackColor 属性，下面对本实例中用到的关键技术进行详细讲解。

（1）DataGridView 控件的 SelectionMode 属性

该属性用于获取或设置一个值，该值指示如何选择 DataGridView 的单元格。语法格式如下：

```
public DataGridViewSelectionMode SelectionMode { get; set; }
```

参数说明

属性值：DataGridViewSelectionMode 类型的枚举值之一。该类型的枚举值及说明如表 13.4 所示。

表 13.4 DataGridViewSelectionMode 枚举值及说明

枚 举 值	说 明
CellSelect	可以选定一个或多个单元格
FullRowSelect	通过单击行的表头或该行所包含的单元格选定整个行
FullColumnSelect	通过单击列的表头或该列所包含的单元格选定整个列
RowHeaderSelect	通过单击行的表头单元格选定此行。通过单击某个单元格可以单独选定此单元格
ColumnHeaderSelect	可以通过单击列的表头单元格选定此列。通过单击某个单元格可以单独选定此单元格

（2）DataGridView 控件公共属性 DefaultCellStyle 的 SelectionForeColor 属性

该属性用于获取或设置 DataGridView 单元格在被选定时的前景色。语法格式如下：

```
public Color SelectionForeColor { get; set; }
```

参数说明

属性值：返回 System.Drawing.Color 的实例，它表示选定单元格的前景色。

（3）DataGridView 控件公共属性 DefaultCellStyle 的 SelectionBackColor 属性

该属性用于获取或设置 DataGridView 单元格在被选定时的背景色。语法格式如下：

```
public Color SelectionBackColor { get; set; }
```

参数说明

属性值：返回 System.Drawing.Color 的实例，它表示选定单元格的背景色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChangeColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.SelectionMode =                                     //设置如何选择单元格
        DataGridViewSelectionMode.FullRowSelect;
    dgv_Message.DefaultCellStyle.SelectionForeColor= Color.Blue;    //选中单元格的前景色
    dgv_Message.DefaultCellStyle.SelectionBackColor= Color.LightYellow;  //选中单元格的背景色
}
```

秘笈心法

心法领悟 334：设置选中行的颜色。

为了使选中行的显示效果更加鲜明和突出，在设置选中行的颜色时，最好同时设置前景色和背景色。

实例 335　在 DataGridView 控件中隔行换色

光盘位置：光盘\MR\13\335　　中级　　趣味指数：★★★★☆

实例说明

在 DataGridView 控件中，若能够实现奇数行与偶数行的颜色交替变换，可以大大增强控件的显示效果，而且还有利于查看数据记录。实例运行效果如图 13.11 所示。

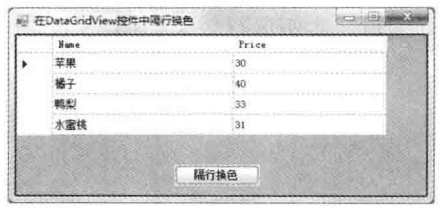


图 13.11　在 DataGridView 控件中隔行换色

关键技术

本实例的设计思路是通过设置奇数行（即行索引值为偶数的行）的背景色来达到隔行换色的目的。本实例实现时主要用到了 DataGridViewRow 类的公共属性 DefaultCellStyle 的 BackColor 属性，该属性用于获取或设置 DataGridView 单元格的背景色。语法格式如下：

```
public Color BackColor { get; set; }
```

参数说明

属性值：返回 System.Drawing.Color 的实例，它表示单元格的背景色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AlternationColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据；添加一个 Button 控件，用来设置隔行换色。

（3）程序的主要代码如下：

```
//DataGridView 控件绑定数据集合
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
    dgv_Message.SelectionMode = DataGridViewSelectionMode.FullRowSelect;  //设置如何选中单元格
}
private void btn_Begin_Click(object sender, EventArgs e)            //单击按钮实现隔行换色
{
    for (int i = 0; i < dgv_Message.Rows.Count; i++)                //遍历所有的行
    {
        if (i % 2 == 0)                                             //判断行索引值为偶数
            dgv_Message.Rows[i].DefaultCellStyle.BackColor = Color.LightYellow;  //隔行更换背景色
    }
}
```

秘笈心法

心法领悟 335：使用模数运算符（%）。

模数运算符（%）计算第二个操作数除第一个操作数后的余数，所有数值类型都具有预定义的模数运算符。本实例巧用模数运算符解决了判断奇偶数的问题。

实例 336 在 DataGridView 控件中显示下拉列表

光盘位置：光盘\MR\13\336　　中级　　趣味指数：★★★★★

实例说明

本实例实现在 DataGridView 控件的单元格中显示下拉列表，使用下拉列表方便地对 DataGridView 控件中的数据进行编辑。实例运行效果如图 13.12 所示。

图 13.12　在 DataGridView 控件中显示下拉列表

关键技术

本实例实现时主要用到了 DataGridViewComboBoxColumn 类和 DataGridView 控件的 Columns 属性的 Add 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）关于 DataGridViewComboBoxColumn 类

该类是 DataGridView 控件的列类型中的一种，可以通过该类型创建下拉列表样式的列，它继承自 DataGridViewColumn 类。

（2）DataGridView 控件的 Columns 属性的 Add 方法

该方法实现将给定的列添加到集合，它的重载形式有两种，本实例用到的方法如下：

```
public virtual int Add(DataGridViewColumn dataGridViewColumn)
```

参数说明

❶ dataGridViewColumn：表示要添加的列对象。该参数值通常是 DataGridViewColumn 派生子类的实例，例如，DataGridViewComboBoxColumn 类的实例。

❷ 返回值：该列的索引。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DropDownList。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    DataGridViewComboBoxColumn dgvc = new DataGridViewComboBoxColumn();    //创建列对象
    dgvc.Items.Add("苹果");                                                //向集合中添加元素
    dgvc.Items.Add("芒果");                                                //向集合中添加元素
    dgvc.Items.Add("鸭梨");                                                //向集合中添加元素
    dgvc.Items.Add("橘子");                                                //向集合中添加元素
    dgvc.HeaderText = "水果";                                              //设置列标题文本
    dgv_Message.Columns.Add(dgvc);                                         //将列添加到集合
}
```

秘笈心法

心法领悟 336：DataGridView 的下拉列表绑定代码表。

DataGridViewComboBoxColumn 类表示 DataGridView 控件的列类型的一种，它派生自 DataGridViewColumn 类。该类的实例在界面和操作上与 ComboBox 控件十分相似，并且像 ComboBox 控件一样可以绑定代码表。

实例 337　在 DataGridView 控件中显示图片

光盘位置：光盘\MR\13\337　　中级　　趣味指数：★★★★★

实例说明

通过 DataGridView 控件可以很好地与数据库实现交互，当显示某一事物的详细信息时，如果可以直接看到该事物的图片，那么可以使用户对该事物有一个更直观的了解。通过对本实例的学习，用户可以实现这个效果。实例运行效果如图 13.13 所示。

图 13.13　在 DataGridView 控件中显示图片

关键技术

本实例实现时主要用到了 DataGridView 控件的 DataSource 属性，关于 DataSource 属性的介绍请参见实例 333。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisplayPictures。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Images>()                    //绑定到图片集合
    {
        new Images(){Im=Image.FromFile("1.bmp")},
        new Images(){Im=Image.FromFile("2.bmp")},
        new Images(){Im=Image.FromFile("3.bmp")},
        new Images(){Im=Image.FromFile("4.bmp")},
        new Images(){Im=Image.FromFile("5.bmp")},
        new Images(){Im=Image.FromFile("6.bmp")},
        new Images(){Im=Image.FromFile("7.bmp")}
    };
    dgv_Message.Columns[0].HeaderText = "图片";                    //设置列文本
    dgv_Message.Columns[0].Width = 70;                             //设置列宽度
    for (int i = 0; i < dgv_Message.Rows.Count; i++)
    {
        dgv_Message.Rows[i].Height = 70;                           //设置行高度
    }
}
```

秘笈心法

心法领悟 337：如何获取图像？

可以通过调用 Image 类的 FromFile 方法来获取图像实例，该方法是个静态方法。

实例 338　为 DataGridView 控件实现复选功能

光盘位置：光盘\MR\13\338　　中级　　趣味指数：★★★★☆

实例说明

本实例实现在 DataGridView 控件的单元格中添加复选框，当用户对 DataGridView 控件中的数据进行筛选时，可以通过选中复选框来筛选数据记录。实例运行效果如图 13.14 所示。

图 13.14　为 DataGridView 控件实现复选功能

关键技术

本实例实现时主要用到了 DataGridViewCheckBoxColumn 类和 DataGridView 控件的 Columns 属性的 Add 方法，下面对本实例中用到的关键技术进行详细讲解。

DataGridViewCheckBoxColumn 类是 DataGridView 控件的列类型中的一种，可以通过该类型创建复选框样式的列，它继承自 DataGridViewColumn 类。

说明：关于 Add 方法的详细说明请参见实例 336。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CheckBoxInDataGridView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据；添加一个 Button 控件，用来删除选中复选框的数据行。

（3）程序的主要代码如下。

在窗体的 Load 事件中，DataGridView 控件绑定数据集合。代码如下：

```
private List<Fruit> P_Fruit;
private void Frm_Main_Load(object sender, EventArgs e)
{
    DataGridViewCheckBoxColumn dgvc = new DataGridViewCheckBoxColumn();          //创建列对象
    dgvc.HeaderText = "状态";                                                    //设置列标题文本
    dgv_Message.Columns.Add(dgvc);                                               //添加列
    P_Fruit = new List<Fruit>() {                                                //创建数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.DataSource = P_Fruit;                                            //绑定数据集合
    dgv_Message.Columns[0].Width = 50;                                           //设置列宽度
    dgv_Message.Columns[1].Width = 170;                                          //设置列宽度
    dgv_Message.Columns[2].Width = 150;                                          //设置列宽度
}
```

单击窗体上的按钮，实现删除选中复选框所在的数据行。代码如下：

```
private void btn_Remove_Click(object sender, EventArgs e)
{
    for (int i = 0; i < dgv_Message.Rows.Count; i++)                             //遍历行集合
    {
        if (dgv_Message.Rows[i].Cells[0].Value != null && dgv_Message.Rows[i].Cells[1].Value != null &&
            dgv_Message.Rows[i].Cells[2].Value != null )                         //判断值是否为空
        {
            if (Convert.ToBoolean(dgv_Message.Rows[i].Cells[0].Value.ToString()))//判断是否选中项
            {
                P_Fruit.RemoveAll(                                               //标记集合中的指定项
                    (pp) =>
                    {
                        if (pp.Name == dgv_Message.Rows[i].Cells[1].Value.ToString() &&
                            pp.Price == Convert.ToSingle(dgv_Message.Rows[i].Cells[2].Value.ToString()))
                            pp.ft = true;                                        //开始标记
                        return false;                                            //不删除项
```

```
                    });
                }
            }
        }
        P_Fruit.RemoveAll(                                          //删除集合中的指定项
            (pp) =>
            {
                return pp.ft;
            });
        dgv_Message.DataSource = null;                              //绑定为空
        dgv_Message.DataSource = P_Fruit;                           //绑定到数据集合
        dgv_Message.Columns[0].Width = 50;                          //设置列宽度
        dgv_Message.Columns[1].Width = 170;                         //设置列宽度
        dgv_Message.Columns[2].Width = 150;                         //设置列宽度
    }
```

秘笈心法

心法领悟 338：Lambda 表达式。

在本实例中应用了 Lambda 表达式，Lambda 表达式支持用语句块或表达式作为方法体，语法上比匿名方法更灵活。

实例 339　在 DataGridView 控件中添加“合计”和“平均值”　高级

光盘位置：光盘\MR\13\339　　趣味指数：★★★★★

实例说明

本实例实现为 DataGridView 控件中的第一个列的所有行求和，为第二个列的所有行求平均数。实例运行效果如图 13.15 所示。

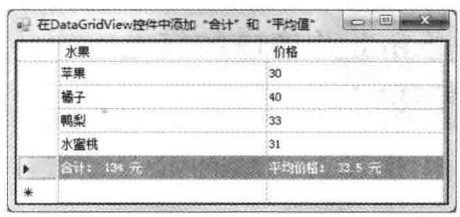


图 13.15　在 DataGridView 控件中添加“合计”和“平均值”

关键技术

本实例在实现时主要用到了 List<T>的 ForEach 方法，该方法实现对 List<T>的每个元素执行指定操作，本实例使用该方法来计算 float 类型元素的和。语法格式如下：

```
public void ForEach(Action<T> action)
```

参数说明

action：要对 List<T>的每个元素执行的 System.Action<T>委托。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SumAndAverage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示绑定的数据。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    G_Fruit = new List<Fruit>() {                                   //创建集合并添加元素
```

```
        new Fruit(){Name="苹果",Price=30},
        new Fruit(){Name="橘子",Price=40},
        new Fruit(){Name="鸭梨",Price=33},
        new Fruit(){Name="水蜜桃",Price=31}};
        dgv_Message.Columns.Add("Fruit", "水果");                    //添加列
        dgv_Message.Columns.Add("Pric", "价格");                     //添加列
        foreach (Fruit f in G_Fruit)                                 //添加元素
        {
            dgv_Message.Rows.Add(new string[]
            {
                f.Name,
                f.Price.ToString()
            });
        }
        dgv_Message.Columns[0].Width = 200;                          //设置列宽度
        dgv_Message.Columns[1].Width = 170;                          //设置列宽度
        float sum = 0;                                               //定义 float 类型变量
        G_Fruit.ForEach(
            (pp) =>
            {
                sum += pp.Price;                                     //求和
            });
        dgv_Message.Rows.Add(new string[]                            //在新列中显示平均值及合计信息
        {
            "合计：  "+sum.ToString()+" 元",
            "平均价格：  "+(sum/G_Fruit.Count).ToString()+" 元"
        });
    }
```

秘笈心法

心法领悟 339：填充 DataGridView 控件的单元格。

在使用 DataGridView 控件显示数据时，可以使用字符串数组为 DataGridView 控件的新增行填充单元格。

实例 340　使用主从表查看生产单详细信息

光盘位置：光盘\MR\13\340　　中级　　趣味指数：★★★★☆

实例说明

在数据库应用程序开发中，经常需要将主从表关联起来查询数据。本实例把生产厂家和商品通过厂家编号关联起来，然后输出各种商品的名称、开始生产日期及生产厂家等相关信息。实例运行效果如图 13.16 所示。

图 13.16　使用主从表查看生产单详细信息

关键技术

本实例在实现时主要用到了 OleDbDataAdapter 类的构造方法和 DbDataAdapter 类的 Fill 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）OleDbDataAdapter 类的构造方法

该方法实现创建一个适配器对象，该对象用于填充数据集和更新数据源。该方法的重载形式有多种，本实例用到的方法如下：

```
public OleDbDataAdapter(string selectCommandText, string selectConnectionString)
```

参数说明

❶ selectCommandText：一个字符串，它是 SQL Select 语句或将由 OleDbDataAdapter 的 SelectCommand 属性使用的存储过程。

❷ selectConnectionString：数据库连接字符串。

（2）DbDataAdapter 类的 Fill 方法

该方法实现在 DataSet 的指定范围中添加或刷新行，以匹配使用 DataSet、DataTable 和 IDataReader 名称的数据源中的行。该方法的重载形式有多种，本实例用到的方法如下：

```
public int Fill(DataTable dataTable)
```

参数说明

❶ dataTable：用于表映射的 DataTable 的名称。

❷ 返回值：已在 DataSet 中成功添加或刷新的行数，这不包括受不返回行的语句影响的行。

说明：OleDbDataAdapter 类继承自 DbDataAdapter 抽象基类，所以 OleDbDataAdapter 类的对象可以调用抽象基类的 Fill 方法。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FromTable。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示数据；添加一个 Button 控件，用来实现查询数据。

（3）程序的主要代码如下。

自定义类 DataTier，在该类中定义一个 GetDate 方法，该方法实现从主从数据表中获取 DataTable 类型的数据源。代码如下：

```
public DataTable GetDate()
{
    string P_Connection = string.Format(                                    //创建数据库连接字符串
      "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
    OleDbDataAdapter P_DataAdapter = new OleDbDataAdapter(                  //创建数据适配器对象
        @"select id as 编号,Name as 名称,Begin as 开始时间,
        Factory as 配件厂家名称,Phone as 电话,Address as 联系地址 from [tb_Ware]
        inner join [tb_Number] on [tb_Ware].Number=[tb_Number].Number",P_Connection);
    DataTable dt = new DataTable();                                         //创建数据表
    P_DataAdapter.Fill(dt);                                                 //填充数据表
    return dt;                                                              //返回数据表
}
```

单击窗体中的“查询详细信息”按钮，实现把数据源绑定到 DataGridView 控件。代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    dgv_Message.DataSource = new DataTier().GetDate();                      //绑定到数据表
}
```

秘笈心法

心法领悟 340：关联主从表。

编写关联主从表的 Transact-SQL 语句，可使用 Inner Join 逻辑运算符。

实例 341　将 DataGridView 中数据导出到 Word

光盘位置：光盘\MR\13\341　　高级　　趣味指数：★★★★★

实例说明

Microsoft Word 具有很强的文档处理功能，将 C#程序与 Word 文档相结合，就能够设计出更加强大的应用

程序。本实例通过使用 Microsoft Word 自动化对象模型中的 Cell 对象，将 DataGridView 控件中的数据导出到 Word 文档表格中。实例运行效果如图 13.17 所示。

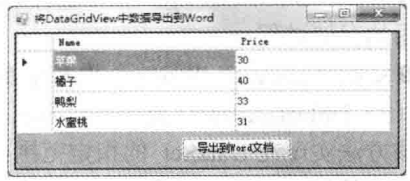


图 13.17　将 DataGridView 中数据导出到 Word

关键技术

本实例在实现时主要用到了 Word 对象模型中 Tables 接口的 Add 方法和 Table 接口的 Cell 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）Tables 接口的 Add 方法

该方法实现在文档中添加指定行数和列数的表格。语法格式如下：

```
Table Add(Range Range, int NumRows, int NumColumns, ref object DefaultTableBehavior, ref object AutoFitBehavior)
```

Add 方法的参数及说明如表 13.5 所示。

表 13.5　Add 方法的参数及说明

枚 举 值	说　明
Range	表示文档范围
NumRows	Word 表格的行数
NumColumns	Word 表格的列数
DefaultTableBehavior	文档中表格行为
AutoFitBehavior	文档中表格格式
返回值	Word 表格对象

（2）Table 接口的 Cell 方法

该方法实现获取 Word 表格中指定行和列的表格。语法格式如下：

```
Cell Cell(int Row, int Column)
```

参数说明

❶ Row：指定的行。

❷ Column：指定的列。

❸ 返回值：Word 表格中指定行和列的表格。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GridToWord。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示数据；添加一个 Button 控件，用来实现把 DataGridView 控件中的数据导出到 Word 文档。

（3）程序的主要代码如下。

在窗体的 Load 事件中，首先绑定 DataGridView 控件到数据集合。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
}
```

单击窗体中的“导出到 Word 文档”按钮，实现把 DataGridView 控件中的数据导出到 Word 文档。代码如下：

```
private Word.Application G_wa;                                      //定义 Word 应用程序字段
private object G_missing = System.Reflection.Missing.Value;         //定义 G_missing 字段并添加引用
private void btn_OutPut_Click(object sender, EventArgs e)
{
    List<Fruit> P_Fruit = new List<Fruit>();                        //创建数据集合
```

```
    foreach (DataGridViewRow dgvr in dgv_Message.Rows)
    {
        P_Fruit.Add(new Fruit()                                          //向数据集合添加数据
        {
            Name = dgvr.Cells[0].Value.ToString(),
            Price = Convert.ToSingle(dgvr.Cells[1].Value.ToString())
        });
    }
    SaveFileDialog P_SaveFileDialog = new SaveFileDialog();              //创建保存文件对话框对象
    P_SaveFileDialog.Filter = "*.doc|*.doc";
    if (DialogResult.OK == P_SaveFileDialog.ShowDialog())                //确认是否保存文件
    {
        ThreadPool.QueueUserWorkItem(                                    //开始线程池
        (pp) =>                                                          //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();      //创建应用程序对象
            object P_obj = "Normal.dot";                                 //定义文档模板
            Word.Document P_wd = G_wa.Documents.Add(                     //向 Word 应用程序中添加文档
                ref P_obj, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Range(                             //得到文档范围
                ref G_missing, ref G_missing);
            object o1 = Word.WdDefaultTableBehavior.wdWord8TableBehavior; //设置文档中表格的格式
            object o2 = Word.WdAutoFitBehavior. wdAutoFitWindow;         //设置文档中表格的格式
            Word.Table P_WordTable = P_Range.Tables.Add(P_Range,         //在文档中添加表格
                P_Fruit.Count ,2, ref o1, ref o2);
            P_WordTable.Cell(1, 1).Range.Text = "水果";                  //向表格中添加信息
            P_WordTable.Cell(1, 2).Range.Text = "价格";                  //向表格中添加信息
            for (int i = 2; i < P_Fruit.Count + 1; i++)
            {
                P_WordTable.Cell(i, 1).Range.Text = P_Fruit[i - 2].Name;          //向表格中添加信息
                P_WordTable.Cell(i, 2).Range.Text = P_Fruit[i - 2].Price.ToString(); //向表格中添加信息
            }
            object P_Path = P_SaveFileDialog.FileName;
            P_wd.SaveAs(                                                 //保存 Word 文件
                ref P_Path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                  //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                 //调用窗体线程
                (MethodInvoker)(() =>                                    //使用 Lambda 表达式
                {
                    MessageBox.Show("成功创建 Word 文档！", "提示！");      //弹出消息对话框
                }));
        });
    }
}
```

秘笈心法

心法领悟 341：使用 Lambda 表达式。

在本实例中应用了 Lambda 表达式，带有表达式体的 Lambda 表达式可以转换为表达式树。

实例 342　将 DataGridView 中数据导出到 Excel

高级

光盘位置：光盘\MR\13\342

趣味指数：★★★★☆

实例说明

Microsoft Excel 具有强大的数据统计功能，并且使用它设计报表简单方便、美观实用，所以在软件开发中常需要把应用程序中的数据导出到 Excel 文件中。本实例实现把 DataGridView 中的数据导出到 Excel。实例运行

效果如图 13.18 所示。

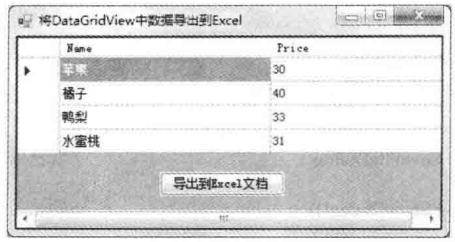


图 13.18　将 DataGridView 中数据导出到 Excel

关键技术

本实例在实现时主要用到了 Microsoft Excel 对象模型的 Workbook 接口的 Add 方法，该方法实现创建一个新的工作簿。语法格式如下：

```
Workbook Add(object Template)
```

参数说明

❶ Template：工作簿的模板。

❷ 返回值：新建的工作簿实例。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GridToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 DataGridView 控件，用来显示数据；添加一个 Button 控件，用来实现把 DataGridView 控件中的数据导出到 Excel 文档。

（3）程序的主要代码如下。

在窗体的 Load 事件中，首先绑定 DataGridView 控件到数据集合。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    dgv_Message.DataSource = new List<Fruit>() {                    //绑定数据集合
    new Fruit(){Name="苹果",Price=30},
    new Fruit(){Name="橘子",Price=40},
    new Fruit(){Name="鸭梨",Price=33},
    new Fruit(){Name="水蜜桃",Price=31}};
    dgv_Message.Columns[0].Width = 200;                             //设置列宽度
    dgv_Message.Columns[1].Width = 170;                             //设置列宽度
}
```

单击窗体中的“导出到 Excel 文档”按钮，实现把 DataGridView 控件中的数据导出到 Excel 文档。代码如下：

```
private Excel.Application G_ea;                                     //定义 Word 应用程序字段
private object G_missing = System.Reflection.Missing.Value;         //定义 G_missing 字段并添加引用
private void btn_OutPut_Click(object sender, EventArgs e)
{
    List<Fruit> P_Fruit = new List<Fruit>();                        //创建数据集合
    foreach (DataGridViewRow dgvr in dgv_Message.Rows)
    {
        P_Fruit.Add(new Fruit()                                     //向数据集合添加数据
        {
            Name = dgvr.Cells[0].Value.ToString(),
            Price = Convert.ToSingle(dgvr.Cells[1].Value.ToString())
        });
    }
    SaveFileDialog P_SaveFileDialog = new SaveFileDialog();         //创建保存文件对话框对象
    P_SaveFileDialog.Filter = "*.xls|*.xls";
    if (DialogResult.OK == P_SaveFileDialog.ShowDialog())           //确认是否保存文件
    {
        ThreadPool.QueueUserWorkItem(                               //开始线程池
        (pp) =>                                                     //使用 Lambda 表达式
        {
```

```
            G_ea = new Microsoft.Office.Interop.Excel.Application();          //创建应用程序对象
            Excel.Workbook P_wk = G_ea.Workbooks.Add(G_missing);              //创建 Excel 文档
            Excel.Worksheet P_ws = (Excel.Worksheet)P_wk.Worksheets.Add(G_missing, G_missing, G_missing, G_missing);//创建工作区域

            for (int i = 0; i < P_Fruit.Count; i++)
            {
                P_ws.Cells[i + 1, 1] = P_Fruit[i].Name;                       //向 Excel 文档中写入内容
                P_ws.Cells[i + 1, 2] = P_Fruit[i]. Price.ToString();          //向 Excel 文档中写入内容
            }
            P_wk.SaveAs(                                                      //保存 Word 文件
                P_SaveFileDialog.FileName, G_missing, G_missing, G_missing,
                G_missing, G_missing, Excel.XlSaveAsAccessMode.xlShared, G_missing,
                G_missing, G_missing, G_missing, G_missing);
            ((Excel._Application)G_ea.Application).Quit();                    //退出应用程序
            this.Invoke(                                                      //调用窗体线程
                (MethodInvoker)(() =>                                         //使用 Lambda 表达式
                {
                    MessageBox.Show("成功创建 Excel 文档！", "提示！"); //弹出消息对话框
                }));
        });
    }
}
```

秘笈心法

心法领悟 342：使用 ThreadPool 创建线程。

在应用程序开发中，如果做一些比较耗时的工作或启动其他应用程序，那么可以通过线程池创建一个新的线程来处理这些任务，这样可以防止因主线程繁忙而出现“假死”现象。

实例 343　像 Excel 一样复制 DataGridView 中数据

高级

光盘位置：光盘\MR\13\343

趣味指数：★★★★★

实例说明

在使用 DataGridView 控件显示数据时，有时需要对无规律性的数据进行显示，在这种情况下是无法用代码进行筛选的，本实例完成在 DataGridView 控件中像 Excel 一样对单元格复制的操作，以实现显示任意选择区域的单元格。实例运行效果如图 13.19 所示。

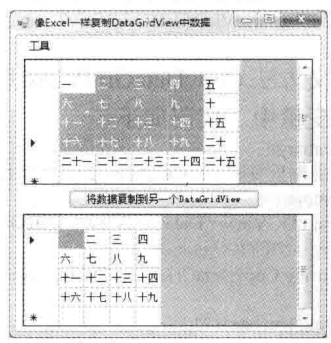


图 13.19　像 Excel 一样复制 DataGridView 中数据

关键技术

本实例在实现时主要用到了 Clipboard 类，该类提供将数据置于系统剪贴板中以及从中检索数据的方法。在本实例的实现过程中，首先把在 DataGridView 控件中选中的内容添加到剪贴板中，然后再用 Clipboard 类读取剪贴板中的信息。Clipboard 类的常用方法及说明如表 13.6 所示。

表 13.6　Clipboard 类的常用方法及说明

枚 举 值	说　明
Clear	从剪贴板中移除所有数据
ContainsAudio	指示在剪贴板中是否存在 WaveAudio 格式的数据
ContainsData	指示剪贴板中是否存在指定格式的数据，或可转换成此格式的数据
ContainsFileDropList	指示剪贴板中是否存在 FileDrop 格式或可转换成此格式的数据

续表

枚 举 值	说 明
ContainsImage	指示剪贴板中是否存在 Bitmap 格式或可转换成此格式的数据
ContainsText	已重载。指示剪贴板中是否存在文本数据
GetAudioStream	检索剪贴板上的音频流
GetData	从剪贴板中检索指定格式的数据
GetDataObject	检索当前位于系统剪贴板中的数据
GetFileDropList	从剪贴板中检索文件名的集合
GetImage	检索剪贴板上的图像
GetText	已重载。从剪贴板中检索文本数据
SetAudio	已重载。将 WaveAudio 格式的数据添加到剪贴板中
SetData	将指定格式的数据添加到剪贴板中
SetDataObject	已重载。将数据置于系统剪贴板中
SetFileDropList	将 FileDrop 格式的文件名集合添加到剪贴板中
SetImage	将 Bitmap 格式的 Image 添加到剪贴板中
SetText	已重载。将文本数据添加到剪贴板中

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetGridData。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 DataGridView 控件，分别用来显示源数据和粘贴的数据；添加一个 Button 控件，用来实现复制数据。

（3）程序的主要代码如下。

在 dataGridView1 控件中选择要复制的单元格，用 button1 控件的 Click 事件将 dataGridView1 控件中所选择的单元格内容添加到 dataGridView2 控件中。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    str = CopyDataGridView(dataGridView1);                    //复制选中的数据
    AddDataGridView(dataGridView2, str, Bool_Blank, Bool_All);  //粘贴数据
}
```

自定义方法 CopyDataGridView 用于将选中的单元格内容添加到剪贴板中，然后将剪贴板中的文本信息存入到字符串变量中。代码如下：

```
/// <summary>
///通过剪贴板复制 DataGridView 控件中所选中的内容
/// </summary>
/// <param DGView="DataGridView">DataGridView 类</param>
/// <return>字符串</return>
public string CopyDataGridView(DataGridView DGView)
{
    string tem_str = "";
    if (DGView.GetCellCount(DataGridViewElementStates.Selected) > 0)
    {
        try
        {
            Clipboard.SetDataObject(DGView.GetClipboardContent());   //将数据添加到剪贴板中
            tem_str = Clipboard.GetText();                           //从剪贴板中获取信息
        }
        catch (System.Runtime.InteropServices.ExternalException)
        {
            return "";
        }
    }
    return tem_str;
}
```

自定义方法 AddDataGridView 是通过在剪贴板中获得的文本信息，按照文本中字符串的相应格式，以及单

元格的相应位置，将其添加到 dataGridView2 控件中。代码如下：

```
/// <summary>
///将字符串按指定的格式添加到 DataGridView 控件中（如果有被选中的单元格，则修改单元格中的内容）
/// </summary>
/// <param DGView="DataGridView">DataGridView 类</param>
/// <param s="string">要替换的单元格字符串</param>
/// <param Blank="bool">标识，如果不是空格是否替换成空格</param>
/// <param All="bool">标识，是否全部替换</param>
public void AddDataGridView(DataGridView DGView, string s, bool Blank, bool All)
{
    string tem_str = s;
    int tem_n = 0;
    int RowCount = 0;                                                    //行数
    int CellCount = 0;                                                   //列数
    bool tem_bool = true;
    string tem_s = "";
    if (s.IndexOf("\r\n") != -1)                                         //如果替换的为多行
        while (tem_bool)                                                 //获取单元格的行数和列数
        {
            tem_s = "";
            if (tem_str.IndexOf("\r\n") != -1)                           //如果获取的不是最后一行
            {
                tem_s = tem_str.Substring(0, tem_str.IndexOf("\r\n") + 2);   //获取当前行中的数据
                //获取当前行中能被识别的数据
                tem_str = tem_str.Substring(tem_str.IndexOf("\r\n") + 2, tem_str.Length - tem_str.IndexOf("\r\n") – 2);

                tem_n = 1;
                while (tem_s.IndexOf("\t") > -1)                         //遍历当前行中的空格
                {
                    //去除已读取的空格
                    tem_s = tem_s.Substring(tem_s.IndexOf("\t") + 1, tem_s.Length - tem_s.IndexOf("\t") - 1);
                    tem_n += 1;                                          //获取列数
                }
                if (tem_n > CellCount)                                   //判断当前列数是否为最大列数
                    CellCount = tem_n;                                   //获取最大的列数
            }
            else                                                         //如果读取的是最后一行
            {
                tem_n = 1;
                while (tem_s.IndexOf("\t") > -1)
                {
                    tem_s = tem_s.Substring(tem_s.IndexOf("\t") + 1, tem_s.Length - tem_s.IndexOf("\t") - 1);
                    tem_n += 1;
                }
                if (tem_n > CellCount)
                    CellCount = tem_n;
                tem_bool = false;                                        //遍历结束
            }
            ++RowCount;                                                  //读取行数
        }
    else                                                                 //如果读取的为单行数据
    {
        tem_n = 1;
        tem_s = s;
        while (tem_s.IndexOf("\t") > -1)
        {
            tem_s = tem_s.Substring(tem_s.IndexOf("\t") + 1, tem_s.Length - tem_s.IndexOf("\t") - 1);
            tem_n += 1;
        }
        if (tem_n > CellCount)
            CellCount = tem_n;
        ++RowCount;                                                      //读取行数
    }
    string[,] Strarr = new string[RowCount, CellCount];                  //定义一个数组，用于记录复制的单元格信息
    tem_str = s;
    tem_n = 0;
    //将单元格信息添加到数组中
    for (int i = 0; i < RowCount; i++)                                   //遍历单元格的行
```

```
{
    for (int j = 0; j < CellCount; j++)                          //遍历单元格的列
    {
        tem_s = "";
        if (tem_str.IndexOf("\r\n") != -1)                       //如果不是最后一行
        {
            if (tem_str.IndexOf("\t") <= -1)                     //设置读取数据的位置
                tem_n = tem_str.IndexOf("\r");                   //最后一个数据的位置
            else
                tem_n = tem_str.IndexOf("\t");                   //不是最后一个数据的位置
            //读取单元格数据
            tem_s = tem_str.Substring(0, tem_str.IndexOf("\r\n") + 2);
        }
        else                                                     //如果是最后一行
        {
            if (tem_str.IndexOf("\t") <= -1)                     //设置读取数据的位置
                tem_n = tem_str.Length;                          //最后一个数据的位置
            else
                tem_n = tem_str.IndexOf("\t");                   //不是最后一个数据的位置
            tem_s = tem_str;                                     //读取单元格数据
        }
        if (tem_s.Length > 0)                                    //如果当前行有数据
        {
            if (tem_s.Substring(0, 1) == "\t")                   //如果第一个字符为空
                Strarr[i, j] = "";                               //向数组中添加一个空记录
            else
            {
                Strarr[i, j] = tem_s.Substring(0, tem_n);        //向数组中添加数据
            }
        }
        else
            Strarr[i, j] = "";                                   //向数组中添加空记录
        if (tem_s.Length > tem_n)                                //如果记录没有读取完
            //获取没有读取的记录
            tem_str = tem_s.Substring(tem_n + 1, tem_s.Length - tem_n - 1);
    }
    if (s.IndexOf("\r\n") > -1)                                  //如果不是最后一行数据
    {
        //读取下一行数据
        s = s.Substring(s.IndexOf("\r\n") + 2, s.Length - s.IndexOf("\r\n") - 2);
        tem_str = s;
    }
}
if (All)                                                         //如果要全部替换
    DGView.Rows.Clear();                                         //清空 DataGridView 控件
//如果 DataGridView 中没有数据
if (DGView.SelectedRows.Count == 0 && DGView.SelectedCells.Count == 0)
{
    DGView.ColumnCount = CellCount;                              //设置列数
    string[] stra = new string[CellCount];                       //定义一个一维数组
    //向 DataGridView 中添加行数据
    for (int i = 0; i < RowCount; i++)                           //读取行
    {
        for (int j = 0; j < CellCount; j++)                      //读取列
        {
            if (Strarr[i, j] == "")                              //如果当前单元格为空
            {
                if (Blank)                                       //如果用*号替换
                    stra[j] = "*";                               //在空单元格中添加*号
                else
                    stra[j] = "";                                //以空格显示空单元格
            }
            else
            {
                stra[j] = Strarr[i, j];                          //记录当前行中的信息
            }
        }
        DGView.Rows.Add(stra);                                   //将行中的信息添加到 DataGridView 控件
```

```
        }
        DGView.AutoResizeColumns();                                    //向 DataGridView 中添加所有的单元格信息
        //将所选择的单元格复制到剪贴板中
        DGView.ClipboardCopyMode = DataGridViewClipboardCopyMode.EnableWithoutHeaderText;
    }
    else                                                               //如果 DataGridView 中有数据
    {
        int maxrow = 0;                                                //记录 DataGridView 中最小单元格的行数
        int maxcell = 0;                                               //记录 DataGridView 中最小单元格的列数
        //获取选中单元格中最大单元格的行数和列数
        for (int i = 0; i < DGView.SelectedCells.Count; i++)
        {
            if (DGView.SelectedCells[i].RowIndex > maxrow)             //如果单元格的行数大于当前指定的行数
                maxrow = DGView.SelectedCells[i].RowIndex;             //记录当前单元格的行数
            if (DGView.SelectedCells[i].ColumnIndex > maxcell)         //如果单元格的列数大于当前指定的列数
                maxcell = DGView.SelectedCells[i].ColumnIndex;         //记录当前单元格的列数
        }
        int minrow = maxrow;                                           //记录 DataGridView 中最大单元格的行数
        int mincell = maxcell;                                         //记录 DataGridView 中最大单元格的列数
        //获取选中单元格中最小单元格的行数和列数
        for (int i = 0; i < DGView.SelectedCells.Count; i++)
        {
            if (DGView.SelectedCells[i].RowIndex < minrow)             //如果单元格的行数小于当前指定的行数
                minrow = DGView.SelectedCells[i].RowIndex;             //记录当前单元格的行数
            if (DGView.SelectedCells[i].ColumnIndex < mincell)         //如果单元格的列数小于当前指定的列数
                mincell = DGView.SelectedCells[i].ColumnIndex;         //记录当前单元格的列数
        }
        //向 DataGridView 控件中添加选中单元格中最小单元格与最大单元格中的所有单元格
        for (int i = 0; i < maxrow - (minrow - 1); i++)                //遍历行数
        {
            if (i >= RowCount)                                         //如果超出要添加的行数
                break;                                                 //退出循环
            for (int j = 0; j < maxcell - (mincell - 1); j++)          //遍历列数
            {
                if (j >= CellCount)                                    //如果超出要添加的列数
                    break;                                             //退出循环
                if (Strarr[i, j]=="")                                  //如果添加的单元格为空
                {
                    if (Blank)                                         //如果用*号替换空格
                        //用*号替换空单元格
                        DGView.Rows[i + minrow].Cells[j + mincell].Value = "*";
                }
                else
                    //设置当前单元格的值
                    DGView.Rows[i + minrow].Cells[j + mincell].Value = Strarr[i, j];
            }
        }
    }
}
```

秘笈心法

心法领悟 343：DataGridView 控件的 AutoResizeColumns 方法。

DataGridView 控件的 AutoResizeColumns 方法能够实现调整所有列的宽度，以适应其所有单元格（包括标题单元格）的内容。

实例 344　从 DataGridView 中拖放数据到 TreeView　高级

光盘位置：光盘\MR\13\344　　趣味指数：★★★★★

实例说明

日常操作中，通过 DataGridView 控件可以显示数据库中的记录，由于受记录数目和界面美观的限制，对于

多条记录的显示，DataGridView 控件存在很多不足。例如，一条记录中有很多不同分类信息，直接看到它的全部内容几乎不可能，此时必须拖动滚动条才能看到，这样做虽然看到了内容，但有些麻烦，本实例通过 C#程序实现选定记录拖至 TreeView 控件直接显示。首先运行本实例，然后选定目标记录，接着按下鼠标左键，移动鼠标至 TreeView 控件，最后显示数据记录。实例运行效果如图 13.20 所示。

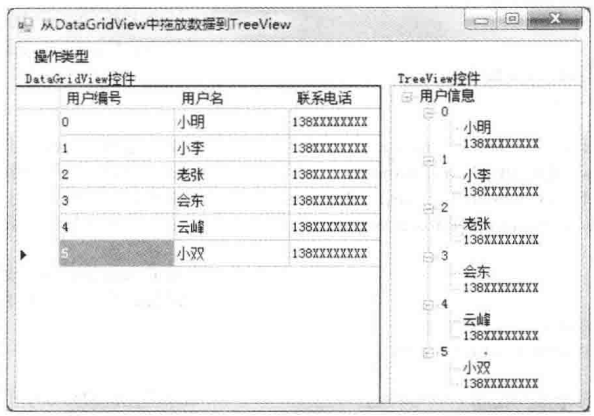


图 13.20　从 DataGridView 中拖放数据到 TreeView

关键技术

本实例主要用到了 DataGridView 控件的 MouseDown 事件和 TreeView 控件的 MouseEnter 事件，下面对本实例中用到的关键技术进行详细讲解。

（1）DataGridView 控件的 MouseDown 事件的处理方法

MouseDown 事件是指当鼠标指针在 DataGridView 控件上方并按下鼠标按键时发生，该事件在本实例中的处理方法如下：

```
private void dataGridView1_MouseDown(object sender, MouseEventArgs e)
```

参数说明

❶ sender：表示支持.NET Framework 类层次结构中所有类的基类对象。

❷ e：表示为 MouseDown 事件提供数据。

（2）TreeView 控件的 MouseEnter 事件的处理方法

MouseEnter 事件是指当鼠标进入 TreeView 控件可见部分时发生，该事件在本实例中的处理方法如下：

```
private void treeView1_MouseEnter(object sender, EventArgs e)
```

参数说明

❶ sender：表示支持.NET Framework 类层次结构中所有类的基类对象。

❷ e：表示为 MouseEnter 事件提供数据。

技巧：在进行拖放操作时，既可以选定单行，又可以选定多行。方法一：先选定一条记录，在 DataGridView 控件中默认状态下是行标题选定；然后按 Ctrl 键，选定需要的记录，拖放到 TreeView 控件上。方法二：选定多条连续记录中任意一个端点记录，按 Shift 键，选中另一端点记录，拖放到 TreeView 控件上。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DateToTreeView。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中首先添加一个 DataGridView 控件，设置其 AllowUserToAddRows 属性为 false，主要用来显示数据库中数据的记录；添加一个 TreeView 控件，设置其 AllowDrop 属性为 true，主要用来接收从 DataGridView 控件中拖放的数据记录。

（3）程序的主要代码如下。

Frm_Main 窗体加载时，首先从数据库中读取记录显示在 DataGridView 控件中。代码如下：

```
private void Form1_Load(object sender,EventArgs e)
{
    string P_Connection = string.Format(                                    //创建数据库连接字符串
     "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb;User Id=Admin");
    OleDbDataAdapter P_OLeDbDataAdapter = new OleDbDataAdapter(
        "select au_id as 用户编号,au_lname as 用户名,phone as 联系电话   from authors",
        P_Connection);                                                      //创建 OleDbDataAdapter 类的对象
    DataSet ds = new DataSet();                                             //创建数据集对象
    P_OLeDbDataAdapter.Fill(ds,"UserInfo");                                 //把数据填充到数据集中
    dataGridView1.DataSource = ds.Tables["UserInfo"].DefaultView;           //DataGridView 控件绑定数据源
    TreeNode treeNode = new TreeNode("用户信息",0,0);                       //创建根节点
    treeView1.Nodes.Add(treeNode);                                          //TreeView 控件添加根节点
    追加节点 ToolStripMenuItem.Checked = true;                              //默认情况下设置为追加节点状态
}
```

在 DataGridView 控件上按下鼠标按键，通过二维数组 recordInfo 保存选定记录。代码如下：

```
//DataGridView 的按下鼠标事件
private void dataGridView1_MouseDown(object sender,MouseEventArgs e)
{
    if (dataGridView1.SelectedCells.Count != 0)                             //判断 DataGridView 控件中是否有选定记录
    {
        //定义一个二维数组，数组中的每一行代表 DataGridView 中的一条记录
        recordInfo = new string[dataGridView1.Rows.Count, dataGridView1.Columns.Count];
        //当按下鼠标左键时，首先获取选定行，记录每一行对应的信息
        for (int i = 0; i < dataGridView1.Rows.Count; i++)
        {
            if (dataGridView1.Rows[i].Selected)                             //判断 DataGridView 中是否有选中行
            {
                for (int j = 0; j < dataGridView1.Columns.Count; j++)       //循环遍历 DataGridView 中选定行的每一列内容
                {
                    //用数组 recordInfo 记录选定信息
                    recordInfo[i, j] = dataGridView1.Rows[i].Cells[j].Value.ToString();
                }
            }
        }
    }
}
```

把鼠标移动到 TreeView 控件上，遍历数组 recordInfo 中的每一条记录，添加到 TreeView 控件下。代码如下：

```
private void treeView1_MouseEnter(object sender, EventArgs e)               //当鼠标进入 TreeView 控件时触发的操作
{
    if (追加节点 ToolStripMenuItem.Checked == true)                         //判断拖放操作的类型是否为追加节点
    {
        if (recordInfo != null && recordInfo.Length != 0)                   //判断数组 recordInfo 是否存在以及是否存在内容
        {
            for (int i = 0; i < recordInfo.GetLength(0); i++)               //用双重 for 循环遍历数组 recordInfo 中的内容
            {
                for (int j = 0; j < recordInfo.GetLength(1); j++)
                {
                    if (recordInfo[i, j] != null)                           //判断数组中的值是否为空
                    {
                        if (j == 0)                                         //当循环遍历至第 0 列时
                        {
                            //指定的标签文本初始化 TreeNode 对象
                            TreeNode Node1 = new TreeNode(recordInfo[i, j].ToString());
                            treeView1.SelectedNode.Nodes.Add(Node1);        //将先前创建的树节点添加到树节点集合的末尾
                            treeView1.SelectedNode = Node1;                 //设置当前树视图控件中选定节点为刚创建完的节点
                        }
                        else
                        {
                            //指定的标签文本初始化 TreeNode 对象
                            TreeNode Node2 = new TreeNode(recordInfo[i, j].ToString());
                            treeView1.SelectedNode.Nodes.Add(Node2);        //将先前创建的树节点添加到树节点集合的末尾
                        }
                    }
```

```
                }
                treeView1.SelectedNode = treeView1.Nodes[0];                    //设置当前树视图控件中的选定节点为根节点
                treeView1.ExpandAll();                                          //在 TreeView 控件中展开所有节点
            }
            for (int m = 0; m < recordInfo.GetLength(0); m++)                   //用循环遍历数组 recordInfo 清空 recordInfo 中的记录
            {
                for (int n = 0; n < recordInfo.GetLength(1); n++)
                {
                    recordInfo[m, n] = null;                                    //设定数组 recordInfo 中的内容为不存在
                }
            }
        }
    }
    if (清空内容 ToolStripMenuItem.Checked == true)                             //判断拖放操作的类型是否为追加节点
    {
        if (treeView1.SelectedNode.Nodes.Count != 0)                            //判断数组 recordInfo 是否存在内容
        {
            treeView1.SelectedNode.Remove();                                    //清空 TreeView 控件中的内容
            TreeNode treeNode = new TreeNode("用户信息", 0, 0);                 //指定的标签文本初始化 TreeNode 对象
            treeView1.Nodes.Add(treeNode);                                      //将先前创建的树节点添加到树节点集合的末尾
            treeView1.SelectedNode = treeNode;                                  //设置当前树控件中的选定节点为刚创建完的节点
            if (recordInfo != null && recordInfo.Length != 0)                   //判断数组 recordInfo 是否存在以及是否存在内容
            {
                for (int i = 0; i < recordInfo.GetLength(0); i++)               //用双重 for 循环遍历数组 recordInfo 中的内容
                {
                    for (int j = 0; j < recordInfo.GetLength(1); j++)
                    {
                        if (recordInfo[i, j] != null)                           //判断数组中的值是否为空
                        {
                            if (j == 0)                                         //当循环遍历至第 0 列时
                            {
                                //指定的标签文本初始化 TreeNode 对象
                                TreeNode Node1 = new TreeNode(recordInfo[i, j].ToString());
                                treeView1.SelectedNode.Nodes.Add(Node1);//将先前创建的树节点添加到树节点集合的末尾
                                treeView1.SelectedNode = Node1;                 //设置当前树视图控件中的选定节点为刚创建完的节点
                            }
                            else
                            {
                                //指定的标签文本初始化 TreeNode 对象
                                TreeNode Node2 = new TreeNode(recordInfo[i, j].ToString());
                                treeView1.SelectedNode.Nodes.Add(Node2);//将先前创建的树节点添加到树节点集合的末尾
                            }
                        }
                    }
                    treeView1.SelectedNode = treeView1.Nodes[0];                //设置当前树视图控件中的选定节点为根节点
                    treeView1.ExpandAll();                                      //在 TreeView 控件中展开所有节点
                }
                for (int m = 0; m < recordInfo.GetLength(0); m++)               //清空 recordInfo 中的记录
                {
                    for (int n = 0; n < recordInfo.GetLength(1); n++)
                    {
                        recordInfo[m, n] = null;                                //设定数组 recordInfo 中的内容为不存在
                    }
                }
            }
            追加节点 ToolStripMenuItem.Checked = true;
            清空内容 ToolStripMenuItem.Checked = false;
        }
    }
}
```

秘笈心法

心法领悟 344：判断 DataGridView 中的某行是否被选定。

通过 DataGridView 控件的行对象（即 DataGridViewRow 类的实例）的 Selected 属性可以判断该行是否被选定。如果该行被选定，则 Selected 属性值为 true，否则为 false。

第14章

自定义用户控件

- 重绘 Windows 标准控件
- 自定义 Windows 控件

14.1 重绘 Windows 标准控件

实例 345	美化 ComboBox 控件下拉列表	高级
	光盘位置：光盘\MR\14\345	趣味指数：★★★★★

实例说明

ComboBox 控件在实际开发中经常会被使用，在实际生活中也随处可见。例如，当双击电脑桌面上的“我的电脑”快捷方式后，会打开“我的电脑”窗口，在其地址栏中显示当前位置，单击地址栏右侧的向下箭头会打开下拉列表，列出与之相关的项目，这个地址栏就类似 ComboBox 控件。但在 C#中，ComboBox 控件默认并不显示图标，本实例通过编写代码来美化 ComboBox 下拉列表，使其可以显示图标，并且能够更改选择项的颜色。实例运行效果如图 14.1 所示。

图 14.1 美化 ComboBox 控件下拉列表

关键技术

本实例主要是通过在 ComboBox 控件的 DrawItem 事件中对下拉列表进行重绘实现的，下面对 ComboBox 控件的 DrawItem 事件进行详细说明。

ComboBox 控件的 DrawItem 事件是指在所有者描述的 ComboBox 的可视方位更改时发生。语法格式如下：

```
public event DrawItemEventHandler DrawItem
```

技巧：DrawItem 事件由所有者描述的 ComboBox 使用，开发人员可以使用该事件来执行在 ComboBox 中绘制项所需的任务。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulComboBox。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，以便对其进行美化。

（3）程序的主要代码如下：

```
private void beautyComboBox_DrawItem(object sender,DrawItemEventArgs e)
{
    Graphics gComboBox = e.Graphics;                                    //声明一个 GDI+绘图图面类的对象
    Rectangle rComboBox = e.Bounds;                                     //声明一个表示矩形的位置和大小类的对象
    Size imageSize = imageList1.ImageSize;                              //声明一个有序整数对的对象
    FontDialog typeFace = new FontDialog();                             //定义一个字体对象
    Font Style = typeFace.Font;                                         //定义一个特定的文本格式对象
    if(e.Index >= 0)                                                    //当绘制的索引项存在时
    {
        string temp = (string)beautyComboBox.Items[e.Index];            //获取 ComboBox 控件索引项下的文本内容
        StringFormat stringFormat = new StringFormat();                 //定义一个封装文本布局信息类的对象
        stringFormat.Alignment = StringAlignment.Near;                  //设定文本的布局方式
        //当绘制项没有键盘加速键和焦点可视化提示时
        if(e.State == (DrawItemState.NoAccelerator | DrawItemState.NoFocusRect))
        {
            //用指定的颜色填充自定义矩形的内部
            e.Graphics.FillRectangle(new SolidBrush(Color.Red),rComboBox);
            //在指定位置绘制指定索引的图片
            imageList1.Draw(e.Graphics,rComboBox.Left,rComboBox.Top,e.Index);
            //在指定的位置并且用指定的 Font 对象绘制指定的文本字符串
            e.Graphics.DrawString(temp,Style,new SolidBrush(Color.Black),rComboBox.Left + imageSize.Width,rComboBox.Top);
```

```
            e.DrawFocusRectangle();                                    //在指定的边界范围内绘制聚焦框
        }
        else                                                           //当绘制项有键盘加速键或者焦点可视化提示时
        {
            //用指定的颜色填充自定义矩形的内部
            e.Graphics.FillRectangle(new SolidBrush(Color.LightBlue),rComboBox);
            //在指定位置绘制指定索引的图片
            imageList1.Draw(e.Graphics,rComboBox.Left,rComboBox.Top,e.Index);
            //在指定的位置并且用指定的 Font 对象绘制指定的文本字符串
            e.Graphics.DrawString(temp,Style,new SolidBrush(Color.Black),rComboBox.Left + imageSize.Width,rComboBox.Top);
            e.DrawFocusRectangle();                                    //在指定的边界范围内绘制聚焦框
        }
    }
}
```

秘笈心法

心法领悟 345：如何实现带查询功能的 ComboBox 控件？

C#中可以通过设置 ComboBox 控件的 AutoCompleteSource 属性和 AutoCompleteMode 属性实现从 ComboBox 控件中查询已存在的项，从而自动完成控件内容的输入功能。代码如下：

```
this.comboBox1.Items.Add(comboBox1.Text);                          //将地址添加到 ComboBox 中
this.comboBox1.AutoCompleteMode = AutoCompleteMode.SuggestAppend;  //设置 ComboBox 自动查询完成
this.comboBox1.AutoCompleteSource = AutoCompleteSource.ListItems;  //设置 ComboBox 控件的查询以列表项作为数据源
```

实例 346　美化单选按钮控件　　高级

光盘位置：光盘\MR\14\346　　趣味指数：★★★★★

实例说明

用户使用 C#自带的控件制作应用程序时，为了使界面更加美观，更具有特色，可以对控件的样式进行重绘。本实例将对单选按钮控件进行重绘，以用户自定义样式进行显示。实例运行效果如图 14.2 所示。

关键技术

本实例在美化单选按钮控件时，主要是通过重绘单选按钮实现的，具体实现时，首先需要创建用户控件，然后在用户控件中使用 Graphics 类的 DrawEllipse 和 FillEllipse 方法实现重绘单选按钮操作。下面对本实例中用到的关键技术进行详细讲解。

（1）创建用户控件并使用

创建用户控件的步骤如下：

① 选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建项”命令，弹出如图 14.3 所示的“添加新项”对话框。

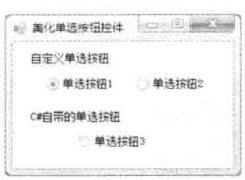


图 14.2　美化单选按钮控件

图 14.3　“添加新项”对话框

② 选择“用户控件”选项，在“名称”文本框中输入用户控件的名称，单击“添加”按钮，即可在当前项目中添加一个用户控件，如图 14.4 所示。

③ 在用户控件中，如果需要添加 Windows 标准控件，可以从工具箱中直接拖放使用；如果需要编写代码，则单击“单击此处切换到代码视图”超链接，进入后台代码视图，以便编写所需的代码。

④ 用户控件制作完成后，选中制作完成的用户控件，用鼠标将其拖曳到工具箱中，如图 14.5 所示。

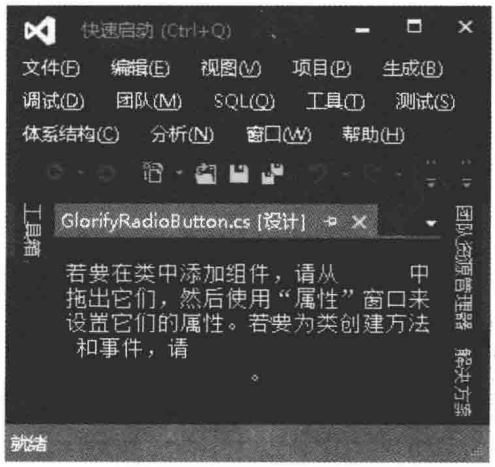


图 14.4 添加的用户控件

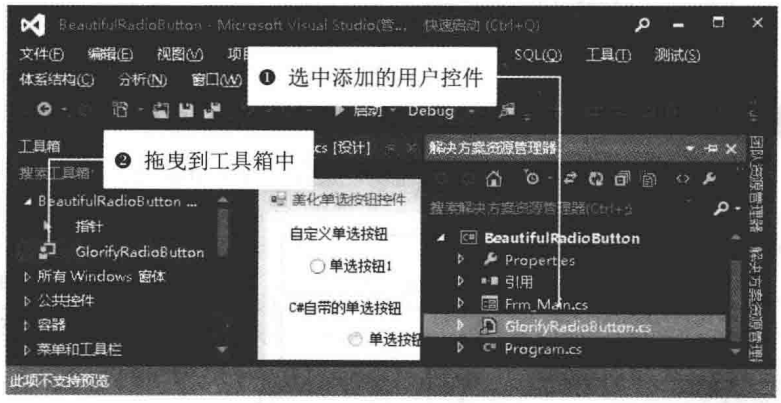


图 14.5 将用户控件拖曳到工具箱中

⑤ 像使用 Windows 标准控件一样，对用户控件进行拖曳即可。

（2）DrawEllipse 方法

该方法用来绘制一个由边框（该边框由一对坐标、高度和宽度指定）定义的椭圆，该方法有 4 种重载形式，本实例中用到的重载形式如下：

```
public void DrawEllipse(Pen pen,Rectangle rect)
```

参数说明

❶ pen：Pen 对象，它确定曲线的颜色、宽度和样式。

❷ rect：Rectangle 结构，它定义椭圆的边界。

（3）FillEllipse 方法

该方法用来填充边框所定义的椭圆的内部，该边框由一对坐标、一个宽度和一个高度指定，该方法有 4 种重载形式，本实例中用到的重载形式如下：

```
public void FillEllipse(Brush brush,Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ rect：Rectangle 结构，它表示定义椭圆的边框。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulRadioButton。

（2）在当前项目中添加一个用户控件，并将其命名为 GlorifyRadioButton。将用户控件继承的 UserControl 类改为 RadioButton 类。

（3）程序的主要代码如下。

在 GlorifyRadioButton 控件中定义变量的代码如下：

```
private bool FontAspect= false;                    //判断字体的方向
private int Measurement = 255;                     //设置渐变的初始值
LinearGradientBrush Periphery_br;                  //外圆的颜色
LinearGradientBrush Central_br;                    //移入控件时中圆的颜色
LinearGradientBrush NoCentral_br;                  //无操作时中圆的颜色
```

```
LinearGradientBrush Stipple_br;                                    //内圆选中的颜色
LinearGradientBrush NoStipple_br;                                  //内圆无选中的颜色
```

在 GlorifyRadioButton 控件的 OnPaint 事件中对控件的样式进行重绘，主要是通过 SystemInformation 类的 SmallIconSize 属性来获取单选按钮左边单选按钮的大小及位置，然后对单选按钮进行重绘。代码如下：

```
/// <summary>
///控件在需要重绘时触发
/// </summary>
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
    e.Graphics.FillRectangle(SystemBrushes.Control, e.ClipRectangle);          //填充矩形
    //清除锯齿
    e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;
    Rectangle boxrect = new Rectangle(e.ClipRectangle.X, e.ClipRectangle.Y, SystemInformation.SmallIconSize.Width, e.ClipRectangle.Height);
    //获取左面图标的区域
    //获取绘制的文本的区域
    Rectangle strrect = new Rectangle(e.ClipRectangle.X + SystemInformation.SmallIconSize.Width, e.ClipRectangle.Y, e.ClipRectangle.Width + 2 -
SystemInformation.SmallIconSize.Width, e.ClipRectangle.Height);
    if (FontAspect)                                                            //判断字体的读取方式
    {
        //设置椭圆的位置
        boxrect.X = e.ClipRectangle.X + e.ClipRectangle.Width - SystemInformation.SmallIconSize.Width;
        strrect.X = e.ClipRectangle.X;                                         //设置字体位置
    }
    Point MousePos = this.PointToClient(Control.MousePosition);                //获取鼠标的位置
    bool Above = e.ClipRectangle.Contains(MousePos);                           //获取鼠标是否在当前控件上
    DrawBox(e.Graphics, boxrect, Above);                                       //绘制单选图案
    DrawText(e.Graphics, strrect);                                             //绘制文字
    if (!Enabled)
        e.Graphics.FillRectangle(new SolidBrush(Color.FromArgb(127, SystemColors.Control)), e.ClipRectangle);
}
```

自定义方法 DrawBox 主要通过 LinearGradientBrush 类来设置单选按钮的绘制颜色，然后用 Graphics 类进行绘制。代码如下：

```
/// <summary>
///  绘制单选控件的图案
/// </summary>
/// <param g="Graphics">封装一个绘图的对象</param>
/// <param rect="Rectangle">单选图案的绘制区域</param>
/// <param Above="bool">判断鼠标是否在控件上方</param>
private void DrawBox(Graphics g, Rectangle rect, bool Above)
{
    //设置外椭圆的渐变色
    int opacity = Measurement;
    Periphery_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 2, PeripheryColor), Color.FromArgb(opacity, PeripheryColor),
LinearGradientMode.ForwardDiagonal);
    //设置中间椭圆形选中时的渐变色
    opacity = (int)(.4f * opacity + .5f);
    Central_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 10, CentralColor), Color.FromArgb(opacity, CentralColor),
LinearGradientMode.ForwardDiagonal);
    //设置中间椭圆形无操作时的渐变色
    opacity = (int)(.4f * opacity + .5f);
    NoCentral_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 10, NoCentralColor), Color.FromArgb(opacity, NoCentralColor),
LinearGradientMode.ForwardDiagonal);
    //设置内圆形选中时的渐变色
    opacity = Measurement;
    Stipple_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 5, StippleColor), Color.FromArgb(opacity, StippleColor),
LinearGradientMode.BackwardDiagonal);
    //设置内圆形无操作时的渐变色
    opacity = (int)(.4f * opacity + .5f);
    NoStipple_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 10, NoStippleColor), Color.FromArgb(opacity, NoStippleColor),
LinearGradientMode.BackwardDiagonal);
    int size = this.Font.Height;                                               //获取字体的高度
    //获取外椭圆的区域
    Rectangle box = new Rectangle(rect.X + ((rect.Width - size) / 2), rect.Y + ((rect.Height - size) / 2), size - 2, size - 2);
    Rectangle glyph = new Rectangle(box.X + 3, box.Y + 3, box.Width - 6, box.Height - 6);      //设置内圆的绘制区域
```

```
        g.FillEllipse(new SolidBrush(SystemColors.Window), box);                    //以白色填充单选图案
        if (Above && this.Enabled)                                                  //如果鼠标移入该控件
        {
            g.DrawEllipse(new Pen(Central_br, (float)(3)), new Rectangle(box.X + 2, box.Y + 2, box.Width - 4, box.Height - 4));    //绘制中心椭圆
        }
        else
        {
            g.DrawEllipse(new Pen(NoCentral_br, (float)(3)), new Rectangle(box.X + 2, box.Y + 2, box.Width - 4, box.Height - 4));  //绘制中心椭圆
        }
        g.DrawEllipse(new Pen(Periphery_br, (float)(1.5)), box);                    //绘制外椭圆
        g.FillEllipse(new SolidBrush(SystemColors.Window), glyph);                  //以白色填充内圆
        if (this.Checked)                                                           //如果选中当前控件
            g.FillEllipse(Stipple_br, glyph);                                       //填充内圆
        else
            g.FillEllipse(NoStipple_br, glyph);                                     //填充内圆
    }
```

自定义方法 DrawText 主要是通过判断单选按钮控件的显示方向在控件的指定位置绘制文本信息。代码如下：

```
    /// <summary>
    ///绘制文本
    /// </summary>
    /// <param g="Graphics">封装一个绘图的对象</param>
    /// <param rect="Rectangle">绘制文本的区域</param>
    private void DrawText(Graphics g, Rectangle rect)
    {
        StringFormat tem_StringF = new StringFormat();                              //创建 StringFormat 对象
        tem_StringF.Alignment = StringAlignment.Near;                               //指定文本靠近布局对齐
        tem_StringF.LineAlignment = StringAlignment.Center;                         //文本居中对齐
        if (FontAspect)                                                             //如果控件是原始方向显示
            tem_StringF.FormatFlags = StringFormatFlags.DirectionRightToLeft;       //按从右到左的顺序显示文本
        if (!FontAspect)                                                            //如果控件是反方向显示
            //绘制文本（从左端开始绘制）
            g.DrawString(this.Text, this.Font, SystemBrushes.ControlText, rect, tem_StringF);
        else
        {
            rect.X = rect.X - SystemInformation.SmallIconSize.Width / 2 + 2;        //设置文本的起始位置（在小图标后）
            //绘制文本
            g.DrawString(this.Text, this.Font, SystemBrushes.ControlText, rect, tem_StringF);
        }
    }
```

自定义方法 getAspect 主要用于获取单选按钮控件的显示方向。代码如下：

```
    /// <summary>
    ///获取文本的读取方向
    /// </summary>
    /// <return>布尔型</return>
    private bool getAspect()
    {
        bool tem_Aspect = false;
        if (this.RightToLeft == RightToLeft.Yes)                                    //从右到左进行读取
            tem_Aspect = true;
        if (this.RightToLeft == RightToLeft.No)                                     //从左到右进行读取
            tem_Aspect = false;
        return tem_Aspect;
    }
```

秘笈心法

心法领悟 346：RadioButton 控件的使用。

RadioButton 控件，又称为单选按钮控件，它为用户提供由两个或多个互斥选项组成的选项集，当用户选中某单选按钮时，同一组中的其他单选按钮不能同时选中。例如，在开发考试管理系统时，如果考试题中有单项选择题，则可以使用 RadioButton 控件来表示单项选择题的各个选项。

实例 347　美化复选框控件

光盘位置：光盘\MR\14\347　　高级　　趣味指数：★★★★★

实例说明

用户使用 C#自带的控件制作应用程序时，为了使界面更加美观，更具有特色，可以对控件的样式进行重绘。本实例将对复选框控件进行重绘，以用户自定义样式进行显示。实例运行效果如图 14.6 所示。

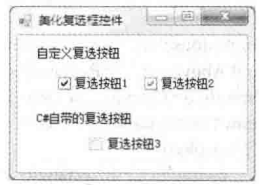


图 14.6　美化复选框控件

关键技术

本实例在美化复选框控件时，主要是通过重绘复选框实现的，具体实现时，首先需要创建用户控件，然后在用户控件中使用 Graphics 类的 DrawRectangle 和 FillRectangle 方法实现重绘复选框操作。下面对本实例中用到的关键技术进行详细讲解。

（1）DrawRectangle 方法

该方法用来绘制由坐标对、宽度和高度指定的矩形，有 3 种重载形式，本实例中用到的重载形式如下：

```
public void DrawRectangle(Pen pen,Rectangle rect)
```

参数说明

❶ pen：Pen 对象，它确定矩形的颜色、宽度和样式。

❷ rect：表示要绘制的矩形的 Rectangle 结构。

（2）FillRectangle 方法

该方法用来填充由一对坐标、一个宽度和一个高度指定的矩形的内部，有 4 种重载形式，本实例中用到的重载形式如下：

```
public void FillRectangle(Brush brush,Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ rect：Rectangle 结构，它表示要填充的矩形。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulCheckBox。

（2）在当前项目中添加一个用户控件，并将其命名为 GlorifyCheckBox。将用户控件继承的 UserControl 类改为 CheckBox 类。

（3）程序的主要代码如下。

在 GlorifyCheckBox 控件中定义变量的代码如下：

```
private bool FontAspect = false;                                    //判断字体的方向
private int Measurement = 255;                                      //设置渐变的初始值
LinearGradientBrush Periphery_br;                                   //外框的颜色
LinearGradientBrush Central_br;                                     //移入控件时中框的颜色
LinearGradientBrush NoCentral_br;                                   //无操作时中框的颜色
```

在 GlorifyCheckBox 控件的 OnPaint 事件中对控件的样式进行重绘，主要是通过 SystemInformation 类的 SmallIconSize 属性来获取复选框控件左边复选框的大小及位置，然后对复选框进行重绘。代码如下：

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
    e.Graphics.FillRectangle(SystemBrushes.Control, e.ClipRectangle);          //填充矩形
    e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;  //清除锯齿
    //获取左面图标的区域
```

```
    Rectangle boxrect = new Rectangle(e.ClipRectangle.X, e.ClipRectangle.Y, SystemInformation.SmallIconSize.Width, e.ClipRectangle.Height);
    //获取绘制的文本的区域
    Rectangle strrect = new Rectangle(e.ClipRectangle.X + SystemInformation.SmallIconSize.Width, e.ClipRectangle.Y, e.ClipRectangle.Width + 5 -
SystemInformation.SmallIconSize.Width, e.ClipRectangle.Height);
    if (FontAspect)                                                          //判断字体的读取方式
    {
        //设置框的位置
        boxrect.X = e.ClipRectangle.X + e.ClipRectangle.Width - SystemInformation.SmallIconSize.Width;
        strrect.X = e.ClipRectangle.X;                                       //设置字体位置
    }
    Point MousePos = this.PointToClient(Control.MousePosition);              //获取鼠标的位置
    bool Above = e.ClipRectangle.Contains(MousePos);                         //获取鼠标是否在当前控件上
    DrawBox(e.Graphics, boxrect, Above);                                     //绘制单选图案
    DrawText(e.Graphics, strrect);                                           //绘制文字
    if (!Enabled)
        e.Graphics.FillRectangle(new SolidBrush(Color.FromArgb(127, SystemColors.Control)), e.ClipRectangle);
}
```

自定义方法 DrawBox 主要通过 LinearGradientBrush 类来设置复选框的绘制颜色，然后用 Graphics 类进行绘制。代码如下：

```
private void DrawBox(Graphics g, Rectangle rect, bool Above)
{
    int opacity = Measurement;                                               //设置外框的渐变色
    Periphery_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 2, PeripheryColor), Color.FromArgb(opacity, PeripheryColor),
LinearGradientMode.ForwardDiagonal);
    opacity = (int)(.4f * opacity + .5f);                                    //设置中间框形选中时的渐变色
    Central_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 10, CentralColor), Color.FromArgb(opacity, CentralColor),
LinearGradientMode.ForwardDiagonal);
    opacity = (int)(.4f * opacity + .5f);                                    //设置中间框形无操作时的渐变色
    NoCentral_br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 10, NoCentralColor), Color.FromArgb(opacity, NoCentralColor),
LinearGradientMode.ForwardDiagonal);
     int size = this.Font.Height;                                            //获取字体的高度
     //获取外框的区域
    Rectangle box = new Rectangle(rect.X + ((rect.Width - size) / 2), rect.Y + ((rect.Height - size) / 2), size - 2, size - 2);
    //设置内框的绘制区域
    Rectangle glyph = new Rectangle(box.X + 3, box.Y + 3, box.Width - 6, box.Height - 6);
    Rectangle right = new Rectangle(box.X, box.Y - 1, box.Width + 2, box.Height + 2);
    g.FillEllipse(new SolidBrush(SystemColors.Window), box);                 //以白色填充单选图案
    if (this.CheckState != CheckState.Unchecked)                             //如果是选中状态
    {
        base.ForeColor = Color.DarkBlue;
        //绘制对号
        ControlPaint.DrawMenuGlyph(g, right, MenuGlyph.Checkmark,this.StippleColor,Color.White);
        g.DrawRectangle(new Pen(new SolidBrush(SystemColors.Control), (float)(3)), box);  //绘制外框
    }
    if (this.CheckState == CheckState.Indeterminate)
        g.FillRectangle(new SolidBrush(Color.FromArgb(127, SystemColors.Control)), right);
    if (Above && this.Enabled)                                               //如果鼠标移入该控件
    {
        g.DrawRectangle(new Pen(Central_br, 2), new Rectangle(box.X + 2, box.Y + 2, box.Width - 4, box.Height - 4));          //绘制中心框
    }
    else
    {
        g.DrawRectangle(new Pen(NoCentral_br, 2), new Rectangle(box.X + 2, box.Y + 2, box.Width - 4, box.Height - 4));       //绘制中心框
    }
    g.DrawRectangle(new Pen(Periphery_br, (float)(1.5)), box);               //绘制外框
}
```

自定义方法 DrawText 主要是通过判断复选框控件的显示方向在控件的指定位置绘制文本信息。代码如下：

```
private void DrawText(Graphics g, Rectangle rect)
{
    StringFormat tem_StringF = new StringFormat();
    tem_StringF.Alignment = StringAlignment.Near;
    tem_StringF.LineAlignment = StringAlignment.Center;                      //文本居中对齐
    if (FontAspect)
        tem_StringF.FormatFlags = StringFormatFlags.DirectionRightToLeft;    //按从左到右的顺序显示文本
    if (!FontAspect)
```

```
            g.DrawString(this.Text, this.Font, SystemBrushes.ControlText, rect, tem_StringF);      //绘制文本
        else
        {
            rect.X = rect.X - SystemInformation.SmallIconSize.Width / 2 + 2;
            g.DrawString(this.Text, this.Font, SystemBrushes.ControlText, rect, tem_StringF);
        }
    }
```

自定义方法 getAspect 主要用于获取复选框控件的显示方向。代码如下：

```
private bool getAspect()
{
    bool tem_Aspect = SystemInformation.RightAlignedMenus;
    if (this.RightToLeft == RightToLeft.Yes)                                    //从右到左进行读取
        tem_Aspect = true;
    if (this.RightToLeft == RightToLeft.No)                                     //从左到右进行读取
        tem_Aspect = false;
    return tem_Aspect;
}
```

秘笈心法

心法领悟 347：如何使窗体中的所有 CheckBox 控件处于取消选中状态？

使窗体中的所有 CheckBox 控件处于取消选中状态时，可以通过 foreach 遍历来实现。代码如下：

```
foreach (Control ctl in this.Controls)                                          //遍历窗体中的控件
{
    if (ctl.GetType().Name == "CheckBox")                                       //判断控件类型是否是 CheckBox
    {
        CheckBox cbox = (CheckBox)ctl;                                          //将遍历到的控件强制转换为 CheckBox 类型
        cbox.Checked = false;                                                   //设置复选框不选中
    }
}
```

实例 348　重绘 ListBox 控件

高级

光盘位置：光盘\MR\14\348

趣味指数：★★★★★

实例说明

当用户使用 ListBox 控件显示数据时，如果数据过多，很难在项集合中查找指定的数据。本实例用两种颜色或两种渐变颜色设置相隔项的背景颜色，这样不但可以美化控件，还便于查找。实例运行效果如图 14.7 所示。

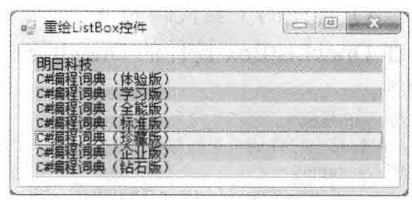


图 14.7　重绘 ListBox 控件

关键技术

本实例实现时主要使用 LinearGradientBrush 类和 Graphics 类的 FillRectangle 方法对 ListBox 控件进行重绘，下面分别对它们进行详细讲解。

（1）LinearGradientBrush 类

该类使用线性渐变封装 Brush（画刷），也就是封装双色渐变和自定义多色渐变。所有渐变都是由矩形的宽度或两个点指定的直线绘制的。默认情况下，双色渐变是沿指定直线从起始色到结束色的均匀水平线性混合。使用 Blend 类、SetSigmaBellShape 方法或 SetBlendTriangularShape 方法自定义混合图案。通过在构造函数中指定 LinearGradientMode 枚举或角度自定义渐变的方向。

下面是 LinearGradientBrush 类的构造函数，该函数主要根据一个矩形、起始颜色和结束颜色以及方向模式，创建 LinearGradientBrush 类的新实例。语法格式如下：

```
public LinearGradientBrush (RectangleF rect, Color color1, Color color2, LinearGradientMode linearGradientMode)
```

参数说明

❶ rect：一个 RectangleF 结构，它指定线性渐变的界限。

❷ color1：表示渐变起始色的 Color 结构。

❸ color2：表示渐变结束色的 Color 结构。

❹ linearGradientMode：LinearGradientMode 枚举值之一，它指定渐变方向，渐变方向决定渐变的起点和终点。LinearGradientMode 枚举值及说明如表 14.1 所示。

表 14.1 LinearGradientMode 枚举值及说明

枚 举 值	描 述
BackwardDiagonal	指定从右上到左下的渐变
ForwardDiagonal	指定从左上到右下的渐变
Horizontal	指定从左到右的渐变
Vertical	指定从上到下的渐变

（2）Graphics 类的 FillRectangle 方法

Graphics 类的 FillRectangle 方法主要用来填充由一对坐标、一个宽度和一个高度指定的矩形的内部，该方法有 4 种重载形式，本实例中用到的重载形式如下：

```
public void FillRectangle(Brush brush,Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ rect：Rectangle 结构，它表示要填充的矩形。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulListBox。

（2）在当前项目中添加一个用户控件，并将其命名为 DrawListBox，将用户控件继承的 UserControl 类改为 ListBox 类。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

（3）程序的主要代码如下。

在 DrawListBox 控件的 DrawItem 事件中，当项重绘时，按指定的颜色对当前项的背景和文本进行重绘。代码如下：

```
/// <summary>
///鼠标移出控件的可见区域时触发
/// </summary>
protected virtual void ListBox_DrawItem(object sender, DrawItemEventArgs e)
{
    Rectangle r = new Rectangle(0, 0, this.Width, this.Height);                    //设置重绘的区域
    SolidBrush SolidB1 = new SolidBrush(this.Color1);                              //设置上一行颜色
    SolidBrush SolidB2 = new SolidBrush(this.Color2);                              //设置下一行颜色
    //设置上一行的渐变色
    LinearGradientBrush LinearG1 = new LinearGradientBrush(r, this.Color1, this.Color1Gradual, LinearGradientMode.BackwardDiagonal);
    //设置下一行的渐变色
    LinearGradientBrush LinearG2 = new LinearGradientBrush(r, this.Color2, this.Color2Gradual, LinearGradientMode.BackwardDiagonal);
    //将单色与渐变色存入 Brush 数组中
    listBoxBrushes = new Brush[] { SolidB1, LinearG1, SolidB2, LinearG2 };
    if (this.Items.Count <= 0)                                                     //如果当前控件为空
        return;
    if (e.Index == (this.Items.Count - 1))                                         //如果绘制的是最后一个项
    {
        bool tem_bool = true;
        if (e.Index == 0 && tem_bool)                                              //如果当前绘制的是第一个或最后一个项
            naught = false;                                                        //不进行重绘
    }
```

```
        if (naught)                                                    //对控件进行重绘
        {
            //获取当前绘制的颜色值
            Brush brush = listBoxBrushes[place = (GradualC) ? (((e.Index % 2) == 0) ? 1 : 3) : (((e.Index % 2) == 0) ? 0 : 2)];
            e.Graphics.FillRectangle(brush, e.Bounds);                 //用指定画刷填充列表项范围所形成的矩形
            //判断当前项是否被选中
            bool selected = ((e.State & DrawItemState.Selected) == DrawItemState.Selected) ? true : false;
            if (selected)                                              //如果当前项被选中
            {
                e.Graphics.FillRectangle(new SolidBrush(ColorSelect), e.Bounds); //绘制当前项
            }
            //绘制当前项中的文本
            e.Graphics.DrawString(this.Items[e.Index].ToString(), this.Font, Brushes.Black, e.Bounds);
        }
        e.DrawFocusRectangle();                                        //绘制聚焦框
    }
```

秘笈心法

心法领悟 348：对 ListBox 控件中的数据进行排序。

对 ListBox 控件中的数据进行排序时，只需要将其 Sorted 属性设置为 true 即可。代码如下：

```
listBox1.Sorted = true;
```

14.2　自定义 Windows 控件

实例 349	自定义水晶按钮控件	高级
	光盘位置：光盘\MR\14\349	趣味指数：★★★★★

实例说明

用户在制作应用程序时，为了使按钮更加美观，可以自己制作按钮。本实例将制作一个半透明式的水晶按钮。实例运行效果如图 14.8 所示。

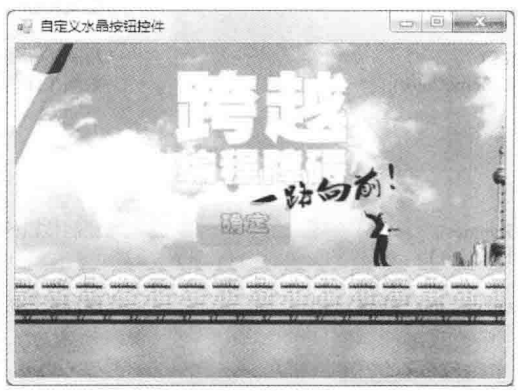


图 14.8　自定义水晶按钮控件

关键技术

本实例实现时主要使用 GraphicsPath 类的 AddArc 和 AddLine 方法绘制一个圆角矩形，下面对本实例中用到的关键技术进行详细讲解。

（1）AddArc 方法

该方法主要用来向当前图形追加一段椭圆弧。语法格式如下：

```
public void AddArc (RectangleF rect,float startAngle,float sweepAngle)
```

参数说明

❶ rect：一个 RectangleF 结构，表示从中截取弧线的椭圆的矩形边界。

❷ startAngle：弧线的起始角度，以度为单位从 X 轴顺时针测量。

❸ sweepAngle：startAngle 和弧线末尾之间的角度。

（2）AddLine 方法

该方法主要用来向 GraphicsPath 追加一条线段。语法格式如下：

```
public void AddLine (float x1, float y1, float x2, float y2)
```

AddLine 方法中的参数及说明如表 14.2 所示。

表 14.2　AddLine 方法中的参数及说明

参　数	描　述
x1	直线起始点的 X 坐标
y1	直线起始点的 Y 坐标
x2	直线终结点的 X 坐标
y2	直线终结点的 Y 坐标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulButton。

（2）在当前项目中添加一个用户控件，并将其命名为 TransparencyButton。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

（3）程序的主要代码如下。

在 TransparencyButton 控件的 Paint 事件中对当前控件进行重绘。代码如下：

```
private void TransparencyButton_Paint(object sender, PaintEventArgs e)
{
    this.BackColor = Color.Transparent;                                          //使当前控件透明
    sm = e.Graphics.SmoothingMode;                                               //设置呈现质量
    Color shineColor = Color.Black;
    Rectangle rect2 = new Rectangle(0, 0, this.Width, this.Height);              //设置绘制按钮的矩形区域
    Rectangle rect1 = new Rectangle(0, this.Height / 2, this.Width, this.Height / 2);  //设置绘制按钮下半部的矩形区域
    if (this.CTransparence == 0)                                                 //如果按钮的透明度为 0
    {
        CobOblongDown(rect2, e.Graphics);                                        //绘制按钮的背景
        CobOblong(rect2, e.Graphics, this.ShineColor);                           //绘制按钮的背景
    }
    else
    {
        if (this.CTransparence > 0)                                              //如果按钮的透明度不为 0
        {
            CobOblongDown(rect2, e.Graphics);                                    //绘制按钮的背景
            for (int i = 0; i < CTransparence; i++)
            {
                CobOblong(rect2, e.Graphics, this.ShineColor);                   //绘制按钮的背景颜色
            }
        }
    }
    int tem_n = (int)(this.CTransparence / 3);                                   //获取一个值，用于设置下半部按钮的颜色深度
    if (tem_n == 0)                                                              //如果为 0
        CobAjar(rect1, e.Graphics, this.ShineColor);                             //绘制按钮的下半部背景
    else
    {
        if (tem_n > 0)                                                           //如果不为 0
        {
            for (int i = 0; i < tem_n; i++)                                      //加深下部按钮的颜色
            {
                CobAjar(rect1, e.Graphics, this.ShineColor);                     //绘制按钮的下半部背景颜色
            }
        }
```

```
    }
    CobOblong(rect2, e.Graphics, this.UndersideShine);                    //设置下半部按钮的光泽度
    if (pub_ButtonClick == false)                                         //判断按钮是否按下（false 为按下）
    {
        CobOblongDown(rect2, e.Graphics);                                 //绘制按钮的背景
    }
    if (this.NText.Length > 0)                                            //如果 Text 属性中有值
        ProtractText(e.Graphics);                                         //绘制透明按钮的文本信息
}
```

自定义方法 CobOblongDown 用于绘制按钮按下时的背景效果，其矩形按钮是以圆角绘制，并用渐变色填充。代码如下：

```
/// <summary>
///绘制透明按钮按下时的效果
/// </summary>
/// <param rect="Rectangle">绘制按钮的区域</param>
/// <param g="Graphics">封装一个绘图的对象</param>
private void CobOblongDown(Rectangle rect, Graphics g)
{
    using (GraphicsPath bh = CreateCobOblong(rect, this.BDegree))         //按钮的圆角绘制
    {
        int opacity = pub_RGB_r1;                                         //设置按钮的 R 色值
        Color tem_Color = Color.Black;                                    //设置按钮的背景颜色为黑色
        if (pub_ButtonClick == true)                                      //如果按钮没有按下
        {
            opacity = pub_RGB_r0;                                         //设置按钮的 R 色值
            tem_Color = Color.White;                                      //设置按钮的背景颜色为白色
        }
        opacity = (int)(.4f * opacity + .5f);                             //设置渐变的变化值
        using (LinearGradientBrush br = new LinearGradientBrush(rect, Color.FromArgb(opacity + 20, tem_Color), Color.FromArgb(opacity,
tem_Color), LinearGradientMode.Vertical))                                 //设置按钮的渐变颜色
        {
            g.FillPath(br, bh);                                           //填充按钮背景
        }
        g.SmoothingMode = sm;                                             //设置呈现的质量
    }
}
```

自定义方法 CreateCobOblong 主要是用 GraphicsPath 类绘制一个圆角矩形，并返回其绘制的路径。代码如下：

```
/// <summary>
///按钮的圆角绘制
/// </summary>
/// <param rect="Rectangle">绘制按钮的区域</param>
/// <param radius="int">圆角的度数</param>
private static GraphicsPath CreateCobOblong(Rectangle rectangle, int radius)
{
    GraphicsPath path = new GraphicsPath();                               //创建 GraphicsPath 对象
    int l = rectangle.Left;                                               //获取矩形左上角的 X 坐标
    int t = rectangle.Top;                                                //获取矩形左上角的 Y 坐标
    int w = rectangle.Width;                                              //获取矩形的宽度
    int h = rectangle.Height;                                             //获取矩形的高度
    path.AddArc(l, t, 2 * radius, 2 * radius, 180, 90);                   //在矩形的左上角绘制圆角
    path.AddLine(l + radius, t, l + w - radius, t);                       //绘制左上角圆角与右上角之间的线段
    path.AddArc(l + w - 2 * radius, t, 2 * radius, 2 * radius, 270, 90);  //绘制右上角圆角
    //绘制左上角、右上角和右下角所形成的三角形
    path.AddLine(l + w, t + radius, l + w, t + h - radius);
    path.AddArc(l + w - 2 * radius, t + h - 2 * radius, 2 * radius, 2 * radius, 0, 90);   //绘制右下角圆角
    path.AddLine(l + radius, t + h, l + w - radius, t + h);               //绘制右下角圆角与左上角之间的线段
    path.AddArc(l, t + h - 2 * radius, 2 * radius, 2 * radius, 90, 90);   //绘制左下角圆角
    path.AddLine(l, t + radius, l, t + h - radius);                       //绘制左上角、左下角和右下角之间的三角形
    return path;
}
```

自定义方法 CobOblong 主要用于绘制按钮在没有按下时的背景效果，其矩形按钮是以圆角绘制，并用渐变色填充。代码如下：

```
/// <summary>
///绘制透明按钮的背景色
```

```
/// </summary>
/// <param rect="Rectangle">绘制按钮的区域</param>
/// <param g="Graphics">封装一个绘图的对象</param>
/// <param fillColor="Color">填充的颜色</param>
private void CobOblong(Rectangle rect, Graphics g, Color fillColor)
{
    using (GraphicsPath bh = CreateCobOblong(rect, this.BDegree))                    //绘制一个圆角矩形
    {
        int opacity = pub_RGB_r0;                                                    //设置按钮的 R 色值
        opacity = (int)(.4f * opacity + .5f);                                        //设置渐变的变化值
        //设置按钮的渐变颜色
        using (LinearGradientBrush br = new LinearGradientBrush(rect, Color.FromArgb(opacity / 5, fillColor), Color.FromArgb(opacity, fillColor),
LinearGradientMode.Vertical))
        {
            g.FillPath(br, bh);                                                      //填充按钮背景
        }
        g.SmoothingMode = sm;                                                        //设置呈现的质量
    }
}
```

自定义方法 ProtractText 主要是在按钮上绘制文本，其文本的字体颜色是以渐变色显示的。代码如下：

```
#region 自定义方法
/// <summary>
///绘制透明按钮的文本
/// </summary>
/// <param g="Graphics">封装一个绘图的对象</param>
private void ProtractText(Graphics g)
{
    Graphics TitG = this.CreateGraphics();                                           //创建 Graphics 对象
    string TitS = this.NText;                                                        //获取图表标题的名称
    SizeF TitSize = TitG.MeasureString(TitS, this.Font);                             //将绘制的字符串进行格式化
    float TitWidth = TitSize.Width;                                                  //获取字符串的宽度
    float TitHeight = TitSize.Height;                                                //获取字符串的高度
    float TitX = 0;                                                                  //标题的横向坐标
    float TitY = 0;                                                                  //标题的纵向坐标
    if (this.Height > TitHeight)                                                     //如果按钮的高度大于文本的高度
        TitY = (this.Height - TitHeight) / 2;                                        //使文本水平方向居中
    else
        TitY = this.BDegree;                                                         //文本置顶
    if (this.Width > TitWidth)                                                       //如果按钮的宽度大于文本的宽度
        TitX = (this.Width - TitWidth) / 2;                                          //使文本水平居中
    else
        TitX = this.BDegree;                                                         //文本置左
    //设置文本的绘制区域
    Rectangle rect = new Rectangle((int)Math.Floor(TitX), (int)Math.Floor(TitY), (int)Math.Ceiling(TitWidth), (int)Math.Ceiling(TitHeight));
    int opacity = pub_RGB_r1;                                                        //设置 R 色值
    opacity = (int)(.4f * opacity + .5f);                                            //设置渐变值
    for (int i = 0; i < DCFontDeepness; i++)                                         //设置文本的深度
    {
        //设置文本的渐变颜色
        using (LinearGradientBrush br = new LinearGradientBrush(rect, Color.FromArgb(opacity, this.ForeColor), Color.FromArgb(opacity,
this.ForeColor), LinearGradientMode.Vertical))
        {
            g.DrawString(TitS, this.Font, br, new PointF(TitX, TitY));               //绘制带有渐变效果的文本
        }
    }
}
```

在对按钮的大小进行改变时，用 SizeChanged 事件对按钮进行重绘。代码如下：

```
private void TransparencyButton_SizeChanged(object sender, EventArgs e)
{
    Invalidate();                                                                    //对控件进行重绘
}
```

当用鼠标按下按钮时，对按钮进行重绘，加深按钮的背景颜色。代码如下：

```
private void TransparencyButton_MouseDown(object sender, MouseEventArgs e)
{
    pub_ButtonClick = false;                                                         //按下按钮
```

```
    Invalidate();                                                    //对控件进行重绘
}
```

当在按钮上松开鼠标时，对按钮进行重绘，恢复按钮的原始状态。代码如下：

```
private void TransparencyButton_MouseUp(object sender, MouseEventArgs e)
{
    pub_ButtonClick = true;                                          //松开按钮
    Invalidate();                                                    //对控件进行重绘
}
```

秘笈心法

心法领悟 349：使 Button 控件大小自动适应文本长度。

可以设置 Button 控件的 AutoSize 属性使按钮控件基于其文本内容调整大小。AutoSize 属性是一个 bool 类型，如果要想设置 Button 控件的 AutoSize 属性使按钮控件基于其文本内容调整大小，只需设置该值为 true 即可；否则设为 false。

实例 350　自制数值文本框组件　　高级

光盘位置：光盘\MR\14\350　　趣味指数：★★★★★

实例说明

在用户制作财务报表时，有些文本框只能填写数值型数据，为了便于用户的正确填写，本实例制作了一个只限于输入数值型的文本框，并可以在输入数值时，限定输入的范围，以及对其进行四舍五入、取整等。实例运行效果如图 14.9 所示，单击“保留两位小数”按钮，效果如图 14.10 所示。

图 14.9　自制数值文本框组件

图 14.10　保留两位小数

关键技术

本实例在设置自定义控件的属性时，如果要将创建的属性放置在一个分类中并对其进行说明，可以用 BrowsableAttribute 类中的 Browsable 方法来控制当前添加的属性是否显示在“属性”窗口中，用 CategoryAttribute 类中的 Category 属性设置分类的别名，用 DescriptionAttribute 类中的 Description 属性设置当前属性的说明性文字。下面对自定义属性的设置进行详细说明。

首先在控件的“代码编辑”窗口中找到如下代码：

```
public partial class NumberBox : TextBox
{
    public NumberBox()
    {
        InitializeComponent();
    }
}
```

说明：因为本实例是在 TextBox 控件的基础上设置自定义控件，在创建用户控件后，将用户控件的 UserControl 类改为 TextBox 类。这样，当前控件就继承了 TextBox 控件的所有属性、事件和方法。

然后在 public NumberBox 代码段的下面添加自定义属性。代码如下：

```
private int TDecimalDigit = 2;
//在“属性”窗口中显示 DecimalDigit 属性
[Browsable(true), Category("数据文本框"), Description("保留的小数位数")]
public int DecimalDigit
```

```
{
    get { return TDecimalDigit; }
    set
    {
        TDecimalDigit = value;
        //当属性值改变时，在这里进行相应的操作
    }
}
```

在设置控件的属性时，属性的设置分为两种，一种是C#自带类型的属性，如string、int、bool、Color、Font、DateSet等类型，它们的创建方式与DecimalDigit属性的创建方式相同，只要将该属性的int类型改为所需的类型即可；另一种是具有下拉列表的属性，该列表中的值是开发者自定义的。

下面对具有下拉列表属性的创建进行说明。

首先在public ChartPanel代码的下面定义一个枚举型变量，用于设置下拉列表的值。代码如下：

```
public enum Reserved
{
    MinInt = 0,                                    //保留最小整数
    Round = 1,                                     //四舍五入
    MaxInt = 2,                                    //保留最大整数
    Tropism = 3,                                   //小数取位（不进行舍入）
}
```

然后在该枚举型的下面设置一个具有Reserved值的下拉列表属性。代码如下：

```
private Reserved TReservedStyle = Reserved.Round;
//在“属性”窗口中显示 ReservedStyle 属性
[Browsable(true), Category("数据文本框"), Description("小数保留的类型")]
public Reserved ReservedStyle
{
    get { return TReservedStyle; }
    set
    {
        TReservedStyle = value;
        //当属性值改变时，在这里进行相应的操作
    }
}
```

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为BeautifulTextBox。

（2）在当前项目中添加一个用户控件，并将其命名为NumberBox，将用户控件继承的UserControl类改为TextBox类。

说明：关于如何创建用户控件，请参见实例346中的关键技术。

（3）程序的主要代码如下。

限制文本框的输入范围主要是在文本框的KeyPress事件中进行，如果在自定义控件中触发该事件，则对其进行重载。代码如下：

```
public NumberBox()
{
    InitializeComponent();
    //对 KeyPress 事件进行重载
    this.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.numberBox1_KeyPress);
    //对 Leave 事件进行重载
    this.Leave += new System.EventHandler(this.numberBox1_Leave);
}
```

numberBox1控件的KeyPress事件主要用于限制文本框中输入的字符只能是数值型。代码如下：

```
///<summary>
///执行自定义控件的 KeyPress 事件
///</summary>
protected virtual void numberBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    Estimate_Key(e, ((TextBox)sender).Text, Convert.ToInt32(this.DataStyle));
}
```

自定义方法 Estimate_Key 主要是判断文本框中输入的字符是整型还是单精度型，如果是单精度型，可以在文本框中输入数字、“.”或“-”；如果是整型，只能在文本框中输入数字和“-”。代码如下：

```
///<summary>
///文本框中只能输入数字型和单精度型的字符串
///</summary>
///<param name="e">KeyPressEventArgs 类</param>
///<param name="s">文本框的字符串</param>
///<param name="n">标识，判断是数字型还是单精度型</param>
public void Estimate_Key(KeyPressEventArgs e, string s, int n)
{
    string tem_s = "";
    if (e.KeyChar == '-')                                                    //如果键值为“-”
        //如果“-”不在起始位输入，或已存在“-”
        if (this.SelectionStart != 0 && this.Text.Substring(0, 1) == "-" && this.SelectedText.IndexOf('-') < 0)
            e.Handled = true;                                                //处理 KeyPress 事件
    if (e.KeyChar != '\b')                                                   //如果当前键值不为 Backspace 键
    {
        if (e.KeyChar <= '9' && e.KeyChar >= '0')                            //如果输入的是数字
        {
            //根据键值组合输入文本
            tem_s = s.Substring(0, this.SelectionStart) + e.KeyChar.ToString() + s.Substring(this.SelectionStart,
this.Text.Length - this.SelectionLength - this.SelectionStart);
            if (!Int64Bound(tem_s))                                          //判断是否在指定范围内
                e.Handled = true;                                            //处理 KeyPress 事件
        }
    }
    switch (n)
    {
        case 0: break;                                                       //字符串型
        case 1:                                                              //整数型
            {
                //当输入的键值不为 0~9 或 Enter 键、Backspace 键时
                if (!(e.KeyChar <= '9' && e.KeyChar >= '0') && e.KeyChar != '\r' && e.KeyChar != '\b' &&
e.KeyChar != '-')
                {
                    e.Handled = true;                                        //处理 KeyPress 事件
                }
                break;
            }
        case 2:                                                              //小数
            {
                //当输入的键值不为 0~9 或 Enter 键、Backspace 键、“.”时
                if ((!(e.KeyChar <= '9' && e.KeyChar >= '0')) && e.KeyChar != '.' && e.KeyChar != '\r' && e.KeyChar != '\b' && e.KeyChar != '-')
                {
                    e.Handled = true;                                        //处理 KeyPress 事件
                }
                else
                {
                    if (e.KeyChar == '.' || e.KeyChar == '\r' || e.KeyChar == '\b')   //如果输入'.'
                    {
                        if (e.KeyChar != '\r' && e.KeyChar != '\b')
                        {
                            if (s == "")                                     //当前文本框为空
                                e.Handled = true;                            //处理 KeyPress 事件
                            else
                            {
                                if (s.Length > 0)                            //当文本框不为空时
                                {
                                    if (s.IndexOf(".") > -1)                 //查找是否已输入过“.”
                                        if (this.SelectedText.IndexOf('.') < 0)
                                            e.Handled = true;                //处理 KeyPress 事件
                                }
                            }
                        }
                    }
                    else
```

```
                {
                    if (s.IndexOf(".") > -1)                                    //如果输入了“.”
                        if (((s.Length - s.IndexOf(".")) > DecimalDigit))       //如果超出了小数的保留位数
                        {
                            if (this.SelectionStart > s.IndexOf("."))           //如果在整数位输入键值
                            {
                                if (this.SelectedText.Length == 0)              //光标定位
                                    e.Handled = true;

                            }
                        }
                    }
                }
                break;
            }
        }
        if (this.Text.Length > 0)                                               //如果值不为空
        {
            if (this.DataStyle != StyleSort.Null && e.KeyChar == '\r')          //如果当前输入的是整数或小数，并且按 Enter 键
            {
                SetTextBox();                                                   //对值进行处理
            }
        }
    }
```

自定义方法 Int64Bound 用于判断向文本框中输入的数值型数据是否在 Int64 范围内，如果超出该范围，将停止在文本框中的输入。代码如下：

```
/// <summary>
///计算指定的字符串是否可以转换成 Int64 范围内的数字
/// </summary>
/// <param IB="string">字符串</param>
/// <return>布尔型</return>
public bool Int64Bound(string IB)
{
    if (IB.IndexOf('-')>0)                                                      //如果在文本框中除第一位以外，还有“-”符号
        return false;                                                           //数据错误
    double tem_d = Convert.ToDouble(IB);                                        //将字符型转换成双精度型
    tem_d = Math.Floor(tem_d);                                                  //取整
    if (tem_d <= UpLine64 && tem_d >= DownLine64)                               //判断整数位是否在 Int64 的范围内
        return true;                                                            //数据正确
    else
        return false;
}
```

numberBox1 控件的 Leave 事件主要是在该控件不是窗体的活动控件时，根据 DataStyle 属性对文本框中的数据进行相应的操作。代码如下：

```
/// <summary>
///执行自定义控件的 Leave 事件
/// </summary>
public virtual void numberBox1_Leave(object sender, System.EventArgs e)
{
    SetTextBox();
}
```

自定义方法 SetTextBox 的主要功能是通过 DataStyle 属性值，对文本框中的数据进行相应的操作。例如，对文本框中的数据进行取整、保留小数，以及对小数进行四舍五入等。代码如下：

```
/// <summary>
///根据属性设置文本框中的内容
/// </summary>
public void SetTextBox()
{
    bool tem_BoolInt = true;                                                    //定义一个变量，判断是否为数值
    if (this.Multiline == true)                                                 //如果允许输入多行
        return;                                                                 //退出当前操作
    if (this.Text.Length == 0)                                                  //如果 Text 属性为空
        return;                                                                 //退出当前操作
    if (this.Text.Trim() == "-")
```

```
{
    this.Text = "";
    return;                                                    //退出当前操作
}
else
{
    char tem_char = '0';
    for (int i = 0; i < this.Text.Length - 1; i++)             //循环遍历文本框中的数值
    {
        tem_char = Convert.ToChar(this.Text.Substring(i, 1));  //获取单个字符
        if ((tem_char > '9' || tem_char < '0'))                //如果字符不是数字
        {
            if (!(tem_char == '.' || tem_char == '-'))         //如果字符不是“.”和“-”
            {
                //当前文本不能转换成数值型数据
                MessageBox.Show("无法将字符串转换成整数或小数");
                ifInt = false;
                this.DataStyle = StyleSort.Null;
                return;
            }
        }
    }
    if (tem_BoolInt)                                           //如果是数值型
    {
        Decimal tem_value = Convert.ToDecimal(this.Text);      //获取当前的值
        switch (Convert.ToInt32(this.DataStyle))               //根据数据类型来进行操作
        {
            case 1:                                            //整型
            case 2:                                            //小数
                {
                    if (Math.Floor(tem_value) == tem_value)    //如果输入的是整型
                        break;                                 //不进行操作
                    switch (Convert.ToInt32(this.ReservedStyle))   //判断小数的保留类型
                    {
                        case 0:                                //保留最小整数
                            {
                                tem_value = Math.Floor(tem_value);
                                break;
                            }
                        case 1:                                //对小数进行四舍五入
                            {
                                if (Convert.ToInt32(this.DataStyle) == 1)   //对第一位小数进行四舍五入
                                {
                                    tem_value = Math.Round(tem_value, 1);
                                }
                                else                           //对指定位数的小数进行四舍五入
                                {
                                    tem_value = Math.Round(tem_value, this.ReservedDigit);
                                }
                                break;
                            }
                        case 2:                                //保留最大整数
                            {
                                tem_value = Convert.ToDecimal(this.Text);   //将文本框转换成双精度
                                tem_value = Math.Ceiling(tem_value);        //取最小整数
                                break;
                            }
                        case 3:                                //保留指定的小数位数
                            {
                                string var_str = this.Text;
                                if (Convert.ToInt32(this.DataStyle) == 2)
                                {
                                    tem_value = Convert.ToDecimal(var_str.Substring(0, var_str.IndexOf('.') + ReservedDigit + 1));
                                }
                                break;
                            }
                    }
                    break;
```

```
                }
            }
            this.Text = tem_value.ToString();                    //显示保留后的数据
        }
    }
}
```

秘笈心法

心法领悟 350：如何创建只读文本框？

通过将 TextBox 控件的 ReadOnly 属性设置为 true 可以创建只读文本框，之后用户仍可滚动并突出显示文本框中的文本，但不允许进行更改，另外。“复制”命令在文本框中仍然有效，但“剪切”和“粘贴”命令无效。

实例 351 设计带行数和标尺的 RichTextBox 控件

光盘位置：光盘\MR\14\351　　高级　　趣味指数：★★★★★

实例说明

当用户在 RichTextBox 控件中输入代码和图片时，为了便于对数据的观察，可以在代码的前面显示行号，也可在 RichTextBox 控件的顶端和左端显示标尺，以测量图片的大小。本实例将制作一个带有行数和标尺的 RichTextBox 控件。实例运行效果如图 14.11 所示。

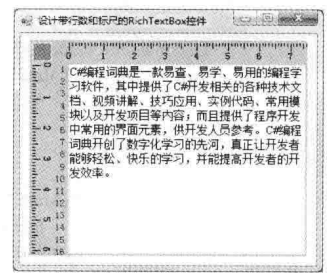


图 14.11　设计带行数和标尺的 RichTextBox 控件

关键技术

本实例主要是在自定义的用户控件中继承了 RichTextBox 控件，然后用 Graphics 类的 DpiX 和 DpiY 属性计算在控件中绘制毫米刻度的值，以及用 RichTextBox 控件的 GetPositionFromCharIndex 方法获取 RichTextBox 控件显示区域中第一个字符的坐标位置。下面对其进行详细介绍。

（1）在用户控件中继承 RichTextBox 控件

在用户控件中继承 RichTextBox 控件，与设置自定义属性基本相同，只是将其类型设置为 RichTextBox，因为该属性组继承 RichTextBox 控件，所以在设置属性时不需要设置 set“访问器”。代码如下：

```
[Browsable(true), Category("设置标尺控件"), Description("设置 RichTextBox 控件的相关属性")]
public RichTextBox NRichTextBox
{
    get { return richTextBox1; }
}
```

（2）DpiX 属性

该属性用于获取 Graphics 的水平分辨率。语法格式如下：

```
public float DpiX { get; }
```

参数说明

属性值：Graphics 支持的水平分辨率的值（以每英寸点数为单位）。

（3）DpiY 属性

该属性用于获取 Graphics 的垂直分辨率。语法格式如下：

```
public float DpiY{ get; }
```

参数说明

属性值：Graphics 支持的垂直分辨率的值（以每英寸点数为单位）。

（4）GetPositionFromCharIndex 方法

该方法用于检索控件内指定字符索引处的位置。语法格式如下：

```
public override Point GetPositionFromCharIndex (int index)
```

参数说明

❶ index：要检索其位置的字符索引。

❷ 返回值：指定字符的位置。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulRichTextBox。

（2）在当前项目中添加一个用户控件，并将其命名为 GuageRichTextBox。在用户控件中添加一个 RichTextBox 控件，将 BorderStyle 属性设置为 None，将 Location 属性设置为“30,30”。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

（3）程序的主要代码如下。

在 GuageRichTextBox 控件的 Paint 事件中，根据标尺的类型，在 RichTextBox 控件的顶端和左端绘制标尺，然后根据 CodeShow 属性在文本框的左边绘制行号。代码如下：

```
private void GuageRichTextBox_Paint(object sender, PaintEventArgs e)
{
    //绘制外边框
    e.Graphics.DrawRectangle(new Pen(Color.DarkGray), 0, 0, this.Width - 1, this.Height - 1);
    if (CodeShow)                                                    //如插在文本框左边添加行号
    {
        //获取行号的宽度
        float tem_code = (float)StringSize((Convert.ToInt32(CodeSize+(float)(richTextBox1.Height / (StringSize(CodeSize.ToString(), richTextBox1.
Font, false))))).ToString(),this.Font, true);
        richTextBox1.Top = Distance_X;                               //设置控件的顶端距离
        richTextBox1.Left = Distance_Y + (int)Math.Ceiling(tem_code);//设置控件的左端距离
        //设置控件的宽度
        richTextBox1.Width = this.Width - Distance_X - 2 - (int)Math.Ceiling(tem_code);
        richTextBox1.Height = this.Height - Distance_Y - 2;          //设置控件高度
        thisleft = Distance_Y + tem_code;                            //设置标尺的左端位置
    }
    else
    {
        richTextBox1.Top = Distance_X;                               //设置控件的顶端距离
        richTextBox1.Left = Distance_Y;                              //设置控件的左端距离
        richTextBox1.Width = this.Width - Distance_X - 2;            //设置控件的宽度
        richTextBox1.Height = this.Height - Distance_Y - 2;          //设置控件的高度
        thisleft = Distance_Y;                                       //设置标尺的左端位置
    }
    //绘制文本框的边框
    e.Graphics.DrawRectangle(new Pen(Color.LightSteelBlue), richTextBox1.Location.X - 1, thisleft - 1, richTextBox1.Width + 1, richTextBox1.Height + 1);
    //文本框的上边框
    e.Graphics.FillRectangle(new SolidBrush(Color.Silver), 1, 1, this.Width - 2, Distance_Y - 2);
    //文本框的左边框
    e.Graphics.FillRectangle(new SolidBrush(Color.Silver), 1, 1, Distance_X - 2, this.Height - 2);
    //绘制左上角的方块边框
    e.Graphics.FillRectangle(new SolidBrush(Color.Gray), 3, 3, Distance_X - 7, Distance_Y - 8);
    //绘制左上角的方块
    e.Graphics.DrawRectangle(new Pen(SystemColors.Control), 3, 3, Distance_X - 8, Distance_Y - 8);
    if (RulerStyle == Ruler.Rule)                                    //标尺
    {
        //绘制左上角的方块边框
        e.Graphics.FillRectangle(new SolidBrush(Color.Gray), thisleft - 3, 3, this.Width - (thisleft - 2) , Distance_Y  -  9); e.Graphics.DrawLine(new
Pen(SystemColors.Control), thisleft - 3, 3, this.Width - 2, 3);                                    //绘制方块的上边线
        //绘制方块的下边线
        e.Graphics.DrawLine(new Pen(SystemColors.Control), thisleft - 3, Distance_Y - 5, this.Width - 2, Distance_Y – 5);
        //绘制方块的中间块
        e.Graphics.FillRectangle(new SolidBrush(Color.WhiteSmoke), thisleft - 2, 9, this.Width - (thisleft - 2) - 1, Distance_Y - 19);
        //绘制左边的方块
        e.Graphics.FillRectangle(new SolidBrush(Color.Gray), 3, Distance_Y - 3, Distance_X - 7, this.Height - (Distance_Y - 3) - 2);
        //绘制方块的左边线
        e.Graphics.DrawLine(new Pen(SystemColors.Control), 3, Distance_Y - 3, 3, this.Height - 2);
```

```
            //绘制方块的右边线
            e.Graphics.DrawLine(new Pen(SystemColors.Control), Distance_X - 5, Distance_Y - 3, Distance_X - 5, this.Height - 2);
            //绘制方块的中间块
            e.Graphics.FillRectangle(new SolidBrush(Color.WhiteSmoke), 9, Distance_Y - 3, Distance_X - 19, this.Height - (Distance_Y - 3) - 2);
        }
        int tem_temHeight = 0;
        string tem_value = "";
        int tem_n = 0;
        int divide = 5;
        Pen tem_p = new Pen(new SolidBrush(Color.Black));
        //横向刻度的设置
        if (UnitStyle == Unit.Cm)                                    //如果刻度的单位是厘米
            Degree = e.Graphics.DpiX / 25.4F;                        //将厘米转换成像素
        if (UnitStyle == Unit.Pels)                                  //如果刻度的单位是像素
            Degree = 10;                                             //设置 10 像素为一个刻度
        int tem_width = this.Width - 3;
        tem_n = (int)StartBitH;                                      //记录横向滚动条的位置
        if (tem_n != StartBitH)                                      //如果横向滚动条的位置值为小数
            StartBitH = (int)StartBitH;                              //对横向滚动条的位置进行取整
        for (float i = 0; i < tem_width; )                           //在文本框的顶端绘制标尺
        {
            tem_temHeight = Scale1;                                  //设置刻度线的最小长度
            float j = (i + (int)StartBitH) / Degree;                 //获取刻度值
            tem_value = "";
            j = (int)j;                                              //对刻度值进行取整
            if (j % (divide * 2) == 0)                               //如果刻度值是 10 进位
            {
                tem_temHeight = Scale10;                             //设置最长的刻度线
                if (UnitStyle == Unit.Cm)                            //如果刻度的单位为厘米
                    tem_value = Convert.ToString(j / 10);            //记录刻度值
                if (UnitStyle == Unit.Pels)                          //如果刻度的单位为像素
                    tem_value = Convert.ToString((int)j * 10);       //记录刻度值
            }
            else if (j % divide == 0)                                //如果刻度值的进位为 5
            {
                tem_temHeight = Scale5;                              //设置刻度线为中等
            }
            tem_p.Width = 1;
            if (RulerStyle == Ruler.Graduation)                      //如果是以刻度值进行测量
            {
                //绘制刻度线
                e.Graphics.DrawLine(tem_p, i + 1 + thisleft, SpaceBetween, i + 1 + thisleft, SpaceBetween + tem_temHeight);
                if (tem_value.Length > 0)                            //如果有刻度值
                    //绘制刻度值
                    ProtractString(e.Graphics, tem_value.Trim(), i + 1 + thisleft, SpaceBetween, i + 1 + thisleft, SpaceBetween + tem_temHeight, 0);
            }
            if (RulerStyle == Ruler.Rule)                            //如果是以标尺进行测量
            {
                if (tem_value.Length > 0)                            //如果有刻度值
                {
                    //绘制顶端的刻度线
                    e.Graphics.DrawLine(tem_p, i + 1 + thisleft, 4, i + 1 + thisleft, 7);
                    //绘制底端的刻度线
                    e.Graphics.DrawLine(tem_p, i + 1 + thisleft, Distance_Y - 9, i + 1 + thisleft, Distance_Y - 7);
                    //设置文本的横向位置
                    float tem_space = 3 + (Distance_X - 19F - 9F - StringSize(tem_value.Trim(),this.Font, false)) / 2F;
                    ProtractString(e.Graphics, tem_value.Trim(), i + 1 + thisleft, (float)Math.Ceiling(tem_space), i + 1 + thisleft, (float)Math.
Ceiling(tem_space) + tem_temHeight, 0);                              //绘制文本
                }
            }
            i += Degree;                                             //累加刻度的宽度
        }
        //纵向刻度的设置
        if (UnitStyle == Unit.Cm)                                    //如果刻度的单位是厘米
            Degree = e.Graphics.DpiX / 25.4F;                        //将厘米转换成像素
        if (UnitStyle == Unit.Pels)                                  //如果刻度的单位是像素
            Degree = 10;                                             //刻度值设为 10 像素
```

```
    int tem_height = this.Height - 3;
    tem_n = (int)StartBitV;                                          //记录纵向滚动条的位置
    if (tem_n != StartBitV)                                          //如果纵向滚动条的位置为小数
        StartBitV = (int)StartBitV;                                  //对其进行取整
    for (float i = 0; i < tem_height; )                              //在文本框的左端绘制标尺
    {
        tem_temHeight = Scale1;                                      //设置刻度线的最小值
        float j = (i + (int)StartBitV) / Degree;                     //获取当前的刻度值
        tem_value = "";
        j = (int)j;                                                  //对刻度值进行取整
        if (j % 10 == 0)                                             //如果刻度值是 10 进位
        {
            tem_temHeight = Scale10;                                 //设置刻度线的长度为最长
            if (UnitStyle == Unit.Cm)                                //如果刻度的单位是厘米
                tem_value = Convert.ToString(j / 10);                //获取厘米的刻度值
            if (UnitStyle == Unit.Pels)                              //如果刻度的单位是像素
                tem_value = Convert.ToString((int)j * 10);           //获取像素的刻度值
        }
        else if (j % 5 == 0)                                         //如果刻度值是 5 进位
        {
            tem_temHeight = Scale5;                                  //设置刻度线的长度为中等
        }
        tem_p.Width = 1;
        if (RulerStyle == Ruler.Graduation)                          //如果是以刻度值进行测量
        {
            //绘制刻度线
            e.Graphics.DrawLine(tem_p, SpaceBetween, i + 1 + Distance_Y, SpaceBetween + tem_temHeight, i + 1 + Distance_Y);
            if (tem_value.Length > 0)                                //如果有刻度值
                ProtractString(e.Graphics, tem_value.Trim(), SpaceBetween, i + 1 + Distance_Y, SpaceBetween +
tem_temHeight, i + 1 + Distance_Y, 1);                               //绘制刻度值
        }
        if (RulerStyle == Ruler.Rule)                                //如果是以标尺进行测量
        {
            if (tem_value.Length > 0)                                //如果有刻度值
            {
                //绘制左端刻度线
                e.Graphics.DrawLine(tem_p, 4, i + 1 + Distance_Y, 7, i + 1 + Distance_Y);
                //绘制右端刻度线
                e.Graphics.DrawLine(tem_p, Distance_Y - 9, i + 1 + Distance_Y, Distance_Y - 7, i + 1 + Distance_Y);
                //设置文本的纵向位置
                float tem_space = 3 + (Distance_X - 19F - 9F - StringSize(tem_value.Trim(),this.Font, false)) / 2F;
                ProtractString(e.Graphics, tem_value.Trim(), (float)Math.Floor(tem_space), i + 1 + Distance_Y, (float)Math.Floor(tem_space) +
tem_temHeight, i + 1 + Distance_Y, 1);                               //绘制文本
            }
        }
        i += Degree;                                                 //累加刻度值
    }
    if (CodeShow)                                                    //如果显示行号
    {
        float tem_FontHeight = (float)(richTextBox1.Height / (StringSize(CodeSize.ToString(), richTextBox1.Font,
false)));                                                            //设置文本的高度
        float tem_tep = richTextBox1.Top;                            //获取文本框的顶端位置
        int tem_mark = 0;
        for (int i = 0; i < (int)tem_FontHeight; i++)                //绘制行号
        {
            tem_mark = i + (int)CodeSize;                            //设置代码编号的宽度
            //绘制行号
            e.Graphics.DrawString(tem_mark.ToString(), this.Font, new SolidBrush(Color.Red), new PointF(richTextBox1.Left - StringSize(tem_
mark.ToString(), this.Font, true) - 2, tem_tep));
            tem_tep = tem_tep + StringSize("懂", richTextBox1.Font, false);   //设置下一个行号的 X 坐标值
        }
    }
}
```

自定义方法 ProtractString 主要是在指定的位置根据参数 n 绘制横向或纵向字符串。代码如下：

```
/// <summary>
///在指定的位置绘制文本信息
```

```
/// </summary>
/// <param e="Graphics">封装一个绘图的对象</param>
/// <param str="string">文本信息</param>
/// <param x1="float">左上角 x 坐标</param>
/// <param y1="float">左上角 y 坐标</param>
/// <param x2="float">右下角 x 坐标</param>
/// <param y2="float">右下角 y 坐标</param>
/// <param n="float">标识，判断是在横向标尺上绘制文字还是在纵向标尺上绘制文字</param>
public void ProtractString(Graphics e, string str, float x1, float y1, float x2, float y2, float n)
{
    float TitWidth = StringSize(str,this.Font, true);                    //获取字符串的宽度
    if (n == 0)                                                          //在横向标尺上绘制文字
        e.DrawString(str, this.Font, new SolidBrush(Color.Black), new PointF(x2 - TitWidth / 2, y2 + 1));
    else                                                                 //在纵向标尺上绘制文字
    {
        StringFormat drawFormat = new StringFormat();                    //创建 StringFormat 对象
        drawFormat.FormatFlags = StringFormatFlags.DirectionVertical;    //设置文本为垂直对齐
        //绘制指定的文本
        e.DrawString(str, this.Font, new SolidBrush(Color.Black), new PointF(x2 + 1, y2 - TitWidth / 2), drawFormat);
    }
}
```

自定义方法 StringSize 主要是根据字符串和字符串的字体样式，通过参数 n 获取字符串的高度或宽度。代码如下：

```
/// <summary>
///获取文本的高度或宽度
/// </summary>
/// <param str="string">文本信息</param>
/// <param font="Font">字体样式</param>
/// <param n="bool">标识，判断返回的是高度还是宽度</param>
public float StringSize(string str,Font font,bool n)                     //n==true 为 width
{
    Graphics TitG = this.CreateGraphics();                               //创建 Graphics 对象
    SizeF TitSize = TitG.MeasureString(str, font);                       //将绘制的字符串进行格式化
    float TitWidth = TitSize.Width;                                      //获取字符串的宽度
    float TitHeight = TitSize.Height;                                    //获取字符串的高度
    if (n)
        return TitWidth;                                                 //返回文本信息的宽度
    else
        return TitHeight;                                                //返回文本信息的高度
}
```

GuageRichTextBox 控件的 Resize 事件主要是在控件大小改变时，设置 richTextBox1 控件的大小，以及对该控件进行重绘。代码如下：

```
private void GuageRichTextBox_Resize(object sender, EventArgs e)
{
    richTextBox1.Top = Distance_X;                                       //设置控件的顶端位置
    richTextBox1.Left = Distance_Y;                                      //设置控件的左端位置
    richTextBox1.Width = this.Width - Distance_X - 2;                    //设置控件的宽度
    richTextBox1.Height = this.Height - Distance_Y - 2;                  //设置控件的高度
    this.Invalidate();                                                   //控件重绘
}
```

richTextBox1 控件的 HScroll 事件是在改变该控件的横向滚动条时，获取当前第一个字符 X 坐标的位置。代码如下：

```
private void richTextBox1_HScroll(object sender, EventArgs e)
{
    //检索控件横向内指定字符索引处的位置
    StartBitH = (int)(Math.Abs((float)richTextBox1.GetPositionFromCharIndex(0).X - 1));
    this.Invalidate();
}
```

richTextBox1 控件的 VScroll 事件是在改变该控件的纵向滚动条时，获取当前第一个字符 Y 坐标的位置。如果显示行号，获取行号的高度，代码如下：

```
private void richTextBox1_VScroll(object sender, EventArgs e)
{
```

```
        //检索控件纵向内指定字符索引处的位置
        StartBitV = (int)(Math.Abs((float)richTextBox1.GetPositionFromCharIndex(0).Y - 1));
        if (CodeShow)                                                    //如果显示行号
            //设置行号的高度
            CodeSize = (int)Math.Abs((richTextBox1.GetPositionFromCharIndex(0).Y / StringSize("懂", richTextBox1.Font, false)));
        this.Invalidate();
    }
```

秘笈心法

心法领悟 351：RichTextBox 与 TextBox 控件的区别。

RichTextBox 控件用于显示、输入和操作带有格式的文本，而 TextBox 控件用于获取用户输入或显示文本，通常用于可编辑文本，也可以成为只读控件。它们两者之间的区别在于，RichTextBox 控件可以显示字体、颜色和链接，从文件中加载文本和嵌入的图像，撤销和重复编辑操作以及查找指定的字符，而 TextBox 控件不能。

实例 352　自制平滑进度条控件

光盘位置：光盘\MR\14\352　　高级　　趣味指数：★★★★★

实例说明

一般情况下，进度条都是以块状的形式出现，如当电脑启动时在 Windows 图标下的滚动条。一种形式的事物看的时间久了，即使再美观也难以吸引人们的眼球。通过对本实例的学习，用户可以实现平滑滚动条。实例运行效果如图 14.12 所示。

图 14.12　自制平滑进度条控件

关键技术

本实例在绘制平滑进度条时，主要用到了 Graphics 类的 FillRectangle 和 DrawLine 方法，其中，FillRectangle 方法用来绘制平滑进度条的矩形区域，DrawLine 方法用来绘制平滑进度条的上、下、左、右边线。下面对本实例中用到的关键技术进行详细讲解。

Graphics 类的 DrawLine 方法用来绘制一条连接由坐标对指定的两个点的线条，该方法有 4 种重载形式，本实例中用到的重载形式如下：

```
public void DrawLine(Pen pen,Point pt1,Point pt2)
```

参数说明

❶ pen：Pen 对象，它确定线条的颜色、宽度和样式。

❷ pt1：Point 结构，它表示要连接的第一个点。

❸ pt2：Point 结构，它表示要连接的第二个点。

说明：关于 Graphics 类的 FillRectangle 方法的详细讲解请参见实例 348 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulProgressBar。

（2）创建 BeautifulProgressBar 窗体应用程序时，程序自动生成一个 BeautifulProgressBar 解决方案，在该解决方案中添加一个 Windows 窗体控件库项目，并将其命名为 WindowsFormsControlLibrary，修改默认用户控件为 SmoothProgressBar.cs。

（3）程序的主要代码如下。

在 SmoothProgressBar 控件中定义进度条的最小值属性。代码如下：

```
public int Minimum
{
    get
```

```
    {
        return min;                                          //返回最小值
    }
    set
    {
        if(value < 0)                                        //当该值小于 0 时
        {
            min = 0;                                         //设置最小值为 0
        }
        if(value > max)                                      //当该值大于最大值时
        {
            min = value;                                     //设定最小值为该值
        }
        if(val < min)                                        //当前值小于最小值时
        {
            val = min;                                       //设定当前值为最小值
        }
        this.Invalidate();                                   //使当前操作区域无效从而导致重绘事件
    }
}
```

在 SmoothProgressBar 控件中定义进度条的最大值属性。代码如下：

```
public int Maximum
{
    get
    {
        return max;                                          //返回最大值
    }
    set
    {
        if(value < min)                                      //当前值小于最小值时
        {
            min = value;                                     //设置最小值为当前值
        }
        max = value;                                         //设置最大值为当前值
        if(val > max)                                        //当前值大于最大值时
        {
            val = max;                                       //设置当前值为最大值
        }
        this.Invalidate();                                   //使当前操作区域无效并导致重绘事件
    }
}
```

在 SmoothProgressBar 控件中定义进度条的当前值属性。代码如下：

```
public int Value
{
    get
    {
        return val;                                          //返回该值
    }
    set
    {
        int oldValue = val;                                  //保存当前值
        if(value < min)                                      //当该值小于最小值时
        {
            val = min;                                       //设置该值为最小值
        }
        else if(value > max)                                 //当该值大于最大值时
        {
            val = max;                                       //设置该值为最大值
        }
        else                                                 //当该值介于最小值和最大值之间时
        {
            val = value;                                     //设置当前值为该值
        }
        float percent;                                       //保存进度条中的值所占总长度的百分比
        Rectangle newValueRect = this.ClientRectangle;       //初始化一个新的工作区域对象
        Rectangle oldValueRect = this.ClientRectangle;       //初始化一个旧的工作区域对象
```

```
        percent = (float)(val - min) / (float)(max - min);                    //用进度条的当前值初始化变量 percent
        newValueRect.Width = (int)((float)newValueRect.Width * percent);      //设置新工作区域对象的宽度
        percent = (float)(oldValue - min) / (float)(max - min);               //用进度条的旧值初始化变量 percent
        oldValueRect.Width = (int)((float)oldValueRect.Width * percent);      //设置旧工作区域对象的宽度
        Rectangle updateRect = new Rectangle();                               //初始化一个更新区域的对象
        if(newValueRect.Width > oldValueRect.Width)                           //当新工作区域大于旧工作区域时
        {
            updateRect.X = oldValueRect.Size.Width;                           //设置更新区域左上角的 X 坐标
            updateRect.Width = newValueRect.Width - oldValueRect.Width;       //设置更新区域的宽度
        }
        else                                                                  //当新工作区域小于或等于旧工作区域时
        {
            updateRect.X = newValueRect.Size.Width;                           //设置更新区域左上角的 X 坐标
            updateRect.Width = oldValueRect.Width - newValueRect.Width;       //设置更新区域的宽度
        }
        updateRect.Height = this.Height;                                      //设置更新区域的高度
        this.Invalidate(updateRect);                                          //使当前操作区域无效并导致重绘事件
    }
}
```

SmoothProgressBar 控件中重写控件的 OnPaint 方法，在该方法中实现绘制平滑进度条的功能。代码如下：

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;                                                  //初始化一个 GDI+绘图图面对象
    SolidBrush brush = new SolidBrush(BarColor);                              //初始化一个单色画笔类
    float percent = (float)(val - min) / (float)(max - min);                  //保存当前滚动条中值所占总长度的百分比
    Rectangle rect = this.ClientRectangle;                                    //设置当前的工作区
    rect.Width = (int)((float)rect.Width * percent);                          //计算进度条的宽度
    g.FillRectangle(brush,rect);                                              //用指定的对象填充工作区
    Draw3DBorder(g);                                                          //使控件实现 3D 效果
    brush.Dispose();                                                          //释放画刷所占用的资源
    g.Dispose();                                                              //释放 GDI+绘图图面对象所占用的资源
}
```

秘笈心法

心法领悟 352：使用 Process 组件打开系统进程。

Process 组件提供对正在计算机上运行的进程的访问，使用它可以打开系统中的程序。例如，使用 Process 组件打开 IE 浏览器的实现代码如下：

```
process1.StartInfo.FileName = "IExplore.exe";                                //指定要打开的进程名
process1.Start();                                                            //启动进程
```

实例 353　制作 Vista 风格的日历

光盘位置：光盘\MR\14\353　　高级　　趣味指数：★★★★★

实例说明

在人们的日常生活中，查询日期是非常普遍的。本实例将制作一个简单的 Vista 风格的日历，只显示当前系统日期的年、月、日，以及当前日期所对应的甲子年和农历。实例运行效果如图 14.13 所示。

图 14.13　制作 Vista 风格的日历

关键技术

本实例主要使用 DateTime 结构来获取当前系统的日期，以及获取日期中的年、月、日，然后用 ChineseLunisolarCalendar 类的 GetSexagenaryYear 方法获取指定日期的甲子年。下面对本实例中用到的关键技术进行详细讲解。

（1）DateTime 结构

该结构表示时间上的一刻，通常以日期和当天的时间表示，本实例主要使用其相关属性获取日期中的年、

月、日等信息。DateTime 结构的常用属性及说明如表 14.3 所示。

表 14.3　DateTime 结构的常用属性及说明

属　性	说　明
Date	获取 DateTime 结构的日期部分
Day	获取 DateTime 结构所表示的日期为该月中的第几天
DayOfWeek	获取 DateTime 结构所表示的日期是星期几
DayOfYear	获取 DateTime 结构所表示的日期是该年中的第几天
Hour	获取 DateTime 结构所表示日期的小时部分
Millisecond	获取 DateTime 结构所表示日期的毫秒部分
Minute	获取 DateTime 结构所表示日期的分钟部分
Month	获取 DateTime 结构所表示日期的月份部分
Now	获取一个 DateTime 对象，该对象设置为此计算机上的当前日期和时间，表示为本地时间
TimeOfDay	获取 DateTime 结构的当天的时间
Today	获取当前日期
Year	获取 DateTime 结构所表示日期的年份部分

（2）ChineseLunisolarCalendar 类的 GetSexagenaryYear 方法

该方法用于计算与指定日期对应的甲子（60 年）循环中的年。语法格式如下：

```
public virtual int GetSexagenaryYear (DateTime time)
```

参数说明

❶ time：要读取的 DateTime 对象。

❷ 返回值：甲子循环中的一个从 1～60 的数字，它与 date 参数对应。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BeautifulCalendar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，设置其 FormBorderStyle 属性为 None，BackColor 和 TransparencyKey 属性为 Color.red，以便使窗体处于透明状态。

（3）在当前项目中添加一个用户控件，并将其命名为 DateTimeControl。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

（4）程序的主要代码如下。

在 dateTimeControl 控件的“代码编辑”窗口中设置变量和常量值。代码如下：

```
int yy = 0;                                                    //年
int mm = 0;                                                    //月
int dd = 0;                                                    //日
string yymm = "";                                              //当前年月
public const string CelestialStem = "甲乙丙丁戊己庚辛壬癸";
public const string TerrestrialBranch = "子丑寅卯辰巳午未申酉戌亥";
public const string Animals = "鼠牛虎兔龙蛇马羊猴鸡狗猪";
public const string ChineseNumber = "〇一二三四五六七八九";
```

在 dateTimeControl 控件的 Paint 事件中，将当前日期的年、月、日，以及相应的农历年份在指定的位置进行绘制。代码如下：

```
private void DateTimeControl_Paint(object sender, PaintEventArgs e)
{
    Image img = new Bitmap(Properties.Resources.backdrop1);                    //从资源文件中读取背景图片
    e.Graphics.DrawImageUnscaled(img, new Point(0, 0));                        //绘制背景图片
    yymm = yy.ToString() + "年" + "   " + mm.ToString() + "月";               //记录当前的年月
    Font Myfont = new Font("幼圆", 12, FontStyle.Bold);                        //设置字体样式
    float Date_Y = (this.Width - StringSize(yymm, Myfont, true)) / 2F;         //设置文本的显示位置
    //绘制日期
```

```
    e.Graphics.DrawString(yymm, Myfont, new SolidBrush(Color.White), new PointF(Date_Y, 20F));
    Myfont = new Font("新宋体", 65,FontStyle.Bold);                          //设置天的字体样式
    float Date_D = (this.Width - StringSize(dd.ToString(), Myfont, true)) / 2F;          //设置文本的位置
    //绘制天
    e.Graphics.DrawString(dd.ToString(), Myfont, new SolidBrush(Color.Black), new PointF(Date_D + 3, 30F));
    //绘制天的阴影
    e.Graphics.DrawString(dd.ToString(), Myfont, new SolidBrush(Color.White), new PointF(Date_D, 30F));
    string MyWeek = GetDayOfWeekString(this.Date);                           //获取今天是周几
    Myfont = new Font("幼圆", 12, FontStyle.Bold);                           //设置周的字体样式
    Date_Y = (this.Width - StringSize(MyWeek, Myfont, true)) / 2F;           //设置文本的位置
    //绘制今天是周几
    e.Graphics.DrawString(MyWeek, Myfont, new SolidBrush(Color.White), new PointF(Date_Y, 110F));
    int tem_n = calendar.GetSexagenaryYear(Date);                            //获取今天在哪个甲子年中
    //获取农历年干支
    string Armour = CelestialStem.Substring((tem_n - 1) % 10, 1) + TerrestrialBranch.Substring((tem_n - 1) % 12, 1);
    tem_n = calendar.GetSexagenaryYear(Date);                                //获取今天在哪个甲子年中
    string Resemble = Animals.Substring((tem_n - 1) % 12, 1);                //获取生肖
    //农历日期
    string tem_embolism = "";                                                //记录是否为闰年
    string tem_Lunardata = "";                                               //记录农历日期
    GetLunarCalendar(Date, out tem_embolism, out tem_Lunardata);             //获取农历的相关信息
    //记录农历的信息
    string LunarInfo = Armour + Resemble + tem_embolism + " " + LunarYear(Date.Year) + "年" + tem_Lunardata;
    Myfont = new Font("黑体", 7.5F);                                         //设置文本的字体样式
    Date_Y = (this.Width - StringSize(LunarInfo, Myfont, true)) / 2F;        //设置农历的显示位置
    //绘制农历
    e.Graphics.DrawString(LunarInfo, Myfont, new SolidBrush(Color.White), new PointF(Date_Y, 130F));
}
```

自定义方法 LunarYear 将获取的农历年份转换成大写形式。代码如下：

```
///<summary>
///将农历年份转换成大写
///</summary>
///<param name="date">阳历年份</param>
///<returns>农历年份</returns>
public string LunarYear(int Y)
{
    string m_LunarYearText = "";
    StringBuilder sb = new StringBuilder();                                  //创建 StringBuilder 对象
    int year = Y;                                                            //记录年份
    int d;
    do
    {
        d = year % 10;                                                       //获取年份的最后一位数字
        sb.Insert(0, ChineseNumber[d]);                                      //将数字转换成大写
        year = year / 10;                                                    //舍去年份的最后一个数字
    } while (year > 0);                                                      //如果年份大于 0，继续循环
    m_LunarYearText = sb.ToString();                                         //获取当前年份的大写
    return m_LunarYearText;
}
```

自定义方法 GetLunarCalendar 根据指定的日期获取当前日期是否为闰年，以及该日期所对应的农历日期，并用 embolism 和 Lunardata 参数返回。代码如下：

```
///<summary>
///获取对应日期的农历及其他信息
///</summary>
///<param name="dtDay">农历日期</param>
///<returns>农历信息</returns>
public static void GetLunarCalendar(DateTime dtDay, out string embolism, out string Lunardata)
{
    string sYear = dtDay.Year.ToString();                                    //获取当前日期的年份
    string sMonth = dtDay.Month.ToString();                                  //获取当前日期的月份
    string sDay = dtDay.Day.ToString();                                      //获取当前日期的日
    int year;
    int month;
    int day;
    try
```

```
{
    year = int.Parse(sYear);                                    //将年转换成 32 位整数
    month = int.Parse(sMonth);                                  //将月转换成 32 位整数
    day = int.Parse(sDay);                                      //将日转换成 32 位整数
}
catch
{
    year = DateTime.Now.Year;                                   //获取系统时间的年份
    month = DateTime.Now.Month;                                 //获取系统时间的月份
    day = DateTime.Now.Day;                                     //获取系统时间的日
}
int nTheDate;
int nIsEnd;
int k, m, n, nBit, i;
string calendar = string.Empty;
//计算到初始时间 1921 年 2 月 8 日的天数：1921-2-8（正月初一）
nTheDate = (year - 1921) * 365 + (year - 1921) / 4 + day + MonthAdd[month - 1] - 38;
if ((year % 4 == 0) && (month > 2))
    nTheDate += 1;
//计算天干，地支，月，日
nIsEnd = 0;
m = 0;
k = 0;
n = 0;
while (nIsEnd != 1)
{
    if (LunarData[m] < 4095)
        k = 11;
    else
        k = 12;
    n = k;
    while (n >= 0)
    {
        //获取 LunarData[m]的第 n 个二进制位的值
        nBit = LunarData[m];
        for (i = 1; i < n + 1; i++)
            nBit = nBit / 2;
        nBit = nBit % 2;
        if (nTheDate <= (29 + nBit))
        {
            nIsEnd = 1;
            break;
        }
        nTheDate = nTheDate - 29 - nBit;
        n = n - 1;
    }
    if (nIsEnd == 1)
        break;
    m = m + 1;
}
year = 1921 + m;
month = k - n + 1;
day = nTheDate;
#region 格式化日期显示为三月廿四
if (k == 12)
{
    if (month == LunarData[m] / 65536 + 1)
        month = 1 - month;
    else if (month > LunarData[m] / 65536 + 1)
        month = month - 1;
}
//农历月
if (month < 1)
{
    embolism = "闰";
    Lunardata = MonthName[-1 * month].ToString() + "月";
}
else
```

```
        {
            embolism = "";
            Lunardata = MonthName[month].ToString() + "月";
        }
        //农历日
        Lunardata += DayName[day].ToString();
        #endregion
    }
```

秘笈心法

心法领悟 353：使用 DateTime.Now 属性获取系统时间。

使用 DateTime.Now 属性可以方便地获取系统时间，而且可以使用格式化方式方便地得到系统时间的字符串表示形式，代码如下：

```
string P_str = DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒");
```

实例 354　DataGridView 和 BindingSource 复合控件

高级

光盘位置：光盘\MR\14\354

趣味指数：★★★★★

实例说明

复合控件是封装在公共容器内的 Windows 窗体控件的集合，这种控件通常称为用户控件，其中包含的子控件称为构成控件。复合控件包含与每个构成控件相关联的所有固有功能，允许程序设计人员有选择地公开和绑定它们的属性。本实例将使用 DataGridView 控件和 BindingSource 组件制作复合控件来显示学生档案信息。实例运行效果如图 14.14 所示。

图 14.14　DataGridView 和 BindingSource 复合控件

关键技术

本实例实现时主要用到了 DataGridView 控件和 BindingSource 组件，下面分别对它们进行详细讲解。

（1）DataGridView 控件

DataGridView 控件，又称为数据表格控件，它提供一种强大而灵活的以表格形式显示数据的方式，可以使用 DataGridView 控件来显示少量数据的只读视图，也可以对其进行缩放以显示特大数据集的可编辑视图。使用 DataGridView 控件，可以显示和编辑来自多种不同类型的数据源的表格数据。将数据绑定到 DataGridView 控件非常简单和直观，在大多数情况下，只需设置 DataSource 属性即可。另外，DataGridView 控件具有极高的可配置性和可扩展性，它提供有大量的属性、方法和事件，可以用来对该控件的外观和行为进行自定义。当需要在 Windows 窗体应用程序中显示表格数据时，首先考虑使用 DataGridView 控件。若要以小型网格显示只读值或者使用户能够编辑具有数百万条记录的表，DataGridView 控件将提供可以方便地进行编程以及有效地利用内存的解决方案。

DataGridView 控件的常用属性及说明如表 14.4 所示。

表 14.4　DataGridView 控件的常用属性及说明

属　性	说　明
AllowUserToAddRows	获取或设置一个值，该值指示是否向用户显示添加行的选项
AllowUserToDeleteRows	获取或设置一个值，该值指示是否允许用户从 DataGridView 中删除行
AlternatingRowsDefaultCellStyle	获取或设置应用于 DataGridView 的奇数行的默认单元格样式
ColumnCount	获取或设置 DataGridView 中显示的列数

续表

属　性	说　明
Columns	获取一个包含控件中所有列的集合
CurrentCell	获取或设置当前处于活动状态的单元格
CurrentRow	获取包含当前单元格的行
DataMember	获取或设置数据源中 DataGridView 显示其数据的列表或表的名称
DataSource	获取或设置 DataGridView 所显示数据的数据源
MultiSelect	获取或设置一个值，该值指示是否允许用户一次选择 DataGridView 的多个单元格、行或列
NewRowIndex	获取新记录所在行的索引
RowCount	获取或设置 DataGridView 中显示的行数
Rows	获取一个集合，该集合包含 DataGridView 控件中的所有行
SortOrder	获取一个值，该值指示是按升序或降序对 DataGridView 控件中的项进行排序，还是不排序

DataGridView 控件的常用事件及说明如表 14.5 所示。

表 14.5　DataGridView 控件的常用事件及说明

事　件	说　明
CellClick	在单元格的任何部分被单击时发生
CellContentClick	在单元格中的内容被单击时发生
CellContentDoubleClick	在用户双击单元格的内容时发生
CellDoubleClick	在用户双击单元格中的任何位置时发生
CellMouseClick	在用户用鼠标单击单元格中的任何位置时发生
CellMouseDoubleClick	在双击 DataGridView 中的单元格时发生
Sorted	在 DataGridView 控件完成排序操作时发生

（2）BindingSource 组件

BindingSource 组件，又称为数据源绑定组件，它主要用于封装窗体的数据源。

BindingSource 组件的常用属性及说明如表 14.6 所示。

表 14.6　BindingSource 组件的常用属性及说明

属　性	说　明
AllowNew	获取或设置一个值，该值指示是否可以使用 AddNew 方法向列表中添加项
AllowRemove	获取一个值，它指示是否可从基础列表中移除项
Count	获取基础列表中的总项数
Current	获取列表中的当前项
DataMember	获取或设置连接器当前绑定到的数据源中的特定列表
DataSource	获取或设置连接器绑定到的数据源
Item	获取或设置指定索引处的列表元素
List	获取连接器绑定到的列表
Position	获取或设置基础列表中当前项的索引
Sort	获取或设置用于排序的列名称以及用于查看数据源中的行的排序顺序
SortDirection	获取列表中项的排序方向
SupportsSearching	获取一个值，它指示数据源是否支持使用 Find 方法进行搜索
SupportsSorting	获取一个值，它指示数据源是否支持排序

BindingSource 组件的常用方法及说明如表 14.7 所示。

表 14.7　BindingSource 组件的常用方法及说明

方　法	说　明
Add	将现有项添加到内部列表中
AddNew	向基础列表添加新项
ApplySort	使用指定的排序说明对数据源进行排序
Clear	从列表中移除所有元素
CopyTo	将 List 中的内容复制到指定数组，从指定索引值处开始
Find	在数据源中查找指定的项
GetEnumerator	检索 List 的一个枚举数
IndexOf	搜索指定的对象，并返回整个列表中第一个匹配项的索引
Insert	将一项插入列表中指定的索引处
MoveFirst	移至列表中的第一项
MoveLast	移至列表中的最后一项
MoveNext	移至列表中的下一项
MovePrevious	移至列表中的上一项
Remove	从列表中移除指定的项
RemoveAt	移除此列表中指定索引处的项
RemoveCurrent	从列表中移除当前项

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ComplexControl。

（2）在当前项目中添加一个用户控件，并将其命名为 UserControl1。在用户控件中添加一个 DataGridView 控件和一个 BindingSource 组件，并在 DataGridView 控件的 Columns 属性中添加描述学生信息的字段。

说明：关于如何创建用户控件，请参见实例 346 中的关键技术。

（3）程序的主要代码如下：

```
public void BindData()
{
    //创建数据库连接对象
    SqlConnection con = new SqlConnection("server=mrwxk\\wangxiaoke;uid=sa;pwd=;database=db_TomeOne;");
    con.Open();                                                          //打开数据库连接
    SqlDataAdapter sda = new SqlDataAdapter("Select * From tb_Student", con);   //创建桥接器对象
    DataTable dt = new DataTable();                                      //创建 DataTable 对象
    try
    {
        sda.Fill(dt);                                                    //填充 DataTable
    }
    catch (Exception ex)
    {
        throw ex;                                                        //抛出异常
    }
    bindingSource1.DataSource = dt;                                      //指定 BindingSource 数据源
    dataGridView1.DataSource = bindingSource1;                           //将 BindingSource 指定给 DataGridView
}
```

秘笈心法

心法领悟 354：使 DataGridView 单元格中的内容换行。

通过设置 DataGridView 控件的 DefaultCellStyle 属性的 WrapMode 属性可以实现单元格文本换行功能。代码如下：

```
dataGridView1.DefaultCellStyle.WrapMode = DataGridViewTriState.True;
```

文件操作篇

- 第 15 章　文件基本操作
- 第 16 章　文件夹基本操作
- 第 17 章　文件流操作
- 第 18 章　加密、解密及解压缩文件

第15章

文件基本操作

- 获取文件基本信息
- 创建和删除文件
- 查找文件
- 修改文件
- 复制文件
- 指定类型的文件操作
- 其他

15.1　获取文件基本信息

实例 355	获取文件的大小	初级
	光盘位置：光盘\MR\15\355	趣味指数：★★★★☆

实例说明

本实例实现了获取文件大小的功能，运行本实例，单击“选择文件判断大小”按钮，弹出“打开”对话框，在该对话框中选择一个文件，单击“确定”按钮，即可弹出对话框显示选中文件的大小。实例运行效果如图 15.1 所示。

图 15.1　获取文件的大小

关键技术

本实例实现时，使用 File 类的 Open 方法打开指定的文件，然后使用 FileStream 类的 Length 属性获取文件的长度，下面对本实例中用到的关键技术进行详细讲解。

（1）File 类的 Open 方法

File 类的 Open 方法用来打开指定路径上的 FileStream，具有读/写访问权限。语法格式如下：

```
public static FileStream Open(string path,FileMode mode)
```

参数说明

❶ path：要打开的文件。

❷ mode：FileMode 枚举值之一，用于指定在文件不存在时是否创建该文件，并确定是保留还是覆盖现有文件的内容。FileMode 枚举值及说明如表 15.1 所示。

❸ 返回值：以指定模式打开的指定路径上的 FileStream，具有读/写访问权限并且不共享。

表 15.1　FileMode 枚举值及说明

枚 举 值	描　述
CreateNew	指定操作系统应创建新文件
Create	指定操作系统应创建新文件。如果文件已存在，它将被覆盖
Open	指定操作系统应打开现有文件
OpenOrCreate	指定操作系统应打开文件（如果文件存在）；否则，应创建新文件
Truncate	指定操作系统应打开现有文件。文件一旦打开，就将被截断为零字节大小
Append	打开现有文件并查找到文件尾，或创建新文件

（2）FileStream 类的 Length 属性

FileStream 类的 Length 属性用来获取用字节表示的流长度。语法格式如下：

```
public override long Length { get; }
```

参数说明

属性值：用字节表示流长度的数值。

说明：① 程序中使用 File 类时，首先需要在命名空间区域添加 System.IO 命名空间。

② 程序中使用与文件相关的类时，都需要添加 System.IO 命名空间，下面遇到类似情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetFileLength。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来打开指定的文件，并获取其大小。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                    //创建打开文件对话框对象
    if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)                      //判断是否选中文件
    {
        MessageBox.Show("文件长度：" +                                          //弹出消息对话框
            File.Open(P_OpenFileDialog.FileName, FileMode.Open).
            Length.ToString()+"字节", "提示！");
    }
}
```

秘笈心法

心法领悟 355：使用 OpenFileDialog 控件作为“打开”对话框。

打开对话框使用 OpenFileDialog 控件来表示，它表示一个通用对话框，用户可以使用此对话框来指定一个或多个要打开的文件的文件名。开发人员可以使用 OpenFileDialog 控件的 ShowDialog 方法显示“打开”对话框。

实例 356 获取文件扩展名

光盘位置：光盘\MR\15\356　　初级　　趣味指数：★★★★☆

实例说明

本实例实现了获取文件扩展名的功能，运行本实例，单击“获取文件扩展名”按钮，弹出“打开”对话框，在该对话框中选择一个文件，单击“确定”按钮，即可弹出对话框显示选中文件的扩展名。实例运行效果如图 15.2 所示。

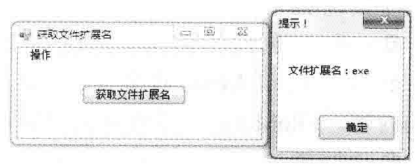


图 15.2　获取文件扩展名

关键技术

本实例获取文件扩展名时，主要是通过从文件名中截取相应的字符串来实现的，其中主要用到了 string 类的 SubString 方法，下面对其进行详细介绍。

SubString 方法用来从指定的字符位置开始截取指定长度的字符串。语法格式如下：

```
public string Substring ( int startIndex, int length)
```

参数说明

❶ startIndex：子字符串的起始位置的索引。

❷ length：要截取的字符数。

❸ 返回值：一个 String 文本字符串，表示截取的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileExt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来打开指定的文件，并获取其扩展名。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
```

```
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                    //创建打开文件对话框对象
    if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)                      //判断是否选中文件
    {
        MessageBox.Show("文件扩展名：" +                                        //弹出消息对话框
            P_OpenFileDialog.FileName.Substring(
            P_OpenFileDialog.FileName.LastIndexOf(".")+1,
            P_OpenFileDialog.FileName.Length-
        P_OpenFileDialog.FileName.LastIndexOf(".")-1), "提示！ ");
    }
}
```

秘笈心法

心法领悟 356：如何获取“打开”对话框的返回值？

“打开”对话框的返回值可以使用 DialogResult 枚举来获取，该枚举指定标识符以指示对话框的返回值。例如，要获取“打开”对话框的返回值是否为“确定”按钮，可以使用如下代码实现：

```
if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)
```

实例 357　获取文件创建时间

光盘位置：光盘\MR\15\357　　初级　　趣味指数：★★★☆

实例说明

本实例实现了获取文件创建时间的功能，运行本实例，单击“获取创建时间”按钮，弹出“打开”对话框，在该对话框中选择一个文件，单击“确定”按钮，即可弹出对话框显示选中文件的创建时间。实例运行效果如图 15.3 所示。

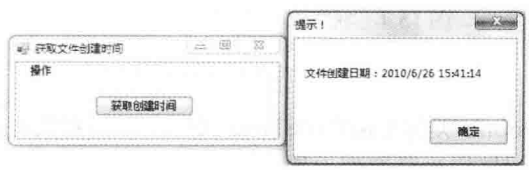


图 15.3　获取文件创建时间

关键技术

本实例实现时主要用到了 FileInfo 类的 CreationTime 属性，下面对其进行详细讲解。

FileInfo 类主要提供创建、复制、删除、移动和打开文件的实例方法，并且帮助创建 FileStream 对象，其 CreationTime 属性用来获取或设置当前 FileInfo 对象的创建时间。语法格式如下：

```
public DateTime CreationTime { get; set; }
```

参数说明

属性值：当前 FileInfo 对象的创建日期和时间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetFileBothDay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来打开指定的文件，并获取其创建时间。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                    //创建打开文件对话框对象
    if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)                      //判断是否选中文件
    {
        FileInfo f = new FileInfo(P_OpenFileDialog.FileName);                  //创建 FileInfo 对象
```

```
            MessageBox.Show("文件创建日期：" +                                    //弹出消息对话框
                f.CreationTime.ToString(), "提示！");
        }
    }
```

秘笈心法

心法领悟 357：使用 FileInfo 类的相关属性获取文件最新修改时间。

获取文件的最新修改时间时，可以使用 FileInfo 类的 LastWriteTime 属性来实现。代码如下：

```
label2.Text = FInfo.LastWriteTime.ToString();                                    //显示文件最新修改时间
```

实例 358　获取文件最后一次修改时间

初级

光盘位置：光盘\MR\15\358　　趣味指数：★★★★☆

实例说明

本实例实现了获取文件最后一次修改时间的功能，运行本实例，单击“获取文件修改时间”按钮，弹出“打开”对话框，在该对话框中选择一个文件，单击“确定”按钮，即可弹出对话框显示选中文件的最后修改时间。实例运行效果如图 15.4 所示。

图 15.4　获取文件最后一次修改时间

关键技术

本实例实现时主要用到了 FileInfo 类的 LastWriteTime 属性，下面对其进行详细讲解。

FileInfo 类的 LastWriteTime 属性用来获取或设置上次写入当前文件的时间。语法格式如下：

```
public DateTime LastWriteTime { get; set; }
```

参数说明

属性值：上次写入当前文件的时间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ModifyTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来打开指定的文件，并获取其最后一次修改时间。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                      //创建打开文件对话框对象
    if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)                        //判断是否选中文件
    {
        FileInfo f = new FileInfo(P_OpenFileDialog.FileName);                    //创建 FileInfo 对象
        MessageBox.Show("文件最后一次修改时间：" +                                  //弹出消息对话框
            f.LastWriteTime.ToString(), "提示！");
    }
}
```

秘笈心法

心法领悟 358：使用 FileInfo 类的相关属性获取文件创建时间。

获取文件的创建时间时，可以使用 FileInfo 类的 CreationTime 属性来实现。代码如下：

```
label2.Text = FInfo.CreationTime.ToString();                    //显示文件创建时间
```

实例 359　获取文件名中禁止使用的字符

初级

光盘位置：光盘\MR\15\359

趣味指数：★★★☆

实例说明

在 Windows 操作系统中新建文件时，如果输入了非法字符，系统会提示错误，那么文件的名称究竟不允许使用哪些非法字符呢？本实例使用 C#获取到了文件名中禁止使用的所有字符。实例运行效果如图 15.5 所示。

图 15.5　获取文件名中禁止使用的字符

关键技术

本实例实现时主要用到了 Path 类的 GetInvalidFileNameChars 方法，下面对其进行详细介绍。

Path 类用来对包含文件或目录路径信息的 String 实例执行操作，其 GetInvalidFileNameChars 属性主要用来获取包含不允许在文件名中使用的字符的数组。语法格式如下：

```
public static char[] GetInvalidFileNameChars()
```

参数说明

返回值：包含不允许在文件名中使用的字符的数组。

说明：Path 类位于 System.IO 命名空间中。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NoChar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来获取路径名中禁止使用的字符；添加一个 RichTextBox 控件，用来显示路径名中所有禁止使用的字符。

（3）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    foreach (char c in Path.GetInvalidFileNameChars())          //得到不允许使用的字符数组
    {
        txt_Str.Text += c + "\r\n";                             //输出字符数组内容
    }
}
```

秘笈心法

心法领悟 359：获取路径名中禁止使用的字符。

获取路径名中禁止使用的字符时，可以使用 Path 类的 GetInvalidPathChars 方法来实现，该方法用来获取包含不允许在路径名中使用的字符的数组。

15.2 文件的创建和删除

实例 360	创建和删除文件 光盘位置：光盘\MR\15\360	初级 趣味指数：★★★★☆

实例说明

本实例实现了创建和删除文件的功能，运行本实例，单击“创建指定文件”按钮，在指定的位置创建一个文件；单击“删除指定文件”按钮，可以删除选择的文件。实例运行效果如图 15.6 所示。

图 15.6 创建和删除文件

关键技术

本实例实现时主要用到了 File 类的 Create 和 Delete 方法，下面分别对它们进行详细介绍。

（1）File 类的 Create 方法

File 类的 Create 方法用来在指定路径中创建文件，该方法为可重载方法，本实例中用到的该方法的重载形式如下：

```
public static FileStream Create(string path)
```

参数说明

❶ path：要创建的文件的路径及名称。

❷ 返回值：一个 FileStream，它提供对 path 中指定的文件的读/写访问。

（2）File 类的 Delete 方法

File 类的 Delete 方法用来删除指定的文件，如果指定的文件不存在，则不引发异常。语法格式如下：

```
public static void Delete(string path)
```

参数说明

path：要删除的文件的名称。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateAndRemoveFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行创建和删除文件操作。

（3）程序的主要代码如下：

```
private void btn_Create_Click(object sender, EventArgs e)
{
    SaveFileDialog P_SaveFileDialog = new SaveFileDialog();                    //创建保存文件对话框对象
    if (P_SaveFileDialog.ShowDialog() == DialogResult.OK)                      //判断是否确定保存文件
    {
        File.Create(P_SaveFileDialog.FileName);                                //创建文件
    }
}
private void btn_Remove_Click(object sender, EventArgs e)
{
    OpenFileDialog P_OpenFileDialog = new OpenFileDialog();                    //创建打开文件对话框对象
    if (P_OpenFileDialog.ShowDialog() == DialogResult.OK)                      //判断是否确定删除文件
    {
        File.Delete(P_OpenFileDialog.FileName);                                //删除文件
    }
}
```

秘笈心法

心法领悟 360：使用 SaveFileDialog 控件作为“另存为”对话框。

“另存为”对话框使用 SaveFileDialog 控件来表示，它表示一个通用对话框，用户可以使用此对话框来指定一个要将文件另存为的文件名。开发人员可以使用 SaveFileDialog 控件的 ShowDialog 方法显示“另存为”对话框。

实例 361　生成随机文件名或文件夹名

光盘位置：光盘\MR\15\361　　中级　　趣味指数：★★★★

实例说明

在实际开发中，如果要创建的文件名或文件夹名称不确定，可以随机生成一个文件名或文件夹名，本实例使用 C#实现了以上功能。实例运行效果如图 15.7 所示。创建完成的随机文件名及文件夹名如图 15.8 所示。

图 15.7　生成随机文件名或文件夹名

图 15.8　生成的随机文件名及文件夹名

关键技术

本实例实现时主要用到了 Guid 结构的 NewGuid 方法，下面对其进行详细讲解。

Guid 结构表示全局唯一标识符（GUID），其 NewGuid 方法用来初始化 Guid 结构的一个新实例。语法格式如下：

```
public static Guid NewGuid()
```

参数说明

返回值：新的 Guid 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RandomFileName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来以随机名称创建文件和文件夹。

（3）程序的主要代码如下：

```
private void btn_file_Click(object sender, EventArgs e)
{
    FolderBrowserDialog P_FolderBrowserDialog =new FolderBrowserDialog(); //创建文件夹对话框对象
    if (P_FolderBrowserDialog.ShowDialog()==DialogResult.OK)              //判断是否选择文件夹
    {
        File.Create(P_FolderBrowserDialog.SelectedPath + "\\" +           //根据 GUID 生成文件名
            Guid.NewGuid().ToString()+ ".txt");
    }
}
private void btn_Drictory_Click(object sender, EventArgs e)
{
    FolderBrowserDialog P_FolderBrowserDialog = new FolderBrowserDialog(); //创建文件夹对话框对象
    if (P_FolderBrowserDialog.ShowDialog() == DialogResult.OK)             //判断是否选择文件夹
    {
```

```
            Directory.CreateDirectory(P_FolderBrowserDialog.SelectedPath +          //根据 GUID 生成文件夹名
                "\\" + Guid.NewGuid().ToString());
        }
    }
```

秘笈心法

心法领悟 361：使用 FolderBrowserDialog 控件作为“浏览文件夹”对话框。

“浏览文件夹”对话框使用 FolderBrowserDialog 控件来表示，它主要用来提示用户选择文件夹。开发人员可以使用 FolderBrowserDialog 控件的 ShowDialog 方法显示“浏览文件夹”对话框。

实例 362	建立临时文件	中级
	光盘位置：光盘\MR\15\362	趣味指数：★★★★

实例说明

当用户在程序中新建立一个文件，但是尚未保存和赋予名称时，所建立的文件是被保存在内存中的，显然这样非常危险。如果由于突然死机或者其他原因造成了程序在未保存文件之前就关闭，那么损失将非常严重。如果在文件建立的同时就在磁盘中存储数据，将不会造成这样的损失，这时需要一个临时文件来保存数据。本实例将介绍如何建立临时文件来存储数据。实例运行效果如图 15.9 所示。

图 15.9　建立临时文件

关键技术

本实例实现时，首先使用 Path 类的 GetTempFileName 方法得到临时文件名称，然后使用 FileInfo 类的 AppendText 方法向临时文件中写入数据，下面对本实例中用到的关键技术进行详细讲解。

（1）Path 类的 GetTempFileName 方法

Path 类的 GetTempFileName 方法用来创建磁盘上唯一命名的零字节的临时文件并返回该文件的完整路径。语法格式如下：

```
public static string GetTempFileName()
```

参数说明

返回值：包含临时文件的完整路径的字符串。

（2）FileInfo 类的 AppendText 方法

FileInfo 类的 AppendText 方法用来创建一个 StreamWriter 对象，它向 FileInfo 的对象表示的文件追加文本。语法格式如下：

```
public StreamWriter AppendText()
```

参数说明

返回值：新的 StreamWriter 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TempFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入临时文件内容；添加一个 TextBox 控件，用来显示临时文件的保存路径及名称；添加两个 Button 控件，分别用来执行创建临时文件和关闭当前窗体操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = Path.GetTempFileName();                    //得到临时文件名称
```

```
    FileInfo fin = new FileInfo(textBox1.Text);                    //创建文件对象
    fin.AppendText();                                              //向文件内追加文本
}
```

秘笈心法

心法领悟 362：TextBox 控件的使用。

TextBox 控件，又称为文本框控件，它主要用于获取用户输入的数据或者显示文本，它通常用于可编辑文本，也可以使其成为只读控件。文本框可以显示多行，开发人员可以使文本换行以便符合控件的大小。

实例 363　根据日期动态建立文件

光盘位置：光盘\MR\15\363　　初级　　趣味指数：★★★★

实例说明

本实例主要实现以当前日期时间为根据创建文件的功能，运行本实例，单击“根据系统日期建立文件”按钮，以当前的日期时间为名称在指定位置创建一个文件。实例运行效果如图 15.10 所示。创建的文件名称如图 15.11 所示。

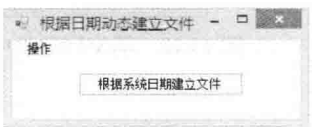


图 15.10　根据日期动态建立文件

图 15.11　创建的文件名称

关键技术

本实例实现时，首先需要使用 DateTime 结构的 Now 属性获取系统当前日期时间，并对获取到的日期时间进行格式化，然后使用 File 类的 Create 方法创建文件，下面对本实例中用到的关键技术进行详细讲解。

DateTime 结构表示时间上的一刻，通常以日期和当天的时间表示，其 Now 属性用来获取一个 DateTime 对象，该对象设置为此计算机上的当前日期和时间，表示为本地时间。本实例中使用 DateTime 结构的 Now 属性获取系统当前日期时间之后，调用 ToString 方法对其进行格式化。代码如下：

```
string strName = DateTime.Now.ToString("yyyyMMddhhmmss");          //对当前日期进行格式化
```

说明：（1）本实例中将当前日期时间格式化为“20100719114743”格式，其中 2010 表示年份，07 表示月份，19 表示天数，11 表示时钟，47 表示分钟，43 表示秒钟。

（2）关于 File 类的 Create 方法的详细讲解请参见实例 360 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来根据当前日期时间动态地创建文件。

（3）程序的主要代码如下：

```
private void btn_Create_Click(object sender, EventArgs e)
{
    FolderBrowserDialog P_FolderBrowserDialog = new FolderBrowserDialog();    //创建浏览文件夹对话框对象
    if (P_FolderBrowserDialog.ShowDialog() == DialogResult.OK)                //判断是否选择了文件夹
```

```
    {
        File.Create(P_FolderBrowserDialog.SelectedPath + "\\" + DateTime.Now.ToString("yyyyMMddhhmmss") + ".txt");      //创建文件
    }
}
```

秘笈心法

心法领悟 363：使用 DateTime.Now 属性获取系统时间。

使用 DateTime.Now 属性可以方便地获取系统时间，而且可以使用格式化方式方便地得到系统时间的字符串表示形式。代码如下：

```
string P_str = DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒");
```

实例 364　清空回收站中的所有文件

光盘位置：光盘\MR\15\364　　中级　　趣味指数：★★★★

实例说明

“回收站”顾名思义就是用来存储垃圾的。在 Windows 操作系统中，回收站是一个存放已删除文件的地方。其实回收站是一个系统文件夹，在 DOS 模式下进入磁盘根目录，输入“dir/a”则会看到一个名为 RECYCLED 的文件夹，这个就是回收站。为了防止误删除操作，Windows 操作系统将用户删除的文件先暂存到回收站中，并且删除的文件可以恢复，待确认删除后再将回收站清空即可。本实例将以编程的方式完成清空回收站的工作，为用户清理出一些磁盘空间。实例运行效果如图 15.12 所示。

图 15.12　清空回收站中的所有文件

关键技术

本实例实现时主要用到了系统 API 函数 SHEmptyRecycleBin，下面对其进行详细讲解。

SHEmptyRecycleBin 是一个内核 API 函数，该函数能够清空回收站中的文件，它在 C#中需要手动地引入方法所在的类库。语法格式如下：

```
[DllImportAttribute("shell32.dll")]                                    //声明 API 函数
private static extern    int SHEmptyRecycleBin(IntPtr handle, string root, int falgs);
```

参数说明

❶ handle：父窗口句柄。

❷ root：将要清空的回收站的地址，如果为 null 值时，将清空所有驱动器上的回收站。

❸ falgs：用于清空回收站的功能参数。

说明：程序中使用系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间，下面再遇到类似的情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ClearRecycle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行清空回收站操作。

（3）程序的主要代码如下：

```
const int SHERB_NOCONFIRMATION = 0x000001;                    //整型常量在 API 中表示删除时没有确认对话框
const int SHERB_NOPROGRESSUI = 0x000002;                      //在 API 中表示不显示删除进度条
const int SHERB_NOSOUND = 0x000004;                           //在 API 中表示删除完毕时不播放声音
[DllImportAttribute("shell32.dll")]                           //声明 API 函数
private static extern int SHEmptyRecycleBin(IntPtr handle, string root, int falgs);
private void button1_Click(object sender, EventArgs e)
```

```
{
    //清空回收站
    SHEmptyRecycleBin(this.Handle, "", SHERB_NOCONFIRMATION + SHERB_NOPROGRESSUI + SHERB_NOSOUND);
}
```

秘笈心法

心法领悟 364：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

15.3　查 找 文 件

实例 365	检查文件是否存在 光盘位置：光盘\MR\15\365	初级 趣味指数：★★★☆

实例说明

在使用文件时不只是要知道文件路径，更重要的是知道这个文件是否存在，如果不存在，程序调用文件时会发生错误。本实例可以检查文件是否存在。实例运行效果如图 15.13 所示。

图 15.13　检查文件是否存在

关键技术

本实例实现时主要用到了 File 类的 Exists 方法，下面对其进行详细讲解。

File 类的 Exists 方法用来确定指定的文件是否存在。语法格式如下：

```
public static bool Exists(string path)
```

参数说明

❶ path：要检查的文件。

❷ 返回值：如果调用方具有要求的权限并且 path 包含现有文件的名称，则为 true；否则为 false。如果 path 为 null、无效路径或零长度字符串，则此方法也将返回 false。如果调用方不具有读取指定文件所需的足够权限，则不引发异常并且该方法返回 false，这与 path 是否存在无关。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileIsExists。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的文件；添加两个 Button 控件，分别用来执行选择文件和判断文件是否存在操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (File.Exists(textBox1.Text))                                    //判断文件是否存在
        MessageBox.Show("该文件已经存在");
    else
        MessageBox.Show("该文件不存在");
}
```

秘笈心法

心法领悟 365：使用 MessageBox 显示消息对话框。

向用户显示信息和消息的对话框称为消息对话框，MessageBox 是一个预定义对话框，主要用于向用户显示与应用程序相关的信息及请求来自用户的信息，其 Show 方法用来显示消息对话框。

实例 366　获取文件夹下的所有子文件夹及文件的名称　初级

光盘位置：光盘\MR\15\366　趣味指数：★★★★

实例说明

本实例主要是根据指定的文件夹路径，获取该文件夹下的所有子文件夹和文件的名称，并将获取的名称按照先后顺序，以节点的方式添加到 TreeView 控件中。实例运行效果如图 15.14 所示。

图 15.14　获取文件夹下的所有子文件夹及文件的名称

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 GetFileSystemInfos 方法、FileInfo 类的 DirectoryName 和 Name 属性，下面分别对它们进行详细讲解。

（1）DirectoryInfo 类的 GetFileSystemInfos 方法

DirectoryInfo 类的 GetFileSystemInfos 方法用来检索表示当前目录的文件和子目录的强类型 FileSystemInfo 对象的数组，该方法有两种重载形式，本实例中用到的重载形式如下：

```
public FileSystemInfo[] GetFileSystemInfos()
```

参数说明

返回值：强类型 FileSystemInfo 项的数组。

（2）FileInfo 类的 DirectoryName 属性

FileInfo 类的 DirectoryName 属性用来获取表示目录的完整路径的字符串。语法格式如下：

```
public string DirectoryName { get; }
```

参数说明

属性值：表示目录的完整路径的字符串。

（3）FileInfo 类的 Name 属性

FileInfo 类的 Name 属性用来获取文件名。语法格式如下：

```
public override string Name { get; }
```

参数说明

返回值：文件名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RansackFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用于显示要遍历的文件夹路径；添加一个 FolderBrowserDialog 控件，用于弹出“浏览文件夹”对话框；添加一个 Button 控件，用于选择文件夹的路径；添加一个 TreeView 控件，用于显示指定文件夹下的所有子文件夹及文件名称。

（3）程序的主要代码如下。

“…”按钮的 Click 事件，用于获取要遍历的文件夹路径，并将该路径下的所有子文件夹及文件的名称通过线程显示在 TreeView 控件上。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)          //打开文件夹对话框
    {
        textBox1.Text = folderBrowserDialog1.SelectedPath;             //显示选择的文件夹路径
        tempstr = folderBrowserDialog1.SelectedPath;                   //存储选择的文件夹路径
```

```
        thdAddFile = new Thread(new ThreadStart(SetAddFile));          //创建一个线程
        thdAddFile.Start();                                            //执行当前线程
    }
}
```

自定义方法 SetAddFile 主要实现委托线程的应用。代码如下：

```
public delegate void AddFile();                                        //定义委托线程
public void SetAddFile()
{
    this.Invoke(new AddFile(RunAddFile));                              //对指定的线程进行托管
}
```

自定义方法 RunAddFile 主要是通过线程来实现对文件夹下所有子文件夹和文件的遍历。代码如下：

```
public void RunAddFile()
{
    TreeNode TNode = new TreeNode();                                   //创建一个线程
    Files_Copy(treeView1, tempstr, TNode, 0);
    Thread.Sleep(0);                                                   //挂起主线程
    thdAddFile.Abort();                                                //执行线程
}
```

自定义方法 Files_Copy 是一个递归方法，主要用于实现遍历指定文件夹下的所有子文件夹和文件，并按照其先后顺序将文件夹名称和文件的名称显示在 TreeView 控件上，参数 TV 表示 TreeView 控件，参数 Sdir 表示指定文件夹的路径，参数 TNode 表示 TreeView 控件的当前节点，参数 n 表示当前操作的是文件，还是文件夹，如果为 0，表示是文件夹，否则为文件。代码如下：

```
private void Files_Copy(TreeView TV, string Sdir, TreeNode TNode, int n)
{
    DirectoryInfo dir = new DirectoryInfo(Sdir);
    Try
    {
        if (!dir.Exists)                                               //判断所指的文件或文件夹是否存在
            return;
        DirectoryInfo dirD = dir as DirectoryInfo;                     //如果给定参数不是文件夹则退出
        if (dirD == null)                                              //判断文件夹是否为空
            return;
        else
        {
            if (n == 0)
            {
                TNode = TV.Nodes.Add(dirD.Name);                       //添加文件夹的名称
                TNode.Tag = 1;
            }
            else
            {
                TNode = TNode.Nodes.Add(dirD.Name);                    //在文件夹中添加各子文件夹的名称
                TNode.Tag = 1;
            }
        }
        FileSystemInfo[] files = dirD.GetFileSystemInfos();            //获取文件夹中所有文件和文件夹
        foreach (FileSystemInfo FSys in files)                         //对单个 FileSystemInfo 进行判断，如果是文件夹则进行递归操作
        {
            FileInfo file = FSys as FileInfo;
            if (file != null)                                          //如果是文件，进行文件的复制操作
            {
                FileInfo SFInfo = new FileInfo(file.DirectoryName + "\\" + file.Name); //获取文件所在的原始路径
                TNode.Nodes.Add(file.Name);                            //添加文件
                TNode.Tag = 1;
            }
            else
            {
                string pp = FSys.Name;                                 //获取当前搜索到的文件夹名称
                Files_Copy(TV, Sdir + "\\" + FSys.ToString(), TNode, 1);   //如果是文件夹，则进行递归调用
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
```

```
            return;
        }
    }
```

自定义方法 UpAndDown 用于返回指定路径的父级路径，参数 dir 表示指定的路径。代码如下：

```
public string UpAndDown_Dir(string dir)
{
    string Change_dir = "";                                    //定义字符串变量
    Change_dir = Directory.GetParent(dir).FullName;            //返回当前路径的父级路径
    return Change_dir;
}
```

秘笈心法

心法领悟 366：TreeView 控件的使用技巧。

TreeView 控件经常用来设计 Windows 窗体的左侧导航菜单。

实例 367　搜索文件

光盘位置：光盘\MR\15\367　　中级　　趣味指数：★★★★

实例说明

在许多杀毒软件中，如瑞星、KV3650 等，一般都是通过检查磁盘中的文件来确定计算机中是否存在病毒。用户在执行这些软件时，会发现这些软件以极快的速度遍历磁盘中的文件，这是如何实现的呢？本实例实现了搜索文件的功能。实例运行效果如图 15.15 所示。

图 15.15　搜索文件

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 GetFileSystemInfos 方法、FileInfo 类的 FullName、Length 和 CreationTime 属性，下面分别对它们进行详细讲解。

（1）FileInfo 类的 FullName 属性

FileInfo 类的 FullName 属性用来获取目录或文件的完整目录。语法格式如下：

```
public virtual string FullName { get; }
```

参数说明

属性值：包含完整目录的字符串。

（2）FileInfo 类的 Length 属性

FileInfo 类的 Length 属性用来获取当前文件的大小（字节）。语法格式如下：

```
public long Length { get; }
```

参数说明

属性值：当前文件的大小（字节）。

说明：① 关于 DirectoryInfo 类的 GetFileSystemInfos 方法的详细讲解请参见实例 366 中的关键技术。

② 关于 FileInfo 类的 CreationTime 属性的详细讲解请参见实例 357 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SearchFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来输入要搜索的文件和目录；添加一个 Button 控件，用来执行搜索文件操作；添加一个 ListView 控件，用来显示搜索到的文件。

（3）程序的主要代码如下：

```
public void SearchFile(string fileDirectory)
{
    DirectoryInfo dir = new DirectoryInfo(fileDirectory);                    //创建 DirectoryInfo 对象
    FileSystemInfo[] f = dir.GetFileSystemInfos();                           //获取文件夹下的文件
    foreach (FileSystemInfo i in f)                                          //对指定的文件夹进行遍历
    {
        if (i is DirectoryInfo)                                              //如果是文件夹
        {
            SearchFile(i.FullName);                                          //递归调用
        }
        else
        {
            if (i.Name == textBox1.Text)                                     //如果等于指定的文件名
            {
                FileInfo fin = new FileInfo(i.FullName);                     //创建 FileInfo 对象
                listView1.Items.Add(fin.Name);                               //为 ListView 添加文件的名称
                //为 ListView 添加文件的路径
                listView1.Items[listView1.Items.Count - 1].SubItems.Add(fin.FullName);
                //为 ListView 添加文件的大小
                listView1.Items[listView1.Items.Count - 1].SubItems.Add(fin.Length.ToString());
                //为 ListView 添加文件的创建时间
                listView1.Items[listView1.Items.Count - 1].SubItems.Add(fin.CreationTime.ToString());
            }
        }
    }
}
```

秘笈心法

心法领悟 367：有效利用 ListView 控件。

ListView 是一种列表控件，在实现诸如显示文件详细信息这样的功能时，推荐使用该控件；另外，由于 ListView 有多种显示样式，因此在实现类似 Windows 系统的“缩略图”“平铺”“图标”“列表”和“详细信息”等功能时，经常需要使用 ListView 控件。

15.4　修 改 文 件

实例 368　修改文件属性　　初级

光盘位置：光盘\MR\15\368　　趣味指数：★★★★☆

实例说明

Windows 操作系统中，每个文件都有一些由系统定义的属性，有只读、隐藏、存档等。在系统中，用户可以更改这些属性，但是有些文件的属性不能够随意更改，如 C:\Command.com 文件，该文件的只读属性就不允许更改，因为它是一个系统文件，具有系统属性。本实例将设计一个专门修改文件属性的应用软件，可以随意修改任何文件的属性。实例运行效果如图 15.16 所示。

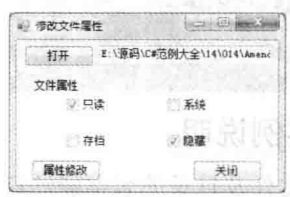


图 15.16　修改文件属性

关键技术

本实例实现时主要用到了 FileInfo 类的 Attributes 属性，下面对其进行详细讲解。

FileInfo 类的 Attributes 属性用来获取和设置文件的属性。语法格式如下：

```
public FileAttributes Attributes { get; set; }
```

参数说明

属性值：FileAttributes 枚举值之一。FileAttributes 枚举值及说明如表 15.2 所示。

表 15.2 FileAttributes 枚举值及说明

枚 举 值	说 明	枚 举 值	说 明
ReadOnly	只读属性	System	系统文件属性
Hidden	隐藏属性	Archive	存档属性

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AmendFileAttribute。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开”对话框；添加一个 TextBox 控件，用来显示打开的文件完全路径；添加 4 个 CheckBox 控件，分别用来让用户选择文件的属性；添加 3 个 Button 控件，分别用来执行打开文件、修改属性和关闭窗体操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    System.IO.FileInfo f = new System.IO.FileInfo(textBox1.Text);          //创建 FileInfo 对象
    if (checkBox1.Checked == true)                                          //如果只读复选框选中
    {
        f.Attributes = System.IO.FileAttributes.ReadOnly;                   //设置文件为只读
    }
    if (checkBox2.Checked == true)                                          //如果系统复选框选中
    {
        f.Attributes = System.IO.FileAttributes.System;                     //设置文件为系统
    }
    if (checkBox3.Checked == true)                                          //如果存档复选框选中
    {
        f.Attributes = System.IO.FileAttributes.Archive;                    //设置文件为存档
    }
    if (checkBox4.Checked == true)                                          //如果隐藏复选框选中
    {
        f.Attributes = System.IO.FileAttributes.Hidden;                     //设置文件为隐藏
    }
}
```

秘笈心法

心法领悟 368：使用 CheckBox 控件作为复选框。

CheckBox 控件，又称为复选框控件，它主要用来表示是否选取了某个选项条件，它常用于为用户提供具有是/否或真/假值的选项。

实例 369 修改文件及目录的名字

光盘位置：光盘\MR\15\369　　初级　　趣味指数：★★★☆

实例说明

文件及目录的名字能够标识文件系统中的一个对象，一个文件的全名通常表示为目录名加上文件名，如“C:\Windows\System32\Cmd.exe”。在 Windows 操作系统中，文件和目录名是可以修改的，但是名称中不能包含“\、 /、 :、 *、 ?、 "、 <、 >、 |、 ”这些字符。通常，修改目录及文件名是在系统中进行的，本实例将使用 C#实现在程序中修改文件及目录名字的功能。实例运行效果如图 15.17 所示。

图 15.17 修改文件及目录的名字

关键技术

本实例实现时主要用到了 File 类的 Move 方法和 Directory 类的 Move 方法，下面分别对它们进行详细讲解。

（1）File 类的 Move 方法

File 类的 Move 方法用来将指定文件移到新位置，并提供指定新文件名的选项。语法格式如下：

```
public static void Move(string sourceFileName,string destFileName)
```

参数说明

❶ sourceFileName：要移动的文件的名称。

❷ destFileName：文件的新路径。

（2）Directory 类的 Move 方法

Directory 类的 Move 方法用来将文件或目录及其内容移到新位置。语法格式如下：

```
public static void Move(string sourceDirName,string destDirName)
```

参数说明

❶ sourceDirName：要移动的文件或目录的路径。

❷ destDirName：指向 sourceDirName 的新位置的路径。如果 sourceDirName 是一个文件，则 destDirName 也必须是一个文件名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UpdateFileAndDirectory。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来打开文件夹；添加两个 TextBox 控件，分别用来显示文件夹的路径和输入修改文件夹及文件的名称；添加 3 个 Button 控件，分别用来执行打开文件夹、修改目录名及修改文件名操作；添加一个 ListBox 控件，用来显示文件夹中的文件名称和文件夹名称。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text))
        Directory.Move(textBox1.Text, textBox2.Text);                                    //移动目录
}
private void button3_Click(object sender, EventArgs e)
{
    File.Move(textBox1.Text + "\\" + listBox1.SelectedItem.ToString(), textBox1.Text + "\\" + textBox2.Text);    //移动文件
}
```

秘笈心法

心法领悟 369：如何判断文本框内容是否为空？

判断文本框内容是否为空主要有以下 3 种方法：

```
if (textBox1.Text=="")                                  //第一种方法
if (textBox1.Text==String.Empty)                        //第二种方法
if (String.IsNullOrEmpty(textBox1.Text))                //第三种方法
```

实例 370　将长文件名转换成短文件名

中级

光盘位置：光盘\MR\15\370

趣味指数：★★★★★

实例说明

长文件名和短文件名的转换在 Windows 操作系统中经常遇到。例如，有一个比较长的路径名“D:\Program Files\Microsoft SQL Server\MSSQL\README.TXT”，那么在 Windows 操作系统中显示时，系统可能会自动将其转换成短文件名，如“D:\PROGRA~1\MICROS~1\MSSQL\README.TXT”，那么，在 C#中能不能实现类似

的功能呢？答案是肯定的。本实例就使用 C#实现了将长文件名转换为短文件名的功能。实例运行效果如图 15.18 所示。

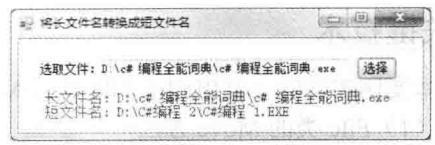


图 15.18　将长文件名转换成短文件名

关键技术

本实例实现时主要用到了系统 API 函数 GetShortPathName，下面对其进行详细讲解。

GetShortPathName 是一个内核 API 函数，该函数能够将长文件名转换为短文件名，它在 C#中需要手动引入方法所在的类库。语法格式如下：

```
[DllImport("Kernel32.dll")]                                                    //声明 API 函数
private static extern Int16 GetShortPathName(string lpszLongPath, StringBuilder lpszShortPath, Int16 cchBuffer);
```

参数说明

❶ lpszLongPath：指定欲获取短路径名的那个文件的名字。可以是个完整路径，或者由当前目录决定。

❷ lpszShortPath：指定一个缓冲区，用于装载文件的短路径和文件名。

❸ cchBuffer：lpszShortPath 缓冲区长度。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetShortPathName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来选择文件；添加一个 TextBox 控件，用来显示选择的文件路径；添加一个 Label 文件，用来显示原来的文件名及转换后的短文件名。

（3）程序的主要代码如下：

```
[DllImport("Kernel32.dll")]                                                    //声明 API 函数
private static extern Int16 GetShortPathName(string lpszLongPath, StringBuilder lpszShortPath, Int16 cchBuffer);
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog OFDialog = new OpenFileDialog();                            //创建 OpenFileDialog 对象
    if (OFDialog.ShowDialog() == DialogResult.OK)                              //判读是否选择了文件
    {
        textBox1.Text = OFDialog.FileName;                                     //显示选择的文件名
        string longName = textBox1.Text;                                       //记录选择的文件名
        StringBuilder shortName = new System.Text.StringBuilder(256);          //创建 StringBuilder 对象
        GetShortPathName(longName, shortName, 256);                            //调用 API 函数转换成短文件名
        string myInfo = "长文件名：" + longName;                               //显示长文件名
        myInfo += "\n 短文件名：" + shortName;                                 //显示短文件名
        label2.Text = myInfo;
    }
}
```

秘笈心法

心法领悟 370：StringBuilder 类和 string 类的使用场合。

当程序中需要大量地对某个字符串进行操作时，应该考虑应用 StringBuilder 类处理该字符串，其设计目的就是针对大量 string 操作的一种改进办法，避免产生太多的临时对象。当程序中只是对某个字符串进行一次或几次操作时，采用 string 类即可。

实例 371	文件批量更名	高级
	光盘位置：光盘\MR\15\371	趣味指数：★★★★★

实例说明

相信读者对更改文件名并不感觉陌生，修改的方法也非常简单。但是，如果有多个文件需要更名，那将是

一件非常繁琐的工作。所以，开发出本实例解决了这个难题，通过本实例可以批量对文件名进行修改。程序中提供了更名的模板，选择后可以快速更名，也可以自定义更名模板，对于文件名还可以进行简体和繁体的相互转换。实例运行效果如图 15.19 所示。

图 15.19　文件批量更名

关键技术

本实例在实现文件批量更名功能时，主要用到了 File 类的 Copy 和 Delete 方法，具体实现时，首先通过 Copy 方法将现有文件复制到新文件，然后再调用 Delete 方法删除原文件，这样就可以实现对文件名称的修改，下面对本实例中用到的关键技术进行详细讲解。

（1）Copy 方法

该方法将现有文件复制到新文件。语法格式如下：

```
public static void Copy(string sourceFileName,string destFileName)
```

参数说明

❶ sourceFileName：要复制的文件。

❷ destFileName：目标文件的名称，它不能是一个目录或现有文件。

（2）Delete 方法

该方法删除指定的文件，如果指定的文件不存在，则不引发异常。语法格式如下：

```
public static void Delete(string path)
```

参数说明

path：要删除的文件的名称。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileBatchChangeName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 StatusStrip 控件，用来作为窗体的状态栏；添加一个 OpenFileDialog 控件，用来选择要批量更名的文件；添加一个 SaveFileDialog 控件，用来导出文件列表；添加 5 个 RadioButton 控件，分别用来选择“文件名大写”“文件名小写”“第一个字母大写”“扩展名大写”和“扩展名小写”5 种更名方式；添加一个 ComboBox 控件，用来选择预设模板；添加一个 TextBox 控件，用来设置模板；添加两个 NumericUpDown 控件，分别用来选择起始数字和增量值；添加一个 ListView 控件，用来显示要批量更名的文件的详细信息。

（3）程序的主要代码如下：

```
private void ChangeName()
{
    int flag = 0;                                          //记录出现错误的文件数量
    try
    {
```

```
            toolStripProgressBar1.Minimum = 0;                                    //设置进度条的起始值为 0
            toolStripProgressBar1.Maximum = listView1.Items.Count - 1;            //设置进度条的最大值为文件数量
            for (int i = 0; i < listView1.Items.Count; i++)                       //开始读取每一项
            {
                string path = listView1.Items[i].SubItems[4].Text;                //获取文件所在目录
                string sourcePath = path + listView1.Items[i].SubItems[0].Text;   //获取文件的完整路径
                string newPath = path + listView1.Items[i].SubItems[1].Text;      //设置更名后的文件的完整路径
                File.Copy(sourcePath, newPath);                                   //将更名后的文件复制到原目录下
                File.Delete(sourcePath);                                          //删除原目录下的原始文件
                toolStripProgressBar1.Value = i;                                  //设置进度条进度
                //设置更新后的文件名，并显示在 ListView 控件中
                listView1.Items[i].SubItems[0].Text = listView1.Items[i].SubItems[1].Text;
                listView1.Items[i].SubItems[6].Text = " √ 成功";                  //设置更改成功后的提示信息
            }
        }
        catch(Exception ex)
        {
            flag++;                                                               //如果发生异常，则使 flag 加 1
            MessageBox.Show(ex.Message);
        }
        finally
        {
            tsslError.Text = flag.ToString() + " 个错误";                         //显示出现错误的数量
        }
    }
```

秘笈心法

心法领悟 371：使用 Exception 类捕捉异常。

所有.NET 中的异常都可被称为 Exception。虽然说异常多种多样，但基本上它们都是 Exception。这个说法可以打一个简单比喻，就像是无论黑猫、白猫、花猫，它们都是“猫”一样。

15.5 复 制 文 件

实例 372	使用 FileStream 复制大文件	中级
	光盘位置：光盘\MR\15\372	趣味指数：★★★★☆

实例说明

使用 FileStream 复制大文件，实际上就是将文件以流的方式进行复制。运行本实例，选择原文件路径和目的文件路径，单击“复制”按钮，即可使用 FileStream 流实现复制文件的功能。实例运行效果如图 15.20 所示。

图 15.20 使用 FileStream 复制大文件

关键技术

本实例使用 FileStream 类将文件以指定的文件流大小进行分割，然后在指定的位置创建一个同名的空文件，将分割后的文件流追加到空文件中，以实现大文件的复制，下面对 FileStream 类进行详细介绍。

FileStream 类公开了以文件为主的 Stream，它表示在磁盘或网络路径上指向文件的流。一个 FileStream 类的实例实际上代表一个磁盘文件，它通过 Seek 方法进行对文件的随机访问，也同时包含了流的标准输入、标准输出、标准错误等。FileStream 默认对文件的打开方式是同步的，但它同样很好地支持异步操作。

FileStream 类的常用属性及说明如表 15.3 所示。

表 15.3　FileStream 类的常用属性及说明

属　性	说　明
CanRead	获取一个值，该值指示当前流是否支持读取
CanSeek	获取一个值，该值指示当前流是否支持查找
CanTimeout	获取一个值，该值确定当前流是否可以超时
CanWrite	获取一个值，该值指示当前流是否支持写入
IsAsync	获取一个值，该值指示 FileStream 是异步还是同步打开的
Length	获取用字节表示的流长度
Name	获取传递给构造函数的 FileStream 的名称
Position	获取或设置此流的当前位置
ReadTimeout	获取或设置一个值，该值确定流在超时前尝试读取多长时间
WriteTimeout	获取或设置一个值，该值确定流在超时前尝试写入多长时间

FileStream 类的常用方法及说明如表 15.4 所示。

表 15.4　FileStream 类的常用方法及说明

方　法	说　明
BeginRead	开始异步读操作
BeginWrite	开始异步写操作
Close	关闭当前流并释放与之关联的所有资源
EndRead	等待挂起的异步读取完成
EndWrite	结束异步写入，在 I/O 操作完成之前一直阻止
Lock	允许读取访问的同时防止其他进程更改 FileStream
Read	从流中读取字节块并将该数据写入给定的缓冲区中
ReadByte	从文件中读取一个字节，并将读取位置提升一个字节
Seek	将该流的当前位置设置为给定值
SetLength	将该流的长度设置为给定值
Unlock	允许其他进程访问以前锁定的某个文件的全部或部分
Write	使用从缓冲区读取的数据将字节块写入该流
WriteByte	将一个字节写入文件流的当前位置

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileCopy。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来选择源文件；添加一个 FolderBrowserDialog 控件，用来选择目的文件的路径；添加两个 TextBox 控件，分别用来显示源文件和目的文件的路径；添加 3 个 Button 控件，分别用来选择源文件和目的文件的路径，以及实现文件的复制功能。

（3）程序的主要代码如下：

```
public void CopyFile(string FormerFile, string toFile, int SectSize)
{
    FileStream fileToCreate = new FileStream(toFile, FileMode.Create);          //创建目的文件，如果已存在将被覆盖
    fileToCreate.Close();                                                       //关闭所有资源
    fileToCreate.Dispose();                                                     //释放所有资源
    FormerOpen = new FileStream(FormerFile, FileMode.Open, FileAccess.Read);    //以只读方式打开源文件
    ToFileOpen = new FileStream(toFile, FileMode.Append, FileAccess.Write);     //以写方式打开目的文件
    //根据一次传输的大小，计算传输的个数
    int FileSize;                                                               //要复制的文件的大小
    //如果分段复制，即每次复制内容小于文件总长度
```

```
    if (SectSize < FormerOpen.Length)
    {
        byte[] buffer = new byte[SectSize];                    //根据传输的大小，定义一个字节数组
        int copied = 0;                                        //记录传输的大小
        while (copied <= ((int)FormerOpen.Length - SectSize))  //复制主体部分
        {
            FileSize = FormerOpen.Read(buffer, 0, SectSize);   //从 0 开始读，每次最大读 SectSize
            FormerOpen.Flush();                                //清空缓存
            ToFileOpen.Write(buffer, 0, SectSize);             //向目的文件写入字节
            ToFileOpen.Flush();                                //清空缓存
            ToFileOpen.Position = FormerOpen.Position;         //使源文件和目的文件流的位置相同
            copied += FileSize;                                //记录已复制的大小
        }
        int left = (int)FormerOpen.Length - copied;            //获取剩余大小
        FileSize = FormerOpen.Read(buffer, 0, left);           //读取剩余的字节
        FormerOpen.Flush();                                    //清空缓存
        ToFileOpen.Write(buffer, 0, left);                     //写入剩余的部分
        ToFileOpen.Flush();                                    //清空缓存
    }
    //如果整体复制，即每次复制内容大于文件总长度
    else
    {
        byte[] buffer = new byte[FormerOpen.Length];           //获取文件的大小
        FormerOpen.Read(buffer, 0, (int)FormerOpen.Length);    //读取源文件的字节
        FormerOpen.Flush();                                    //清空缓存
        ToFileOpen.Write(buffer, 0, (int)FormerOpen.Length);   //释放字节
        ToFileOpen.Flush();                                    //清空缓存
    }
    FormerOpen.Close();                                        //释放所有资源
    ToFileOpen.Close();                                        //释放所有资源
    MessageBox.Show("文件复制完成");
}
```

秘笈心法

心法领悟 372：do while 循环与 while 循环有什么不同吗？

do while 与 while 语句的运行方式基本相同，只是 do while 语句要先执行一次循环体的内容，然后再判断布尔条件，如果满足条件则执行循环体，不满足布尔条件时退出循环体。

实例 373　复制文件时显示复制进度

光盘位置：光盘\MR\15\373　　中级　　趣味指数：★★★★★

实例说明

复制文件时显示复制进度实际上就是用文件流来复制文件，并在每一块文件复制后，用进度条来显示文件的复制情况。本实例实现了复制文件时显示复制进度的功能。实例运行效果如图 15.21 所示。

图 15.21　复制文件时显示复制进度

关键技术

本实例主要是以线程和委托的方式，在使用 FileStream 类对文件进行复制的同时，使用 ProgressBar 控件来显示文件复制进度。下面对本实例中用到的关键技术进行详细讲解。

（1）线程的构造函数

该构造函数主要用于初始化 Thread 类的新实例。语法格式如下：

```
public Thread (ThreadStart start)
```

参数说明

start：ThreadStart 委托，它表示此线程开始执行时要调用的方法。

（2）委托

在拥有此控件的基础窗口句柄的线程上执行指定的委托。语法格式如下：

```
public Object Invoke (Delegate method)
```

参数说明

❶ method：包含要在控件的线程上下文中调用的方法的委托。

❷ 返回值：正在被调用的委托的返回值，或者如果委托没有返回值，则为空引用。

📖 **说明**：关于 FileStream 类的详细讲解请参见实例 372 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileCopyPlan。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来选择源文件；添加一个 FolderBrowserDialog 控件，用来选择目的文件的路径；添加两个 TextBox 控件，分别用来显示源文件和目的文件的路径；添加 3 个 Button 控件，分别用来选择源文件和目的文件的路径，以及实现文件的复制功能；添加一个 ProgressBar 控件，用来显示复制进度条。

（3）程序的主要代码如下：

```
public void CopyFile(string FormerFile, string toFile, int SectSize, ProgressBar progressBar1)
{
    progressBar1.Value = 0;                                          //设置进度栏的当前位置为 0
    progressBar1.Minimum = 0;                                        //设置进度栏的最小值为 0
    FileStream fileToCreate = new FileStream(toFile, FileMode.Create);   //创建目的文件，如果已存在将被覆盖
    fileToCreate.Close();                                            //关闭所有资源
    fileToCreate.Dispose();                                          //释放所有资源
    FormerOpen = new FileStream(FormerFile, FileMode.Open, FileAccess.Read);   //以只读方式打开源文件
     ToFileOpen = new FileStream(toFile, FileMode.Append, FileAccess.Write);   //以写方式打开目的文件
     //根据一次传输的大小，计算传输的个数
    int max = Convert.ToInt32(Math.Ceiling((double)FormerOpen.Length / (double)SectSize));
    progressBar1.Maximum = max;                                      //设置进度栏的最大值
    int FileSize;                                                    //要复制的文件的大小
    if (SectSize < FormerOpen.Length)                                //如果分段复制，即每次复制内容小于文件总长度
    {
        byte[] buffer = new byte[SectSize];                          //根据传输的大小，定义一个字节数组
        int copied = 0;                                              //记录传输的大小
        int tem_n = 1;                                               //设置进度栏中进度块的增加个数
        while (copied <= ((int)FormerOpen.Length - SectSize))        //复制主体部分
        {
            FileSize = FormerOpen.Read(buffer, 0, SectSize);         //从 0 开始读，每次最大读 SectSize
            FormerOpen.Flush();                                      //清空缓存
            ToFileOpen.Write(buffer, 0, SectSize);                   //向目的文件写入字节
            ToFileOpen.Flush();                                      //清空缓存
            ToFileOpen.Position = FormerOpen.Position;               //使源文件和目的文件流的位置相同
            copied += FileSize;                                      //记录已复制的大小
            progressBar1.Value = progressBar1.Value + tem_n;         //增加进度栏的进度块
        }
        int left = (int)FormerOpen.Length - copied;                  //获取剩余大小
        FileSize = FormerOpen.Read(buffer, 0, left);                 //读取剩余的字节
        FormerOpen.Flush();                                          //清空缓存
        ToFileOpen.Write(buffer, 0, left);                           //写入剩余的部分
        ToFileOpen.Flush();                                          //清空缓存
    }
    else                                                             //如果整体复制，即每次复制内容大于文件总长度
    {
        byte[] buffer = new byte[FormerOpen.Length];                 //获取文件的大小
        FormerOpen.Read(buffer, 0, (int)FormerOpen.Length);          //读取源文件的字节
        FormerOpen.Flush();                                          //清空缓存
        ToFileOpen.Write(buffer, 0, (int)FormerOpen.Length);         //释放字节
```

```
            ToFileOpen.Flush();                                    //清空缓存
        }
        FormerOpen.Close();                                        //释放所有资源
        ToFileOpen.Close();                                        //释放所有资源
        if (MessageBox.Show("复制完成") == DialogResult.OK)          //显示“复制完成”提示对话框
        {
            progressBar1.Value = 0;                                //设置进度栏的当前位置为 0
            textBox1.Clear();                                      //清空文本
            textBox2.Clear();
            str = "";
        }
    }
```

秘笈心法

心法领悟 373：使用线程代替 Timer 组件。

Timer 组件可以按用户定义的时间间隔来引发 Tick 事件，该事件一般为周期性的，每隔若干秒或若干毫秒执行一次，Timer 组件工作在窗体线程中，如果 Timer 组件中执行了较为耗时的操作，会增加窗体线程的负担，导致窗体中其他操作不能及时得到 CPU 资源，出现窗体长时间或短时间无响应的情况。在适当的情况下可以使用线程来代替 Timer 组件，这样会减少窗体线程的负担。

实例 374　批量复制文件

光盘位置：光盘\MR\15\374　　中级　　趣味指数：★★★★☆

实例说明

复制文件是经常用到的技术，在备份文件时，要将文件复制到其他地方。本实例在系统中可以同时操作多个文件，将用户所选的多个文件复制到指定的目录中，对于大批量的文件复制操作，使用该软件将给用户带来方便。实例运行效果如图 15.22 所示。

图 15.22　批量复制文件

关键技术

本实例主要使用 File 类的 Copy 方法实现批量复制文件功能，下面对其进行详细介绍。

File 类的 Copy 方法用来将现有文件复制到新文件，该方法有两种重载形式，本实例中用到的重载形式如下：

```
public static void Copy(string sourceFileName,string destFileName)
```

参数说明

❶ sourceFileName：要复制的文件。

❷ destFileName：目标文件的名称，它不能是一个目录或现有文件。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CopyLotFiles。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 FolderBrowserDialog 控件，用来选择文件夹；添加一个 ListBox 控件，用来显示目录下的文件信息；添加两个 TextBox 控件，分别用来显示和输入目录；添加两个 Button 控件，分别用来执行打开目录和复制文件操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    foreach (object o in listBox1.SelectedItems)                   //遍历所有文件
    {
```

```
            File.Copy(textBox1.Text + "\\" + o.ToString(), textBox2.Text + "\\" + o.ToString());          //复制文件
        }
    }
```

秘笈心法

心法领悟 374：foreach 语句的使用。

foreach 语句用于循环列举一个集合的元素，并对该集合中的每个元素执行一次相关的语句。使用 foreach 语句时，变量的类型一定要与集合类型相同，例如，如果想遍历一个字符串数组中的每一项，那么此处变量的类型就应该是 string 类型，依此类推。

15.6　指定类型的文件操作

实例 375	使用 C#操作 INI 文件	中级
	光盘位置：光盘\MR\15\375	趣味指数：★★★★☆

实例说明

计算机中的配置文件有很多，大家对它们都不熟悉，了解程度也仅限于格式。细心的读者会发现，相当多的一部分配置文件都是 INI 格式。本实例将详细讲解 C#中有关 INI 文件的操作。实例运行效果如图 15.23 所示。

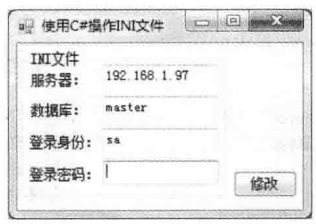


图 15.23　使用 C#操作 INI 文件

关键技术

本实例中对 INI 文件操作时，主要用到了系统 API 函数 GetPrivateProfileString 和 WritePrivateProfileString，下面分别对它们进行详细讲解。

（1）GetPrivateProfileString 函数

该函数主要用来读取 INI 文件的内容。语法格式如下：

```
[DllImport("kernel32")]
private static extern int GetPrivateProfileString(string lpAppName,string lpKeyName,string lpDefault,StringBuilder lpReturnedString,int nSize,string lpFileName);
```

GetPrivateProfileString 函数中的参数说明如表 15.5 所示。

表 15.5　GetPrivateProfileString 函数中的参数说明

参　数	描　述
lpAppName	表示 INI 文件内部根节点的值
lpKeyName	表示根节点下子标记的值
lpDefault	表示当标记值未设定或不存在时的默认值
lpReturnedString	表示返回读取节点的值
nSize	表示读取的节点内容的最大容量
lpFileName	表示文件的全路径

（2）WritePrivateProfileString 函数

该函数主要用于向 INI 文件写入数据。语法格式如下：

```
[DllImport("kernel32")]
private static extern long WritePrivateProfileString(string mpAppName,string mpKeyName,string mpDefault,string mpFileName);
```

WritePrivateProfileString 函数中的参数说明如表 15.6 所示。

表 15.6　WritePrivateProfileString 函数中的参数说明

参　数	描　述
mpAppName	表示 INI 文件内部根节点的值
mpKeyName	表示将要修改的标记名称
mpDefault	表示想要修改的内容
mpFileName	表示 INI 文件的全路径

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 INIFileOperate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 TextBox 控件，分别用来显示 INI 文件中对应的 4 个节点的内容；添加一个 Button 控件，用来对 INI 文件执行修改操作。

（3）程序的主要代码如下。

Frm_Main 窗体加载时，在文本框中显示 INI 文件。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    str = Application.StartupPath + "\\ConnectString.ini";                    //INI 文件的物理地址
    strOne = System.IO.Path.GetFileNameWithoutExtension(str);                 //获取 INI 文件的文件名
    if (File.Exists(str))                                                     //判断是否存在该 INI 文件
    {
        server.Text = ContentReader(strOne, "Data Source", "");               //读取 INI 文件中服务器节点的内容
        database.Text = ContentReader(strOne, "DataBase", "");                //读取 INI 文件中数据库节点的内容
        uid.Text = ContentReader(strOne, "Uid", "");                          //读取 INI 文件中用户节点的内容
        pwd.Text = ContentReader(strOne, "Pwd", "");                          //读取 INI 文件中密码节点的内容
    }
}
```

需要修改节点内容时，直接在文本框中进行修改，修改完成后，单击“修改”按钮，即可将编辑的内容写入到 INI 文件中。“修改”按钮的 Click 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (File.Exists(str))                                                     //判断是否存在 INI 文件
    {
        WritePrivateProfileString(strOne, "Data Source", server.Text, str);   //修改 INI 文件中服务器节点的内容
        WritePrivateProfileString(strOne, "DataBase", database.Text, str);    //修改 INI 文件中数据库节点的内容
        WritePrivateProfileString(strOne, "Uid", uid.Text, str);              //修改 INI 文件中用户节点的内容
        WritePrivateProfileString(strOne, "Pwd", pwd.Text, str);              //修改 INI 文件中密码节点的内容
        MessageBox.Show("恭喜你，修改成功！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("对不起，你所要修改的文件不存在，请确认后再进行修改操作！", "提示信息", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 375：如何获取程序可执行文件的路径？

获取程序可执行文件的路径时，可以使用 Application 类的 StartupPath 属性来实现，该属性用来获取启动了应用程序的可执行文件的路径，不包括可执行文件的名称。

实例 376　使用 C#操作 XML 文件

光盘位置：光盘\MR\15\376　　中级　　趣味指数：★★★★☆

实例说明

本实例使用 C#中的 LINQ to XML 技术实现了创建、添加、修改和删除 XML 文件操作。运行本实例，如果 XML 文件不存在，则“创建 XML 文件”区域的全部控件可用，这时输入相应的 XML 元素值，单击“创建”按钮，即可在项目文件夹下创建一个新的 XML 文件，同时将 XML 文件中的内容显示在窗体下方的数据表格中；在“操作 XML 文件”区域输入值，单击“添加”按钮，可以向 XML 文件中添加元素；如果用户在窗体下方的数据表格中选定一条记录，则单击“修改”和“删除”按钮，即可修改和删除 XML 文件中的元素。实例运行效果如图 15.24 所示。

图 15.24　使用 C#操作 XML 文件

关键技术

本实例对 XML 文件进行操作时，主要用到 LINQ to XML 技术中的 XElement 类的 Load、SetAttributeValue、Add、ReplaceNodes 和 Save 方法以及 XDocument 类的 Save 方法、XDeclaration 类、IEnumerable<T>泛型接口的 First 和 Remove 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）XElement 类的 Load 方法

XElement 类表示一个 XML 元素，其 Load 方法用来从文件加载 XElement。语法格式如下：

```
public static XElement Load(string uri)
```

参数说明

❶ uri：一个 URI 字符串，用来引用要加载到新 XElement 中的文件。

❷ 返回值：一个包含所指定文件的内容的 XElement。

（2）XElement 类的 SetAttributeValue 方法

SetAttributeValue 方法用来设置属性的值、添加属性或移除属性。语法格式如下：

```
public void SetAttributeValue(XName name,Object value)
```

参数说明

❶ name：一个 XName，其中包含要更改的属性的名称。

❷ value：分配给属性的值。如果该值为 null，则移除该属性；否则会将值转换为其字符串表示形式，并分配给该属性的 Value 属性。

（3）XElement 类的 Add 方法

Add 方法用来将指定的内容添加为此 XContainer 的子级。语法格式如下：

```
public void Add(Object content)
```

参数说明

content：要添加的包含简单内容的对象或内容对象集合。

（4）XElement 类的 ReplaceNodes 方法

ReplaceNodes 方法用来使用指定的内容替换此文档或元素的子节点。语法格式如下：

```
public void ReplaceNodes(Object content)
```

参数说明

content：一个用于替换子节点的包含简单内容的对象或内容对象集合。

（5）XElement 类的 Save 方法

Save 方法用来序列化此元素的基础 XML 树，可以将输出保存到文件、XmlTextWriter、TextWriter 或 XmlWriter。语法格式如下：

```
public void Save(string fileName)
```

参数说明

fileName：一个包含文件名称的字符串。

（6）XDocument 类的 Save 方法

XDocument 类表示 XML 文档，其 Save 方法用来将此 XDocument 序列化为文件、TextWriter 或 XmlWriter。语法格式如下：

```
public void Save(string fileName)
```

参数说明

fileName：一个包含文件名称的字符串。

（7）XDeclaration 类

该类表示一个 XML 声明。语法格式如下：

```
public XDeclaration(string version,string encoding,string standalone)
```

参数说明

❶ version：XML 的版本，通常为“1.0”。

❷ encoding：XML 文档的编码。

❸ standalone：包含“yes”或“no”的字符串，用来指定 XML 是独立的还是需要解析外部的实体。

说明：使用 LINQ to XML 技术中的类时，需要添加 System.Linq.Xml 命名空间。

（8）IEnumerable<T>泛型接口的 First 方法

IEnumerable<T>泛型接口公开枚举数，该枚举数支持在指定类型的集合上进行简单迭代，其 First 方法用来返回序列中的第一个元素。语法格式如下：

```
public static TSource First<TSource>(this IEnumerable<TSource> source)
```

参数说明

❶ source：要返回其第一个元素的 IEnumerable<T>。

❷ 返回值：返回指定序列中的第一个元素。

（9）IEnumerable<T>泛型接口的 Remove 方法

Remove 方法用来将源集合中的每个节点从其父节点中移除。语法格式如下：

```
public static void Remove<T>(this IEnumerable<T> source) where T : XNode
```

参数说明

source：一个包含源集合的 XNode 的 IEnumerable<T>。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OperateXML。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，该窗体中主要用到的控件及说明如表 15.7 所示。

表 15.7　Frm_Main 窗体中用到的控件及说明

控件类型	控件名称	属性设置	说　明
TextBox	textBox1	ReadOnly 属性设置为 True，Text 属性设置为 Peoples	顶级节点名称
	textBox2	ReadOnly 属性设置为 True，Text 属性设置为 People	子节点名称
	textBox3	ReadOnly 属性设置为 True，Text 属性设置为 ID	子节点属性
	textBox4	ReadOnly 属性设置为 True，Text 属性设置为 Name	第一个元素名称
	textBox5	无	第一个元素值
	textBox6	ReadOnly 属性设置为 True，Text 属性设置为 Sex	第二个元素名称
	textBox7	无	第二个元素值
	textBox8	ReadOnly 属性设置为 True，Text 属性设置为 Salary	第三个元素名称
	textBox9	无	第三个元素值
	textBox10	ReadOnly 属性设置为 True，Text 属性设置为 001	子节点属性值
	textBox11	无	职工姓名
	textBox12	无	职工薪水
ComboBox	comboBox1	DropDownStyle 属性设置为 DropDownList，Items 属性中添加“男”和“女”	职工性别
DataGridView	dataGridView1	无	显示 XML 文件信息
Button	button1	Text 属性设置为“创建”	创建 XML 文件
	button2	Text 属性设置为“添加”	添加 XML 文件元素
	button3	Text 属性设置为“修改”	修改 XML 文件元素
	button4	Text 属性设置为“删除”	删除 XML 文件元素

（3）程序的主要代码如下。

Frm_Main 窗体加载时，判断指定的 XML 文件是否存在，如果存在，则将“创建 XML 文件”区域的所有控件的状态设置为不可用，并将 XML 文件中的数据显示在 DataGridView 控件；否则，将“创建 XML 文件”区域的所有控件的状态设置为可用。Form1 窗体的 Load 事件代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    if (File.Exists(strPath))                          //判断 XML 文件是否存在
    {
        groupBox1.Enabled = false;                     //将容器控件设置为不可用
        getXmlInfo();                                  //显示 XML 文件中的所有信息
    }
    else
        groupBox1.Enabled = true;                      //将容器控件设置为可用
}
```

上面的代码中用到 getXmlInfo 方法，该方法为自定义的无返回值类型方法，它主要用来将 XML 文件中的内容绑定到 DataGridView 控件中。getXmlInfo 方法的实现代码如下：

```
#region 将 XML 文件内容绑定到 DataGridView 控件
/// <summary>
/// 将 XML 文件内容绑定到 DataGridView 控件
/// </summary>
private void getXmlInfo()
{
    DataSet myds = new DataSet();                      //创建 DataSet 数据集对象
    myds.ReadXml(strPath);                             //读取 XML 结构
    dataGridView1.DataSource = myds.Tables[0];         //显示 XML 文件中的信息
}
#endregion
```

单击“创建”按钮，根据用户输入的元素值创建一个 XML 文件，并保存到程序文件夹下的 bin 文件夹中。“创建”按钮的 Click 事件代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    XDocument doc = new XDocument(                                          //创建 XML 文档对象
        new XDeclaration("1.0", "utf-8", "yes"),                            //添加 XML 文件声明
        new XElement(textBox1.Text,                                         //创建 XML 元素
            new XElement(textBox2.Text, new XAttribute(textBox3.Text, textBox10.Text),//为 XML 元素添加属性
                new XElement(textBox4.Text, textBox5.Text),
                new XElement(textBox6.Text, textBox7.Text),
                new XElement(textBox8.Text, textBox9.Text))
            )
        );
    doc.Save(strPath);                                                      //保存 XML 文档
    groupBox1.Enabled = false;
    getXmlInfo();
}
```

单击“添加”按钮，使用 LINQ to XML 技术向指定的 XML 文件中插入用户输入的数据，并重新保存 XML 文件。“添加”按钮的 Click 事件代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    XElement xe = XElement.Load(strPath);                                   //加载 XML 文档
    IEnumerable<XElement> elements1 = from element in xe.Elements("People")  //创建 IEnumerable 泛型接口
                                      select element;
    //生成新的编号
    string str = (Convert.ToInt32(elements1.Max(element => element.Attribute("ID").Value)) + 1).ToString("000");
    XElement people = new XElement(                                         //创建 XML 元素
        "People", new XAttribute("ID", str),                                //为 XML 元素设置属性
        new XElement("Name", textBox11.Text),
        new XElement("Sex", comboBox1.Text),
        new XElement("Salary", textBox12.Text)
        );
    xe.Add(people);                                                         //添加 XML 元素
    xe.Save(strPath);                                                       //保存 XML 元素到 XML 文件
    getXmlInfo();
}
```

当用户在 DataGridView 控件中选择某记录时，使用 LINQ to XML 技术在 XML 文件中查找选中记录的详细信息，并显示到相应的文本框和下拉列表中。代码如下：

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    strID = dataGridView1.Rows[e.RowIndex].Cells[3].Value.ToString();       //记录选中的 ID 编号
    XElement xe = XElement.Load(strPath);                                   //加载 XML 文档
    IEnumerable<XElement> elements = from PInfo in xe.Elements("People")    //根据编号查找信息
                                     where PInfo.Attribute("ID").Value == strID
                                     select PInfo;
    foreach (XElement element in elements)                                  //遍历查找到的所有信息
    {
        textBox11.Text = element.Element("Name").Value;                     //显示员工姓名
        comboBox1.SelectedItem = element.Element("Sex").Value;              //显示员工性别
        textBox12.Text = element.Element("Salary").Value;                   //显示员工薪水
    }
}
```

单击“修改”按钮，首先判断是否选定要修改的记录，如果已经选定，则使用 LINQ to XML 技术修改 XML 文件中的指定记录，并重新保存 XML 文件。“修改”按钮的 Click 事件代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    if (strID != "")                                                        //判断是否选择了编号
    {
        XElement xe = XElement.Load(strPath);                               //加载 XML 文档
        IEnumerable<XElement> elements = from element in xe.Elements("People")  //根据编号查找信息
                                         where element.Attribute("ID").Value == strID
                                         select element;
```

```
        if (elements.Count() > 0)                                    //判断是否找到了信息
        {
            XElement newXE = elements.First();                       //获取找到的第一条记录
            newXE.SetAttributeValue("ID", strID);                    //为 XML 元素设置属性值
            newXE.ReplaceNodes(                                      //替换 XML 元素中的值
                new XElement("Name", textBox11.Text),
                new XElement("Sex", comboBox1.Text),
                new XElement("Salary", textBox12.Text)
                );
        }
        xe.Save(strPath);                                            //保存 XML 元素到 XML 文件
    }
    getXmlInfo();
}
```

单击“删除”按钮，首先判断是否选定要删除的记录，如果已经选定，则使用 LINQ to XML 技术删除 XML 文件中的指定记录，并重新保存 XML 文件。“删除”按钮的 Click 事件代码如下：

```
private void button4_Click(object sender, EventArgs e)
{
    if (strID != "")                                                 //判断是否选择了编号
    {
        XElement xe = XElement.Load(strPath);                        //加载 XML 文档
        IEnumerable<XElement> elements = from element in xe.Elements("People")   //根据编号查找信息
                                         where element.Attribute("ID").Value == strID
                                         select element;
        if (elements.Count() > 0)                                    //判断是否找到了信息
            elements.First().Remove();                               //删除找到的 XML 元素信息
        xe.Save(strPath);                                            //保存 XML 元素到 XML 文件
    }
    getXmlInfo();
}
```

秘笈心法

心法领悟 376：什么是 XML？

XML 是 eXtensible Markup Language 的简写，是一种提供数据描述格式的标记语言。XML 以一种简单的文本格式存储数据，可以在不同系统间进行数据交换，这使它成为在 Internet 上传输数据的绝佳格式。XML 是 Visual Studio.NET 和.NET Framework 的很多功能的核心。

实例 377　创建 PDF 文档

光盘位置：光盘\MR\15\377　　中级　　趣味指数：★★★★

实例说明

PDF 文档凭借界面美观、漂亮、大方赢得了众多 PDF 迷的喜爱，该文档简单易用，可以实现打印、查找及阅读等常用的功能。本实例将介绍如何使用 C#编程语言创建 PDF 文档。实例运行效果如图 15.25 所示。创建完的 PDF 文档如图 15.26 所示。

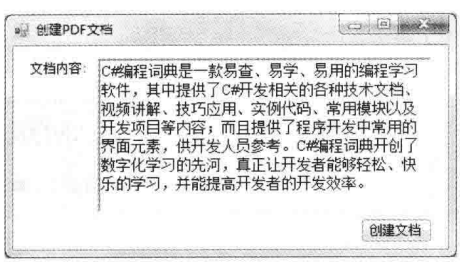


图 15.25　创建 PDF 文档

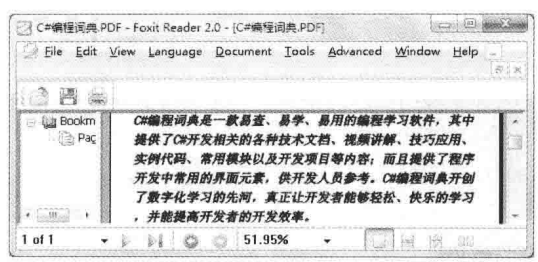


图 15.26　创建完的 PDF 文档

关键技术

本实例在创建 PDF 文档时，用到了第三方组件 itextsharp.dll。具体实现时，首先创建一个 itextSharp.text.Document 对象，然后使用该对象创建一个 Writer 对象，打开当前 Document 对象，向该对象内写入内容，最后关闭 Document 对象，完成创建 PDF 文档工作。

例如，本实例中，使用 Document 对象的 Add 方法将内容添加到 PDF 文档中的实现代码如下：

```
//使用指定的路径和创建模式初始化文件流对象
PdfWriter.getInstance(document, new FileStream(filePath, FileMode.Create));
document.Open();                                                    //打开文档
BaseFont baseFont = BaseFont.createFont(@"c:\windows\fonts\SIMSUN.TTC,1", BaseFont.IDENTITY_H, BaseFont.NOT_EMBEDDED);
iTextSharp.text.Font font =new iTextSharp.text.Font(baseFont, 20);  //设置文档字体样式
document.Add(new Paragraph(richTextBox1.Text,font ));               //添加内容至 PDF 文档中
document.Close();                                                   //关闭文档
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreatePDFDocument。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 RichTextBox 控件，用来输入要保存的内容；添加一个 Button 控件，用来创建 PDF 文档。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();                //给出文件保存信息，确定保存位置
    saveFileDialog.Filter = "PDF 文件（*.PDF）|*.PDF";
    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        filePath = saveFileDialog.FileName;
        //开始创建 PDF 文档，首先声明一个 Document 对象
        Document document = new Document();
        //使用指定的路径和创建模式初始化文件流对象
        PdfWriter.getInstance(document, new FileStream(filePath, FileMode.Create));
        document.Open();                                                 //打开文档
        BaseFont baseFont = BaseFont.createFont(@"c:\windows\fonts\SIMSUN.TTC,1", BaseFont.IDENTITY_H, BaseFont.NOT_EMBEDDED);
        iTextSharp.text.Font font = new iTextSharp.text.Font(baseFont, 20);  //设置文档字体样式
        document.Add(new Paragraph(richTextBox1.Text, font));            //添加内容至 PDF 文档中
        document.Close();                                                //关闭文档
        MessageBox.Show("祝贺你，文档创建成功！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 377：合理使用第三方组件。

.NET 框架本身提供了丰富的 dll 组件，通常情况下这些组件完全可以满足开发人员的需求。但有些第三方组件具有特定的功能，并且执行效率非常高，可以满足开发人员的某些特殊要求，这时可以考虑使用第三方组件来解决开发中遇到的一些特殊需求。

15.7 其　　他

实例 378	获取窗口文本	初级
	光盘位置：光盘\MR\15\378	趣味指数：★★★★

实例说明

在系统中按 Alt+Ctrl+Del 组合键，打开任务管理器，会看到当前运行的程序，本实例可以获取当前运行程

序窗口的文本。实例运行效果如图 15.27 所示。

图 15.27　获取窗口文本

关键技术

本实例在实现获取窗口文本功能时，主要用到了系统 API 函数 GetWindow 和 GetWindowText，下面分别对它们进行详细讲解。

（1）GetWindow 函数

该函数可以获得一个窗体的句柄。语法格式如下：

```
[DllImportAttribute("user32.dll")]
private static extern IntPtr GetWindow(IntPtr handle,int ucmd);
```

参数说明

❶ handle：当前窗口的句柄。

❷ ucmd：标识变量。

❸ 返回值：目标窗体的句柄。

（2）GetWindowText 函数

该函数可以通过窗体的句柄获得该窗体的标题文字。语法格式如下：

```
[DllImportAttribute("user32.dll")]
private static extern int GetWindowText(IntPtr handle, StringBuilder Text, int MaxCount);
```

参数说明

❶ handle：欲获取文字的那个窗口的句柄。

❷ Text：预定义的一个缓冲区，至少有 MaxCount 个字符大小；该缓冲区随同窗口文字载入。

❸ MaxCount：Text 缓冲区的长度。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetWindowsText。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 RichTextBox 控件，用来显示获取到的窗口文本；添加两个 Button 控件，分别用来执行获取窗口文本和关闭当前窗体操作。

（3）程序的主要代码如下：

```
private   void button1_Click(object sender, EventArgs e)
{
    richTextBox1.Clear();                                          //清空 richTextBox1 控件
    StringBuilder text = new StringBuilder(2560);                  //创建 StringBuilder 对象
    IntPtr currentHandle;                                          //定义变量
    currentHandle = GetWindow(this.Handle, GW_HWNDNEXT);           //获得窗体句柄
    int v = GetWindowText(currentHandle, text, 2560);              //获得窗体文字
    richTextBox1.Text = text.ToString();                           //显示窗体文字
}
```

秘笈心法

心法领悟 378：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 379 判断文件是否正在被使用

光盘位置：光盘\MR\15\379

中级

趣味指数：★★★☆

实例说明

当程序中访问外部文件时，如果被访问的文件正在被其他的程序所使用，则程序可能会发出异常，并停止运行。所以在对外部文件操作之前最好先进行判断，判断文件是否正在被使用，然后再进行其他操作。本实例将介绍如何判断文件是否正在被使用。实例运行效果如图 15.28 所示。

图 15.28 判断文件是否正在被使用

关键技术

本实例实现时，首先使用 File 类的 Move 方法移动要判断的文件，如果可以移动，说明没有被使用；如果不能移动，则表明文件正在被使用。

说明：关于 File 类的 Move 方法的详细讲解请参见实例 369 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 JudgeFileOpen。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来选择文件；添加两个 CheckBox 控件，分别用来显示文件是否正在被使用；添加两个 Button 控件，分别用来检测文件是否正在被使用和关闭当前窗体。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    checkBox1.Checked = false;                                          //没有被选中
    checkBox2.Checked = true;                                           //被选中
    openFileDialog1.ShowDialog();                                       //打开文件对话框
    try
    {
        System.IO.File.Move(openFileDialog1.FileName, openFileDialog1.FileName);   //移动文件
    }
    catch
    {
        checkBox2.Checked = false;                                      //没有被选中
        checkBox1.Checked = true;                                       //被选中
    }
}
```

秘笈心法

心法领悟 379：使用 try…catch 语句捕获异常。

在大部分情况下，开发人员不希望异常状况发生导致程序结束，因此，可以使用 try…catch 程序语句块捕捉程序中的 Exception 对象，再使用自定义的程序逻辑处理异常状况。如果有需要，用户也可以使用多重 try…catch 语句块（如一个 try 块后跟一个或多个 catch 子句构成）。

实例 380　C#中实现文件拖放

光盘位置：光盘\MR\15\380　　初级　　趣味指数：★★★★

实例说明

在 Windows 操作系统的文件夹中，经常会以文件拖放的形式对文件进行复制，使文件便于管理和使用。本实例制作了一个拖放文件的窗体，将所选的文件拖放到该窗体中，将在该窗体中显示被拖放文件的路径及文件名。实例运行效果如图 15.29 所示。

图 15.29　C#中实现文件拖放

关键技术

本实例实现时主要用到了 DragEventArgs 类的 Data 属性及 DataObject 类的 GetDataPresent 和 GetData 方法，下面分别对它们进行详细讲解。

（1）DragEventArgs 类的 Data 属性

DragEventArgs 类包含与所有拖放事件（DragEnter、DragLeave、DragOver 和 Drop）相关的参数，其 Data 属性用来获取一个数据对象，该对象包含与对应拖动事件关联的数据。Data 属性的语法格式如下：

```
public IDataObject Data { get; }
```

参数说明

属性值：一个数据对象，该对象包含与对应拖动事件关联的数据。

（2）DataObject 类的 GetDataPresent 方法

DataObject 类主要实现基数据传输机制，其 GetDataPresent 方法用来确定 DataObject 对象中存储的数据是否与指定的格式关联，该方法有 3 种重载形式，本实例中用到的重载形式如下：

```
public virtual bool GetDataPresent(string format)
```

参数说明

❶ format：要检查的格式。

❷ 返回值：如果 DataObject 对象中存储的数据与指定的格式关联，或者可以转换成指定的格式，则为 true；否则为 false。

（3）DataObject 类的 GetData 方法

DataObject 类的 GetData 方法用来返回与所指定数据格式关联的数据，它有 3 种重载形式，本实例中用到的重载形式如下：

```
public virtual Object GetData(string format)
```

参数说明

❶ format：要检索的数据的格式。

❷ 返回值：与指定格式关联的数据，或为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AllowDropFile。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来显示拖放到窗体中的文件名。

（3）程序的主要代码如下：

```
private void Form1_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.FileDrop))
    {
        string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);          //获取拖入文件的基本信息
        for (int i = 0; i < files.Length; i++)                                    //把窗体的文件的文件名加入 ListBox
```

```
        {
            listBox1.Items.Add(files[i]);                    //添加文件的路径
        }
    }
}
```

秘笈心法

心法领悟 380：程序中尽量不要使用太多的嵌套 for 语句。

由于嵌套 for 语句将消耗很大的资源，所以在实际开发项目时，能不使用嵌套 for 语句尽量不要使用。

实例 381 根据内容对文件进行比较

光盘位置：光盘\MR\15\381　　初级　　趣味指数：★★★☆

实例说明

本实例通过对两个文件的内容进行比较来确定它们是否相等，运行本实例，单击“源文件”按钮，选择第一个文件；单击“目标文件”按钮，选择第二个文件；单击“文件比较”按钮，对选择的两个文件的内容进行比较。实例运行效果如图 15.30 所示。

图 15.30　根据内容对文件进行比较

关键技术

本实例主要使用 StreamReader 类对两个文件的内容进行比较。实现时，首先将源文件用 StreamReader 类读入，并将目标文件用另一个 StreamReader 类读入，然后使用 StreamReader 类的 ReadToEnd 方法将文件内容读出进行比较，下面对本实例中用到的关键技术进行详细讲解。

StreamReader 类用来实现 TextReader 抽象类，使其以一种特定的编码从字节流中读取字符，其 ReadToEnd 方法用来从流的当前位置到末尾读取流。语法格式如下：

```
public override string ReadToEnd()
```

参数说明

返回值：字符串形式的流的其余部分（从当前位置到末尾）。如果当前位置位于流的末尾，则返回空字符串（""）。

注意：使用 StreamReader 类实现文件的读取速度非常快，但有一个弊端，就是一次读取的文件不能过大，否则会提示内存不足。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileEqual。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 Button 控件，分别用来执行选择源文件、选择目标文件和比较文件内容操作；添加两个 TextBox 控件，分别用来显示选择的源文件和目标文件路径。

（3）程序的主要代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    StreamReader sr1 = new StreamReader(textBox1.Text);          //创建 StreamReader 对象
    StreamReader sr2 = new StreamReader(textBox2.Text);          //创建 StreamReader 对象
    if (object.Equals(sr1.ReadToEnd(), sr2.ReadToEnd()))         //读取文件内容并判断
    {
        MessageBox.Show("两个文件相等");
    }
    else
```

```
    {
        MessageBox.Show("两个文件不相等");
    }
}
```

秘笈心法

心法领悟 381：区分 ReadLine 方法和 ReadToEnd 方法。

StreamReader 类的 ReadLine 方法用于从流中读取第一行字符，而 ReadToEnd 方法用于从流的当前位置到末尾读取流。

实例 382　获取文件夹中的图标资源

光盘位置：光盘\MR\15\382　　高级　　趣味指数：★★★★☆

实例说明

在制作类似于 FTP 的上传文件时，要在自制的文件列表中显示服务器端的所有文件及文件夹的名称，为了快速地判断文件类型，可以在文件的前面显示该文件的系统图标。本实例的主要功能是获取相应文件及文件夹的系统图标。运行本实例，选择文件夹的目录，单击“显示”按钮，在列表中将显示该文件夹下的所有文件及子文件夹的相关信息，以及相应的图标。实例运行效果如图 15.31 所示。

图 15.31　获取文件夹中的图标资源

关键技术

本实例主要用 API 函数 SHGetFileInfo 获取文件的基本信息及其图标，然后用 API 函数 ExtractIconEx 获取图标句柄进行相应的显示，下面对这两个 API 函数进行详细讲解。

（1）SHGetFileInfo 函数

该函数主要用于获取包含在可执行文件或 DLL 中的图标数或图标句柄。语法格式如下：

```
[DllImport("shell32.dll", EntryPoint = "SHGetFileInfo")]
public static extern IntPtr SHGetFileInfo(string pszPath, uint dwFileAttribute, ref SHFILEINFO psfi, uint cbSizeFileInfo, uint Flags);
```

SHGetFileInfo 函数中的参数说明如表 15.8 所示。

表 15.8　SHGetFileInfo 函数中的参数说明

参　数	描　述
pszPath	文件的详细路径
dwFileAttribute	一个或多个文件属性标志的结合，但是如果该函数的最后一个参数 uFlags 不包括 SHGFI_USEFILEATTRIBUTES 标志，则该参数被忽略
psfi	用于返回文件信息（用 SHFILEINFO 结构保存）
cbSizeFileInfo	通过参数 psfi 指向 SHFILEINFO 结构的大小，以字节表示
Flags	N 个标志参数的结合
返回值	取决于参数 uFlags

（2）ExtractIconEx 函数

该函数从限定的可执行文件、动态链接库（DLL），或者图标文件中生成图标句柄数组。语法格式如下：

```
[DllImport("shell32.dll")]
public static extern uint ExtractIconEx(string lpszFile, int nIconIndex, int[] phiconLarge, int[] phiconSmall, uint nIcons);
```

ExtractIconEx 函数中的参数说明如表 15.9 所示。

表 15.9 ExtractIconEx 函数中的参数说明

参　数	描　述
lpszFile	定义可获取图标的可执行文件、DLL，或者图标文件的名字的空结束字符串指针
nIconIndex	指定抽取第一个图标基于零的指针
phiconLarge	指向图标句柄数组的指针，它可接收从文件获取的大图标的句柄。如果该参数是 NULL，没有从文件抽取大图标
phiconSmall	指向图标句柄数组的指针，它可接收从文件获取的小图标的句柄。如果该参数是 NULL，没有从文件抽取小图标
nIcons	指定要从文件中抽取图标的数目
返回值	如果 nIconIndex 参数是-1，PhiconLarge 和 PhiconSmall 参数是 NULL，返回值是包含在指定文件中的图标数目；否则，返回值是成功地从文件中获取图标的数目

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileIco。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的文件夹路径；添加两个 Button 控件，分别用来选择文件夹和获取文件夹中的图标资源；添加一个 ImageList 组件，用来存储文件及文件夹的图标；添加一个 ListView 控件，用来显示文件的基本信息。

（3）程序的主要代码如下：

```
public void GetListViewItem(string path, ImageList imglist, ListView lv)                    //获取指定路径下的所有文件及其图标
{
    lv.Items.Clear();
    SHFILEINFO shfi = new SHFILEINFO();                                                      //创建 SHFILEINFO 对象
    try
    {
        string[] dirs = Directory.GetDirectories(path);                                      //获取指定目录中子目录的名称
        string[] files = Directory.GetFiles(path);                                           //获取目录中文件的名称
        for (int i = 0; i < dirs.Length; i++)                                                //遍历子文件夹
        {
            string[] info = new string[4];                                                   //定义一个数组
            DirectoryInfo dir = new DirectoryInfo(dirs[i]);                                  //根据文件夹的路径创建 DirectoryInfo 对象
            if (!(dir.Name == "RECYCLER" || dir.Name == "RECYCLED" || dir.Name == "Recycled" || dir.Name == "System Volume
Information"))
            {
                //获取文件夹的图标及类型
                SHGetFileInfo(dirs[i], (uint)0x80, ref shfi, (uint)System.Runtime.InteropServices.Marshal.SizeOf(shfi), (uint)(0x100 | 0x400));
                imglist.Images.Add(dir.Name, (Icon)Icon.FromHandle(shfi.hIcon).Clone());  //添加图标
                info[0] = dir.Name;                                                          //获取文件夹的名称
                info[1] = "";                                                                //获取文件夹的大小
                info[2] = "文件夹";                                                          //获取类型
                info[3] = dir.LastWriteTime.ToString();                                      //获取更改时间
                ListViewItem item = new ListViewItem(info, dir.Name);                        //创建 ListViewItem 对象
                lv.Items.Add(item);                                                          //添加当前文件夹的基本信息
                DestroyIcon(shfi.hIcon);                                                     //销毁图标
            }
        }
        for (int i = 0; i < files.Length; i++)                                               //遍历文件
        {
            string[] info = new string[4];                                                   //定义一个数组
            FileInfo fi = new FileInfo(files[i]);                                            //根据文件路径创建 FileInfo 对象
            //获取文件的类型
            string Filetype = fi.Name.Substring(fi.Name.LastIndexOf(".") + 1, fi.Name.Length - fi.Name.LastIndexOf(".") - 1);
            string newtype = Filetype.ToLower();                                             //将文件类型转换为小写
            if (!(newtype == "sys" || newtype == "ini" || newtype == "bin" || newtype == "log" || newtype == "com" || newtype == "bat" || newtype ==
"db"))
            {
                SHGetFileInfo(files[i], (uint)0x80, ref shfi, (uint)System.Runtime.InteropServices.Marshal.SizeOf(shfi), (uint)(0x100 | 0x400));
                //获取文件的图标及类型
```

```
                imglist.Images.Add(fi.Name, (Icon)Icon.FromHandle(shfi.hIcon).Clone());  //添加图标
                info[0] = fi.Name;                                                       //获取文件的名称
                info[1] = fi.Length.ToString();                                          //获取文件的大小
                info[2] = fi.Extension.ToString();                                       //获取文件的类型
                info[3] = fi.LastWriteTime.ToString();                                   //获取文件的更改时间
                ListViewItem item = new ListViewItem(info, fi.Name);                     //创建 ListViewItem 对象
                lv.Items.Add(item);                                                      //添加当前文件的基本信息
                DestroyIcon(shfi.hIcon);                                                 //销毁图标
            }
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 382：string 类的 Substring 方法的使用场合。

在程序开发中连接 Access 数据库时，可以使用 string 类的 Substring 方法获得 Access 数据库的文件名称；另外，对于一些其他的类似获取文件路径这样的功能，都可以借助 string 类的 Substring 方法来实现。

实例 383　将一个文件分割为多个小文件　高级

光盘位置：光盘\MR\15\383　　趣味指数：★★★★★

实例说明

现在网络已经非常普及，在网络上可以自由地传输文件，但是，如果文件过大，则不方便在网络上传输，这时可以将大文件分割成小文件，分批进行传输。本实例实现了将选定文件按照指定的大小进行分割的功能。实例运行效果如图 15.32 所示。分割后的文件效果如图 15.33 所示。

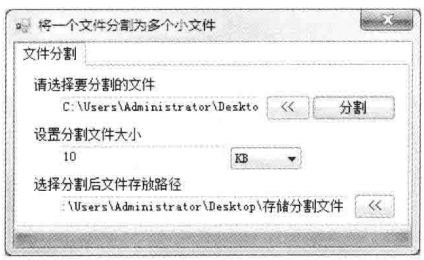


图 15.32　将一个文件分割为多个小文件

图 15.33　分割后的文件效果

关键技术

本实例对文件进行分割操作时，主要是通过 FileStream、BinaryReader 和 BinaryWriter 类实现的。具体实现时，以文件的全路径对应的字符串和文件打开模式来初始化 FileStream 文件流，然后再以 FileStream 文件流来初始化 BinaryReader 文件阅读器，最后通过 for 语句循环将大文件分割成多个小文件。

（1）BinaryReader 类

该类主要是用特定的编码将基元数据类型读作二进制值，其构造函数主要使用 UTF8Encoding 初始化 BinaryReader 类的新实例。BinaryReader 类的构造函数有两种重载形式，本实例中用到的重载形式如下：

```
public BinaryReader(Stream input)
```

参数说明

input：流。

本实例在读取要分割文件的内容时，用到了 BinaryReader 类的 ReadBytes 方法，该方法用来从当前流中将 count 个字节读入字节数组，并使当前位置提升 count 个字节。语法格式如下：

```
public virtual byte[] ReadBytes(int count)
```

参数说明

❶ count：要读取的字节数。

❷ 返回值：包含从基础流中读取的数据的字节数组。如果到达了流的末尾，则该字节数组可能小于所请求的字节数。

（2）BinaryWriter 类

该类主要是以二进制形式将基元类型写入流，并支持用特定的编码写入字符串，其构造函数主要使用 UTF-8 作为字符串编码来初始化 BinaryWriter 类的新实例。BinaryWriter 类的构造函数有 3 种重载形式，本实例中用到的重载形式如下：

```
public BinaryWriter(Stream output)
```

参数说明

output：输出流。

本实例在将内容写入到分割后的文件中时，用到了 BinaryWriter 类的 Write 方法，该方法用来将值写入当前流，它有数十种重载形式，本实例中用到的重载形式如下：

```
public virtual void Write(byte[] buffer)
```

参数说明

buffer：包含要写入的数据的字节数组。

说明：关于 FileStream 类的详细讲解请参见实例 372 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileComminuteUnite。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 TextBox 控件，分别用来显示要分割的文件、分割文件的大小和分割后文件的存放路径；添加一个 ComboBox 控件，用来选择分割单位；添加 3 个 Button 控件，分别用来选择要分割的文件、选择分割后文件的存放路径及执行分割文件操作；添加一个 ProgressBar 控件，用来显示分割进度条。

（3）程序的主要代码如下：

```
#region 分割文件
/// <summary>
///分割文件
/// </summary>
/// <param name="strFlag">分割单位</param>
/// <param name="intFlag">分割大小</param>
/// <param name="strPath">分割后的文件存放路径</param>
/// <param name="strFile">要分割的文件</param>
/// <param name="PBar">进度条显示</param>
public void SplitFile(string strFlag, int intFlag, string strPath, string strFile, ProgressBar PBar)
{
    int iFileSize = 0;
    //根据选择来设定分割的小文件的大小
    switch (strFlag)
    {
        case "Byte":
            iFileSize = intFlag;
            break;
        case "KB":
            iFileSize = intFlag * 1024;
            break;
        case "MB":
            iFileSize = intFlag * 1024 * 1024;
            break;
        case "GB":
            iFileSize = intFlag * 1024 * 1024 * 1024;
            break;
    }
    //以文件的全路径对应的字符串和文件打开模式来创建 FileStream 文件流对象
```

```
    FileStream SplitFileStream = new FileStream(strFile, FileMode.Open);
    //以 FileStream 文件流来创建 BinaryReader 文件阅读器
    BinaryReader SplitFileReader = new BinaryReader(SplitFileStream);
    byte[] TempBytes;                                                    //每次分割读取的最大数据
    int iFileCount = (int)(SplitFileStream.Length / iFileSize);          //小文件总数
    PBar.Maximum = iFileCount;
    if (SplitFileStream.Length % iFileSize != 0) iFileCount++;
    string[] TempExtra = strFile.Split('.');
    //循环将大文件分割成多个小文件
    for (int i = 1; i <= iFileCount; i++)
    {
        //确定小文件的文件名称
        string sTempFileName = strPath + @"\" + i.ToString().PadLeft(4, '0') + "." + TempExtra[TempExtra.Length - 1];
        //根据文件名称和文件打开模式来创建 FileStream 文件流对象
        FileStream TempStream = new FileStream(sTempFileName, FileMode.OpenOrCreate);
        //以 FileStream 实例来创建 BinaryWriter 书写器对象
        BinaryWriter TempWriter = new BinaryWriter(TempStream);
        TempBytes = SplitFileReader.ReadBytes(iFileSize);                //从大文件中读取指定大小数据
        TempWriter.Write(TempBytes);                                     //把此数据写入小文件
        TempWriter.Close();                                              //关闭书写器，形成小文件
        TempStream.Close();                                              //关闭文件流
        PBar.Value = i - 1;
    }
    SplitFileReader.Close();                                             //关闭大文件阅读器
    SplitFileStream.Close();                                             //关闭数据流
    MessageBox.Show("文件分割成功!");
}
#endregion
```

秘笈心法

心法领悟 383：如何以创建模式创建 FileStream 对象？

通常使用 FileStream 类的构造方法来创建一个已知文件的文件流，若把构造方法的枚举参数值改为 FileMode.Create，则会以创建模式创建 FileStream 类，这样程序将创建一个新文件。

实例 384　将多个小文件合并为一个文件

光盘位置：光盘\MR\15\384　　高级　　趣味指数：★★★★★

实例说明

一个文件可以分割为多个小文件，同理，用户还可以将分割后的多个小文件合并为一个文件，本实例就实现了将分割后的多个文件合并为一个文件的功能。实例运行效果如图 15.34 所示。合并后的文件效果如图 15.35 所示。

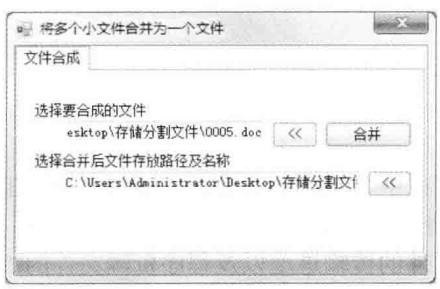


图 15.34　将多个小文件合并为一个文件

图 15.35　合并后的文件效果

关键技术

本实例实现文件合并操作时，主要是通过 FileStream、BinaryReader 和 BinaryWriter 类实现的。具体实现时，

首先以合并后的文件名称和打开方式来创建、初始化 FileStream 文件流，然后以 FileStream 文件流初始化 BinaryWriter 书写器，用以合并分割的文件，最后循环合并小文件。

说明：（1）关于 FileStream 类的详细讲解请参见实例 372 中的关键技术。

（2）关于 BinaryReader 类和 BinaryWriter 类的详细讲解请参见实例 383 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FileComminuteUnite。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示要合成的文件和合成后文件的存放路径；添加 3 个 Button 控件，分别用来选择要合成的文件、选择合成后文件的存放路径及执行合并文件操作；添加一个 ProgressBar 控件，用来显示合并进度条。

（3）程序的主要代码如下：

```
#region 合并文件
/// <summary>
///合并文件
/// </summary>
/// <param name="list">要合并的文件集合</param>
/// <param name="strPath">合并后的文件名称</param>
/// <param name="PBar">进度条显示</param>
public void CombinFile(string[] strFile, string strPath, ProgressBar PBar)
{
    PBar.Maximum = strFile.Length;
    FileStream AddStream = null;
    //以合并后的文件名称和打开方式来创建 FileStream 文件流
    AddStream = new FileStream(strPath, FileMode.Append);
    //以 FileStream 文件流来创建 BinaryWriter 书写器，用以合并分割的文件
    BinaryWriter AddWriter = new BinaryWriter(AddStream);
    FileStream TempStream = null;
    BinaryReader TempReader = null;
    //循环合并小文件，并生成合并文件
    for (int i = 0; i < strFile.Length; i++)
    {
        //以小文件所对应的文件名称和打开模式来创建 FileStream 文件流，起读取分割文件作用
        TempStream = new FileStream(strFile[i].ToString(), FileMode.Open);
        TempReader = new BinaryReader(TempStream);
        //读取分割文件中的数据，并生成合并后文件
        AddWriter.Write(TempReader.ReadBytes((int)TempStream.Length));
        TempReader.Close();                                    //关闭 BinaryReader 文件阅读器
        TempStream.Close();                                    //关闭 FileStream 文件流
        PBar.Value = i + 1;
    }
    AddWriter.Close();                                         //关闭 BinaryWriter 文件书写器
    AddStream.Close();                                         //关闭 FileStream 文件流
    MessageBox.Show("文件合并成功！");
}
#endregion
```

秘笈心法

心法领悟 384：使用 ProgressBar 控件显示进度条。

ProgressBar 控件是一个进度条控件，它通过在水平放置的方框中显示适当数目的矩形块来指示工作的进度，工作完成时，进度条被填满，进度条通常用于帮助用户了解等待一项工作完成的进度。ProgressBar 控件经常在多线程应用程序中使用，用来显示文件、文件夹的复制进度、文件的传输进度以及程序资源文件的加载进度等。

第16章

文件夹基本操作

- 获取文件夹信息
- 文件夹操作

16.1 获取文件夹信息

实例 385	获得文件夹创建时间	初级
	光盘位置：光盘\MR\16\385	趣味指数：★★★☆

实例说明

在 Windows 操作系统中新建一个文件夹之后，该文件夹会存在一些基本的信息，如文件夹名称、创建时间、修改时间等。本实例使用 C#语言实现了获取文件夹创建时间的功能。实例运行效果如图 16.1 所示。

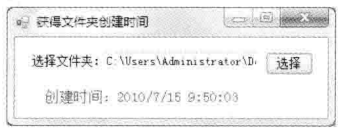


图 16.1 获得文件夹创建时间

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 CreationTime 属性，下面对其进行详细讲解。

DirectoryInfo 类公开了用于创建、移动和枚举文件夹和子文件夹的实例方法，其 CreationTime 属性用来获取或设置当前 DirectoryInfo 对象的创建时间，该属性的语法格式如下：

```
public DateTime CreationTime { get; set; }
```

参数说明

属性值：当前 DirectoryInfo 对象的创建日期和时间。

说明：（1）程序中使用 DirectoryInfo 类时，首先需要在命名空间区域添加 System.IO 命名空间。

（2）程序中使用与文件夹相关的类时，都需要添加 System.IO 命名空间，下文遇到类似情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetFoldCTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来选择文件夹；添加一个 TextBox 控件，用来显示选择的文件夹；添加一个 Label 控件，用来显示选中文件夹的创建时间。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    FolderBrowserDialog FBDialog = new FolderBrowserDialog();                    //创建 FolderBrowserDialog 对象
    if (FBDialog.ShowDialog() == DialogResult.OK)                                //判断是否选择文件夹
    {
        string strPath = FBDialog.SelectedPath;                                  //记录选择的文件夹
        textBox1.Text = strPath;                                                 //显示选择的文件夹
        DirectoryInfo DInfo = new DirectoryInfo(strPath);                        //创建 DirectoryInfo 对象
        label2.Text = "创建时间：" + DInfo.CreationTime.ToString();               //显示文件夹创建时间
    }
}
```

秘笈心法

心法领悟 385：使用 DirectoryInfo 类的相关属性获取文件夹的最新修改时间。

获取文件夹的最新修改时间时，可以使用 DirectoryInfo 类的 LastWriteTime 属性来实现。代码如下：

```
label2.Text = DInfo.LastWriteTime.ToString();                                //显示文件夹最新修改时间
```

实例 386　获得临时文件目录

光盘位置：光盘\MR\16\386　　初级　　趣味指数：★★★★☆

实例说明

为了提高运行速度，系统会在 Temp 文件夹下生成运行过的程序和文件、浏览网站时留下的临时文件，那么如何获得临时文件夹，以便清除这些临时文件呢？本实例通过 Path 对象的 GetTempPath 方法实现了获取临时文件目录的功能。实例运行效果如图 16.2 所示。

图 16.2　获取临时文件目录

关键技术

本实例主要使用 C#中提供的 GetTempPath 方法得到系统的临时文件目录。语法格式如下：

```
public static string GetTempPath ( )
```

参数说明

返回值：包含临时目录的路径信息的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetTempDirectory。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来获取系统的临时文件目录；添加一个 TextBox 控件，用来显示获取到的临时文件目录。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = Path.GetTempPath();                    //获取临时文件目录路径
}
```

秘笈心法

心法领悟 386：TextBox 控件的使用。

TextBox 控件，又称为文本框控件，主要用于获取用户输入的数据或者显示文本，它通常用于可编辑文本，也可以使其成为只读控件。文本框可以显示多行，开发人员可以使文本换行以便符合控件的大小。

实例 387　获得应用程序所在目录

光盘位置：光盘\MR\16\387　　初级　　趣味指数：★★★★☆

实例说明

获取程序中的报表等文件要使用相对路径，如果使用绝对路径，用户移动程序时程序将会出错，这时就需要获取应用程序所在的目录，设置报表等文件的路径为相对路径。本实例实现了获取应用程序所在目录的功能。实例运行效果如图 16.3 所示。

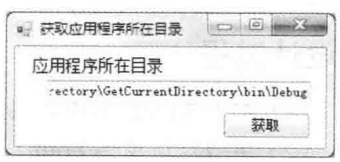


图 16.3　获取应用程序所在目录

关键技术

本实例实现时主要用到了 Application 类的 StartupPath 属性，下面对其进行详细讲解。

Application 类用来提供静态方法和属性以管理应用程序，其 StartupPath 属性用于获取当前目录（即该进程

从中启动的目录）的完全限定路径。语法格式如下：

```
public static string StartupPath { get; }
```

参数说明

属性值：启动应用程序的可执行文件的路径。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetCurrentDirectory。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来获取应用程序所在的目录；添加一个 TextBox 控件，用来显示获取到的应用程序所在的目录。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = Application.StartupPath;                    //获取应用程序的启动路径
}
```

秘笈心法

心法领悟 387：使用 Application.Exit 方法退出应用程序。

当程序中有隐藏窗体时，调用 this.Close 方法只能关闭当前窗体，这时可以在窗体的 FormClosing 事件中通过调用 Application.Exit 方法退出应用程序。

实例 388　获得系统当前目录

光盘位置：光盘\MR\16\388　　初级　　趣味指数：★★★☆

实例说明

系统当前目录是程序设计时经常考虑的一种外界环境因素。在程序设计时，因为不能确定用户当前的文件系统结构，所以经常会编写一些访问相对路径的代码。在程序运行时可以动态地得到系统当前路径，这样就能够解决这种问题。本实例将在程序运行时动态地得到系统当前的目录。实例运行效果如图 16.4 所示。

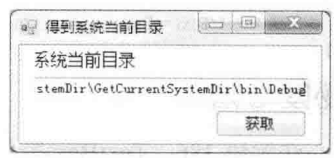


图 16.4　得到系统当前目录

关键技术

本实例实现时主要用到了 Environment 类的 CurrentDirectory 属性，下面对其进行详细讲解。

Environment 类用来提供有关当前环境和平台的信息以及操作它们的方法，其 CurrentDirectory 属性主要用来获取或设置当前工作目录的完全限定路径。语法格式如下：

```
public static string CurrentDirectory { get; set; }
```

参数说明

属性值：包含目录路径的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetCurrentSystemDir。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来获取系统当前目录；添加一个 TextBox 控件，用来显示获取到的系统当前目录。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = System.Environment.CurrentDirectory;        //获取系统当前目录
}
```

秘笈心法

心法领悟 388：如何获取本地计算机的 NetBIOS 名称。

获取本地计算机的 NetBIOS 名称时，可以使用 Environment 类的 MachineName 属性来实现，代码如下：

```
textBox1.Text = System.Environment.MachineName;                    //获取本地计算机的 NetBIOS 名称
```

实例 389　获取路径名禁止使用的字符

初级

光盘位置：光盘\MR\16\389

趣味指数：★★★☆

实例说明

在 Windows 操作系统中新建文件夹时，如果输入了非法字符，系统会提示错误，那么文件夹的名称究竟不允许使用哪些非法字符呢？本实例使用 C#获取到了文件夹路径名中禁止使用的所有字符。实例运行效果如图 16.5 所示。

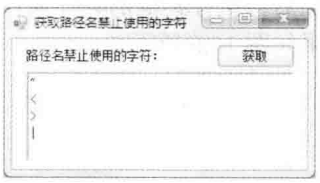


图 16.5　获取路径名禁止使用的字符

关键技术

本实例实现时主要用到了 Path 类的 GetInvalidPathChars 方法，下面对其进行详细介绍。

Path 类用来对包含文件或目录路径信息的 String 实例执行操作，其 GetInvalidPathChars 方法主要用来获取包含不允许在路径名中使用的字符的数组。语法格式如下：

```
public static char[] GetInvalidPathChars()
```

参数说明

返回值：包含不允许在路径名中使用的字符的数组。

说明：Path 类位于 System.IO 命名空间中。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetEnjoinChar。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来获取路径名中禁止使用的字符；添加一个 RichTextBox 控件，用来显示路径名中禁止使用的字符。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    richTextBox1.Text = string.Empty;                              //清空文本框
    char[] myChars = Path.GetInvalidPathChars();                   //创建字符串数组
    String myInfo = string.Empty;                                  //定义空字符串
    foreach (char myChar in myChars)                               //遍历字符数组
    {
        myInfo += myChar.ToString() + "\n";                        //获取字符数组中的项
    }
    richTextBox1.Text = myInfo;                                    //显示禁用的所有字符
}
```

秘笈心法

心法领悟 389：获取文件名中禁止使用的字符。

获取文件名中禁止使用的字符时，可以使用 Path 类的 GetInvalidFileNameChars 方法来实现，该方法用来获取包含不允许在文件名中使用的字符的数组。

16.2 文件夹操作

实例 390	判断指定文件夹是否存在 光盘位置：光盘\MR\16\390	初级 趣味指数：★★★☆

实例说明

在执行文件夹的一些基本操作（如移动文件夹或者删除文件夹）时，常常需要首先判断指定的文件夹是否存在，然后再执行其他的具体操作，那么如何来判断指定的文件夹是否存在呢？本实例将告诉读者答案。实例运行效果如图 16.6 所示。

图 16.6 判断指定文件夹是否存在

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 Exists 属性，下面对其进行详细讲解。

DirectoryInfo 类的 Exists 属性是用来获取指定文件夹是否存在的值。语法格式如下：

```
public override bool Exists { get; }
```

参数说明

属性值：如果文件夹存在，则为 true；否则为 false。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ISDirExist。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入要判断的文件夹路径；添加一个 Button 控件，用来判断文件夹是否存在。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        DirectoryInfo dirInfo = new DirectoryInfo(textBox1.Text);          //创建 DirectoryInfo 对象
        if (dirInfo.Exists)                                                //如果文件夹存在
        {
            MessageBox.Show("文件夹存在");
        }
        else
        {
            MessageBox.Show("文件夹不存在");
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 390：使用 MessageBox 显示消息对话框。

向用户显示信息和消息的对话框称为消息对话框，MessageBox 是一个预定义对话框，主要用于向用户显示与应用程序相关的信息及请求来自用户的信息，其 Show 方法用来显示消息对话框。

实例 391	创建文件夹 光盘位置：光盘\MR\16\391	初级 趣味指数：★★★☆

实例说明

本实例主要实现创建文件夹功能，运行本实例，首先单击“选择”按钮，选择文件夹的存放路径；然后输入要创建的文件夹名称，单击“创建”按钮，即可在指定的位置创建一个文件夹。实例运行效果如图 16.7 所示。

图 16.7　创建文件夹

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 Create 方法，下面对其进行详细讲解。

DirectoryInfo 类的 Create 方法用来创建文件夹，它有两种重载形式，分别如下：

```
public void Create()
public void Create(DirectorySecurity directorySecurity)
```

参数说明

directorySecurity：要应用于文件夹的访问控制。

说明：本实例使用了 DirectoryInfo 类的 Create 方法的第一种重载形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateDir。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的文件夹路径和输入创建的文件夹名称；添加两个 Button 控件，分别用来执行选择文件夹和创建文件夹操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    FolderBrowserDialog FBDialog = new FolderBrowserDialog();                    //创建 FolderBrowserDialog 对象
    if (FBDialog.ShowDialog() == DialogResult.OK)                                //判断是否选择文件夹
    {
        string strPath = FBDialog.SelectedPath;                                  //记录选择的文件夹
        if (strPath.EndsWith("\\"))                                              //判断是否以“\”结尾
            textBox1.Text = strPath;                                             //显示选择的文件夹
        else
            textBox1.Text = strPath + "\\";
    }
}
private void button2_Click(object sender, EventArgs e)
{
    DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text + textBox2.Text);      //创建 DirectoryInfo 对象
    DInfo.Create();                                                              //创建文件夹
}
```

秘笈心法

心法领悟 391：EndsWith 方法的使用。

EndsWith 方法用来确定 String 实例的末尾是否与指定的字符串匹配。例如，本实例中使用该方法判断文件夹路径是否以“\”结尾，代码如下：

```
if (strPath.EndsWith("\\"))                                                      //判断是否以“\”结尾
```

实例 392　以当前日期时间为根据创建文件夹

初级

光盘位置：光盘\MR\16\392

趣味指数：★★★★

实例说明

本实例主要实现以当前日期时间为根据创建文件夹的功能，运行本实例，单击“选择”按钮，选择文件夹的存放路径；单击“创建”按钮，以当前的日期时间为名称在指定位置创建一个文件夹。实例运行效果如图 16.8 所示。

图 16.8　以当前日期时间为根据创建文件夹

关键技术

本实例实现时，首先需要使用 DateTime 结构的 Now 属性获取系统当前日期时间，并对获取到的日期时间进行格式化，然后使用 DirectoryInfo 类的 Create 方法创建文件夹。下面对本实例中用到的关键技术进行详细讲解。

DateTime 结构表示时间上的一刻，通常以日期和当天的时间表示，其 Now 属性用来获取一个 DateTime 对象，该对象设置为此计算机上的当前日期和时间，表示为本地时间。本实例中使用 DateTime 结构的 Now 属性获取完系统当前日期时间之后，调用 ToString 方法对其进行了格式化，代码如下：

```
string strName = DateTime.Now.ToString("yyyyMMddhhmmss");                    //对当前日期进行格式化
```

说明：（1）本实例中将当前日期时间格式化为“20100717085112”格式，其中 2010 表示年份，07 表示月份，17 表示天数，08 表示时钟，51 表示分钟，12 表示秒钟。

（2）关于 DirectoryInfo 类的 Create 方法的详细讲解请参见实例 391 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateDirByDate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的文件夹路径；添加两个 Button 控件，分别用来执行选择文件夹操作和以当前日期时间为根据创建文件夹操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    string strName = DateTime.Now.ToString("yyyyMMddhhmmss");                //对当前日期进行格式化
    DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text + strName);        //创建 DirectoryInfo 对象
    DInfo.Create();                                                          //创建文件夹
    MessageBox.Show("文件夹创建成功，名称为" + strName + "");
}
```

秘笈心法

心法领悟 392：使用 DateTime.Now 属性获取系统时间。

使用 DateTime.Now 属性可以方便地获取系统时间，而且可以使用格式化方式方便地得到系统时间的字符串表示形式，代码如下：

```
string P_str = DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒");
```

实例 393　移动文件夹

光盘位置：光盘\MR\16\393　　初级　　趣味指数：★★★☆

实例说明

本实例主要实现移动文件夹的功能，运行本实例，首先单击第一个“选择”按钮，选择原文件夹位置；然后单击第二个“选择”按钮，选择目标文件夹位置；最后单击“移动”按钮，将选择的原文件夹移动到目标文件夹。实例运行效果如图 16.9 所示。

图 16.9　移动文件夹

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 MoveTo 方法，下面对其进行详细讲解。

DirectoryInfo 类的 MoveTo 方法用来将 DirectoryInfo 对象及其内容移动到新路径。语法格式如下：

```
public void MoveTo(string destDirName)
```

参数说明

destDirName：要将文件夹移动到的目标位置的名称和路径（目标不能是另一个具有相同名称的磁盘卷或文件夹，它可以是要将原文件夹作为子文件夹添加到其中的一个现有文件夹）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MoveDir。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的源文件夹路径和目标文件夹路径；添加 3 个 Button 控件，分别用来执行选择源文件夹、选择目标文件夹和移动文件夹操作。

（3）程序的主要代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text);                    //创建 DirectoryInfo 对象
        string strPath = textBox2.Text + textBox1.Text.Substring(textBox1.Text.LastIndexOf("\\") + 1, textBox1.Text.Length - textBox1.Text.
LastIndexOf("\\") - 1);                                                            //设置移动路径
        DInfo.MoveTo(strPath);                                                     //移动文件夹
    }
    catch { MessageBox.Show("移动的文件必须在同一盘符下！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information); }
}
```

秘笈心法

心法领悟 393：使用 try…catch 语句捕获异常。

在大多数情况下，开发人员不希望异常状况发生导致程序结束，因此，可以使用 try…catch 程序语句块捕捉程序中的 Exception 对象，再使用自定义的程序逻辑处理异常状况。如果有需要，用户也可以使用多重 try…catch 语句块（如一个 try 块后跟一个或多个 catch 子句构成）。

实例 394　删除文件夹

光盘位置：光盘\MR\16\394　　初级　　趣味指数：★★★☆

实例说明

本实例主要实现删除文件夹的功能，运行本实例，单击“选择”按钮，选择需要删除的文件夹，单击“删

除”按钮，删除选择的文件夹及其包含的所有子文件夹和文件。实例运行效果如图 16.10 所示。

图 16.10　删除文件夹

关键技术

本实例实现时主要用到了 DirectoryInfo 类的 Delete 方法，下面对其进行详细讲解。

DirectoryInfo 类的 Delete 方法用来删除文件夹，它有两种重载形式，分别如下：

```
public override void Delete()
public void Delete(bool recursive)
```

参数说明

recursive：若为 true，则删除该文件夹及其子文件夹和所有文件；否则为 false。

说明：本实例使用了 DirectoryInfo 类的 Delete 方法的第二种重载形式。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DeleteDir。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择要删除的文件夹；添加两个 Button 控件，分别用来执行选择文件夹和删除文件夹操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text);          //创建 DirectoryInfo 对象
    DInfo.Delete(true);                                              //删除文件夹中的所有内容
    MessageBox.Show("删除文件夹成功！");
}
```

秘笈心法

心法领悟 394：Directory 类和 DirectoryInfo 类的区别。

Directory 类是静态类，所以它的调用需要字符串参数为每一个方法调用规定文件夹路径，因此如果要在对象上进行单一方法调用，则可以使用静态 Directory 类，在这种情况下静态调用速度要快一些，因为.NET 框架不必执行实例化新对象并调用其方法的过程。如果要在文件夹上执行几种操作，则创建 DirectoryInfo 对象并使用其方法就更好一些，这样会提高效率，因为对象将在文件夹系统上引用正确的文件夹，而静态类就必须每次都寻找文件夹。

实例 395　获取所有逻辑磁盘目录

光盘位置：光盘\MR\16\395　　中级　　趣味指数：★★★★☆

实例说明

每台计算机都会有多个逻辑分区（如 C 盘或 D 盘），每个逻辑分区下又会存储许多文件，如果想浏览逻辑分区下的文件，相对来说比较费时。为了解决以上问题，这里使用 C#制作了一个程序，该程序可以获取逻辑分区下的所有逻辑磁盘目录，并可以对目录下的数据进行各项操作（例如，剪切、复制、粘贴、打开、删除和重命名等）。实例运行效果如图 16.11 所示。

图 16.11　获取所有逻辑磁盘目录

关键技术

本实例实现时，首先获取计算机中的所有逻辑分区（如 C 盘或 D 盘），主要通过 Directory 类的 GetLogicalDrives 方法实现；然后获取逻辑分区下所有子文件夹和文件，通过 Directory 类的 GetDirectories 和 GetFiles 方法实现。下面对本实例中用到的关键技术进行详细讲解。

（1）GetLogicalDrives 方法

该方法用来检索计算机上逻辑分区的名称。语法格式如下：

```
public static string[] GetLogicalDrives()
```

参数说明

返回值：计算机上的逻辑分区。

（2）GetDirectories 方法

该方法用来获取指定文件夹中子文件夹的名称。语法格式如下：

```
public static string[] GetDirectories(string path)
```

参数说明

❶ path：为其返回子文件夹名称的数组的路径。

❷ 返回值：一个类型 String 的数组，它包含 path 中子文件夹的名称。

（3）GetFiles 方法

该方法返回指定文件夹中的文件的名称。语法格式如下：

```
public static string[] GetFiles(string path)
```

参数说明

❶ path：将从其检索文件的文件夹。

❷ 返回值：指定文件夹中文件名的 String 数组。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetDiskList。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为窗体的工具栏；添加一个 ImageList 组件，用来存储文件及文件夹图标；添加一个 ContextMenuStrip 控件，用来作为程序的右键快捷菜单；添加一个 ListView 控件，用来显示磁盘下文件的相关信息。

（3）程序的主要代码如下：

在公共类 BaseClass 中自定义一个 listFolders 方法，用于获取计算机中的所有逻辑分区。代码如下：

```
public void listFolders(ToolStripComboBox tscb)                    //获取本地磁盘目录
{
    string[] logicdrives = System.IO.Directory.GetLogicalDrives();  //获取所有逻辑分区
    for (int i = 0; i < logicdrives.Length; i++)                    //遍历存储逻辑分区名称的数组
    {
        tscb.Items.Add(logicdrives[i]);                             //将分区名称添加到下拉列表中
```

```
            tscb.SelectedIndex = 0;                                    //设置第一项被选中
        }
    }
```

在公共类 BaseClass 中自定义一个 GetListViewItem 方法，用于根据选择的路径，获取路径下的所有文件及文件夹，并获取相关的信息显示在 ListView 控件中，并且该方法还能获取指定路径下所有文件及文件夹的图标。代码如下：

```
public void GetListViewItem(string path, ImageList imglist, ListView lv)      //获取指定路径下的所有文件及其图标
{
    lv.Items.Clear();                                                          //清空 ListView
    Win32.SHFILEINFO shfi = new Win32.SHFILEINFO();                            //创建 SHFILEINFO 对象
    try
    {
        string[] dirs = Directory.GetDirectories(path);                        //获取指定路径下的所有文件夹
        string[] files = Directory.GetFiles(path);                             //获取指定路径下的所有文件
        for (int i = 0; i < dirs.Length; i++)                                  //遍历文件夹
        {
            string[] info = new string[4];                                     //创建一个存储信息的数组
            DirectoryInfo dir = new DirectoryInfo(dirs[i]);                    //创建 DirectoryInfo 对象
            //判断遍历出来的文件是否是系统文件
            if (dir.Name == "RECYCLER" || dir.Name == "RECYCLED" || dir.Name == "Recycled" || dir.Name == "System Volume Information")
            { }
            else                                                               //如果不是系统文件则获得图标
            {
                Win32.SHGetFileInfo(dirs[i],
                                    (uint)0x80,
                                    ref shfi,
                                    (uint)System.Runtime.InteropServices.Marshal.SizeOf(shfi),
                                    (uint)(0x100 | 0x400));                    //取得 Icon 和 TypeName
                imglist.Images.Add(dir.Name, (Icon)Icon.FromHandle(shfi.hIcon).Clone());//添加图标
                info[0] = dir.Name;                                            //文件夹名称
                info[1] = "";
                info[2] = "文件夹";                                            //类型
                info[3] = dir.LastWriteTime.ToString();                        //最后修改时间
                ListViewItem item = new ListViewItem(info, dir.Name);          //创建 ListViewItem 对象
                lv.Items.Add(item);                                            //将信息添加到 ListView 控件中
                Win32.DestroyIcon(shfi.hIcon);                                 //销毁图标
            }
        }
        for (int i = 0; i < files.Length; i++)                                 //开始遍历文件
        {
            string[] info = new string[4];                                     //存储文件信息的数组
            FileInfo fi = new FileInfo(files[i]);                              //创建 FileInfo 对象
            //获取文件类型
            string Filetype = fi.Name.Substring(fi.Name.LastIndexOf(".")+1,fi.Name.Length- fi.Name.LastIndexOf(".") -1);
            string newtype=Filetype.ToLower();                                 //文件类型小写
            //如果获取的文件是系统文件则不执行操作
            if (newtype == "sys" || newtype == "ini" || newtype == "bin" || newtype == "log" || newtype == "com" || newtype == "bat" || newtype == "db")
            { }
            else                                                               //否则获得图标
            {
                Win32.SHGetFileInfo(files[i],
                                    (uint)0x80,
                                    ref shfi,
                                    (uint)System.Runtime.InteropServices.Marshal.SizeOf(shfi),
                                    (uint)(0x100 | 0x400));                    //取得 Icon 和 TypeName
                //添加图标
                imglist.Images.Add(fi.Name, (Icon)Icon.FromHandle(shfi.hIcon).Clone());
                info[0] = fi.Name;                                             //文件名称
                info[1] = fi.Length.ToString();                                //文件大小
                info[2] = fi.Extension.ToString();                             //表示文件扩展名部分的字符串
                info[3] = fi.LastWriteTime.ToString();                         //最后修改日期
                ListViewItem item = new ListViewItem(info, fi.Name);           //创建 ListViewItem 实例
                lv.Items.Add(item);                                            //添加到 ListView 控件中
                Win32.DestroyIcon(shfi.hIcon);                                 //销毁图标
            }
```

```
        }
    }
    catch
    {
    }
}
```

秘笈心法

心法领悟 395：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 396　获取指定文件夹的上级目录

初级

光盘位置：光盘\MR\16\396

趣味指数：★★★★

实例说明

使用 Windows 操作系统提供的资源管理器浏览文件夹时，可以通过单击“向上”按钮，返回当前文件夹的上级目录，那么如何使用 C#来实现类似的功能呢？本实例给出了准确的答案。实例运行效果如图 16.12 所示。

图 16.12　获取指定文件夹的上级目录

关键技术

本实例实现时主要用到了 Directory 类的 GetParent 方法，下面对其进行详细介绍。

Directory 类公开用于创建、移动和枚举通过文件夹和子文件夹的静态方法，其 GetParent 方法用来检索指定路径的父文件夹，包括绝对路径和相对路径。语法格式如下：

```
public static DirectoryInfo GetParent (string path)
```

参数说明

❶ path：用于检索父文件夹的路径。

❷ 返回值：父文件夹；或者如果 path 是根文件夹，包括 UNC 服务器或共享名的根，则为 null。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetUpDir。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来选择文件夹；添加一个 TextBox 控件，用来显示选择的文件夹；添加一个 Label 控件，用来显示选择的文件夹的当前目录和上级目录。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    FolderBrowserDialog FBDialog = new FolderBrowserDialog();          //创建 FolderBrowserDialog 对象
    if (FBDialog.ShowDialog() == DialogResult.OK)                      //判断是否选择了文件夹
    {
        textBox1.Text = FBDialog.SelectedPath;                         //显示选择的文件夹
        string str1 = textBox1.Text;                                   //记录选择的文件夹
        string str2 = Directory.GetParent(str1).FullName;              //获取上级目录的全名
        string myInfo = "当前目录是：" + str1;                          //显示当前文件夹
        myInfo += "\n 上级目录是：" + str2;                             //显示上级文件夹
        label2.Text = myInfo;
    }
}
```

秘笈心法

心法领悟 396：使用 FolderBrowserDialog 类显示“浏览文件夹”对话框。

“浏览文件夹”对话框使用 FolderBrowserDialog 类来表示，它主要用来提示用户选择文件夹，开发人员可以使用 FolderBrowserDialog 类的 ShowDialog 方法显示“浏览文件夹”对话框。FolderBrowserDialog 类最常用的属性是 SelectedPath 属性，该属性用来获取或设置用户选定的路径。

实例 397　使用递归法删除文件夹中的所有文件

光盘位置：光盘\MR\16\397　　中级　　趣味指数：★★★★

实例说明

使用递归法删除文件夹中的所有文件，即遍历文件夹中的所有文件，并将遍历到的文件进行一一删除。本实例实现了使用递归法删除文件夹中所有文件的功能。实例运行效果如图 16.13 所示。

图 16.13　使用递归法删除文件夹中的所有文件

关键技术

本实例实现时，首先需要创建 DirectoryInfo 对象，以指定文件夹；然后调用 DirectoryInfo 类的 GetFileSystemInfos 方法生成一个 FileSystemInfo 类型的数组，用来记录指定文件夹中的所有子文件夹及文件；最后，循环访问 FileSystemInfo 数组中的文件，并调用 FileInfo 类的 Delete 方法将遍历到的文件一一删除。下面对本实例中用到的关键技术进行详细讲解。

（1）DirectoryInfo 类的 GetFileSystemInfos 方法

该方法用于检索表示当前文件夹的文件和子文件夹的强类型 FileSystemInfo 对象的数组，它有两种重载形式，本实例中用到的重载形式如下：

```
public FileSystemInfo[] GetFileSystemInfos()
```

参数说明

返回值：强类型 FileSystemInfo 项的数组。

（2）FileInfo 类的 Delete 方法

该方法用于永久删除文件。语法格式如下：

```
public override void Delete()
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DeleteDirByDG。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的文件夹；添加两个 Button 控件，分别用来执行选择文件夹操作和使用递归法删除文件夹中所有文件的操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text);                    //创建 DirectoryInfo 对象
    FileSystemInfo[] FSInfo = DInfo.GetFileSystemInfos();                      //获取所有文件
    for (int i = 0; i < FSInfo.Length; i++)                                    //遍历获取到的文件
    {
        FileInfo FInfo = new FileInfo(textBox1.Text + "\\" + FSInfo[i].ToString());   //创建 FileInfo 对象
        FInfo.Delete();                                                        //删除文件
    }
    MessageBox.Show("删除成功", "信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 397：FileSystemInfo 的使用场合。

FileSystemInfo 类包含文件和文件夹操作所共有的方法。FileSystemInfo 对象可以表示文件或文件夹，从而可以作为 FileInfo 或 DirectoryInfo 对象的基础。当在程序中分析许多文件和文件夹时，请使用 FileSystemInfo 类。

实例 398	对指定文件夹中的文件进行分类存储	中级
	光盘位置：光盘\MR\16\398	趣味指数：★★★★☆

实例说明

当一个文件夹中有很多种类型的文件时，查找起来非常不方便，但这时如果将各种类型的文件进行分类存储（如将 txt 类型的文件放在一个文件夹中，将 doc 类型的文件放在另一个文件夹中），则会显得非常方便。本实例使用 C#实现了对指定文件夹中的文件进行分类存储的功能。运行本实例，如图 16.14 所示，单击“选择”按钮，选择要整理的文件夹；单击“整理”按钮，对选择的文件夹中的文件进行分类存储；单击“查看”按钮，可以查看整理后的文件夹，如图 16.15 所示。

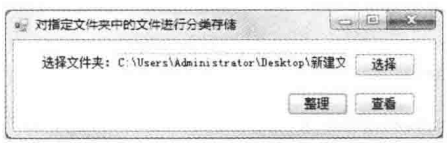


图 16.14　对指定文件夹中的文件进行分类存储

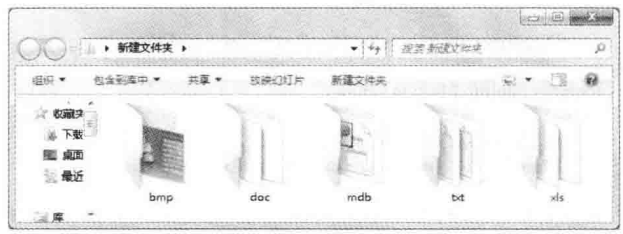


图 16.15　查看整理后的文件夹

关键技术

本实例中首先使用 DirectoryInfo 类的 GetFiles 方法获取指定文件夹中的所有文件，并遍历这些文件，将这些文件的扩展名添加到一个 List 泛型集合中；然后遍历 List 泛型集合，根据其中存储的扩展名类型，使用 Directory 类的 CreateDirectory 方法创建相应的文件夹；最后再次遍历获取的所有文件，并使用 FileInfo 类的 MoveTo 方法将文件移动到对应的文件夹中，从而实现文件的分类存储功能。下面对本实例用到的关键技术进行详细讲解。

（1）DirectoryInfo 类的 GetFiles 方法

该方法用来返回当前文件夹的文件列表，它有 3 种重载形式，本实例中用到的重载形式如下：

```
public FileInfo[] GetFiles()
```

参数说明

返回值：FileInfo 类型数组。

（2）Directory 类的 CreateDirectory 方法

该方法用来创建指定路径中的所有文件夹，它有两种重载形式，本实例中用到的重载形式如下：

```
public static DirectoryInfo CreateDirectory(string path)
```

参数说明

❶ path：要创建的文件夹路径。

❷ 返回值：由 path 指定的 DirectoryInfo 对象。

（3）FileInfo 类的 MoveTo 方法

该方法将指定文件移动到新位置，并提供指定新文件名的选项。语法格式如下：

```
public void MoveTo(string destFileName)
```

参数说明

destFileName：要将文件移动到的路径，可以指定另一个文件名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ManageFileByType。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的文件夹；添加 3 个 Button 控件，分别用来执行选择文件夹、整理文件夹和查看整理后的文件夹的操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    List<string> listExten = new List<string>();                          //创建泛型集合对象
    DirectoryInfo DInfo = new DirectoryInfo(textBox1.Text);               //创建 DirectoryInfo 对象
    FileInfo[] FInfos = DInfo.GetFiles();                                 //获取文件夹中的所有文件
    string strExten = "";                                                 //定义一个变量，用来存储文件扩展名
    foreach (FileInfo FInfo in FInfos)                                    //遍历所有文件
    {
        strExten = FInfo.Extension;                                       //记录文件扩展名
        if (!listExten.Contains(strExten))                                //判断泛型集合中是否已经存在该扩展名
        {
            listExten.Add(strExten.TrimStart('.'));                       //将扩展名去掉之后添加到泛型集合中
        }
    }
    for (int i = 0; i < listExten.Count; i++)                             //遍历泛型集合
    {
        Directory.CreateDirectory(textBox1.Text + listExten[i]);          //创建文件夹
    }
    foreach (FileInfo FInfo in FInfos)                                    //遍历所有文件
     {
        //将文件移动到对应的文件夹中
        FInfo.MoveTo(textBox1.Text + FInfo.Extension.TrimStart('.') + "\\" + FInfo.Name);
    }
    MessageBox.Show("整理完毕！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 398：List<T>泛型集合的使用。

List<T>泛型集合表示可通过索引访问的对象的强类型列表，它是 ArrayList 类的泛型等效类，提供用于对列表进行搜索、排序和操作的方法。

实例 399　伪装文件夹

光盘位置：光盘\MR\16\399　　高级　　趣味指数：★★★★★

实例说明

出于安全性的考虑，现在许多用户喜欢通过一些软件对文件夹进行加密。但是，这样加密后真的安全吗？笔者认为加密的最高境界应该是让人感觉文件夹并没有被加密，这样自然也就不会有人对它实施破解了，安全系数才会真正地大幅度提高。所以，开发本实例用于对文件夹进行伪装。例如，将文件夹伪装成回收站，那么当双击伪装后的文件夹，会进入回收站，而不是文件夹。实例运行效果如图 16.16 所示。

图 16.16　伪装文件夹

关键技术

本实例中，首先通过 File 类的 CreateText 方法在文件夹中创建 desktop.ini 文件，然后通过 StreamWriter 对象的 WriteLine 方法向该文件中写入 Windows 文件标识符，从而达到伪装文件夹的目的，下面对本实例中用到的关键技术进行详细讲解。

（1）File 类的 CreateText 方法

该方法创建或打开一个文件用于写入 UTF-8 编码的文本。语法格式如下：

```
public static StreamWriter CreateText(string path)
```

参数说明

❶ path：要打开以进行写入的文件。

❷ 返回值：一个 StreamWriter，它使用 UTF-8 编码写入指定的文件。

（2）StreamWriter 对象的 WriteLine 方法

该方法将字符串写入文本流。语法格式如下：

```
public virtual void WriteLine(string value)
```

参数说明

value：要写入的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CamouflageFolder。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 FolderBrowserDialog 控件，用来选择要伪装或还原的文件夹；添加两个 TextBox 控件，分别用来显示选择的文件夹和输入类标识符；添加一个 ComboBox 控件，用来选择要伪装的类型；添加 3 个 Button 控件，分别用来执行打开浏览文件夹对话框、伪装文件夹和还原文件夹操作。

（3）程序的主要代码如下。

Frm_Main 窗体的后台代码中，首先自定义一个 GetFolType 方法，用于根据选择的伪装方式获取其对应的 Windows 文件标识符。代码如下：

```
private string GetFolType()
{
    int Tid = comboBox1.SelectedIndex;                                  //获取选择项索引
    switch (Tid)                                                        //根据索引设置 Windows 文件标识符
    {
        case 0: return @"{20D04FE0-3AEA-1069-A2D8-08002B30309D}";       //我的电脑 Windows 文件标识符
        case 1: return @"{450D8FBA-AD25-11D0-98A8-0800361B1103}";       //我的文档 Windows 文件标识符
        case 2: return @"{992CFFA0-F557-101A-88EC-00DD010CCC48}";       //拨号网络 Windows 文件标识符
        case 3: return @"{21EC2020-3AEA-1069-A2DD-08002B30309D}";       //控制面板 Windows 文件标识符
        case 4: return @"{D6277990-4C6A-11CF-8D87-00AA0060F5BF}";       //计划任务 Windows 文件标识符
        case 5: return @"{2227A280-3AEA-1069-A2DE-08002B30309D}";       //打印机 Windows 文件标识符
        case 6: return @"{208D2C60-3AEA-1069-A2D7-08002B30309D}";       //网络邻居 Windows 文件标识符
        case 7: return @"{645FF040-5081-101B-9F08-00AA002F954E}";       //回收站 Windows 文件标识符
        case 8: return @"{85BBD920-42A0-1069-A2E4-08002B30309D}";       //公文包 Windows 文件标识符
        case 9: return @"{BD84B380-8CA2-1069-AB1D-08000948F534}";       //字体 Windows 文件标识符
        case 10: return @"{BDEADF00-C265-11d0-BCED-00A0C90AB50F}";      //Web 文件夹 Windows 文件标识符
    }
    //如果都不符合则返回我的电脑 Windows 文件标识符
    return @"{20D04FE0-3AEA-1069-A2D8-08002B30309D}";
}
```

自定义一个 Camouflage 方法，用于在指定的文件夹下创建一个名为 desktop.ini 的文件，在此文件中写入伪装类型的 Windows 文件标识符。例如，如果想将文件夹伪装成“我的电脑”，那么在调用此方法后，desktop.ini 文件中就被写入“我的电脑”的 Windows 文件标识符“CLSID={20D04FE0-3AEA-1069-A2D8-08002B30309D}”。Camouflage 方法的实现代码如下：

```
private void Camouflage(string str)                                     //用于创建 desktop.ini 文件
{
```

```
    //用 desktop.ini 文件创建 StreamWriter 对象
    StreamWriter sw = File.CreateText(txtFolPath.Text.Trim() + @"\desktop.ini");
    sw.WriteLine(@"[.ShellClassInfo]");                                    //写入“[.ShellClassInfo]”
    sw.WriteLine("CLSID=" + str);                                          //写入 Windows 文件标识符
    sw.Close();                                                            //关闭对象
    //设置 desktop.ini 文件为隐藏
    File.SetAttributes(txtFolPath.Text.Trim() + @"\desktop.ini", FileAttributes.Hidden);
    File.SetAttributes(txtFolPath.Text.Trim(), FileAttributes.System);     //设置文件夹属性为系统属性
    MessageBox.Show("伪装成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

程序中提供了预置伪装类型和自定义伪装类型两种，如果选择自定义伪装类型，需要手动输入要伪装类型的 Windows 文件标识符，然后再调用 Camouflage 方法进行伪装。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (this.txtFolPath.Text == "")                                        //如果没有选择文件
    {
        MessageBox.Show("请选择文件夹路径！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else
    {
        try
        {
            if (txtID.ReadOnly == false)                                   //如果自定义 Windows 文件标识符
            {
                string str = txtID.Text.Trim();                            //获取选择的 Windows 文件标识符
                if (str.StartsWith("."))                                   //如果以“.”开头
                    str = str.Substring(1);                                //则去掉“.”
                if (!str.StartsWith("{") || str.Trim().Length != 38)       //如果不以“{”开头，并且长度错误
                {
                    MessageBox.Show("自定义类型错误！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else                                                       //如果 Windows 文件标识符合法
                {
                    Camouflage(str);                                       //调用 Camouflage 方法开始伪装
                }
            }
            else                                                           //如果选择预定的伪装方式
            {
                //通过 GetFolType 方法获取 Windows 文件标识符，然后使用 Camouflage 方法开始伪装
                Camouflage(GetFolType());
            }
        }
        catch
        {
            MessageBox.Show ("不要进行多次伪装！","提示",MessageBoxButtons.OK,MessageBoxIcon.Error);
        }
    }
}
```

如果想还原伪装过的文件夹，那么首先需要选择伪装过的文件夹，然后单击“还原”按钮去除伪装，其实现原理是删除文件夹下的 desktop.ini 文件，这样文件夹就还原到原始状态了。“还原”按钮的 Click 事件代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    if (txtFolPath.Text == "")                                             //判断是否选择了要还原伪装的文件夹
    {
        MessageBox.Show("请选择加密过的文件夹！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.Error);
    }
    else                                                                   //如果选择了文件夹
    {
        try
        {
            FileInfo fi = new FileInfo(txtFolPath.Text.Trim() + @"\desktop.ini");   //创建 FileInfo 对象
            if (!fi.Exists)                                                //如果不存在 desktop.ini 文件
            {
                MessageBox.Show("该文件夹没有被伪装！", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
```

```
                else
                {
                    System.Threading.Thread.Sleep(1000);                                    //休眠线程
                    File.Delete(txtFolPath.Text + @"\desktop.ini");                         //删除文件夹下的 desktop.ini 文件
                    File.SetAttributes(txtFolPath.Text.Trim(), FileAttributes.System);      //设置文件夹属性为系统属性
                    MessageBox.Show("还原成功", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            catch
            {
                MessageBox.Show("不要多次还原！");
            }
        }
    }
```

秘笈心法

心法领悟 399：Windows 文件标识符的作用。

就像每个公民都有唯一的身份证号码一样，在 Windows 操作系统中，每个系统级别的应用程序（如“我的电脑”“回收站”和“计划任务”等）也都用唯一的标识符来进行管理，当双击某个文件夹时（如双击“计划任务”），操作系统会首先检查该文件夹的文件名，并到注册表中去搜索该标识符所注册的应用程序类型，最后再打开相应的应用程序或使用这个应用程序打开该文件，那么在操作系统与真实文件夹之间起到承接作用的这些数字就被称为“Windows 文件标识符”，英文名称为“CLSID”，它们被保存在注册表中的“HKEY_LOCAL_MACHINE\Software\Classes\CLSID”键值下，通常由 32 个十六进制数构成。

第17章

文件流操作

- 文件的读取和写入
- 常用的文件流操作

17.1　文件的读取和写入

实例 400	创建一个文件用于写入 UTF-8 编码的文本	初级
	光盘位置：光盘\MR\17\400	趣味指数：★★★★☆

实例说明

本实例实现创建一个 UTF-8 编码格式的文件（以创建文本文件为例），首先在窗体中设置文件路径，然后输入要写入文本文件的内容，最后单击“创建”按钮实现将文本内容以 UTF-8 编码格式写入文本文件。实例运行效果如图 17.1 所示。

图 17.1　创建一个文件用于写入 UTF-8 编码的文本

关键技术

本实例主要用到了 File 类的 CreateText 方法和 StreamWriter 类的 WriteLine 方法，下面分别对它们进行详细讲解。

（1）File 类的 CreateText 方法

该方法实现创建或打开一个文件用于写入 UTF-8 编码的文本。语法格式如下：

```
public static StreamWriter CreateText(string path)
```

参数说明

❶ path：要打开以进行写入的文件。

❷ 返回值：一个 System.IO.StreamWriter，它使用 UTF-8 编码写入指定的文件。

（2）StreamWriter 类的 WriteLine 方法

该方法实现将后跟行结束符的字符串写入文本流。该方法的重载形式有多种，本实例用到的语法格式如下：

```
public virtual void WriteLine(string value)
```

参数说明

value：要写入的字符串。如果 value 为 null，则仅写入行结束字符。

说明：本实例创建的文件为 UTF-8 编码格式的文本文件（*.txt）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WriteUTF8File。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来设置文件路径和输入要写入文件的内容；添加两个 Button 控件，分别用来设置文件路径和创建并写入文件内容。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text.Trim()))                    //若文件路径为空
    {
        MessageBox.Show("请设置文件路径！");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text.Trim()))                    //若文本内容为空
    {
        MessageBox.Show("请输入文件内容！");
        return;
    }
    if (!File.Exists(textBox1.Text))
    {
        using (StreamWriter sw = File.CreateText(textBox1.Text))       //创建用于写入 UTF-8 编码的文本
```

```
            {
                sw.WriteLine(textBox2.Text);                    //把字符串写入文本流
                MessageBox.Show("文件创建成功", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        else
        {
            MessageBox.Show("该文件已经存在", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
```

秘笈心法

心法领悟 400：using 语句的使用。

使用 using 语句创建实现了 IDisposable 接口的类，当 using 语句块结束后，程序会自动调用 IDisposable 接口的 Dispose 方法来销毁创建的实例。本实例代码中的 StreamWriter 类实现了 IDisposable 接口，所以使用 using 语句对其进行创建。

实例 401　OpenRead 方法打开现有文件并读取

光盘位置：光盘\MR\17\401　　初级　　趣味指数：★★★☆

实例说明

本实例使用静态类 File 的 OpenRead 方法打开一个文本文件，首先选择要打开的文本文件，然后程序将该文本文件的文本内容显示在窗体下方的 TextBox 控件中。实例运行效果如图 17.2 所示。

图 17.2　OpenRead 方法打开现有文件并读取

关键技术

本实例主要用到了 File 类的 OpenRead 方法和 FileStream 类的 Read 方法，下面分别对它们进行详细讲解。

（1）File 类的 OpenRead 方法

该方法实现打开现有文件以进行读取。语法格式如下：

```
public static FileStream OpenRead(string path)
```

参数说明

❶ path：要打开以进行读取的文件。

❷ 返回值：指定路径上的只读文件流。

（2）FileStream 类的 Read 方法

该方法实现从流中读取字节块并将该数据写入给定的缓冲区中。语法格式如下：

```
public override int Read(byte[] array, int offset, int count)
```

Read 方法中的参数说明如表 17.1 所示。

表 17.1　Read 方法中的参数说明

参　数	说　明
array	写入数据的缓冲区
offset	读取的开始位置索引
count	最多读取的字节数
返回值	读入 buffer 中的总字节数。如果当前的字节数没有所请求那么多，则总字节数可能小于所请求的字节数；或者如果已到达流的末尾，则为零

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件的内容；添加一个 Button 控件，用来选择指定的文件。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.Filter = "文本文件(*.txt)|*.txt";          //设置打开文件的类型
        openFileDialog1.ShowDialog();
        textBox1.Text = openFileDialog1.FileName;                  //设置打开的文件名称
        FileStream fs = File.OpenRead(textBox1.Text);              //打开现有文件以进行读取
        byte[] b = new byte[1024];                                 //定义缓存
        while (fs.Read(b, 0, b.Length) > 0)                        //每次读取 1024 个字节到缓存中
        {
            textBox2.Text = Encoding.Default.GetString(b);         //把字节数组中的所有字节转为一个字符串
        }
    }
    catch { MessageBox.Show("请选择文件"); }
}
```

秘笈心法

心法领悟 401：使用 Encoding 类的 GetString 方法。

在使用各种流读取或写入文件时，经常需要将字节解码为对应的字符串，可以考虑使用 Encoding 类的 GetString 方法，该方法可以实现将字节数组解码为对应的字符串。

实例 402　OpenWrite 方法打开现有文件并进行写入

初级

光盘位置：光盘\MR\17\402

趣味指数：★★★★☆

实例说明

本实例使用静态类 File 的 OpenWrite 方法打开现有文件并进行写入，首先选择一个文本文件，然后在窗体下方的 TextBox 控件中输入要写入该文本文件的内容，最后单击“写入”按钮。实例运行效果如图 17.3 所示。

图 17.3　OpenWrite 方法打开现有文件并进行写入

关键技术

本实例主要用到了 File 类的 OpenWrite 方法、Encoding 抽象类的 GetBytes 方法和 FileStream 类的 Write 方法，下面分别对它们进行详细讲解。

（1）File 类的 OpenWrite 方法

该方法用于实现打开现有文件以进行写入。语法格式如下：

```
public static FileStream OpenWrite(string path)
```

参数说明

❶ path：要打开以进行写入的文件。

❷ 返回值：返回具有写入权限的指定路径上的非共享文件流对象。

（2）Encoding 抽象类的 GetBytes 方法

该方法用于实现将指定的字符串中的所有字符编码为一个字节序列。其重载形式有多种，本实例用到的方法如下：

```
public virtual byte[] GetBytes(string s)
```

参数说明

❶ s：包含要编码的字符的字符串。

❷ 返回值：返回一个字节数组，包含对指定的字符集进行编码的结果。

（3）FileStream 类的 Write 方法

该方法实现使用从缓冲区读取的数据将字节块写入该流。语法格式如下：

```
public override void Write(byte[] array, int offset, int count)
```

Write 方法中的参数说明如表 17.2 所示。

表 17.2 Write 方法中的参数说明

参　数	说　明
array	包含要写入该流的数据的缓冲区
offset	位置索引，从此处开始将字节复制到当前流
count	要写入当前流的最大字节数
返回值	无返回值

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WriteFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和输入要写入文件的内容；添加两个 Button 控件，分别打开指定文件及向该文件中写入指定的文本。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text.Trim()))                    //若指定文件名称为空
    {
        MessageBox.Show("请设置文件！");
        return;
    }
    try
    {
        FileStream FStream = File.OpenWrite(textBox1.Text);            //通过指定文件获取文件流
        Byte[] info = Encoding.Default.GetBytes(textBox2.Text);        //把字符串转换为字节数组
        FStream.Write(info, 0, info.Length);                           //向文件流中写入字节数组
        FStream.Close();
        MessageBox.Show("写入文件成功！");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

秘笈心法

心法领悟 402：调用流的 Close 方法。

在使用完毕各种类型流的实例后，需要调用 Close 方法关闭当前流并释放与之关联的所有资源。

实例 403　打开现有 UTF-8 编码文本文件并进行读取

光盘位置：光盘\MR\17\403　　初级　　趣味指数：★★★☆

实例说明

本实例实现打开 UTF-8 编码格式的文本文件并进行读取，首先选择 UTF-8 编码格式的文本文件，然后程序将该文本文件的内容显示在窗体下方的 TextBox 控件中。实例运行效果如图 17.4 所示。

图 17.4　打开现有 UTF-8 编码文本文件并进行读取

关键技术

本实例主要用到了 File 类的 OpenText 方法和 StreamReader 类的 ReadToEnd 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）File 类的 OpenText 方法

该方法实现打开现有 UTF-8 编码文本文件以进行读取。语法格式如下：

```
public static StreamReader OpenText(string path)
```

参数说明

❶ path：要打开以进行读取的文件。

❷ 返回值：返回指定路径上的 StreamReader 实例。

（2）StreamReader 类的 ReadToEnd 方法

该方法实现从流的当前位置到末尾读取流。语法格式如下：

```
public override string ReadToEnd()
```

参数说明

返回值：返回字符串形式的流的其余部分（从当前位置到末尾）。如果当前位置位于流的末尾，则返回空字符串（""）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadUTF8File。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件的内容；添加一个 Button 控件，用来打开指定的 UTF-8 编码格式的文件。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.Filter = "文本文件(*.txt)|*.txt";          //设置打开的文件类型
        openFileDialog1.ShowDialog();                              //显示对话框
        textBox1.Text = openFileDialog1.FileName;                  //显示文件名及路径
        StreamReader SReader = File.OpenText(textBox1.Text);       //打开 UTF-8 编码文本文件
        textBox2.Text = SReader.ReadToEnd();                       //读取文件的内容
    }
    catch { }
}
```

秘笈心法

心法领悟 403：应用 StreamReader 类读取文本文件。

StreamReader 类实现了 TextReader 抽象类，常用来读取文本文件。

实例 404 读取文件中的第一行数据

光盘位置：光盘\MR\17\404

初级

趣味指数：★★★☆

实例说明

本实例实现读取文件中的第一行数据，首先选择要读取的文件，然后程序将该文件的第一行数据显示在窗体下方的文本框中。实例运行效果如图 17.5 所示。

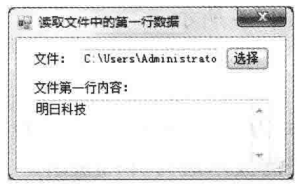


图 17.5 读取文件中的第一行数据

关键技术

本实例主要用到了 StreamReader 类的 ReadLine 方法，该方法实现从当前流中读取一行字符并将数据作为字符串返回。语法格式如下：

```
public override string ReadLine()
```

参数说明

返回值：返回当前流中的第一行数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadFirstLine。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件的第一行数据。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.Filter = "文本文件(*.txt)|*.txt";                    //打开的文件类型
        openFileDialog1.ShowDialog();                                        //显示对话框
        textBox1.Text = openFileDialog1.FileName;                            //显示文件名及路径
        StreamReader SReader = new StreamReader(textBox1.Text, Encoding.Default);  //创建 StreamReader 对象
        textBox2.Text = SReader.ReadLine();                                  //读取文件中的第一行数据
    }
    catch { }
}
```

秘笈心法

心法领悟 404：应用 Encoding.Default 属性。

Encoding.Default 属性在使用流技术处理文件时非常重要，它可实现获取系统当前的 ANSI 代码页的编码。

实例 405 按行读取文本文件中的数据

光盘位置：光盘\MR\17\405

中级

趣味指数：★★★★

实例说明

本实例实现按行读取文本文件中的所有数据，首先选择要读取的文本文件，然后程序将按行读取该文件的

全部数据，并将读取的数据显示在窗体下方的文本框中。实例运行效果如图 17.6 所示。

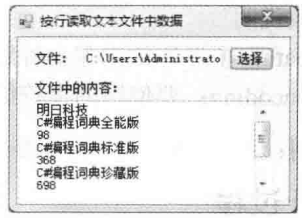


图 17.6　按行读取文本文件中的数据

关键技术

本实例主要使用 StreamReader 类的 ReadLine 方法来实现按行读取文本文件中的数据的功能。

说明： 关于 StreamReader 类的 ReadLine 方法的详细介绍请参见实例 404 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadFileByLine。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件中的全部数据。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.Filter = "文本文件(*.txt)|*.txt";                          //设置选择的文件类型
        openFileDialog1.ShowDialog();                                              //打开对话框
        textBox1.Text = openFileDialog1.FileName;                                  //设置文件路径
        StreamReader SReader = new StreamReader(textBox1.Text, Encoding.Default);  //创建 StreamReader 对象
        string strLine = string.Empty;
        while ((strLine = SReader.ReadLine()) != null)                             //逐行读取文本文件
        {
            textBox2.Text += strLine + Environment.NewLine;                        //在文本框中显示读取内容
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 405：如何使文本换行？

在文本框中输出多行文本时，可以使用静态类 Environment 的 NewLine 属性实现文本的换行显示。

实例 406　读取文件中的所有数据

光盘位置：光盘\MR\17\406　　初级　　趣味指数：★★★☆

实例说明

本实例使用 StreamReader 类的 ReadToEnd 方法实现读取文件中的全部数据，首先选择要读取的文件，然后程序将该文件的全部数据显示在窗体下方的 TextBox 控件中。实例运行效果如图 17.7 所示。

图 17.7　读取文件中的所有数据

关键技术

本实例主要用到了 StreamReader 类的构造方法和 ReadToEnd 方法，下面对本实例中用到的关键技术进行详细讲解。

StreamReader 类的构造方法实现用指定的字符编码，为指定的文件名初始化 StreamReader 类的一个新实例。语法格式如下：

```
public StreamReader(string path, Encoding encoding)
```

参数说明

❶ path：要读取的完整文件路径。

❷ encoding：要使用的字符编码。

说明：关于 StreamReader 类的 ReadToEnd 方法的详细介绍请参见实例 403 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadAllData。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件的全部内容；添加一个 Button 控件，用来选择要读取的文件。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        openFileDialog1.Filter = "文本文件(*.txt)|*.txt";                                    //设置打开的文件类型
        openFileDialog1.ShowDialog();                                                        //显示对话框
        textBox1.Text = openFileDialog1.FileName;                                            //显示文件路径
        StreamReader SReader = new StreamReader(textBox1.Text, Encoding.Default);            //创建 StreamReader 对象
        textBox2.Text = SReader.ReadToEnd();                                                 //读取全部数据
    }
    catch { MessageBox.Show("请选择文件"); }
}
```

秘笈心法

心法领悟 406：区分 ReadLine 方法和 ReadToEnd 方法。

StreamReader 类的 ReadLine 方法用于从流中读取第一行字符，而 ReadToEnd 方法用于从流的当前位置到末尾读取流。

实例 407 向文件中写入追加数据

初级

光盘位置：光盘\MR\17\407

趣味指数：★★★★☆

实例说明

本实例使用 StreamWriter 类的 Write 方法实现向文件中写入数据，首先选择要写入数据的文件，然后在窗体下方的 TextBox 控件中输入要写入的内容，最后单击“写入”按钮写入数据。实例运行效果如图 17.8 所示。

图 17.8 向文件中写入追加数据

关键技术

本实例主要用到了 StreamWriter 类的构造方法和 Write 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）StreamWriter 类的构造方法

该方法实现使用默认编码和缓冲区大小，为指定路径上的指定文件初始化 StreamWriter 类的新实例。语法

格式如下：

```
public StreamWriter(string path)
```

参数说明

path：要向其中写入的完整文件路径。

（2）StreamWriter 类的 Write 方法

该方法实现将字符串写入流，它的重载形式有多种。语法格式如下：

```
public override void Write(string value)
```

参数说明

value：要写入流的字符串。如果 value 为空，则不写入任何内容。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddDataToFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和输入要写入文件的内容；添加两个 Button 控件，分别用来打开指定文件和向该文件中写入内容。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text.Trim()))                    //若未选择文件
    {
        MessageBox.Show("请设置文件");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text.Trim()))                    //若未输入要写入文件的内容
    {
        MessageBox.Show("请输入要写入的文件内容");
        return;
    }
    try
    {
        StreamWriter SWriter = new StreamWriter(textBox1.Text);        //创建 StreamWriter 对象
        SWriter.Write(textBox2.Text);                                  //向文件中写入指定的内容
        SWriter.Close();                                               //关闭 StreamWriter 对象和基础流
        MessageBox.Show("写入文件成功！");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

秘笈心法

心法领悟 407：使用 StreamWriter 类写文本文件。

StreamWriter 类实现了 TextWriter 抽象类，常用来向文本文件中写入数据。

17.2　常用的文件流操作

实例 408	将文本文件转换成网页文件	中级
	光盘位置：光盘\MR\17\408	趣味指数：★★★★☆

实例说明

在“东方网页王”和 Microsoft Word 等软件中，用户只要在其中添加一些文字和图片等元素，就会自动生成

一个网页。实际上，生成网页的技术并不难，因为网页文件本质上就是带有 HTML 控制符号的纯文本文件，用户只要在文本文件的前后加上 HTML 语言标识符，并保存为.htm 文件即可。实例运行效果如图 17.9 所示。

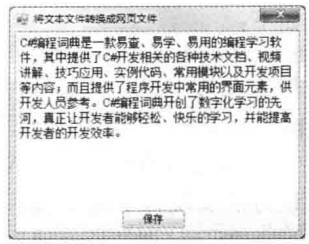


图 17.9　将文本文件转换成网页文件

关键技术

本实例实现将 RichTextBox 控件中的文本转换为网页格式的文本，主要用到了 RichTextBox 控件的 AppendText 和 SaveFile 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）RichTextBox 控件的 AppendText 方法

该方法实现向文本框的当前文本追加文本。语法格式如下：

```
public void AppendText(string text)
```

参数说明

text：要向文本框的当前内容追加的文本。

（2）RichTextBox 控件的 SaveFile 方法

该方法实现将 RichTextBox 中的内容保存到特定类型的文件中。语法格式如下：

```
public void SaveFile(string path, RichTextBoxStreamType fileType)
```

参数说明

❶ path：要保存的文件的名称和位置。

❷ fileType：RichTextBoxStreamType 类型的枚举值之一，其枚举值及说明如表 17.3 所示。

表 17.3　RichTextBoxStreamType 枚举值及说明

枚 举 值	说　明
RichText	RTF 格式流
PlainText	用空格代替对象链接与嵌入（OLE）对象的纯文本流
RichNoOleObjs	用空格代替 OLE 对象的丰富文本格式（RTF 格式）流
TextTextOleObjs	具有 OLE 对象的文本表示形式的纯文本流
UnicodePlainText	包含用空格代替对象链接与嵌入（OLE）对象的文本流

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ConvertTxtToWeb。

（2）在默认窗体 Form1 中添加一个 RichTextBox 控件，用来显示网页的文本和 HTML 语言标识符；添加一个 Button 控件，用来生成网页。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        //设置网页显示的文本
        string strContent = "明日科技有限公司是一家以计算机软件技术为核心的高科技企业，多年来始终致力于行业管理软件开发、数字化出版物制作、计算机网络系统综合应用以及行业电子商务网站开发领域，涉及生产、管理、控制、仓储、物流、营销、服务等行业。公司拥有软件开发和项目实施方面的资深专家和学习型技术团队，多年来积累了丰富的技术文档和学习资料，公司的开发团队不仅是开拓进取的技术实践者，更致力于成为技术的普及和传播者。";
        string strCompany = "吉林省明日科技有限公司";
        string strWeb = "www.mingrisoft.com";
        string strFileName = "公司网页.htm";
        richTextBox1.Clear();
        richTextBox1.AppendText("<HTML>");                    //添加网页的 HTML 语言标识符
        richTextBox1.AppendText("<HEAD>");                    //设置网页的 HEAD 元素
        richTextBox1.AppendText("<TITLE>");                   //设置网页的 TITLE 元素
        richTextBox1.AppendText(strCompany);                  //添加网页的标题内容
        richTextBox1.AppendText("</TITLE>");
```

```
            richTextBox1.AppendText("</HEAD>");
            richTextBox1.AppendText("<BODY BGCOLOR='TAN'>");
            richTextBox1.AppendText("<CENTER>");                                    //设置文本居中的标记
            richTextBox1.AppendText("<H2>" + strCompany + "</H2>");
            String strHyper = "<H4><A HREF='" + strWeb + "'>欢迎访问明日科技公司网站：";
            richTextBox1.AppendText(strHyper + strWeb + "</A></H4>");
            richTextBox1.AppendText("</CENTER>");
            richTextBox1.AppendText(strContent);
            richTextBox1.AppendText("</BODY>");
            richTextBox1.AppendText("</HTML>");
            richTextBox1.SaveFile(strFileName, RichTextBoxStreamType.PlainText);    //保存文本框的内容为纯文本流
            System.Diagnostics.Process.Start(strFileName);                          //通过文档名启动进程资源
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
```

秘笈心法

心法领悟 408：通过指定文档来启动进程资源。

调用 Process.Start 方法可实现通过指定文档来启动进程资源，并将资源与新的 Process 组件关联。

实例 409　读写内存流数据

光盘位置：光盘\MR\17\409　　初级　　趣味指数：★★★★

实例说明

本实例使用 MemoryStream 实现读写内存流，首先在窗体最上端的文本框中输入字符串，然后单击“写入并读取”按钮，程序会将内存流的相关属性和内存流中的数据显示在窗体中间的 RichTextBox 控件中。实例运行效果如图 17.10 所示。

图 17.10　读写内存流数据

关键技术

本实例主要用到了 MemoryStream 类的 Capacity 属性、Write 方法和 Read 方法，下面分别对它们进行详细讲解。

（1）MemoryStream 类的 Capacity 属性

该属性用于或设置分配给该流的字节数。语法格式如下：

```
public virtual int Capacity { get; set; }
```

参数说明

返回值：流的缓冲区可使用部分的长度。

（2）MemoryStream 类的 Write 方法

该方法实现使用从缓冲区读取的数据将字节块写入当前流。语法格式如下：

```
public override void Write(byte[] buffer, int offset, int count)
```

Write 方法中的参数说明如表 17.4 所示。

表 17.4　Write 方法中的参数说明

参　数	说　明	参　数	说　明
buffer	从中写入数据的缓冲区	count	最多写入的字节数
offset	位置索引，从此处开始写入	返回值	无返回值

（3）MemoryStream 类的 Read 方法

该方法实现从当前流中读取字节块并将数据写入 buffer 中。语法格式如下：

```
public override int Read(byte[] buffer, int offset, int count)
```

Read 方法中的参数说明如表 17.5 所示。

表 17.5 Read 方法中的参数说明

参　数	说　明
buffer	写入数据的缓冲区
offset	位置索引，从此处开始读取
count	最多读取的字节数
返回值	写入缓冲区中的总字节数。如果当前可用字节数不到所请求的字节数，则这一总字节数可能小于所请求的字节数，或者如果在读取任何字节前已到达流的末尾，则为零

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OperateMemoryStream。

（2）在默认窗体 Form1 中添加两个 RichTextBox 控件，分别用来显示内存流属性和读取的内存流内容；添加一个 TextBox 控件，用来输入写入内存流的文本；添加一个 Button 控件，用来实现写入并读取内存流。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    byte[] BContent = Encoding.Default.GetBytes(textBox1.Text);                    //把字符串转换为字节数组
    MemoryStream MStream = new MemoryStream(100);                                  //初始化容量为 100 个字节的内存流
    MStream.Write(BContent, 0, BContent.Length);                                   //把从缓冲区读取的字节写入内存流
    //获取内存流的属性，包括容量大小、流的长度、流中的当前位置
    richTextBox1.Text = "分配给该流的字节数：" + MStream.Capacity.ToString() + "\n 流长度："
        + MStream.Length.ToString() + "\n 流的当前位置：" + MStream.Position.ToString();
    MStream.Seek(0, SeekOrigin.Begin);                                             //将当前流中的位置设置为流的开头
    byte[] byteArray = new byte[MStream.Length];                                   //根据流的长度创建缓冲区
    int count = MStream.Read(byteArray, 0, (int)MStream.Length - 1);               //从当前流中读取字节块并将数据写入缓冲区中
    while (count < MStream.Length)                                                 //当缓冲区中的字节数小于流长度
    {
        byteArray[count++] = Convert.ToByte(MStream.ReadByte());                   //使用流中的数据初始化缓冲区
    }
    //根据对缓冲区进行解码所产生的字符个数创建字符数组
    char[] charArray = new char[Encoding.Default.GetCharCount(byteArray, 0, count)];
    Encoding.Default.GetChars(byteArray, 0, count, charArray, 0);                  //将缓冲区中的字节序列解码为指定的字符数组
    for (int i = 0; i < charArray.Length; i++)                                     //遍历字符数组
    {
        richTextBox2.Text += charArray[i].ToString();                              //显示输出字符数组中的数据
    }
}
```

秘笈心法

心法领悟 409：使用 Encoding 类的 GetBytes 方法。

在进行内存流操作时，需要把字符串转换为字节，然后才可以进行流操作，这可以使用 Encoding 类的 GetBytes 方法，该方法实现将指定的字符串中的所有字符编码为一个字节序列，即字节数组。

实例 410　创建并写入二进制文件数据

中级

光盘位置：光盘\MR\17\410

趣味指数：★★★★

实例说明

本实例实现创建并写入二进制文件数据，首先选择要创建的文件的目录，然后设置文件的名称，最后单击

“创建”按钮。实例运行效果如图 17.11 所示。

图 17.11　创建并写入二进制文件数据

关键技术

本实例主要用到了 BinaryWriter 类的构造方法和 Write 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）BinaryWriter 类的构造法方法

该方法实现基于所提供的流，用 UTF-8 作为字符串编码来初始化 BinaryWriter 类的新实例。语法格式如下：

```
public BinaryWriter(Stream output)
```

参数说明

output：表示输出流。

（2）BinaryWriter 类的 Write 方法

该方法实现将 4 字节有符号整数写入当前流，并将流的位置提升 4 个字节。语法格式如下：

```
public virtual void Write(int value)
```

参数说明

value：要写入的 4 字节有符号整数。

注意：为了与其他文件类型进行区分，本实例中创建的文件均以 bin 为扩展名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WriteBinaryFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来设置文件路径和文件名称；添加两个 Button 控件，分别用来选择文件路径和创建二进制文件。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text.Trim()))                    //若文件路径为空
    {
        MessageBox.Show("请选择文件路径");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text.Trim()))                    //若文件名称为空
    {
        MessageBox.Show("请设置文件名称");
        return;
    }
    try
     {
        //创建文件流对象
        FileStream myStream = new FileStream(textBox1.Text + "\\" + textBox2.Text+".bin", FileMode.Create);
        BinaryWriter myWriter = new BinaryWriter(myStream);            //创建 BinaryWriter 对象以写入二进制文件
        for (int i = 0; i < 10; i++)//
        {
            myWriter.Write(i);                                         //将整数写入当前流
        }
        myWriter.Close();                                              //关闭当前资源和基础流
        myStream.Close();                                              //关闭文件流
        MessageBox.Show("创建并写入成功！");
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

秘笈心法

心法领悟 410：以创建模式创建 FileStream 对象。

通常使用 FileStream 类的构造方法来创建一个已知文件的文件流，若把构造方法的枚举参数值改为 FileMode.Create，则会以创建模式创建 FileStream 对象，这样程序将创建一个新文件。

实例 411　读取二进制文件中的内容

光盘位置：光盘\MR\17\411　　中级　　趣味指数：★★★★

实例说明

本实例实现读取二进制文件中的内容，首先选择要读取的二进制文件，然后程序将该二进制文件的内容输出到窗体下方的 TextBox 控件中。实例运行效果如图 17.12 所示。

图 17.12　读取二进制文件中的内容

关键技术

本实例主要用到了 BinaryWriter 类的构造方法和 ReadInt32 方法，下面对本实例中用到的关键技术进行详细讲解。

BinaryWriter 类的 ReadInt32 方法实现从当前流中读取 4 字节有符号整数，并使流的当前位置提升 4 个字节。声明语法如下：

```
public virtual int ReadInt32()
```

参数说明

返回值：从当前流中读取的 4 字节有符号整数。

说明：关于 BinaryWriter 类的详细介绍请参见实例 410 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ReadBinaryFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示文件路径和文件内容；添加两个 Button 控件，分别用来选择二进制文件和显示二进制文件的内容。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text.Trim()))                    //若文件路径为空
    {
        MessageBox.Show("请选择文件");
        return;
    }
    textBox2.Text = string.Empty;
    try
    {
        //根据指定文件路径创建文件流
        FileStream myStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);
        BinaryReader myReader = new BinaryReader(myStream);            //创建 BinaryReader 对象以读取二进制文件
        for (int i = 0; i < myStream.Length; i++)                      //遍历文件流
        {
            textBox2.Text += myReader.ReadInt32();                     //从当前流中读取 4 字节有符号整数
        }
        myReader.Close();                                              //关闭当前流
        myStream.Close();                                              //关闭文件流
    }
    catch { }
}
```

秘笈心法

心法领悟 411：创建只读文件流。

若要创建一个只读的文件流，则需要把 FileStream 构造方法的第二个枚举参数值设置为 FileMode.Open，另

外还需要把该构造方法的第三个枚举参数值设置为 FileAccess.Read。

实例 412　使用缓冲流复制文件

光盘位置：光盘\MR\17\412　　高级　　趣味指数：★★★★☆

实例说明

在使用缓冲流时，需要用到 System.IO 命名空间下的 BufferedStream 类。在本实例中，首先选择要复制的源文件，然后设置该文件被赋值的目标路径，最后单击“复制文件”按钮实现复制文件到指定的路径。实例运行效果如图 17.13 所示。

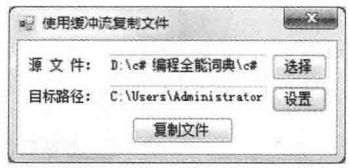


图 17.13　使用缓冲流复制文件

关键技术

BufferedStream 流，也称作缓冲流，它实现给另一个流上的读写操作添加一个缓冲层。缓冲流在内存中创建一个缓冲区，它会以最有效率的增量从磁盘中读取字节到缓冲区。缓冲区是内存中的字节块，用于缓存数据，从而减少对操作系统的调用次数，缓冲区可提高读取和写入性能。本实例主要用到了 BufferedStream 类的 Read、Write 和 Flush 方法，下面对本实例用到的关键技术进行详细讲解。

（1）BufferedStream 类的 Read 方法

该方法实现将字节从当前缓冲流复制到数组。语法格式如下：

```
public override int Read(byte[] array, int offset, int count)
```

Read 方法中的参数说明如表 17.6 所示。

表 17.6　Read 方法中的参数说明

参　数	说　明
array	将字节复制到缓冲区
offset	位置索引，从此处开始读取字节
count	要读取的字节数
返回值	读入 array 字节数组中的总字节数。如果可用的字节没有所请求的那么多，总字节数可能小于请求的字节数；或者如果在可读取任何数据前就已到达流的末尾，则为零

（2）BufferedStream 类的 Write 方法

该方法实现将字节复制到缓冲流，并将缓冲流内的当前位置向前移动与写入字节数相匹配的数。语法格式如下：

```
public override void Write(byte[] array, int offset, int count)
```

Write 方法中的参数说明如表 17.7 所示。

表 17.7　Write 方法中的参数说明

参　数	说　明
array	字节数组，从该字节数组将 count 个字节复制到当前缓冲流中
offset	缓冲区中的偏移量，从此处开始将字节复制到当前缓冲流中
count	要写入当前缓冲流中的字节数
返回值	无返回值

（3）BufferedStream 类的 Flush 方法

该方法实现清除该流的所有缓冲区，使得所有缓冲的数据都被写入到基础设备。语法格式如下：

```
public override void Flush()
```

技巧：由于缓冲流在内存的缓冲区中直接读写数据，而不是从磁盘中直接读写数据，所以它处理大容量的文件尤为适合。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OperateBufferedStream。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示源文件和目标文件路径；添加两个 Button 控件，分别用来选择源文件和设置目标路径。

（3）程序的主要代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        string str1 = textBox1.Text;                                    //获取包含路径的源文件
        string str2 = textBox2.Text + "\\" + textBox1.Text.Substring(textBox1.Text.LastIndexOf("\\") + 1, textBox1.Text.Length -
textBox1.Text.LastIndexOf("\\") - 1);                                   //获取包含路径的目标文件
        Stream myStream1, myStream2;                                    //声明两个 Stream 类型的引用
        BufferedStream myBStream1, myBStream2;                          //声明两个缓冲流的引用
        byte[] myByte = new byte[1024];                                 //创建容量为 1024 字节的缓冲区
        int i;
        myStream1 = File.OpenRead(str1);                                //以读取的方式打开源文件
        myStream2 = File.OpenWrite(str2);                               //以写入的方式打开目标文件
        myBStream1 = new BufferedStream(myStream1);                     //创建源文件缓冲流
        myBStream2 = new BufferedStream(myStream2);                     //创建目标文件缓冲流
        i = myBStream1.Read(myByte, 0, 1024);                           //把字节从当前缓冲流复制到数组
        while (i > 0)                                                   //若复制到数组中的字节数大于零
        {
            myBStream2.Write(myByte, 0, i);                             //把字节数组中的数据复制到缓冲流
            i = myBStream1.Read(myByte, 0, 1024);                       //把字节从当前缓冲流复制到数组
        }
        myBStream2.Flush();                                             //清除流的缓冲区
        myStream1.Close();                                              //关闭流，释放资源
        myBStream2.Close();
        MessageBox.Show("文件复制完成");
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

秘笈心法

心法领悟 412：使用缓冲流的 Flush 方法。

在缓冲流中写入数据后，需要调用该缓冲流的 Flush 方法清除该流的所有缓冲区，并使所有缓冲的数据都被写入到基础设备。

实例 413 解析只有一种格式的文本文件

初级

光盘位置：光盘\MR\17\413

趣味指数：★★★★☆

实例说明

本实例在运行时，首先在窗体上端的文本框中显示若干条具有单一格式的字符串，然后单击窗体中的按钮，程序将按照文本框中文本的格式把数据写入到 DataGridView 控件中。实例运行效果如图 17.14 所示。

图 17.14　解析只有一种格式的文本文件

关键技术

本实例在实现解析只有一种格式的文本文件功能时，主要用到了 TextFieldParser 类的 TextFieldType 属性和 ReadFields 方法，下面对本实例中用到的关键技术进行详细讲解。

（1）TextFieldParser 类的 TextFieldType 属性

该属性用来表示要分析的文件是符号分隔文件还是固定宽度文件。语法格式如下：

```
public FieldType TextFieldType { get; set; }
```

该属性的值是 FieldType 类型的枚举值，FieldType 类型的枚举值及说明如表 17.8 所示。

表 17.8　FieldType 类型的枚举值及说明

枚 举 值	说　明
Delimited	指示字段是分隔的
FixedWidth	指示字段是固定宽度的

（2）TextFieldParser 类的 ReadFields 方法

该方法实现读取当前行上的所有字段，并将这些字段作为字符串数组返回，然后使光标前进到包含数据的下一行。语法格式如下：

```
public string[] ReadFields()
```

参数说明

返回值：返回一个包含当前行上的所有字段的字符串数组。

说明：TextFieldParser 类提供用于分析结构化文本文件的方法和属性，该类位于 Microsoft.VisualBasic.FileIO 命名空间下，并需要手动添加 Microsoft.VisualBasic 引用。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SingleFormatTxt。

（2）在默认窗体 Form1 中添加一个 TextBox 控件，用来显示文本文件的内容；添加一个 DataGridView 控件，用来导入并显示文本文件中的内容；添加一个 Button 控件，用来把文本文件中的内容导入到 DataGridView 控件中。

（3）程序的主要代码如下。

在窗体的 Load 事件中，首先把应用程序当前目录下的 test.txt 文件中的内容显示到文本框中。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    Computer MyComputer = new Computer();                          //创建 Computer 对象以读取文本文件
    txtResult.Text = MyComputer.FileSystem.ReadAllText("test.txt"); //读取文本文件
}
```

单击窗体上的按钮，实现将文本文件中的内容导入到 DataGridView 控件中。代码如下：

```
private void btnParseTextFiles_Click(object sender, EventArgs e)
{
```

```
    using (TextFieldParser myReader = new TextFieldParser("test.txt"))
    {
        myReader.TextFieldType = FieldType.Delimited;                    //表示档案内容是字符分隔
        myReader.Delimiters = new String[] { "," };                      //定义文本文件的字符分隔符
        this.DataGridView1.Rows.Clear();
        DataGridView1.ColumnHeadersVisible = true;
        //设定栏标题样式
        DataGridViewCellStyle columnHeaderStyle = new DataGridViewCellStyle();
        columnHeaderStyle.BackColor = Color.Beige;                       //设置背景色
        //设置字体
        columnHeaderStyle.Font = new Font("Verdana", 9, FontStyle.Bold);
        //设置列标题类型
        DataGridView1.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
        //设置列宽的调整模式
        DataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
        string[] currentRow;                                             //定义字符串数组
        int myRowCount = 1;                                              //定义表示行数的变量
        int myColCount = 0;                                              //定义表示列数的变量
        //循环处理文本文件中所有数据列的所有字段
        while (!myReader.EndOfData)
        {
            try
            {
                currentRow = myReader.ReadFields();                      //读取数据列
                if (myRowCount == 1)
                {
                    foreach (string currentField in currentRow)          //遍历字符串数组
                    {
                        //动态设定 DataGridView 控件的字段数目
                        DataGridView1.ColumnCount = myColCount + 1;
                        //设定 DataGridView 控件各栏的标题名称
                        DataGridView1.Columns[myColCount].Name = currentField;
                        myColCount += 1;
                    }
                }
                else
                {
                    this.DataGridView1.Rows.Add(currentRow);             //DataGridView 控件添加新行
                }
            }
            catch (MalformedLineException ex)
            {
                MessageBox.Show(ex.Message);
            }
            myRowCount += 1;                                             //行数自增 1
        }
    }
}
```

秘笈心法

心法领悟 413：关于 MalformedLineException 异常。

当 ReadFields 方法不能使用指定格式分析行时将引发 MalformedLineException 异常。

实例 414　解析含有多种格式的文本文件

高级

光盘位置：光盘\MR\17\414

趣味指数：★★★★☆

实例说明

本实例在运行时，首先在窗体左侧的文本框中显示若干条具有不同格式的字符串，然后单击窗体中的按钮，程序将按照文本框中文本的特有格式把数据写入到对应的 DataGridView 控件中。实例运行效果如图 17.15 所示。

图 17.15　解析含有多种格式的文本文件

关键技术

在解析含有多种格式的文本文件时，本实例主要用到了 TextFieldParser 类的 PeekChars 方法，该方法只返回指定数目的字符而不会前进至下一行，开发人员通过采用逐一判断每个数据行格式并逐行读取数据行的方式，能够解析含有多种格式的文本文件并顺利读取。语法格式如下：

```
public string PeekChars(int numberOfChars)
```

参数说明

❶ numberOfChars：字符的数目。

❷ 返回值：返回由指定数目的字符组成的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiFormatTxt。

（2）在默认窗体 Form1 中添加 3 个 DataGridView 控件，用来导入并显示对应格式的文本；添加一个 TextBox 控件，用来显示文本文件的内容；添加一个 Button 控件，用来实现把不同格式的文本导入对应的 DataGridView 控件中。

（3）程序的主要代码如下。

在 Form1 窗体的 Load 事件中，把应用程序当前目录下的 test.txt 文件中的内容显示到文本框中；另外，通过代码动态设置 3 个 DataGridView 控件的列标题等。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    txtResult.Text = File.ReadAllText("test.txt");              //读取文件的内容
    DataGridView1.ColumnHeadersVisible = true;                  //设置列标题可见
    //设定栏标题样式
    DataGridViewCellStyle columnHeaderStyle = new DataGridViewCellStyle();
    columnHeaderStyle.BackColor = Color.Beige;
    columnHeaderStyle.Font = new Font("Verdana", 9, FontStyle.Bold);
    DataGridView1.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
    //设定 DataGridView 控件的字段数目
    DataGridView1.ColumnCount = 3;
    //设定各栏的标题名称
    DataGridView1.Columns[0].Name = "类别编号";
    DataGridView1.Columns[1].Name = "类别名称";
    DataGridView1.Columns[2].Name = "说明";
    DataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    DataGridView2.ColumnHeadersVisible = true;
    DataGridView2.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
    //设定 DataGridView 控件的字段数目
    DataGridView2.ColumnCount = 4;
    //设定各栏的标题名称
    DataGridView2.Columns[0].Name = "产品编号";
    DataGridView2.Columns[1].Name = "产品名称";
    DataGridView2.Columns[2].Name = "单位数量";
    DataGridView2.Columns[3].Name = "单价";
    DataGridView2.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
```

```
    DataGridView3.ColumnHeadersVisible = true;
    DataGridView3.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
    //设定 DataGridView 控件的字段数目
    DataGridView3.ColumnCount = 3;
    //设定各栏的标题名称
    DataGridView3.Columns[0].Name = "货运公司编号";
    DataGridView3.Columns[1].Name = "货运公司名称";
    DataGridView3.Columns[2].Name = "电话";
    DataGridView3.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
}
```

单击窗体上的按钮，实现把不同格式的文本导入对应的 DataGridView 控件中。代码如下：

```
private void btnParseTextFiles_Click(object sender, EventArgs e)
{
    using (TextFieldParser myReader = new TextFieldParser("test.txt"))
    {
        int[] FirstFormat = { 5, 10, -1 };              //定义第一种格式的宽度
        int[] SecondFormat = { 6, 10, 17, -1 };         //定义第二种格式的宽度
        string[] ThirdFormat = { "," };                 //定义第三种格式的分隔字符
        this.DataGridView1.Rows.Clear();                //清空第一个 DataGridView 控件
        this.DataGridView2.Rows.Clear();                //清空第二个 DataGridView 控件
        this.DataGridView3.Rows.Clear();                //清空第三个 DataGridView 控件
        string[] CurrentRow;
        while (!myReader.EndOfData)                     //读取数据直到结束
        {
            try
            {
                //读取当前行的前两个字符
                string RowType = myReader.PeekChars(2);
                switch (RowType)                        //根据读取的两个字符进行分类
                {
                    case "CK":                          //对应第一个 DataGridView 控件
                        myReader.TextFieldType = FieldType.FixedWidth;
                        myReader.FieldWidths = FirstFormat;
                        CurrentRow = myReader.ReadFields();
                        this.DataGridView1.Rows.Add(CurrentRow);
                        break;
                    case "PB":                          //对应第二个 DataGridView 控件
                        myReader.TextFieldType = FieldType.FixedWidth;
                        myReader.FieldWidths = SecondFormat;
                        CurrentRow = myReader.ReadFields();
                        this.DataGridView2.Rows.Add(CurrentRow);
                        break;
                    case "SP":                          //对应第三个 DataGridView 控件
                        myReader.TextFieldType = FieldType.Delimited;
                        myReader.Delimiters = ThirdFormat;
                        CurrentRow = myReader.ReadFields();
                        this.DataGridView3.Rows.Add(CurrentRow);
                        break;
                }
            }
            catch (MalformedLineException ex)
            {
                MessageBox.Show("行 " + ex.Message + " 是无效的。略过。");
            }
        }
        //排序各个 DataGridView 控件的内容
        DataGridView1.Sort(DataGridView1.Columns[0], System.ComponentModel.ListSortDirection.Ascending);
        DataGridView2.Sort(DataGridView2.Columns[0], System.ComponentModel.ListSortDirection.Ascending);
        DataGridView3.Sort(DataGridView3.Columns[0], System.ComponentModel.ListSortDirection.Ascending);
    }
}
```

秘笈心法

心法领悟 414：指定 DataGridView 列的内容排序。

DataGridView 控件主要用来显示数据记录，若数据量较大，则查找某条记录比较麻烦，这时可以通过使用 DataGridView 控件的 Sort 方法来指定某个列及排序方式实现对控件中的数据进行升序或降序排列。

第18章

加密、解密及解压缩文件

- 加密与解密文件
- 压缩和解压缩文件

18.1 加密与解密文件

实例 415 对文本文件进行加密与解密

光盘位置：光盘\MR\18\415　　中级　　趣味指数：★★★★★

实例说明

在本实例的窗体中，首先选择要加密或解密的文本文件，然后单击“加密”或“解密”按钮对文本文件进行加密或解密。实例运行效果如图 18.1 所示。

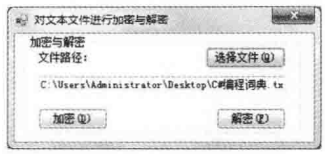


图 18.1 对文本文件进行加密与解密

关键技术

本实例主要用到了 System.Security.Cryptography 命名空间下的 RijndaelManaged 类的 CreateDecryptor 和 CreateEncryptor 方法以及 CryptoStream 类的 Write 方法，下面分别对它们进行详细讲解。

（1）RijndaelManaged 类

该类是访问 System.Security.Cryptography.Rijndael 对称加密算法的托管版本。语法格式如下：

```
public sealed class RijndaelManaged : Rijndael
```

注意：此算法支持 128、192 或 256 位的密钥长度。

（2）CreateDecryptor 方法

该方法位于 RijndaelManaged 类中，使用指定的 Key 和初始化向量（IV）创建对称的 Rijndael 解密器对象。语法格式如下：

```
public override IcryptoTransform CreateDecryptor (byte[] rgbKey,byte[] rgbIV)
```

参数说明

❶ rgbKey：用于对称算法的机密密钥。

❷ rgbIV：用于对称算法的 IV。

❸ 返回值：对称的 Rijndael 解密器对象。

（3）CreateEncryptor 方法

该方法位于 RijndaelManaged 类中，使用指定的 Key 和初始化向量（IV）创建对称的 Rijndael 加密器对象。语法格式如下：

```
public override ICryptoTransform CreateEncryptor (byte[] rgbKey,byte[] rgbIV)
```

参数说明

❶ rgbKey：用于对称算法的机密密钥。

❷ rgbIV：用于对称算法的 IV。

❸ 返回值：对称的 Rijndael 加密器对象。

（4）CryptoStream 类

该类定义将数据流链接到加密转换的流，它的构造器语法格式如下：

```
public CryptoStream (Stream stream,ICryptoTransform transform,CryptoStreamMode mode)
```

参数说明

❶ stream：对其执行加密转换的流。

❷ transform：要对流执行的加密转换。

❸ mode：CryptoStreamMode 枚举值之一。CryptoStreamMode 枚举值及说明如表 18.1 所示。

表 18.1　CryptoStreamMode 枚举值及说明

枚 举 值	说　明
Read	对加密流的读访问
Write	对加密流的写访问

（5）CryptoStream 类的 Write 方法

该方法将一个字节序列写入当前 CryptoStream 类中，并从当前位置写入指定的字节数。语法格式如下：

```
public override void Write (byte[] buffer,int offset,int count)
```

参数说明

❶ buffer：字节数组。该方法将 count 个字节从 buffer 复制到当前流。

❷ offset：buffer 中的字节偏移量，从此偏移量开始将字节复制到当前流。

❸ count：要写入当前流的字节数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EncryptTextFileOne。

（2）在默认 Form1 窗体中，主要添加一个 TextBox 控件，用来显示文件路径；添加一个 OpenFileDialog 控件，用来选择要加密或解密的文件；添加 3 个 Button 控件，用来执行加密、解密和选择文件操作。

（3）程序的主要代码如下。

单击“加密”按钮实现对选择的文本文件进行加密，“加密”按钮的 Click 事件代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "")                                          //若未选择要加密的文本文件
    { MessageBox.Show("请选择要加密的文件"); }                          //如果没有选择则弹出提示
    else
    {
        try{
          string strPath = textBox1.Text;                             //加密文件的路径
          int intLent=strPath.LastIndexOf("\\")+1;                    //设置截取的起始位置
          int intLong = strPath.Length;                               //设置截取的长度
          string strName = strPath.Substring(intLent,intLong-intLent);//要加密的文件名称
          int intTxt = strName.LastIndexOf(".");                      //设置截取的起始位置
          int intTextLeng = strName.Length;                           //设置截取的长度
          string strTxt = strName.Substring(intTxt,intTextLeng-intTxt);//取出文件的扩展名
          strName = strName.Substring(0,intTxt);
          //加密后的文件名及路径
          string strOutName = strPath.Substring(0, strPath.LastIndexOf("\\") + 1) + strName + "Out" + strTxt;
          //加密文件密钥
          byte[] key = { 24, 55, 102, 24, 98, 26, 67, 29, 84, 19, 37, 118, 104, 85, 121, 27, 93, 86, 24, 55, 102, 24, 98, 26, 67, 29, 9, 2, 49, 69, 73, 92 };
          byte[] IV ={ 22, 56, 82, 77, 84, 31, 74, 24, 55, 102, 24, 98, 26, 67, 29, 99 };
          RijndaelManaged myRijndael = new RijndaelManaged();
          FileStream fsOut = File.Open(strOutName, FileMode.Create, FileAccess.Write);
          FileStream fsIn = File.Open(strPath, FileMode.Open, FileAccess.Read);
          //写入加密文本文件
          CryptoStream csDecrypt = new CryptoStream(fsOut, myRijndael.CreateEncryptor(key, IV), CryptoStreamMode.Write);
        BinaryReader br = new BinaryReader(fsIn);                     //创建阅读器来读加密文本
        csDecrypt.Write(br.ReadBytes((int)fsIn.Length), 0, (int)fsIn.Length);//将数据写入加密文本
        csDecrypt.FlushFinalBlock();
        csDecrypt.Close();                                            //关闭 CryptoStream 对象
        fsIn.Close();                                                 //关闭 FileStream 对象
        fsOut.Close();                                                //关闭 FileStream 对象
        if (MessageBox.Show("加密成功!加密后的文件名及路径为:\n" + strOutName + ",是否删除源文件", "信息提示", MessageBoxButtons.
YesNo) == DialogResult.Yes)
        {
            File.Delete(strPath);                                     //删除指定文件
            textBox1.Text = "";                                       //清空文本框
        }else
        { textBox1.Text = ""; }
    }
```

```
    catch (Exception ee)                                        //如果出现异常
    {
        MessageBox.Show(ee.Message);                            //输出异常信息
    }
    }
}
```

单击“解密”按钮实现对加密的文本文件进行解密，“解密”按钮的 Click 事件代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "")                                    //若未选择要解密的文件
    {
        MessageBox.Show("请选择要解密的文件路径");                  //如果没有选择则弹出提示
    }
    else
    {
        string strPath = textBox1.Text;                         //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1;            //设置截取字符串的起始位置
        int intLong = strPath.Length;                           //设置截取长度
        string strName = strPath.Substring(intLent, intLong - intLent);  //要加密的文件名称
        int intTxt = strName.LastIndexOf(".");                  //截取字符串的起始位置
        int intTextLeng = strName.Length;                       //截取长度
        strName = strName.Substring(0, intTxt);                 //获取文件扩展名
        if (strName.LastIndexOf("Out") != -1)
        {
            strName = strName.Substring(0, strName.LastIndexOf("Out"));
        }
        else
        {
            strName = strName + "In";
        }
        //加密后的文件名及路径
        string strInName = strPath.Substring(0, strPath.LastIndexOf("\\") + 1) + strName + ".txt";
        //解密文件密钥
        byte[] key = { 24, 55, 102, 24, 98, 26, 67, 29, 84, 19, 37, 118, 104, 85, 121, 27, 93, 86, 24, 55, 102, 24, 98, 26, 67, 29, 9, 2, 49, 69, 73, 92 };
        byte[] IV ={ 22, 56, 82, 77, 84, 31, 74, 24, 55, 102, 24, 98, 26, 67, 29, 99 };
        RijndaelManaged myRijndael = new RijndaelManaged();     //创建 RijndaelManaged 对象
        //创建 FileStream 对象
        FileStream fsOut = File.Open(strPath, FileMode.Open, FileAccess.Read);
        CryptoStream csDecrypt = new CryptoStream(fsOut, myRijndael.CreateDecryptor(key, IV), CryptoStreamMode.Read);
        StreamReader sr = new StreamReader(csDecrypt);          //把文件读出来
        StreamWriter sw = new StreamWriter(strInName);          //解密后写入一个新文件
        sw.Write(sr.ReadToEnd());
        sw.Flush();
        sw.Close();
        sr.Close();
        fsOut.Close();
        if (MessageBox.Show("解密成功!解密后的文件名及路径为:"+strInName+"，是否删除源文件", "信息提示", MessageBoxButtons.YesNo)
== DialogResult.Yes)
        {
            File.Delete(strPath);                               //删除指定文件
            textBox1.Text = "";                                 //清空文本框
        }
        else
        {
            textBox1.Text = "";
        }
    }
}
```

秘笈心法

心法领悟 415：StreamWriter 类的 Flush 方法。

在使用 StreamWriter 类的实例调用 Write 方法写入数据完毕之后，需要调用该实例的 Flush 方法，该方法实现清理当前编写器的所有缓冲区，并使所有缓冲数据写入基础流。

实例 416　利用图片对文件进行加密与解密

光盘位置：光盘\MR\18\416　　中级　　趣味指数：★★★★☆

实例说明

本实例在加密时，使用指定的图片生成加密密钥，然后对文本文件进行加密；在解密时，使用加密时的图片生成解密密钥，然后对加密的文本文件进行解密。在本实例的窗体中，首先打开一个图片，用来生成加密或解密的密钥，然后选择要加密或解密的文本文件，最后单击“加密”或“解密”按钮，实现对文本文件的加密或解密。实例运行效果如图 18.2 所示。

图 18.2　利用图片对文件进行加密与解密

关键技术

实现本实例时，主要用到了 RC2CryptoServiceProvider 类、BinaryWriter 类的 Write 方法、File 类的 Delete 和 Copy 方法，下面分别对其进行介绍。

（1）RC2CryptoServiceProvider 类

该类定义访问 RC2 算法的加密服务提供程序（CSP）实现的包装对象，无法继承此类。

（2）BinaryWriter 类

该类以二进制形式将基元类型写入流，并支持用特定的编码写入字符串。语法格式如下：

```
public BinaryWriter (Streamoutput)
```

参数说明

output：表示输出流。

（3）BinaryWriter 类的 Write 方法

该方法将一个无符号字节写入当前流，并将流的位置提升一个字节。语法格式如下：

```
public virtual void Write (byte value)
```

参数说明

value：表示要写入的无符号字节。

（4）File 类

该类提供用于创建、复制、删除、移动和打开文件的静态方法，并协助创建 FileStream 对象，该类是个静态类。

（5）File 类的 Delete 方法

该方法删除指定的文件。如果指定的文件不存在，则引发异常。语法格式如下：

```
public static void Delete (string path)
```

参数说明

path：表示要删除的文件的名称。

（6）File 类的 Copy 方法

该方法将现有文件复制到新文件，不允许改写同名的文件。语法格式如下：

```
public static void Copy (string sourceFileName,string destFileName)
```

参数说明

❶ sourceFileName：要复制的文件。

❷ destFileName：目标文件的名称，不能是一个目录或现有文件。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EncryptTextFileTwo。

（2）在 Form1 窗体中，主要添加一个 TextBox 控件，用来显示加密或解密文件的路径；添加一个 OpenFileDialog 控件，用来选择要加密或解密的文件和打开密钥的图片；添加 4 个 Button 控件，用来执行加密、解密、打开文件和打开图片操作；添加一个 PictureBox 控件，用于显示密钥图片。

（3）程序的主要代码如下。

单击“加密”按钮实现利用图片对文本文件进行加密，“加密”按钮的 Click 事件代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        if (pictureBox1.ImageLocation==null)                    //判断是否选择了图片
        { MessageBox.Show("请选择一幅图片用于加密"); return; }    //如果没有选择则弹出提示
        if (textBox1.Text == "")                                //若未选择需要加密的文件
        { MessageBox.Show("请选择加密文件路径"); return; }        //如果没有选择则弹出提示
        //图片流
        FileStream fsPic = new FileStream(pictureBox1.ImageLocation, FileMode.Open, FileAccess.Read);
        //加密文件流
        FileStream fsText = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);
        //初始化对称算法的密钥和向量
        byte[] bykey = new byte[16];                            //定义存储密钥的字节数组
        byte[] byIv = new byte[8];                              //定义存储向量的字节数组
        fsPic.Read(bykey, 0, 16);                               //把图片流写入密钥缓冲区
        fsPic.Read(byIv, 0, 8);                                 //把图片流写入向量缓冲区
        //临时加密文件
        string strPath = textBox1.Text;                         //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1;
        int intLong = strPath.Length;
        string strName = strPath.Substring(intLent, intLong - intLent);   //要加密的文件名称
        string strLinPath = "C:\\" + strName;                   //临时加密文件路径
        FileStream fsOut = File.Open(strLinPath, FileMode.Create, FileAccess.Write);
        //开始加密，首先创建 RC2CryptoServiceProvider 对象
        RC2CryptoServiceProvider desc = new RC2CryptoServiceProvider();
        BinaryReader br = new BinaryReader(fsText);             //创建 BinaryReader 对象
        //创建 CryptoStream 对象，用于写入临时加密文件
        CryptoStream cs = new CryptoStream(fsOut, desc.CreateEncryptor(bykey, byIv), CryptoStreamMode.Write);
        cs.Write(br.ReadBytes((int)fsText.Length), 0, (int)fsText.Length);   //写入加密流
        cs.FlushFinalBlock();
        cs.Flush();
        cs.Close();
        fsPic.Close();
        fsText.Close();
        fsOut.Close();
        File.Delete(textBox1.Text.TrimEnd());                   //删除源文件
        File.Copy(strLinPath, textBox1.Text);                   //复制加密文件
        File.Delete(strLinPath);                                //删除临时文件
        MessageBox.Show("加密成功");
        pictureBox1.ImageLocation = null;
        textBox1.Text = "";
    }
    catch (Exception ee)
    {
        MessageBox.Show(ee.Message);
    }
}
```

单击“解密”按钮实现利用图片对加密的文本文件进行解密，“解密”按钮的 Click 事件代码如下：

```
private void button4_Click(object sender, EventArgs e)
{
    try
    {
        //图片流
        FileStream fsPic = new FileStream(pictureBox1.ImageLocation, FileMode.Open, FileAccess.Read);
        //解密文件流
        FileStream fsOut = File.Open(textBox1.Text, FileMode.Open, FileAccess.Read);
        //初始化对称算法的密钥和向量
        byte[] bykey = new byte[16];                            //定义存储密钥的字节数组
```

```
        byte[] byIv = new byte[8];                                          //定义存储向量的字节数组
        fsPic.Read(bykey, 0, 16);                                           //把图片流写入密钥缓冲区
        fsPic.Read(byIv, 0, 8);                                             //把图片流写入向量缓冲区
        //创建临时解密文件
        string strPath = textBox1.Text;                                     //加密文件的路径
        int intLent = strPath.LastIndexOf("\\") + 1;                        //获取不含文件名的路径长度
        int intLong = strPath.Length;                                       //获取含文件名的路径长度
        //获取要解密文件的名称，即加密文件的名称
        string strName = strPath.Substring(intLent, intLong - intLent);
        string strLinPath = "C:\\" + strName;                               //临时解密文件路径
        FileStream fs = new FileStream(strLinPath, FileMode.Create, FileAccess.Write);
        //开始解密，首先创建 RC2CryptoServiceProvider 对象
        RC2CryptoServiceProvider desc = new RC2CryptoServiceProvider();
        //创建 CryptoStream 对象，用于读取加密文件
        CryptoStream csDecrypt = new CryptoStream(fsOut, desc.CreateDecryptor(bykey, byIv), CryptoStreamMode.Read);
        BinaryReader sr = new BinaryReader(csDecrypt);                      //创建 BinaryReader 对象
        BinaryWriter sw = new BinaryWriter(fs);                             //创建 BinaryWriter 对象
        sw.Write(sr.ReadBytes(Convert.ToInt32(fsOut.Length)));              //写入解密流
        sw.Flush();
        sw.Close();
        sr.Close();
        fs.Close();
        fsOut.Close();
        fsPic.Close();
        csDecrypt.Flush();
        File.Delete(textBox1.Text.TrimEnd());                               //删除源文件
        File.Copy(strLinPath, textBox1.Text);                               //复制加密文件
        File.Delete(strLinPath);                                            //删除临时文件
        MessageBox.Show("解密成功");                                        //弹出提示信息
        pictureBox1.ImageLocation = null;                                   //清空图片
        textBox1.Text = "";                                                 //清空文本框
    }
    catch (Exception ee)                                                    //如果出现异常
    {
        MessageBox.Show(ee.Message);                                        //输出异常
    }
}
```

秘笈心法

心法领悟 416：使用 CryptoStream 类的 FlushFinalBlock 方法。

在调用 CryptoStream 类的 Write 方法写入加密或解密流之后，需要调用该类的 FlushFinalBlock 方法，该方法实现用缓冲区的当前状态更新基础数据源或储存库，随后清除缓冲区。

实例 417　使用 ROT13 算法加密解密文件

光盘位置：光盘\MR\18\417　　中级　　趣味指数：★★★★☆

实例说明

文件加密可以避免造成重要信息的泄漏，复杂的加密算法可以将信息加密得非常繁杂，但是对于一般的应用，没有必要做类似于 PGP、RSA 或 DES 等复杂的加密算法，本实例介绍使用 ROT13 这种简单的加密和解密算法。实例运行效果如图 18.3 所示。

图 18.3　使用 ROT13 算法加密解密文件

关键技术

本实例主要是用 Convert 类的 ToChar 方法来获取单个字符的 Unicode 编码，然后将字母的前 13 个和后 13 个对调，实现加密的功能。

ToChar 方法返回指定的 Unicode 字符值，不执行任何实际的转换。语法格式如下：

```
public static char ToChar (char value)
```

参数说明

value：一个 Unicode 字符。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ROT13Encrypt。

（2）在默认窗体 Form1 中添加两个文本框，分别用来显示原始数据和解密后的数据；添加两个 Button 控件，分别用来实现加密和解密数据。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)                    //使用 ROT13 算法加密数据
{
    textBox2.Text= ROT13Encode(textBox1.Text);
}
public string ROT13Encode(string InputText)                               //ROT13 算法
{
    char tem_Character;                                                   //定义字符变量
    int UnicodeChar;                                                      //表示一个 Unicode 字符
    string EncodedText = "";                                              //表示加密或解密后的字符串
    //遍历字符串中的所有字符，只能加密字符串，无法加密汉字
    for (int i = 0; i < InputText.Length; i++)
    {
        tem_Character = System.Convert.ToChar(InputText.Substring(i, 1));  //获取字符串中指定的字符
        UnicodeChar = (int)tem_Character;                                 //获取当前字符的 Unicode 编码
        if (UnicodeChar >= 97 && UnicodeChar <= 109)                      //对字符进行加密
        {
            UnicodeChar = UnicodeChar + 13;
        }
        else if (UnicodeChar >= 110 && UnicodeChar <= 122)                //对字符进行解密
        {
            UnicodeChar = UnicodeChar - 13;
        }
        else if (UnicodeChar >= 65 && UnicodeChar <= 77)                  //对字符进行加密
        {
            UnicodeChar = UnicodeChar + 13;
        }
        else if (UnicodeChar >= 78 && UnicodeChar <= 90)                  //对字符进行解密
        {
            UnicodeChar = UnicodeChar - 13;
        }
        EncodedText = EncodedText + (char)UnicodeChar;                    //返回设置后的字符
    }
    return EncodedText;
}
private void button2_Click(object sender, EventArgs e)                    //使用 ROT13 算法解密数据
{
    textBox2.Text = ROT13Encode(textBox2.Text);
}
```

秘笈心法

心法领悟 417：关于 ROT13 加密算法的总结。

ROT13 是一种简单的加密方式，主要是把 26 个英文字母的前 13 个和后 13 个对调，起到一定的英文文字加密保护作用。

实例 418　使用对称算法加密、解密文件

光盘位置：光盘\MR\18\418　　中级　　趣味指数：★★★★☆

实例说明

本实例使用对称算法实现加密和解密文件，在本实例的对称算法加密窗体中，首先选择源文件路径，然后输入加密密码和加密后的文件路径，最后单击“加密”按钮实现对源文件加密，如图 18.4 所示。在本实例的对称算法解密窗体中，首先选择要解密的原文件路径，然后输入解密密码和解密后的文件路径，最后单击“解密”按钮实现对源文件解密，如图 18.5 所示。

图 18.4　使用对称算法加密窗体

图 18.5　使用对称算法解密窗体

关键技术

本实例主要用到了 DESCryptoServiceProvider 类的 CreateDecryptor 方法和 CryptoStream 类的相关方法，关于 CryptoStream 类的相关方法请参考本章的实例 415。下面主要对 DESCryptoServiceProvider 类的 CreateDecryptor 方法进行详细讲解。

DESCryptoServiceProvider 类的 CreateDecryptor 方法使用指定的 SymmetricAlgorithm.Key 属性和初始化向量 SymmetricAlgorithm.IV 来创建对称解密器对象。语法格式如下：

```
public abstract ICryptoTransform CreateDecryptor(byte[] rgbKey, byte[] rgbIV)
```

参数说明

❶ rgbKey：用于对称算法的密钥。

❷ rgbIV：用于对称算法的初始化向量。

❸ 返回值：返回对称解密器对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其解决方案命名为 SymmetricalEncrypt，项目名称命名为 EncryptFile，作为实现加密的项目。然后添加一个新项目，命名为 UnEncryptFile，作为实现解密的项目。

（2）在 EncryptFile 项目的默认窗体 Form1 中添加 3 个文本框，分别用来显示源文件路径、输入加密密码和输入加密后的文件路径；添加两个 Button 控件，分别用来选择源文件和实现对文件的加密。对于 UnEncryptFile 项目的默认窗体 Form1，该窗体上控件的设置情况可参考 EncryptFile 项目中 Form1 窗体的设置。

（3）程序的主要代码如下。

在 EncryptFile 项目的默认窗体 Form1 中，单击“加密”按钮实现对源文件的加密。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    string myFile = textBox1.Text;                                    //获取源文件路径
    string myPassword = textBox2.Text;                                //获取加密密码
    string myEnFile = textBox3.Text;                                  //获取加密后的文件路径
    try
    {
        byte[] myIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };   //设置向量
        byte[] myKey = System.Text.Encoding.UTF8.GetBytes(myPassword);      //设置密钥
```

```
        //原文件的文件流
        FileStream myInStream = new FileStream(myFile, FileMode.Open, FileAccess.Read);
        //加密后文件的文件流
        FileStream myOutStream = new FileStream(myEnFile, FileMode.OpenOrCreate, FileAccess.Write);
        myOutStream.SetLength(0);                                   //初始文件流的长度
        byte[] myBytes = new byte[100];                             //定义缓冲区
        long myInLength = 0;                                        //定义不断变化的流的长度
        long myLength = myInStream.Length;                          //获取源文件的文件流的长度
        DES myProvider = new DESCryptoServiceProvider();            //定义标准的加密算法实例
        //实现将数据流链接到加密转换的流
        CryptoStream myCryptoStream = new CryptoStream(myOutStream, myProvider.CreateEncryptor(myKey, myIV), CryptoStreamMode.Write);
        //从原文件流中每次读取 100 个字节，然后写入加密转换的流
        while (myInLength < myLength)
        {
            int mylen = myInStream.Read(myBytes, 0, 100);           //读取源文件流
            myCryptoStream.Write(myBytes, 0, mylen);                //写入加密转换的流
            myInLength += mylen;                                    //计算写入的流长度
        }
        myCryptoStream.Close();                                     //关闭资源
        myInStream.Close();
        myOutStream.Close();
        MessageBox.Show("加密文件成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

在 UnEncryptFile 项目的默认窗体 Form1 中，单击“解密”按钮实现对源文件的解密。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    string str1 = textBox1.Text;                                    //获取源文件路径
    string strPwd = textBox2.Text;                                  //获取解密密码
    string str2 = textBox3.Text;                                    //获取解密后的文件路径
    try
    {
        byte[] myIV = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCD, 0xEF };   //设置向量
        byte[] myKey = System.Text.Encoding.UTF8.GetBytes(strPwd);          //设置密钥
        //源文件的文件流
        FileStream myFileIn = new FileStream(str1, FileMode.Open, FileAccess.Read);
        //解密后文件的文件流
        FileStream myFileOut = new FileStream(str2, FileMode.OpenOrCreate, FileAccess.Write);
        myFileOut.SetLength(0);                                     //初始化文件流的长度
        byte[] myBytes = new byte[100];                             //定义缓冲区
        long myLength = myFileIn.Length;                            //获取源文件流的长度
        long myInLength = 0;                                        //定义不断变化的流的长度
        DES myProvider = new DESCryptoServiceProvider();            //定义标准的加密算法实例
        //实现将数据流链接到解密转换的流
        CryptoStream myDeStream = new CryptoStream(myFileOut, myProvider.CreateDecryptor(myKey, myIV), CryptoStreamMode.Write);
        //从源文件流中每次读取 100 个字节，然后写入解密转换的流
        while (myInLength < myLength)
        {
            int mylen = myFileIn.Read(myBytes, 0, 100);             //读取源文件流
            myDeStream.Write(myBytes, 0, mylen);                    //写入解密转换的流
            myInLength += mylen;                                    //计算写入的流长度
        }
        myDeStream.Close();                                         //关闭资源
        myFileOut.Close();
        myFileIn.Close();
        MessageBox.Show("解密文件成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 418：关于加密与解密过程的总结。

从上面的两段加密和解密代码中可以看出，加密与解密的过程基本相同。首先都是获取源文件的文件流和加密（或解密）后的文件流，然后将源数据流链接到加密（或解密）转换的流，最后把源文件流写入加密（或解密）转换的流。

实例 419　使用口令加密可执行文件

高级

光盘位置：光盘\MR\18\419　　趣味指数：★★★★★

实例说明

exe 可执行文件的加密是软件加密技术的一个重要部分，本实例讲解如何在 C#中使用口令加密 exe 可执行文件。运行本实例，如果用户选择的是 exe 文件，则输入加密口令，单击“加密”按钮，加密选择的 exe 文件；如果用户选择的是 mrexe 文件，则输入加密时用的口令，单击“打开”按钮，打开加密过的 exe 文件。实例运行效果如图 18.6 所示。

图 18.6　使用口令加密可执行文件

关键技术

使用口令加密可执行文件，顾名思义就是在打开加密过的可执行文件时，需要先输入口令，然后才能打开，这时就需要将使用口令加密过的可执行文件的菜单项写入到注册表中，以便打开使用口令加密过的可执行文件时直接弹出“输入口令”对话框。在使用口令加密过的可执行文件的菜单项写入到注册表中时，需要用到 RegistryKey 类的 CreateSubKey 和 SetValue 方法，下面分别对它们进行详细讲解。

（1）CreateSubKey 方法

RegistryKey 类表示 Windows 注册表中的项级节点，它位于 Microsoft.Win32 命名空间下，其 CreateSubKey 方法主要用来创建一个新子项或打开一个现有子项以进行写访问。语法格式如下：

```
public RegistryKey CreateSubKey(string subkey)
```

参数说明

❶ subkey：要创建或打开的子项的名称或路径。

❷ 返回值：RegistryKey 对象，表示新建的子项或 null（如果操作失败）。如果为 subkey 指定了零长度字符串，则返回当前的 RegistryKey 对象。

技巧：要获取 RegistryKey 的实例，请使用 Registry 类的静态成员之一。

（2）SetValue 方法

该方法用来设置注册表项中的名称/值对的值。语法格式如下：

```
public void SetValue(string name,Object value)
```

参数说明

❶ name：要存储的值的名称。

❷ value：要存储的数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EncryptExe。

（2）在默认窗体 Form1 中主要添加如表 18.2 所示的控件。

表 18.2　Form1 窗体主要用到的控件及说明

控件类型	控件名称	属性设置	说　明
TextBox	textBox1	无	显示选择的加密或解密 exe 文件
	textBox2	PasswordChar 属性设置为*	输入加密或解密密码
Button	button1	Text 属性设置为“…”	选择要加密或解密的 exe 文件
	button2	Text 属性设置为“加密”	加密 exe 文件
	button3	Text 属性设置为“打开”	打开加密后的 exe 文件
OpenFileDialog	openFileDialog1	无	显示扫描进度

（3）程序的主要代码如下。

Form1 窗体加载时，首先将 exe 加密文件菜单写入到注册表中，然后判断系统中是否有打开的 exe 加密文件，如果有，则将其显示到“请选择 Exe 文件”文本框中。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    FileMenu(Application.ExecutablePath + ",0", Application.ExecutablePath);          //在注册表中创建 exe 加密文件菜单项
    string[] str = Environment.GetCommandLineArgs();                                  //记录系统中的打开项
    try
    {
        string strFile = "";                                                          //用来记录文件名
        for (int i = 2; i < str.Length; i++)
            strFile += str[i];                                                        //为文件名赋值
        FileInfo FInfo = new FileInfo(strFile);                                       //创建 FileInfo 对象
        if (FInfo.Extension.ToLower() == ".mrexe")
            textBox1.Text = strFile;                                                  //显示文件名
    }
    catch { }
}
```

向注册表中写入 exe 加密文件菜单时用到 FileMenu 方法，该方法为自定义的无返回值类型方法，主要用来向注册表中写入 exe 加密文件菜单，它有两个 string 类型的参数，分别用来表示 exe 加密程序环境变量及 exe 加密程序路径。FileMenu 方法的实现代码如下：

```
//创建快捷菜单
public static void FileMenu(string strPath, string strName)
{
    try
    {
        //在注册表中创建 exe 加密文件子键
        Registry.ClassesRoot.CreateSubKey(".mrexe");
        RegistryKey RKey1 = Registry.ClassesRoot.OpenSubKey(".mrexe", true);          //打开新创建的子键
        RKey1.SetValue("", "mrexefile");                                              //为新创建的子键赋值
        RKey1.Close();                                                                //关闭子键
        Registry.ClassesRoot.CreateSubKey("mrexefile");                               //创建 mrexefile 子键
        RegistryKey RKey2 = Registry.ClassesRoot.OpenSubKey("mrexefile", true);       //打开 mrexefile 子键
        RKey2.CreateSubKey("DefaultIcon");                                            //创建 DefaultIcon 子键
        RKey2.CreateSubKey("shell");                                                  //创建 shell 子键
        RKey2.Close();                                                                //关闭子键
        //打开子键
        RegistryKey RKey3 = Registry.ClassesRoot.OpenSubKey("mrexefile\\DefaultIcon", true);
        RKey3.SetValue("", strPath);                                                  //为子键设置值
        RKey3.Close();                                                                //关闭子键
        //打开子键
        RegistryKey RKey4 = Registry.ClassesRoot.OpenSubKey("mrexefile\\shell", true);
        RKey4.CreateSubKey("使用口令打开");                                           //创建子键
        RKey4.Close();                                                                //关闭子键
        //打开子键
        RegistryKey RKey5 = Registry.ClassesRoot.OpenSubKey("mrexefile\\shell\\使用口令打开", true);
        RKey5.CreateSubKey("command");                                                //创建子键
        RKey5.Close();                                                                //关闭子键
        //打开子键
        RegistryKey RKey6 = Registry.ClassesRoot.OpenSubKey("mrexefile\\shell\\使用口令打开\\command", true);
```

```
            RKey6.SetValue("", strName + " \\F %1");                    //为子键设置值
            RKey6.Close();                                              //关闭子键
        }
        catch{}
    }
```

注意：对注册表进行操作时，需要添加 Microsoft.Win32 命名空间。

单击“加密”按钮，首先判断是否选择要加密的 exe 文件，如果已经选择，则使用用户输入的加密密码对选择的 exe 文件进行加密，执行成功后弹出信息提示。“加密”按钮的 Click 事件代码如下：

```
//加密 exe 文件
private void button2_Click(object sender, EventArgs e)
{
    string strPwd = textBox2.Text;                                      //记录加密密码
    byte[] btRKey = new byte[0];                                        //定义一个字节数组，用来记录加密密码
    if (strPwd.Length == 6)                                             //判断密码位数
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[0],
(byte)strPwd[1] };                                                      //记录加密密码
    }
    if (strPwd.Length == 7)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6],
(byte)strPwd[0] };
    }
    if (strPwd.Length >= 8)
    {
        btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5], (byte)strPwd[6],
(byte)strPwd[7] };
     }
    //创建 FileStream 对象
    FileStream FStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);
    //创建一个 FileStream 对象，用来创建加密文件
    FileStream NewFStream = new FileStream(textBox1.Text + ".mrexe", FileMode.OpenOrCreate, FileAccess.Write);
    NewFStream.SetLength((long)0);                                      //设置加密文件初始长度为 0
    byte[] buffer = new byte[0x400];                                    //定义一个字节数组
    int MinNum = 0;                                                     //设置加密位置从 0 开始
    long length = FStream.Length;                                       //记录文件长度
    int MaxNum = (int)(length / ((long)0x400));                         //计算文件最大值
    DES myDES = new DESCryptoServiceProvider();                         //创建 DES 加密流对象
    //创建 CryptoStream 加密流对象
    CryptoStream CStream = new CryptoStream(NewFStream, myDES.CreateEncryptor(btRKey, btRKey), CryptoStreamMode.Write);
    while (MinNum < length)
    {
        int count = FStream.Read(buffer, 0, 0x400);                     //对要加密的文件进行流读取
        CStream.Write(buffer, 0, count);                                //向加密流中写入数据
        MinNum += count;                                                //记录已经读取的数据位置
    }
    CStream.Close();
    NewFStream.Close();
    FStream.Close();
    File.Delete(textBox1.Text);                                         //删除旧文件
    MessageBox.Show("使用口令加密可执行文件成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

单击“打开”按钮，首先使用用户输入的解密密码对加密过的 exe 文件进行解密，然后使用 Process 类的 Start 方法打开解密过的 exe 文件。“打开”按钮的 Click 事件代码如下：

```
//解密 exe 文件
private void button3_Click(object sender, EventArgs e)
{
    string strPwd = textBox2.Text;                                      //记录解密密码
    FileStream FStream = null;                                          //声明 FileStream 对象，用来记录旧文件
    FileStream NewFStream = null;                                       //声明 FileStream 对象，用来记录新文件
    CryptoStream CStream = null;                                        //声明 CryptoStream 对象，用来解密文件
    try
```

```
    {
        try
        {
            byte[] btRKey = new byte[0];                                    //定义一个字节数组，用来记录解密密码
            if (strPwd.Length == 6)                                         //判断密码位数
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[0], (byte)strPwd[1] };                                         //记录解密密码
            }
            if (strPwd.Length == 7)
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[6], (byte)strPwd[0] };
            }
            if (strPwd.Length >= 8)
            {
                btRKey = new byte[] { (byte)strPwd[0], (byte)strPwd[1], (byte)strPwd[2], (byte)strPwd[3], (byte)strPwd[4], (byte)strPwd[5],
(byte)strPwd[6], (byte)strPwd[7] };
            }
            //创建 FileStream 对象
            FStream = new FileStream(textBox1.Text, FileMode.Open, FileAccess.Read);
            //获取新文件名
            string strNewFile = textBox1.Text.Substring(0, textBox1.Text.Length - 6);
            //创建解密文件
            NewFStream = new FileStream(strNewFile, FileMode.OpenOrCreate, FileAccess.Write);
            NewFStream.SetLength((long)0);                                  //设置解密文件初始长度为 0
            byte[] buffer = new byte[0x400];                                //定义一个字节数组
            int MinNum = 0;                                                 //设置解密位置从 0 开始
            long length = FStream.Length;                                   //记录文件长度
            int MaxNum = (int)(length / ((long)0x400));                     //计算文件最大值
            DES myDES = new DESCryptoServiceProvider();                     //创建 DES 加解密对象
            //创建 CryptoStream 加解密流对象
            CStream = new CryptoStream(NewFStream, myDES.CreateDecryptor(btRKey, btRKey), CryptoStreamMode.Write);
            while (MinNum < length)
            {
                int count = FStream.Read(buffer, 0, 0x400);                 //对要解密的文件进行流读取
                CStream.Write(buffer, 0, count);                            //向解密流中写入数据
                MinNum += count;                                            //记录已经读取的数据位置
            }
            CStream.Close();
            FStream.Close();
            NewFStream.Close();
            File.Delete(textBox1.Text);                                     //删除加密文件
            System.Diagnostics.Process.Start(strNewFile);                   //打开解密后的 exe 文件
        }
        catch
        {
            MessageBox.Show("口令错误！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Error);
            textBox2.Focus();
        }
    }
    finally
    {
        CStream.Close();
        FStream.Close();
        NewFStream.Close();
    }
}
```

秘笈心法

心法领悟 419：获取文件的扩展名。

在处理文件操作的应用程序中，获取文件的扩展名信息很重要，可以使用 FileInfo 类的 Extension 属性获取文件的扩展名，本实例使用该属性来判断指定文件是否为加密后的 mrexe 文件。

18.2　压缩和解压缩文件

实例 420	使用 GZip 压缩文件 光盘位置：光盘\MR\18\420	中级 趣味指数：★★★★☆

实例说明

本实例使用 GZipStream 类的 Write 方法实现对文件的压缩，在本实例的窗体中，首先选择要压缩的源文件，然后输入压缩文件的名称，最后单击“压缩”按钮实现将指定的文件压缩为 GZip 格式。实例运行效果如图 18.7 所示。

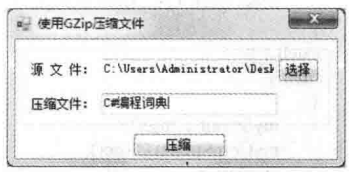


图 18.7　使用 GZip 压缩文件

注意：当压缩文件时，不需要指明压缩文件的扩展名，程序会自动为压缩文件设置扩展名（gzip）。

关键技术

GZipStream 类位于 System.IO.Compression 命名空间下，提供用于压缩和解压缩流的方法和属性。该类用来表示 GZip 数据格式，这种数据格式包括一个检测数据损坏的循环冗余校验值。当使用 GZipStream 类构造一个压缩流后，就可以使用该类的 Write 方法写数据，从而实现压缩文件功能。下面介绍该类的 Write 方法。

Write 方法实现从指定的字节数组中将要压缩的字节写入基础流。语法格式如下：

```
public override void Write(byte[] array, int offset, int count)
```

Write 方法中的参数说明如表 18.3 所示。

表 18.3　Write 方法中的参数说明

参　数	说　明	参　数	说　明
array	用于存储要压缩字节的数组	count	压缩的字节数
offset	数组中开始读取的位置	返回值	无返回值

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GZipFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示源文件路径及输入的压缩文件名称；添加两个 Button 控件，分别用来选择要压缩的文件及实现压缩文件。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text))                    //若未选择源文件
    {
        MessageBox.Show("请选择源文件!","信息提示");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text))                    //若未输入压缩文件名称
    {
        MessageBox.Show("请输入压缩文件名!", "信息提示");
        return;
    }
    string str1 = textBox1.Text;                                //获取源文件路径
    string str2 = textBox2.Text.Trim()+".gzip";                 //获取压缩文件的名称
    byte[] myByte = null;                                       //定义字节数组引用
    FileStream myStream = null;                                 //表示源文件的文件流
```

```
    FileStream myDesStream = null;                          //表示压缩文件的文件流
    GZipStream myComStream = null;                          //表示压缩流
    try
    {
        //获取源文件的文件流
        myStream = new FileStream(str1, FileMode.Open, FileAccess.Read, FileShare.Read);
        myByte = new byte[myStream.Length];                 //根据源文件流的大小创建缓冲区
        myStream.Read(myByte, 0, myByte.Length);            //把源文件的文件流写入缓冲区
        //创建压缩文件的文件流
        myDesStream = new FileStream(str2, FileMode.OpenOrCreate, FileAccess.Write);
        //根据压缩文件的文件流生成压缩流
        myComStream = new GZipStream(myDesStream, CompressionMode.Compress, true);
        myComStream.Write(myByte, 0, myByte.Length);        //把源文件的文件流写入压缩流
        MessageBox.Show("压缩文件完成！");
    }
    catch { }
    finally
    {
        myStream.Close();
        myComStream.Close();
        myDesStream.Close();
    }
}
```

秘笈心法

心法领悟 420：创建压缩流。

通过创建 GZipStream 对象来创建压缩流时，需要给构造方法的第二个参数传入 CompressionMode.Compress 枚举值。

实例 421　使用 Gzip 解压文件

光盘位置：光盘\MR\18\421　　中级　　趣味指数：★★★★☆

实例说明

本实例使用 GZipStream 类的 Read 方法实现对文件的解压，在本实例的窗体中，首先选择要解压的 GZip 格式文件，然后输入解压文件的名称，最后单击“解压”按钮实现解压 GZip 格式的文件。实例运行效果如图 18.8 所示。

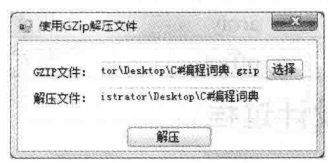


图 18.8　使用 GZip 解压文件

关键技术

若要使用 GZipStream 类构造一个解压缩流，则需要该类的 Read 方法读取数据，从而实现解压缩文件功能。下面将介绍该类的 Read 方法。

Read 方法实现将若干解压缩的字节读入指定的字节数组。语法格式如下：

```
public override int Read(byte[] array, int offset, int count)
```

Read 方法中的参数说明如表 18.4 所示。

表 18.4　Read 方法中的参数说明

参　数	说　明
array	用于存储解压缩的字节的数组
offset	数组中开始读取的位置
count	解压缩的字节数
返回值	解压缩到字节数组中的字节数。如果已到达流的末尾，则返回 0 或已读取的字节数

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UnGzipFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示 GZip 文件路径、输入的解压文件名称；添加两个 Button 控件，分别用来选择 Gzip 格式文件及实现解压文件。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text))                    //若未选择 gzip 文件
    {
        MessageBox.Show("请选择 GZIP 文件!", "信息提示");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text))                    //若未输入解压文件名
    {
        MessageBox.Show("请输入解压文件名!", "信息提示");
        return;
    }
    string str1 = textBox1.Text;                                //获取 gzip 文件的名称
    string str2 = textBox2.Text.Trim();                         //获取解压文件名
    byte[] myByte = null;                                       //定义字节数组引用
    FileStream myStream = null;                                 //表示 gzip 文件的文件流
    FileStream myDesStream = null;                              //表示解压文件的文件流
    GZipStream myDeComStream = null;                            //表示解压缩流
    try
    {
        myStream = new FileStream(str1, FileMode.Open);         //获取 gzip 文件的文件流
        //根据 gzip 文件的文件流生成解压缩流
        myDeComStream = new GZipStream(myStream, CompressionMode.Decompress, true);
        myByte = new byte[4];                                   //创建 4 个字节大小的缓冲区
        int myPosition = (int)myStream.Length - 4;
        myStream.Position = myPosition;                         //设置此流的当前位置
        myStream.Read(myByte, 0, 4);                            //读入 4 个字节到缓冲区
        myStream.Position = 0;                                  //重新设置此流的当前位置
        int myLength = BitConverter.ToInt32(myByte, 0);
        byte[] myData = new byte[myLength + 100];               //创建缓冲区
        int myOffset = 0;                                       //表示解压缩流的开始读取位置
        int myTotal = 0;                                        //表示从解压缩流中总共读取的字节数
        while (true)
        {
            //从解压缩流中读取数据写入缓冲区
            int myBytesRead = myDeComStream.Read(myData, myOffset, 100);
            if (myBytesRead == 0)                               //若读取解压缩流结束
                break;
            myOffset += myBytesRead;                            //设置新的起始位置
            myTotal += myBytesRead;                             //获取总共读取的字节数
        }
        myDesStream = new FileStream(str2, FileMode.Create);    //创建解压文件的文件流
        myDesStream.Write(myData, 0, myTotal);                  //把缓冲区中的数据写入文件流
        myDesStream.Flush();
        MessageBox.Show("解压文件完成！ ");
    }
    catch { }
    finally
    {
        myStream.Close();
        myDeComStream.Close();
        myDesStream.Close();
    }
}
```

秘笈心法

心法领悟 421：创建解压缩流。

通过创建 GZipStream 对象来创建解压缩流时，需要给构造方法的第二个参数传入 CompressionMode.Decompress 枚举值。

实例 422 使用 WinRAR 压缩文件

光盘位置：光盘\MR\18\422　　中级　　趣味指数：★★★★☆

实例说明

本实例通过使用C#语言执行WinRAR软件的压缩命令来实现压缩文件。运行本实例，首先在窗体中选择要压缩的源文件，然后输入压缩文件的名称，最后单击“压缩”按钮实现将指定的文件压缩为 rar 格式。实例运行效果如图 18.9 所示。

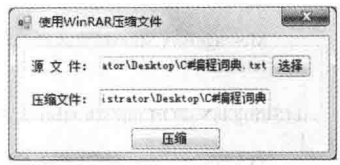


图 18.9　使用 WinRAR 压缩文件

关键技术

WinRAR 软件可以使用命令行模式实现对文件的压缩，本实例使用的压缩文件的语法格式如下：

```
WinRAR a rarFile sourceFile
```

WinRAR 软件的压缩命令参数及说明如表 18.5 所示。

表 18.5　WinRAR 软件的压缩命令参数及说明

参　数	说　明
WinRAR	WinRAR 软件的执行命令
a	WinRAR 软件的压缩命令，表示添加文件到压缩文件
rarFile	压缩文件的名称
sourceFile	被压缩的文件

说明：压缩文件时，生成的压缩文件位于被压缩文件所在的目录。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WinRARFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示源文件路径及输入压缩文件名称；添加两个 Button 控件，分别用来选择要压缩的文件及实现压缩文件。

（3）程序的主要代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text))                    //若未选择源文件
    {
        MessageBox.Show("请选择源文件!", "信息提示");
        return;
    }
    if (String.IsNullOrEmpty(textBox2.Text))                    //若未输入压缩文件名
    {
        MessageBox.Show("请输入压缩文件名!", "信息提示");
        return;
    }
    String myRar;                                               //表示 WinRAR.exe 所在路径
    RegistryKey myReg;                                          //声明 RegistryKey 引用
    Object myObj;                                               //声明 Object 引用
    String myInfo;                                              //表示压缩命令的字符串
    ProcessStartInfo myStartInfo;                               //声明 ProcessStartInfo 引用
    Process myProcess;                                          //声明 Process 类的引用
    string strRar;                                              //表示压缩文件名
    string strFile;                                             //表示源文件名
    try
```

```
    {
        //检索注册表 HKEY_CLASSES_ROOT 基项下的指定子项
        myReg = Registry.ClassesRoot.OpenSubKey(@"Applications\WinRAR.exe\Shell\Open\Command");
        //检索子项中与指定名称关联的值
        myObj = myReg.GetValue("");
        //获取包含 WinRAR.exe 文件所在路径的字符串
        myRar = myObj.ToString();
        myReg.Close();                                                    //关闭指定的注册表项
        //获取 WinRAR.exe 文件所在的完整路径
        myRar = myRar.Substring(1, myRar.Length - 7);
        strRar = textBox2.Text.Trim() + ".rar";                           //设置压缩文件的名称
        strFile = textBox1.Text.Substring(textBox1.Text.LastIndexOf("\\")+1);  //获取源文件的名称
        myInfo = " a " + strRar + " " + strFile + "";                     //设置压缩命令
        myStartInfo = new ProcessStartInfo();                             //创建 ProcessStartInfo 对象
        myStartInfo.FileName = myRar;                                     //设置要启动应用程序的路径
        myStartInfo.Arguments = myInfo;                                   //设置启动程序时的命令参数
        myStartInfo.WindowStyle = ProcessWindowStyle.Hidden;              //隐藏进程窗口
        //设置要启动的进程的初始目录
        myStartInfo.WorkingDirectory = textBox1.Text.Substring(0, textBox1.Text.LastIndexOf("\\"));
        myProcess = new Process();                                        //新建进程
        //设置要传递给进程的 Start 方法的属性
        myProcess.StartInfo = myStartInfo;
        myProcess.Start();                                                //启动进程
        myProcess.WaitForExit();                                          //等待关闭进程
        myProcess.Close();                                                //释放进程资源
        MessageBox.Show("压缩文件成功！");
    }
    catch(Exception ex)
    {
        throw ex;
    }
}
```

秘笈心法

心法领悟 422：隐藏启动进程时使用的窗口。

当启动一个进程之后，若不希望显示进程运行时的窗口，可以通过设置当前进程的 WindowStyle 属性值为 ProcessWindowStyle.Hidden 来实现。

实例 423　使用 WinRAR 解压文件

光盘位置：光盘\MR\18\423　　中级　　趣味指数：★★★★☆

实例说明

本实例通过使用 C#语言执行 WinRAR 软件的解压缩命令来实现解压 rar 文件。运行本实例，首先在窗体中选择要解压的 rar 格式文件，然后单击“解压”按钮实现解压 rar 文件到 rar 文件所在的目录。实例运行效果如图 18.10 所示。

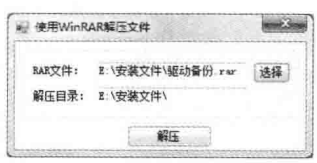


图 18.10　使用 WinRAR 解压文件

关键技术

WinRAR 软件可以使用命令行模式实现对 rar 文件的解压缩，本实例使用的解压缩文件的语法格式如下：

```
WinRAR x rarFile currentDirectory
```

WinRAR 软件的解压命令参数及说明如表 18.6 所示。

表 18.6 WinRAR 软件的解压命令参数及说明

参　数	说　明
WinRAR	WinRAR 软件的执行命令
x	WinRAR 软件的解压缩命令，表示解压缩文件
rarFile	压缩文件名
currentDirectory	压缩文件所在的目录

说明：解压缩文件时，生成的文件位于压缩文件所在的目录。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UnWinRARFile。

（2）在默认窗体 Form1 中添加两个 TextBox 控件，分别用来显示 rar 文件路径及解压目录；添加两个 Button 控件，分别用来选择 rar 格式的文件及实现解压文件。

（3）程序的主要代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    String myRar;                                                    //表示 WinRAR.exe 所在路径
    RegistryKey myReg;                                               //声明 RegistryKey 引用
    Object myObj;                                                    /声明 Object 引用
    String myInfo;                                                   //表示解压命令的字符串
    ProcessStartInfo myStartInfo;                                    /声明 ProcessStartInfo 引用
    Process myProcess;                                               //声明 Process 引用
    string strRar;                                                   //表示压缩文件名
    string strFile;                                                  //表示解压目录
    try
    {
        //检索注册表 HKEY_CLASSES_ROOT 基项下的指定子项
        myReg = Registry.ClassesRoot.OpenSubKey("Applications\\WinRar.exe\\Shell\\Open\\Command");
        //检索子项中与指定名称关联的值
        myObj = myReg.GetValue("");
        //获取包含 WinRAR.exe 文件所在路径的字符串
        myRar = myObj.ToString();
        myReg.Close();                                               //关闭指定的注册表项
        //获取 WinRAR.exe 文件所在的完整路径
        myRar = myRar.Substring(1, myRar.Length - 7);
        strRar = textBox1.Text.Substring(textBox1.Text.LastIndexOf("\\") + 1);   //获取 rar 文件名
        strFile = textBox2.Text.Substring(textBox2.Text.LastIndexOf("\\")+1);    //获取解压目录
        myInfo = " X " + strRar + " " + strFile + "\\";              //设置解压命令
        myStartInfo = new ProcessStartInfo();                        //创建 ProcessStartInfo 对象
        myStartInfo.FileName = myRar;                                //设置要启动应用程序的路径
        myStartInfo.Arguments = myInfo;                              //设置启动程序时的命令参数
        myStartInfo.WindowStyle = ProcessWindowStyle.Hidden;         //隐藏进程窗口
        //设置要启动的进程的初始目录
        myStartInfo.WorkingDirectory = textBox1.Text.Substring(0,textBox1.Text.LastIndexOf("\\"));
        myProcess = new Process();                                   //创建进程
        //设置要传递给进程的 Start 方法的属性
        myProcess.StartInfo = myStartInfo;
        myProcess.Start();                                           //启动进程
        MessageBox.Show("解压文件成功！");
    }
    catch { }
}
```

秘笈心法

心法领悟 423：设置要启动的进程的初始目录。

在启动一个进程之前，有时需要动态设置当前进程的启动目录。例如，本实例设置进程的启动目录为 rar 文件的所在目录，这可以通过设置进程的 WorkingDirectory 属性值来实现。

实例 424　批量压缩和解压缩文件

光盘位置：光盘\MR\18\424　　高级　　趣味指数：★★★★★

实例说明

计算机之间如果想传输多个文件，通常要将这些文件做成压缩包。这样不仅容易携带和传输，而且还压缩了体积。运行本实例，在窗体中可以选择要压缩的多个文件，然后单击“批量压缩”按钮，将选择的多个文件压缩成一个压缩包。也可以选择多个压缩包，然后单击“批量解压缩”按钮，批量解压缩选择的压缩包。实例运行效果如图 18.11 所示。

图 18.11　批量压缩和解压缩文件

关键技术

本实例中的压缩和解压缩主要使用的是 ICSharpCode.dll 组件，它是一个第三方组件，相对于.NET 自带的 Copression 压缩类，它压缩后的文件更小，而且支持的压缩格式也多，因此这里使用了该组件。使用该组件时，首先需要在“添加引用”中将 ICSharpCode.dll 文件添加到项目中，然后在命名空间区域添加如下代码：

```
using ICSharpCode.SharpZipLib;
using ICSharpCode.SharpZipLib.Zip;
using ICSharpCode.SharpZipLib.Checksums;
```

下面针对使用 ICSharpCode.dll 组件压缩和解压缩文件进行详细讲解。

（1）压缩文件

在通过 ICSharpCode.dll 组件压缩文件时，主要使用 ZipOutputStream 类，该类用于保存一个 zip 文件流，使用过程中只要把要压缩的文件流加入即可，其主要步骤如下。

首先，将要压缩的文件流加入到 ZipOutputStream 类中，其中的 zipedFileName 是要生成的 zip 文件的文件路径。代码如下：

```
ZipOutputStream zipedStream = new ZipOutputStream(File.Create(zipedFileName));
```

然后，打开一个文件，取得文件流。代码如下：

```
FileStream fs= File.OpenRead(file);
byte[] buffer = new byte[fs.Length];
fs.Read(buffer, 0, buffer.Length);
```

把文件加入压缩流要用到 ZipEntry 类，生成一个 ZipEntry 实例。代码如下：

```
ZipEntry entry = new ZipEntry(fileName);
```

再把 ZipEntry 加入到压缩流中。代码如下：

```
zipedStream.PutNextEntry(entry);
```

最后把文件写入压缩流完成压缩操作。代码如下：

```
zipedStream.Write(buffer, 0, buffer.Length);
```

（2）解压缩文件

在通过 ICSharpCode.dll 组件解压缩文件时，主要使用 ZipInputStream 类，该类生成一个 GZipInputStream 流，用来打开压缩文件，其主要步骤如下。

首先，使用 File 类的 OpenRead 方法打开压缩文件，然后创建 ZipInputStream 对象。代码如下：

```
ZipInputStream ZIPStream  = new ZipInputStream(File.OpenRead(FileToUpZip));
```

然后，通过 File 类的 Create 方法生成一个文件流，用来生成解压文件。代码如下：

```
FileStream streamWriter = File.Create(fileName);
```

通过 ZipInputStream 对象的 Read 方法读入一个压缩块。代码如下：

```
size = ZIPStream.Read(data, 0, data.Length);
```

最后，通过 FileStream 对象的 Write 方法写入解压文件代表的文件流。代码如下：

```
streamWriter = File.Create(fileName);                    //创建写入流
int size = 2048;                                         //定义缓存大小
byte[] data = new byte[2048];                            //定义缓存
while (true)
{
    size = ZIPStream.Read(data, 0, data.Length);         //开始读取
    if (size > 0)
        streamWriter.Write(data, 0, size);               //写入流
    else
        break;
}
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BatchDecompression。

（2）在默认窗体 Form1 中主要用到的控件及说明如表 18.7 所示。

表 18.7　Form1 窗体主要用到的控件及说明

控件类型	控件名称	属性设置	说　明
StatusStrip	statusStrip1	无	状态栏
SaveFileDialog	saveFileDialog1	Filter 属性设为 RAR|*.rar	选择压缩文件保存路径
TextBox	txtfiles	无	显示要压缩的文件路径
	txtfiles2	无	显示要解压缩的文件路径
OpenFileDialog	openFileDialog1	无	选择要压缩的文件

（3）程序的主要代码如下。

在 Form1 窗体的后台代码中，自定义一个 CopyFile 方法，用于创建一个文件夹，然后将需要批量压缩的文件全部复制到该文件夹下，以便通过压缩文件夹实现批量压缩文件。代码如下：

```
public void CopyFile(string[] list,string strNewPath,ToolStripProgressBar TSPBar)
{
    try
    {
        TSPBar.Maximum = list.Length;                    //文件数量
        string strNewFile = "c:\\" + strNewPath;         //保存的新目录
        if (!Directory.Exists(strNewFile))               //判断目录是否存在
            Directory.CreateDirectory(strNewFile);       //如果不存在则创建
        foreach (object objFile in list)                 //遍历文件数组
        {
            string strFile = objFile.ToString();         //获取文件路径
            string Filename = strFile.Substring(strFile.LastIndexOf("\\") + 1, strFile.Length - strFile.LastIndexOf("\\") - 1);
                                                         //截取文件名
            File.Copy(strFile, strNewFile+"\\"+Filename, true);  //将文件复制到新的目录下
            TSPBar.Value += 1;                           //设置进度条进度
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

当文件全部复制到新建的文件夹后，调用 ZipFileDictory 方法压缩文件夹，其参数分别代表待压缩的文件夹、压缩文件输出流以及压缩后的文件名。该方法首先创建文件夹，然后压缩文件，再递归压缩文件夹。代码如下：

```
private bool ZipFileDictory(string FolderToZip, ZipOutputStream ZOPStream, string ParentFolderName)
{
    bool res = true;                                     //判断操作是否成功
    string[] folders, filenames;                         //声明文件夹和文件的数组
    ZipEntry entry = null;                               //创建一个 ZipEntry 实例
    FileStream fs = null;                                //创建一个 FileStream 实例
    Crc32 crc = new Crc32();                             //创建一个 Crc32 实例
```

```
    try
    {
        //创建当前文件夹，加上“/”才会当成是文件夹创建
        entry = new ZipEntry(Path.Combine(ParentFolderName, Path.GetFileName(FolderToZip) + "/"));
        ZOPStream.PutNextEntry(entry);                                    //加入到压缩流中
        ZOPStream.Flush();
        filenames = Directory.GetFiles(FolderToZip);                      //先压缩文件，再递归压缩文件夹
        foreach (string file in filenames)                                //遍历文件
        {
            fs = File.OpenRead(file);                                     //打开压缩文件
            byte[] buffer = new byte[fs.Length];                          //设置缓存
            fs.Read(buffer, 0, buffer.Length);                            //读取文件流
            entry = new ZipEntry(Path.Combine(ParentFolderName, Path.GetFileName(FolderToZip) + "/" + Path.GetFileName(file)));
            entry.DateTime = DateTime.Now;                                //获取当前日期和时间
            entry.Size = fs.Length;                                       //获取文件流大小
            fs.Close();                                                   //关闭文件流
            crc.Reset();
            crc.Update(buffer);
            entry.Crc = crc.Value;
            ZOPStream.PutNextEntry(entry);                                //加入到压缩流中
            ZOPStream.Write(buffer, 0, buffer.Length);                    //开始写入
        }
    }
    catch
    {
        res = false;
    }
    finally
    {
        if (fs != null)
        {
            fs.Close();
            fs = null;
        }
        if (entry != null)
        {
            entry = null;
        }
        GC.Collect();
        GC.Collect(1);
    }
    folders = Directory.GetDirectories(FolderToZip);                      //获取指定目录下的所有子目录
    foreach (string folder in folders)                                    //遍历目录
    {
        if (!ZipFileDictory(folder, ZOPStream, Path.Combine(ParentFolderName, Path.GetFileName(FolderToZip))))
        {
            return false;                                                 //返回 false
        }
    }
    return res;
}
```

如果想批量解压缩文件，只需选择要解压缩的文件后，单击“批量解压缩”按钮。此时会调用自定义的 UnZip 方法，其参数分别代表待解压的文件和指定的解压目录，步骤是首先初始化压缩文件写入流对象，然后创建解压后的文件名，读取压缩文件，向解压后的文件写入内容。代码如下：

```
public void UnZip(string FileToUpZip, string ZipedFolder)
{
    if (!File.Exists(FileToUpZip))                                        //如果不存在需要解压缩的文件
    {
        return;                                                           //返回
    }
    if (!Directory.Exists(ZipedFolder))                                   //如果不存在解压缩的文件夹
    {
        Directory.CreateDirectory(ZipedFolder);                           //创建文件夹
    }
    ZipInputStream ZIPStream = null;                                      //创建 ZipInputStream 实例
    ZipEntry theEntry = null;                                             //创建 ZipEntry 实例
```

```
        string fileName;
        FileStream streamWriter = null;                                    //创建文件流
        try
        {
            //生成一个 GZipInputStream 流，用来打开压缩文件
            ZIPStream = new ZipInputStream(File.OpenRead(FileToUpZip));
            while ((theEntry = ZIPStream.GetNextEntry()) != null)
            {
                if (theEntry.Name != String.Empty)
                {
                    fileName = Path.Combine(ZipedFolder, theEntry.Name);
                    if (fileName.EndsWith("/") || fileName.EndsWith("\\"))  //判断文件路径是否是文件夹
                    {
                        Directory.CreateDirectory(fileName);               //创建文件夹
                        continue;
                    }
                    streamWriter = File.Create(fileName);                  //生成文件流，它用来生成解压文件
                    int size = 2048;                                       //压缩块的大小，一般为 2048 的倍数
                    byte[] data = new byte[2048];                          //指定缓冲区的大小
                    while (true)
                    {
                        size = ZIPStream.Read(data, 0, data.Length);       //读入一个压缩块
                        if (size > 0)
                        {
                            streamWriter.Write(data, 0, size);             //写入解压文件代表的文件流
                        }
                        else
                        {
                            break;                                         //若读到压缩文件尾，则结束
                        }
                    }
                }
            }
        }
        finally
        {
            if (streamWriter != null)                                      //如果存在 streamWriter
            {
                streamWriter.Close();                                      //关闭流
                streamWriter = null;
            }
            if (theEntry != null)                                          //如果存在 theEntry
            {
                theEntry = null;                                           //清空
            }
            if (ZIPStream != null)                                         //如果存在 ZIPStream
            {
                ZIPStream.Close();                                         //关闭流
                ZIPStream = null;
            }
            GC.Collect();
            GC.Collect(1);
        }
    }
```

秘笈心法

心法领悟 424：合理使用第三方组件。

.NET 框架本身提供了丰富的 dll 组件，通常情况下这些组件完全可以满足开发人员的需求。有些第三方组件具有特定的功能，并且执行效率非常高，可以满足开发人员的某些特殊要求，这时可以考虑使用第三方组件来解决开发中遇到的一些特殊需求。

C#与 Office 高效开发篇

⏭ 第 19 章　C#与 Word 互操作

⏭ 第 20 章　高效应用 Excel

第19章

C#与 Word 互操作

- Word 文档基本操作
- 读取外部数据到 Word 中
- 将 Word 中的数据进行输出处理

19.1　Word 文档基本操作

实例 425	将 Word 文档嵌入到 WinForm 窗体中	初级
	光盘位置：光盘\MR\19\425	趣味指数：★★★★★

实例说明

Office 是人们日常工作中经常用到的办公软件，随着信息技术的飞速发展，已经有越来越多的人使用 Word 制作文档、使用 Excel 制作报表、使用 OutLook 方便地收发电子邮件。本实例将介绍怎样使用 C#在 Form 窗体中显示 Word 文档。实例运行效果如图 19.1 所示。

图 19.1　将 Word 文档嵌入到 WinForm 窗体中

关键技术

本实例重点介绍怎样使用 WebBrowser 控件显示 Word 文档，下面对本实例中用到的关键技术进行详细讲解。

WebBrowser 控件是 Windows 窗体应用程序的标准控件，使用 WebBrowser 控件可以在窗体中显示网页信息，通过 WebBrowser 控件的 Navigate 方法可以将指定位置的文档加载到 WebBrowser 控件中，该方法常用的语法格式如下：

```
public void Navigate(string urlString)
```

参数说明

urlString：要加载文档的 URL 地址。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordInForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 Word 文档嵌入到 WinForm 窗体中”，并向窗体中添加一个 MenuStrip 菜单控件，向菜单中添加两个菜单项“打开”和“退出”，分别用于打开 Word 文档和退出应用程序；向窗体中添加一个 WebBrowser 控件，用于显示 Word 文档。

（3）程序的主要代码如下：

```
private void 打开 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog P_GetFile = new OpenFileDialog();                    //创建打开文件对话框对象
    DialogResult P_dr = P_GetFile.ShowDialog();                         //显示打开文件对话框
    if (P_dr == DialogResult.OK)                                        //是否单击“确定”按钮
    {
        WebBrowser.Navigate(P_GetFile.FileName);                        //打开 Word 文档并显示
    }
}
```

秘笈心法

心法领悟 425：使用 WebBrowser 控件轻松地在窗体中浏览网页信息。

使用 WebBrowser 控件可以在 Windows 窗体客户端应用程序中显示网页，且非常方便，只需执行 Navigate 方法，并在方法参数中加入网址信息字符串即可。

实例 426	创建 Word 文档	初级
	光盘位置：光盘\MR\19\426	趣味指数：★★★★☆

实例说明

操作 Word 文档之前首先要创建 Word 文档，如果系统中已经安装了 Office 办公软件，可以非常容易地完成 Word 文档的创建。那么，怎样使用 C#来创建 Word 文档呢？这正是本实例中将要介绍的内容。实例运行效果如图 19.2 所示。

在实例的运行中单击“创建 Word 文档”按钮，会根据用户选择的路径位置创建 Word 文档，而文档的名称就是当前系统的日期与时间。创建的 Word 文档如图 19.3 所示。

图 19.2　创建 Word 文档

图 19.3　创建的 Word 文档

关键技术

本实例重点介绍怎样使用 Documents 集合的 Add 方法方便地创建 Word 文档，下面对本实例中用到的关键技术进行详细讲解。

使用 C#操作 Word 文档前首先要创建 Word 应用程序对象（Application 对象），Application 对象的 Documents 属性会返回 Documents 对象，Documents 对象是 Word 应用程序打开的 Word 文档的集合，使用 Documents 对象的 Add 方法可以方便地创建 Word 文档。Add 方法的语法格式如下：

```
Document Documents.Add(ref object Template, ref object NewTemplate, ref object DocumentType, ref object Visible);
```

Add 方法中的参数说明如表 19.1 所示。

表 19.1　Add 方法中的参数说明

参　数	描　述
Template	创建 Word 文档所使用的模板，如果省略此参数，Word 会使用 Normal 模板
NewTemplate	是否将文档以模板形式打开，默认值为 false
DocumentType	设定创建文档的类型，参数接收枚举成员，默认值为 wdNewBlankDocument
Visible	打开的文档是否可见，默认值为 true

注意：创建 Word 文档时可以根据 Add 方法的第一个参数（Template）来设置创建 Word 文档所使用的模板。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NewWord。

（2）使用 C#操作 Word 文档，首先要确定本地计算机中已经安装 Office 办公软件，本实例中使用的 Office 版本为 2003。只有安装 Office 2003 后 C#才可以引用与 Word 相关的 COM 组件。C#引用 COM 组件的具体步骤是，选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令，弹出“引用管理器”对话框，

如图 19.4 所示。在“引用管理器”对话框中选择“COM/类型库”，再选择 Microsoft Word Object Library 引用，单击“确定”按钮完成添加。成功添加 COM 组件的引用后，可以在程序代码中引用命名空间“using Word = Microsoft.Office.Interop.Word;”。

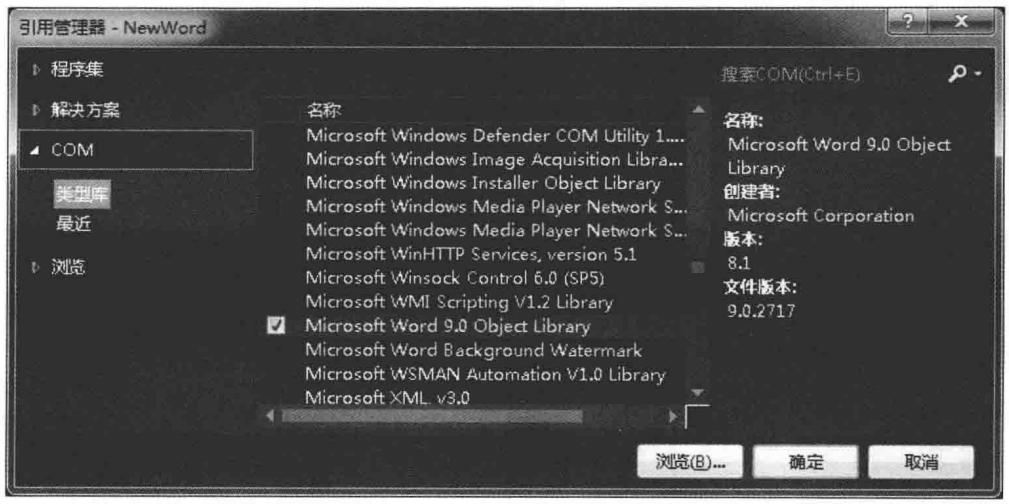


图 19.4　“引用管理器”对话框

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“创建 Word 文档”，并向窗体中添加一个 TextBox 控件，用于显示 Word 文档保存路径；向窗体中添加 3 个 Button 控件，分别用于选择文档保存路径、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                          //将新建按钮设置为不可用
    ThreadPool.QueueUserWorkItem(                                     //开始线程池
        (pp) =>                                                       //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();   //创建应用程序对象
            object P_obj = "Normal.dot";                              //定义文档模板
            Word.Document P_wd = G_wa.Documents.Add(                  //向 Word 应用程序中添加文档
                ref P_obj, ref G_missing, ref G_missing, ref G_missing);
            G_str_path = string.Format(                               //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                              //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(               //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                              //调用窗体线程
                (MethodInvoker)(() =>                                 //使用 Lambda 表达式
                {
                    MessageBox.Show("Word 文档已经创建", "提示！");    //提示已经创建 Word
                    btn_display.Enabled = true;                       //启用显示按钮
                }));
        });
}
```

秘笈心法

心法领悟 426：通过 Visible 属性设置 Word 文档是否可见。

在 Form 窗体编程中可以设置窗体的 Visible 属性，从而控制窗体是否可见，使用 C#操作 Word 文档时，则可以通过设置 Application 对象的 Visible 属性控制 Word 文档是否可见。

实例 427 向 Word 文档中写入文字

光盘位置：光盘\MR\19\427　　初级　　趣味指数：★★★★☆

实例说明

Word 有着强大的文本编辑功能，用户可以轻松地在 Word 中输入文本内容，更改文字字体，设置文字大小、颜色，方便地对文本内容进行排版。在前面的实例中已经详细介绍了怎样使用 C#创建 Word 文档，本实例将会使用 C#方便地向指定的 Word 文档中输入文本内容。实例运行效果如图 19.5 所示。

实例运行中单击“创建 Word 文档”按钮，会根据用户选择的路径位置创建 Word 文档并向文档中添加文字，而文档的名称就是当前系统的日期与时间。创建的 Word 文档如图 19.6 所示。

图 19.5　向 Word 文档中写入文字

图 19.6　被写入文字的 Word 文档

关键技术

本实例重点介绍怎样使用 Word 文档中的 Range 属性向 Word 中添加文本信息，下面对本实例用到的关键技术进行详细讲解。

读取或设置 Word 文档中的文本内容的方式基本相同，首先是得到文本内容的范围，然后通过范围对象读取或设置文本信息。本实例中首先使用 Paragraphs 集合中的索引获取 Word 文档中的段落对象，然后通过 Paragraph 段落对象的 Range 属性得到文档中段落的范围，最后通过范围对象的 Text 属性获取或设置段落内的文本信息。

说明：在 Word 文档的操作中也可以使用 Range 对象的 Collapse 方法折叠 Range 对象。折叠后的 Range 对象将不包含文本或其他信息，只是标注文档中的一个位置。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddLetter。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，Text 属性为“向 Word 文档中写入文字”，并向窗体中添加两个 TextBox 控件，分别用于输入字符串和显示 Word 文档保存路径；向窗体中添加 3 个 Button 控件，分别用于选择文档保存路径、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                    //停用新建按钮
    ThreadPool.QueueUserWorkItem(                               //使用线程池
        (P_temp) =>                                             //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                      //创建 Word 应用程序对象
```

```
            Word.Document P_wd = G_wa.Documents.Add(                          //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;                    //得到文档的范围对象
            P_Range.Text = txt_add.Text;                                      //向文档中添加字符串
            G_str_path = string.Format(                                       //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                                      //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                       //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                      //开始执行窗体线程
                (MethodInvoker)(() =>                                         //使用 Lambda 表达式
                {
                    btn_Display.Enabled = true;                               //启用显示按钮
                    MessageBox.Show("成功创建 Word 文档！", "提示！");          //弹出消息对话框
                }));
        });
    }
```

秘笈心法

心法领悟 427：Range 方法的妙用。

通过 Document 对象的 Range 方法可以返回一个 Range 对象，如果 Range 方法中的参数是默认参数，此时的 Range 是文档中所有内容的范围，可以通过 Range.Text 属性获取或设置 Range 对象范围内的文本内容。

实例 428　在 Word 文档中插入图片

中级

光盘位置：光盘\MR\19\428

趣味指数：★★★★☆

实例说明

在 Word 文档的使用过程中，可以轻松地向文档中插入图片信息，那么，怎样使用 C#向 Word 文档中插入图片呢？本实例中将会介绍一种方法，可以方便地实现上述功能。实例运行效果如图 19.7 所示。

实例运行中单击“创建 Word 文档”按钮，会根据用户选择的路径位置创建 Word 文档并向文档中添加图片，而文档的名称就是当前系统日期与时间。创建的 Word 文档如图 19.8 所示。

图 19.7　在 Word 文档中插入图片

图 19.8　被插入图片的 Word 文档

关键技术

本实例重点介绍怎样使用 InlineShapes 集合的 AddPicture 方法向 Word 文档中插入图片，下面对本实例中用

到的关键技术进行详细讲解。

使用 Documents 对象的 Add 方法可以创建 Word 文档，使用 Open 方法可以打开 Word 文档，当得到文档对象后可以根据 InlineShapes 属性得到 InlineShape 对象的集合，InlineShape 对象包括图片、ActiveX 控件等信息。使用 InlineShapes 集合的 AddPicture 方法可以方便地向文档中插入图片。AddPicture 方法的语法格式如下：

```
InlineShape AddPicture(string FileName, ref object LinkToFile, ref object SaveWithDocument, ref object Range);
```

AddPicture 方法中的参数说明如表 19.2 所示。

表 19.2　AddPicture 方法中的参数说明

参　数	描　述
FileName	将要插入的图片文件名称及路径
LinkToFile	如果为 true，则将图片链接到创建该对象的文件；如果为 false，则将图片作为该文件的独立副本，默认为 false
SaveWithDocument	如果为 true，则链接的图片与工作表一起保存，默认为 false
Range	插入图片的范围

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddImage。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在 Word 文档中插入图片”，并向窗体中添加两个 TextBox 控件，分别用于显示图片路径和显示 Word 文档保存路径；添加 4 个 Button 控件，分别用于选择向 Word 文档中插入的图片、选择 Word 文档保存路径、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                          //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                     //使用线程池
        (P_temp) =>                                                   //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                            //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                  //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;            //得到段落范围
            object P_Ranges = P_Range;                                //创建 object 对象
            P_wd.InlineShapes.AddPicture(                             //向文档中插入图片
                G_OpenFileDialog.FileName, ref G_missing, ref G_missing, ref P_Ranges);
            G_str_path = string.Format(                               //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                              //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(               //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                              //开始执行窗体线程
                (MethodInvoker)(() =>                                 //使用 Lambda 表达式
                {
                    btn_Display.Enabled = true;                       //启用显示按钮
                    MessageBox.Show("成功创建 Word 文档！", "提示！");  //弹出消息对话框
                }));
        });
}
```

秘笈心法

心法领悟 428：可以使用 AddPicture 方法向 Word 中插入多个图片。

实例中已经介绍了如何使用 AddPicture 方法向 Word 文档中插入图片，如果已经理解了插入图片的方法，那么向文档中插入多个图片实现起来也会变得很简单。AddPicture 方法的最后一个参数是指定将图片插入到 Word 文档中的哪个范围，读者可以根据自己的选择将多个图片插入到文档中的不同位置。

实例 429　在 Word 文档中绘制表格

光盘位置：光盘\MR\19\429　　中级　　趣味指数：★★★★☆

实例说明

使用 Excel 可以方便地处理表格信息，但是在 Word 中也可以简单地使用表格。在 Word 菜单中选择“表格”/“插入”/“表格”命令，在插入表格窗口中添加表格的行数和列数，最后单击“确定”按钮插入表格。在本实例中可以使用 C#方便地向指定的 Word 文档中插入表格，并在表格中添加相应信息。实例运行效果如图 19.9 所示。

实例运行中单击“创建 Word 文档”按钮，系统会根据用户选择的路径位置创建 Word 文档并向文档中绘制表格，而文档的名称就是当前系统日期与时间。创建的 Word 文档如图 19.10 所示。

图 19.9　在 Word 文档中绘制表格

图 19.10　Word 文档中的表格

关键技术

本实例重点介绍怎样使用 Tabels 集合的 Add 方法在文档中绘制表格，下面对本实例中用到的关键技术进行详细讲解。

通过 Document 对象的 Tables 属性可以得到 Tables 集合，使用该集合对象的 Add 方法可以轻松地在 Word 文档中绘制表格。Add 方法的语法格式如下：

```
Table Add(Range Range, int NumRows, int NumColumns, ref object DefaultTableBehavior, ref object AutoFitBehavior);
```

Add 方法中的参数说明如表 19.3 所示。

表 19.3　Add 方法中的参数说明

参　数	描　述
Range	插入表格所在的范围
NumRows	插入表格的行数
NumColumns	插入表格的列数
DefaultTableBehavior	WdDefaultTableBehavior 枚举值，指定表格的自适应方式
AutoFitBehavior	WdAutoFitBehavior 枚举值，指定表格的自适应方式

说明：在 Word 文档中插入表格后可以使用表格对象的 Cell 方法获取或设置表格中的文本内容，该方法接收两个参数，这两个参数分别为表格中行与列的索引。需要注意的是，表格中的行与列的索引值从 1 开始。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AddTable。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在 Word 文档中绘制表格”，并向窗体中添加 3 个 TextBox 控件，分别用于显示 Word 文档保存路径、设置插入表格行数和设置插入表格列数；添加 3 个 Button 控件，分别用于选择 Word 文档保存路径、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    G_ToolProgressBar.Minimum = 1;                                          //设置进度条最小值
    G_ToolProgressBar.Maximum = int.Parse(txt_row.Text)+1;                  //设置进度条最大值
    btn_New.Enabled = false;                                                //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                           //使用线程池
        (P_temp) =>                                                         //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                                  //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                        //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;                  //得到文档范围对象
            object P_DefaultTable =                                         //创建表格参数对象
                Word.WdDefaultTableBehavior.wdWord8TableBehavior;
            object P_AutoFit = Word.WdAutoFitBehavior.wdAutoFitWindow;      //创建表格参数对象
            Word.Table P_WordTable = P_Range.Tables.Add(                    //向文档中添加表格
                P_Range,int.Parse(txt_row.Text),int.Parse(txt_column.Text),ref P_DefaultTable, ref P_AutoFit);
            for (int i = 1; i < int.Parse(txt_row.Text) + 1; i++)
            {
                for (int j = 1; j < int.Parse(txt_column.Text) + 1; j++)
                {
                    P_WordTable.Cell(i, j).Range.Text =                     //向表格中添加数据
                        string.Format("{0}行 {1}列", i.ToString(), j.ToString());
                    Thread.Sleep(10);                                       //线程挂起 10 毫秒
                }
                this.Invoke(                                                //调用窗体线程
                    (MethodInvoker)(() =>                                   //使用 Lambda 表达式
                    {
                        G_ToolProgressBar.Value = i + 1;                    //设置进度信息
                    }));
            }
            G_str_path = string.Format(                                     //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                                    //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                     //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                    //开始执行窗体线程
                (MethodInvoker)(() =>                                       //使用 Lambda 表达式
                {
                    btn_Display.Enabled = true;                             //启用显示按钮
                    MessageBox.Show("成功创建 Word 文档！", "提示！");      //弹出消息对话框
                }));
        });
}
```

秘笈心法

心法领悟 429：使用 Cell 方法轻松地获取或设置表格中的文本信息。

表格对象的 Cell 方法提供了两个参数并返回一个 Range 对象，该方法中的两个参数分别为表格对象中的行与列的索引。需要注意的是，参数中行与列的索引都是从 1 开始的，根据两个参数可以找到并返回表格中的单元格的 Range 对象，并可以根据 Range 对象的 Text 属性获取或设置单元格的文本内容。

实例 430　向 Word 文档中插入 Windows 控件

光盘位置：光盘\MR\19\430　　中级　　趣味指数：★★★★★

实例说明

Word 是一个功能强大的文字处理软件，在 Word 文档中可以方便地对文字信息进行排版，也可以向文档中插入生动、直观的图片信息。本实例将会介绍一个有趣的功能，用于向 Word 文档中插入 Windows 控件，并且此控件还可以响应用户事件。例如，向 Word 文档中插入 Button 控件，也可以响应用户的单击事件。实例运行效果如图 19.11 所示。

图 19.11　向 Word 文档中插入 Windows 控件

关键技术

本实例重点介绍怎样使用 Controls 集合的 AddButton 方法向文档中插入 Windows 按钮控件，下面对本实例中用到的关键技术进行详细讲解。

使用 Controls 集合的 AddButton 方法可以方便地向 Word 文档中插入按钮控件。AddButton 的语法格式如下：

```
public Button AddButton(Range range, float width, float height, string name);
```

AddButton 方法中的参数说明如表 19.4 所示。

表 19.4　AddButton 方法中的参数说明

参　数	描　述
range	将要插入 Windows 控件所在的范围
width	插入 Windows 控件的宽度
height	插入 Windows 控件的高度
name	插入 Windows 控件的名称

说明：本实例介绍了向 Word 文档中插入 Windows 控件的方法，当 AddButton 方法执行后将会向 Word 文档中插入 Button 按钮，同时还会返回此 Button 对象的引用，可以通过设置此 Button 对象的属性和事件来控制 Word 文档中 Button 按钮的状态和事件。

设计过程

（1）打开 Visual Studio 2012 开发环境，选择“文件”/“新建”/“项目”命令，弹出“新建项目”对话框，在 “项目类型”列表框中选择 Visual C#/Office 选项，再在右侧的“模板”列表框中选择相应的 Word 版本选项，并在“名称”文本框中输入“InsertControl”，然后选择项目保存位置，最后单击“确定”按钮新建项目，如图 19.12 所示。

图 19.12　在 Visual Studio 2012 中创建 Word 文档

（2）在打开的 Word 文档中单击鼠标右键，在弹出的快捷菜单中选择“查看代码”命令，再在 Startup 事件中写入如下代码：

```
private void ThisDocument_Startup(object sender, System.EventArgs e)
{
    Word.Range P_Range1 = this.Paragraphs[1].Range;                    //得到文档范围
    Microsoft.Office.Tools.Word.Controls.Button P_btn =                //向文档中添加按钮
        this.Controls.AddButton(P_Range1, 50, 20, "button1");
    P_btn.Text = "Button 按钮";                                        //设置按钮文本
    P_btn.Height = 50;                                                 //设置按钮高度
    P_btn.Width = 100;                                                 //设置按钮宽度
    P_btn.Click += new EventHandler(P_btn_Click);                      //添加单击事件
}
```

秘笈心法

心法领悟 430：怎样向 Word 文档中插入多个 Windows 控件？

使用 Controls 集合的 AddButton 方法可以向 Word 文档中添加 Button 控件，同样地，也可以使用其他方法向文档中添加多个 Windows 控件。例如，使用 AddTextBox 方法可以向文档中添加文本框控件，使用 AddDataGridView 方法可以向文档中添加 DataGridView 控件。

实例 431　设置 Word 文档中的字体样式

光盘位置：光盘\MR\19\431　　初级　　趣味指数：★★★★☆

实例说明

使用 C#能够方便地操作指定的 Word 文档，可以向文档中添加文字、图片和表格，甚至还可以添加 Windows

控件。本实例将会介绍一种可以方便地设置 Word 文档中的字体样式的方法。实例运行效果如图 19.13 所示。

实例运行中单击“创建 Word 文档”按钮，会根据用户选择的路径位置创建 Word 文档并向文档中添加带有字体样式的文字，而文档的名称就是当前系统的日期与时间。创建的 Word 文档如图 19.14 所示。

图 19.13　设置 Word 文档中的字体样式

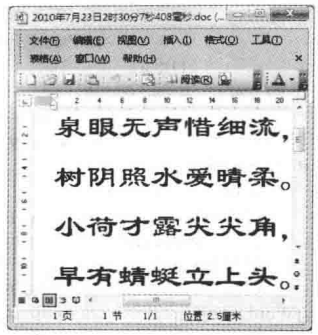


图 19.14　Word 文档中带有字体样式的文字

关键技术

本实例重点介绍怎样使用 Font 对象设置 Word 文档中文本内容的字体，下面对本实例中用到的关键技术进行详细讲解。

操作 Word 文档的过程中可以使用 Document 对象的 Range 方法或 Paragraph 对象的 Range 属性获得范围对象，范围对象的 Font 属性将会返回一个 Font 对象，通过设置 Font 对象的 Name 属性可以更改指定范围内文本的字体名称，设置 Font 对象的 Size 属性可以更改文本内容的字体大小。Font 对象的常用属性如表 19.5 所示。

表 19.5　Font 对象的常用属性

属　性	描　述
Name	指定字体的名称
Size	设置字体的大小，单位为“磅”
Bold	将此属性设置为-1，表示字体为加粗格式
Italic	将此属性设置为-1，表示字体为斜体格式

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FontStyle。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，更改 Text 属性为“设置 Word 文档中的字体样式”，向窗体中添加 3 个 TextBox 控件，分别用于显示文档保存路径、用户输入文字信息和预览文字信息；添加 4 个 Button 控件，分别用于选择 Word 文档保存路径、预览文字信息、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                          //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                     //使用线程池
        (P_temp) =>                                                   //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                            //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                  //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;            //得到段落范围
            this.Invoke(                                              //开始执行窗体线程
                    (MethodInvoker)(() =>                             //使用 Lambda 表达式
                        {
```

```
                P_Range.Text = txt_Text.Text;                                    //向文档中添加文本
                P_Range.Font.Name =                                              //设置文本字体
                    rbtn_Font1.Checked ? rbtn_Font1.Text :
                    rbtn_Font2.Checked ? rbtn_Font2.Text :
                    rbtn_Font3.Checked ? rbtn_Font3.Text : "宋体";
                P_Range.Font.Size =                                              //设置文本大小
                    int.Parse(cbox_Select.SelectedItem.ToString());
            }));
        G_str_path = string.Format(                                              //计算文件保存路径
            @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
            DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
        P_wd.SaveAs(                                                             //保存 Word 文件
            ref G_str_path,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing);
        ((Word._Application)G_wa.Application).Quit(                              //退出应用程序
            ref G_missing, ref G_missing, ref G_missing);
        this.Invoke(                                                             //开始执行窗体线程
            (MethodInvoker)(() =>                                                //使用 Lambda 表达式
            {
                btn_Display.Enabled = true;                                      //启用显示按钮
                MessageBox.Show("成功创建 Word 文档！","提示！");                  //弹出消息对话框
            }));
    });
}
```

秘笈心法

心法领悟 431：Font 对象的妙用。

如果理解了本实例中 Font 对象的使用方法，那么就可以轻松地设置 Name 属性更改文档中字体的名称，设置 Size 属性更改文档中字体的大小，设置 Color 属性更改文档中字体的颜色。

实例 432　统计 Word 文档中的字符数

光盘位置：光盘\MR\19\432　　初级　　趣味指数：★★★★☆

实例说明

在字符串操作中，可以通过字符串对象的 Length 属性得到字符串的长度（字符数量），也可以使用类似的方法来统计 Word 文档中的字符数量。首先通过 Word 文档对象的 Range 方法得到指定 Word 文档的 Range 对象（范围信息），然后通过 Range 的 Text 属性得到文档中文本内容的字符串，通过分析此字符串来统计 Word 文档中的字符数量。实例运行效果如图 19.15 所示。

图 19.15　统计 Word 文档中的字符数

关键技术

本实例重点介绍怎样方便地统计 Word 文档中的字符数量，下面对本实例中用到的关键技术进行详细讲解。

统计 Word 文档中字符数量的方法实现起来很简单，首先得到文档中所有段落的范围，通过范围的 Text 属

性可以得到包括文档中所有文本信息的字符串对象。字符串对象的 Length 属性会返回字符串中字符的数量，现在，可以使用一些方法排除字符串中的回车符或空格等字符，最后得到 Word 文档中字符的数量。

📖 **说明**：计算字符串中字符的数量，可以使用循环的方式判断字符串中每一个字符是否为回车符或空格并开始计数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetCount。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，更改 Text 属性为“统计 Word 文档中的字符数”，向窗体中添加一个 TextBox 控件，用于显示 Word 文档打开位置；添加两个 RadioButton 控件，用于设置统计文档中字符数量的方式。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_Get.Enabled = false;                                                    //停用统计按钮
    ThreadPool.QueueUserWorkItem(                                               //开始线程池
        (pp) =>                                                                 //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();             //创建应用程序对象
            object P_OpenFileDialog = G_OpenFileDialog.FileName;                //创建 object 对象
            Word.Document P_Document = G_wa.Documents.Open(                     //打开 Word 文档
                ref P_OpenFileDialog, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            int P_count = 0;                                                    //定义计数器
            foreach (Word.Paragraph paragraph in P_Document.Paragraphs)
            {
                Word.Range P_Range_temp = paragraph.Range;                      //得到段落文本范围
                foreach (char P_chr in P_Range_temp.Text)                       //遍历每一个字符
                {
                    P_count =                                                   //计数器开始计数
                        P_chr.ToString() != "\r" ?
                        rbtn_Blank.Checked ? ++P_count :P_chr.ToString() != " " ? ++P_count :P_count : P_count;
                }
            }
            ((Word._Application)G_wa.Application).Quit(                         //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                        //调用窗体线程
                (MethodInvoker)(() =>                                           //使用 Lambda 表达式
                {
                    MessageBox.Show(                                            //提示已经创建 Word
                        string.Format("{0}共{1}个字符",
                        rbtn_Blank.Checked ? "记空格" : "不记空格",P_count.ToString()), "提示！ ");
                    btn_Get.Enabled = true;                                     //启用统计按钮
                }));
        });
}
```

秘笈心法

心法领悟 432：合理使用三元运算符。

三元运算符又叫条件运算符（ ？：），它会根据布尔型表达式的值返回两个值中的一个。如果条件为 true，则计算第一表达式并以它的计算结果为准；如果为 false，则计算第二表达式并以它的计算结果为准。在程序的开发过程中，可以适当地使用三元运算符来代替 if 语句，这样会使代码更加简洁清晰。

实例 433 将一个 Word 文档分割为多个小的 Word 文档

光盘位置：光盘\MR\19\433　　高级　　趣味指数：★★★★★

实例说明

将一个 Word 文档分割为多个文档的实现原理非常简单，首先打开指定的 Word 文档并检查文档的信息，然后根据用户选择分割文档的方式来分割文档的内容，最后指定的 Word 文档将会被分割成多个小的 Word 文档。实例运行效果如图 19.16 所示。

实例运行中单击“开始分割”按钮，会根据用户选择的方式分割 Word 文档，并生成多个 Word 文档。分割前的 Word 文档如图 19.17 所示，分割后的 Word 文档如图 19.18～图 19.21 所示

图 19.16　将一个 Word 文档分割为多个小的 Word 文档

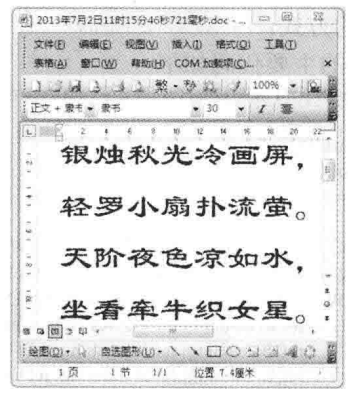


图 19.17　分割前的 Word 文档

图 19.18　分割后的第一个 Word 文档

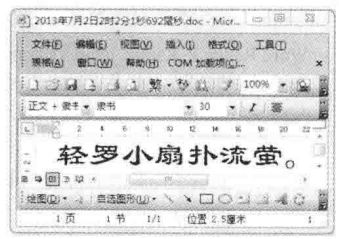


图 19.19　分割后的第二个 Word 文档

图 19.20　分割后的第三个 Word 文档

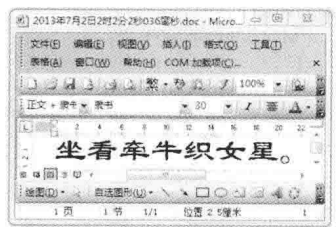


图 19.21　分割后的第 4 个 Word 文档

关键技术

本实例重点介绍怎样使用 Range 对象的 Copy 和 Paste 方法轻松地复制和粘贴 Word 文档中的内容，下面对本实例中用到的关键技术进行详细讲解。

Range 对象表示 Word 文档中文档内容的一段范围，要复制 Word 文档中的内容首先要使用 Range 对象的 Select 方法选中将要复制的 Range 范围，选中范围后就可以使用 Range 对象的 Copy 方法将选中范围复制到系统

剪贴板。现在，文档中指定的内容已经被放入剪贴板，在其他 Word 文档的 Range 对象中可以调用 Paste 方法方便地将剪贴板的内容粘贴到 Word 文档中。

提示：由于 Range 对象的 Copy 方法是将文档指定内容放入系统剪贴板，所以在执行 Copy 方法后，用户在操作系统中的其他文本编辑器中单击鼠标右键，并在弹出的快捷菜单中选择“粘贴”命令，可以将剪贴板中的内容手动粘贴到文本编辑器中。但是这样做是不友好的，在用户不知情的情况下在系统剪贴板中放入大量内容是不对的，所以在使用 Range 对象的 Copy 和 Paste 方法后不要忘记使用 Clipboard.Clear 方法清空系统剪贴板。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordSplit。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，更改 Text 属性为“将一个 Word 文档分割为多个小的 Word 文档”，向窗体中添加两个 TextBox 控件，分别用于显示 Word 文档的打开位置和显示文档的保存位置；添加 3 个 Button 控件，分别用于打开 Word 文档、选择文档的保存路径和分割文档。

（4）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    btn_split.Enabled = false;                                          //停用分割按钮
    ThreadPool.QueueUserWorkItem(                                       //开始线程池
        (pp) =>                                                         //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();     //创建应用程序对象
            object P_OpenFileDialog = G_OpenFileDialog.FileName;        //创建 object 对象
            Word.Document P_Document = G_wa.Documents.Open(             //打开 Word 文档
                ref P_OpenFileDialog, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            bool P_bl = false;                                          //定义布尔变量
            this.Invoke(                                                //调用窗体线程
                (MethodInvoker)(() =>                                   //使用 Lambda 表达式
                {
                    P_bl = cbox_Select.SelectedIndex == 0;
                }));
            if (P_bl)                                                   //判断用什么方式分割文档
            {
                foreach (Word.Paragraph Paragraph in G_wa.ActiveDocument.Paragraphs)
                {
                    Paragraph.Range.Select();                           //选择段落
                    Paragraph.Range.Copy();                             //将段落放入剪贴板
                    AddFile();                                          //将剪贴板中的数据放入文档
                }
            }
            else
            {
                Word.Range P_Range = G_wa.ActiveDocument.Content;       //得到文档区域
                int P_int_count = P_Range.Text.Length;                  //得到文档字符总长度
                int P_int_i = P_int_count / 100;                        //计算循环建立的文档次数
                if (P_int_i > 0)                                        //如果文字大于 100 个
                {
                    for (int i = 0; i < P_int_i; i++)                   //开始循环创建文档
                    {
                        object P_o1 = i == 0 ? 0 : i * 100 + 1;         //复制文档范围的开始部分
                        object P_o2 = i * 100 + 101;                    //复制文档范围的结尾部分
                        Word.Range P_Range_temp =                       //得到文档的范围
                            G_wa.ActiveDocument.Range(ref P_o1, ref P_o2);
                        P_Range.Select();                               //选中文档范围
```

```
                    P_Range_temp.Copy();                                    //将文档范围放入剪贴板
                    AddFile();                                              //将剪贴板中的数据放入文档
                }
                object P_o11 = P_int_i * 100 + 1;                           //复制文档范围的开始部分
                Word.Range P_Range_temp1 =                                  //得到文档的范围
                    G_wa.ActiveDocument.Range(ref P_o11, ref G_missing);
                P_Range.Select();                                           //选中文档范围
                P_Range_temp1.Copy();                                       //将内容放入剪贴板
                AddFile();                                                  //向新建文档中粘贴
            }
            else
            {
                Word.Range P_Range2 = G_wa.ActiveDocument.Content;          //得到文档区域
                P_Range.Select();                                           //选中文档范围
                P_Range2.Copy();                                            //将文档范围放入剪贴板
                AddFile();                                                  //将剪贴板中的数据放入文档
            }
        }
        ((Word._Application)G_wa.Application).Quit(                         //退出应用程序
            ref G_missing, ref G_missing, ref G_missing);
        this.Invoke(                                                        //调用窗体线程
            (MethodInvoker)(() =>                                           //使用 Lambda 表达式
            {
                Clipboard.Clear();                                          //清空剪贴板
                MessageBox.Show("完成！", "提示！");                          //弹出消息对话框
                btn_split.Enabled = true;                                   //启用分割按钮
            }));
    });
}
```

秘笈心法

心法领悟 433：轻松复制 Word 文档中的信息。

如果复制 Word 文档中的信息，首先得到 Word 文档中将要复制内容的范围，然后通过 Range 对象的 Select 方法选中将要复制的内容，调用 Range 对象的 Copy 方法将指定的内容复制到系统剪贴板，最后使用 Paste 方法将剪贴板的内容粘贴到指定位置。需要注意的是，在使用 Range 对象的 Copy 和 Paste 方法后不要忘记使用 Clipboard.Clear 方法清空剪贴板中的内容。

实例 434　将多个 Word 文档合并为一个 Word 文档

光盘位置：光盘\MR\19\434　　高级　　趣味指数：★★★★★

实例说明

将多个 Word 文档合并为一个 Word 文档，首先需要用户手动添加多个 Word 文档的集合，然后依次打开每个 Word 文档，将这些 Word 文档的全部内容复制到剪贴板并粘贴到新建的 Word 文档中。实例运行效果如图 19.22 所示。

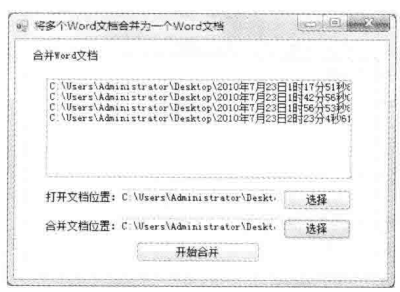


图 19.22　将多个 Word 文档合并为一个 Word 文档

实例运行中单击“选择”按钮，会将用户所选择的 Word 文档的路径放入文件列表中，后期可以遍历文件列表将多个 Word 文档合并到一个 Word 文档中，合并前的多个 Word 文档如图 19.23～图 19.26 所示。

图 19.23　合并前的第一个 Word 文档

图 19.24　合并前的第二个 Word 文档

图 19.25　合并前的第三个 Word 文档

图 19.26　合并前的第 4 个 Word 文档

实例运行中单击“开始合并”按钮，会将用户选择的多个 Word 文档合并为一个 Word 文档，合并后的 Word 文档如图 19.27 所示。

图 19.27　合并后的 Word 文档

关键技术

本实例重点介绍怎样将多个 Word 文档的内容合并为一个 Word 文档以及使用 Documents 对象的 Open 方法打开指定的 Word 文档，下面对本实例中用到的关键技术进行详细讲解。

说明：本实例中用到了 Range 对象的 Copy 和 Paste 方法分别复制和粘贴 Word 文档中的内容，关于 Copy 与 Paste 方法的详细讲解请参照实例 433 中的关键技术。

（1）合并 Word 文档

将多个 Word 文档的内容合并为一个 Word 文档的实现方法很简单，首先创建并打开一个新的 Word 文档用于存放多个文档的内容，然后根据用户指定的 Word 文档路径的集合依次打开每一个 Word 文档，并将文档中的

所有内容粘贴到新建的 Word 文档中。

（2）Documents 对象的 Open 方法

使用 Documents 对象的 Open 方法可以方便地打开 Word 文档。Open 方法的语法格式如下：

```
Document Open(ref object FileName, ref object ConfirmConversions, ref object ReadOnly, ref object AddToRecentFiles, ref object PasswordDocument,
ref object PasswordTemplate, ref object Revert, ref object WritePasswordDocument, ref object WritePasswordTemplate, ref object Format, ref object
Encoding, ref object Visible, ref object OpenAndRepair, ref object DocumentDirection, ref object NoEncodingDialog, ref object XMLTransform);
```

Open 方法中的参数说明如表 19.6 所示。

表 19.6　Open 方法中的参数说明

参　数	描　述
FileName	将要打开的 Word 文档名称及路径
ConfirmConversions	如果 Open 方法中 FileName 参数指定的不是 Word 格式文件，则需要将该参数设置为 true，表示显示文件转换对话框
ReadOnly	是否以只读方式打开文档，默认值为 false
AddToRecentFiles	是否将文件名称添加到文件菜单中的最近使用文件列表中，默认值为 true
PasswordDocument	如果文档有密码保护，可以使用此参数传递密码的字符串值，用于打开文档
PasswordTemplate	如果模板有密码保护，可以使用此参数传递密码的字符串值，用于打开模板
Revert	使用 Open 方法打开已经被打开的文档，如果此参数为 true 时，则放弃已打开文档中尚未保存的修改；如果此参数为 false，则激活已打开文档
WritePasswordDocument	如果文档有密码保护，可以使用此参数传递密码的字符串值，用于保存文档
WritePasswordTemplate	如果模板有密码保护，可以使用此参数传递密码的字符串值，用于保存模板
Format	WdOpenFormat 枚举类型参数，用于在打开文档时指定所使用的文件转换器
Encoding	Office.MsoEncoding 枚举类型参数，用于指定打开文档时所使用的代码页或字符集
Visible	是否在可见窗口中打开文档，如果为 true，则在可见窗口中打开，默认为 true
OpenAndRepair	是否尝试修复损坏文档，true 为修复
DocumentDirection	WdDocumentDirection 枚举类型参数，用于指定已打开文档中文本的水平流动方式
NoEncodingDialog	如果为 true，表示在不能确定文档编码方式时，阻止显示编码对话框
XMLTransform	XML 转换格式

注意： Documents 的 Add 方法中只有第一个参数 FileName 是必选的，其他参数都是可选参数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordMerge。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 中的 Name 属性为 Frm_Main，更改 Text 属性为“将多个 Word 文档合并为一个 Word 文档”，向窗体中添加一个 ListBox 控件，用于显示将要合并的 Word 文档的集合；添加 3 个 Button 控件，分别用于添加将要合并的 Word 文档、选择合并文档的保存位置和开始合并 Word 文档。

（4）程序的主要代码如下：

```
private void btn_split_Click(object sender, EventArgs e)
{
    btn_Merge.Enabled = false;                                          //启用合并按钮
    ThreadPool.QueueUserWorkItem(                                       //开始线程池
        (pp) =>                                                         //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();     //创建应用程序对象
            Word.Document P_MainDocument =                              //新建合并文档对象
                G_wa.Documents.Add(ref G_missing, ref G_missing,
                ref G_missing, ref G_missing);
```

```
            foreach (string P_Str in G_Str_Files)                                    //遍历文档的集合
            {
                object P_strs = P_Str;                                               //创建 object 对象
                Word.Document P_Document = G_wa.Documents.Open(                      //打开 Word 文档
                    ref P_strs, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing);
                Word.Range P_Range_temp =                                            //得到文档全部范围
                    P_Document.Range(ref G_missing, ref G_missing);
                P_Range_temp.Select();                                               //选择文档全部范围
                P_Range_temp.Copy();                                                 //复制文档全部范围
                Word.Range P_Range_temp2 =                                           //得到文档的范围
                    P_MainDocument.Range(ref G_missing, ref G_missing);
                object P_end= Word.WdCollapseDirection.wdCollapseEnd;                //创建 object 对象
                P_Range_temp2.Collapse(ref P_end);                                   //折叠文档范围
                P_Range_temp2.Select();                                              //选择文档的最后位置
                P_Range_temp2.Paste();                                               //粘贴文档内容
                ((Word._Document)P_Document).Close(ref G_missing, ref G_missing,     //关闭文档
                    ref G_missing);
            }
            object P_SavePath=G_SaveFileDialog.FileName;                             //创建 object 对象
            P_MainDocument.SaveAs(                                                   //保存合并后的文档
                ref P_SavePath, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                              //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                             //调用窗体线程
                (MethodInvoker)(() =>                                                //使用 Lambda 表达式
                {
                    Clipboard.Clear();                                               //清空剪贴板
                    MessageBox.Show("成功合并 Word 文档！", "提示！");                //弹出消息对话框
                    btn_Merge.Enabled = true;                                        //停用合并按钮
                }));
        });
    }
```

秘笈心法

心法领悟 434：以只读方式打开 Word 文档。

本实例已经介绍了怎样使用 Documents 对象的 Open 方法方便地打开 Word 文档，如果想使用只读的方式打开 Word 文档，可以设置 Open 方法的第三个参数为 true 使文档以只读方式打开。

实例 435　批量替换 Word 文档中指定的字符串

光盘位置：光盘\MR\19\435　　高级　　趣味指数：★★★★★

实例说明

在字符串操作中可以使用Replace方法方便地替换字符串中指定的内容。本实例中将会使用 Range 对象的 Find 属性方便地在 Word 文档中查找指定的字符串，也可以设置 Find 对象的 Replacement 属性方便地替换 Word 文档中的文本内容。实例运行效果如图 19.28 所示。

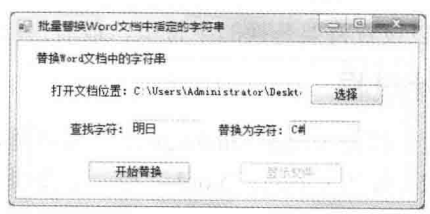


图 19.28　批量替换 Word 文档中指定的字符串

实例运行中单击“开始替换”按钮，会将用户选择 Word 文档内所指定的字符串替换为相应的字符串。替换字符串前的 Word 文档如图 19.29 所示，替换字符串后的 Word 文

档如图 19.30 所示。

图 19.29　替换字符串前的 Word 文档

图 19.30　替换字符串后的 Word 文档

关键技术

本实例重点介绍怎样通过 Find 对象的属性和方法批量替换 Word 文档中指定的字符串，下面对本实例中用到的关键技术进行详细讲解。

如果替换 Word 文档中指定的字符串首先要得到 Find 对象，可以使用 Document 对象的 Range 方法得到 Range 对象，通过 Range 对象的 Find 属性会返回 Find 对象。Find 对象的属性和方法的具体应用如图 19.31 所示。

调用Document对象的Range方法得到Range对象

```
Word.Range P_Range = P_Document.Range(ref G_Missing, ref G_Missing);
Word.Find P_Find = P_Range.Find;          ——▶ Range对象的Find属性返回Find对象
P_Find.Text = "a";                       ——▶ Find对象的Text属性设置将要查找到的字符串
P_Find.Replacement.Text = "d";           ——▶ Replacement对象的Text属性设置将要替换的字符串
object P_Replace = Word.WdReplace.wdReplaceAll;  ——▶ Execute方法的参数，替换所有查找到的文本
P_Find.Execute(                          ——▶ 执行Execute方法开始替换
    ref G_Missing, ref G_Missing, ref G_Missing,
    ref G_Missing, ref G_Missing, ref G_Missing, ref G_Missing,
    ref G_Missing, ref G_Missing, ref G_Missing, ref P_Replace,
    ref G_Missing, ref G_Missing, ref G_Missing, ref G_Missing);
```

图 19.31　Find 对象的属性和方法的具体应用

从图 19.31 中可以看到，首先使用 Document 对象的 Range 方法得到了 Range 对象，然后通过 Range 对象的 Find 属性得到 Find 对象。通过 Find 对象的 Text 属性可以设置将要查找的字符串，通过 Replacement 对象的 Text 属性可以设置将要替换的字符串，最后执行 Execute 方法开始替换操作。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordReplace。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“批量替换 Word 文档中指定的字

符串”，向窗体中添加 3 个 TextBox 控件，分别用于显示打开 Word 文档路径、定义查找字符串和定义替换字符串；添加 3 个 Button 控件，分别用于选择 Word 文档、替换 Word 文档内容和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_Begin_Click(object sender, EventArgs e)
{
    btn_Begin.Enabled = false;                                          //停用替换按钮
    ThreadPool.QueueUserWorkItem(                                       //开始线程池
        (o) =>                                                          //使用 Lambda 表达式
        {
            G_WordApplication = new Microsoft.Office.Interop.Word.Application();   //创建 Word 应用程序对象
            object P_FilePath = G_OpenFileDialog.FileName;              //创建 object 对象
            Word.Document P_Document = G_WordApplication.Documents.Open(   //打开 Word 文档
                ref P_FilePath, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing);
            Word.Range P_Range =                                        //得到文档范围
                P_Document.Range(ref G_Missing, ref G_Missing);
            Word.Find P_Find = P_Range.Find;                            //得到 Find 对象
            this.Invoke(                                                //在窗体线程中执行
                (MethodInvoker)(() =>                                   //使用 Lambda 表达式
                {
                    P_Find.Text = txt_Find.Text;                        //设置查找的文本
                    P_Find.Replacement.Text = txt_Replace.Text;         //设置替换的文本
                }));
            object P_Replace = Word.WdReplace.wdReplaceAll;             //定义替换方式对象
            bool P_bl = P_Find.Execute(                                 //开始替换
                ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing, ref G_Missing,
                ref G_Missing, ref G_Missing, ref G_Missing, ref P_Replace,
                ref G_Missing, ref G_Missing, ref G_Missing, ref G_Missing);
            G_WordApplication.Documents.Save(                           //保存文档
                ref G_Missing, ref G_Missing);
            ((Word._Document)P_Document).Close(                         //关闭文档
                ref G_Missing, ref G_Missing, ref G_Missing);
            ((Word._Application)G_WordApplication).Quit(                //退出 Word 应用程序
                ref G_Missing, ref G_Missing, ref G_Missing);
            this.Invoke(                                                //在窗体线程中执行
                (MethodInvoker)(() =>                                   //使用 Lambda 表达式
                {
                    if (P_bl)                                           //查看是否找到并替换
                    {
                        MessageBox.Show("找到字符串并替换", "提示！");   //弹出消息对话框
                        btn_Display.Enabled = true;                     //启用显示文件按钮
                    }
                    else
                    {
                        MessageBox.Show("没有找到字符串", "提示！");     //弹出消息对话框
                    }
                    btn_Begin.Enabled = true;                           //启用开始替换按钮
                }));
        });
}
```

秘笈心法

心法领悟 435：使用 Replace 方法简单地代替 Find 对象的方法实现批量替换字符串。

在 Word 文档操作中，可以使用 Find 对象的属性和方法批量替换 Word 文档中指定的字符串，也可以使用字符串对象的 Replace 方法简单地代替 Find 对象的方法，使用方法如下：

```
P_Range.Text = P_Range.Text.Replace("查找的字符串","替换的字符串");
```

19.2 读取外部数据到 Word 中

实例 436	读取文本文件到 Word 中 光盘位置：光盘\MR\19\436	中级 趣味指数：★★★★★

实例说明

在实际的工作过程中，经常需要将文本文件的信息复制到 Word 文档中，首先打开文本文件选中需要的文本内容并复制文本内容，然后将剪贴板内的文本粘贴到 Word 中。本实例将会使用 C#实现类似的功能，将文本文件的内容读取到 Word 文档中，首先使用 StreamReader 读取文本文件内容，然后将文本内容放入 Word 文档。实例运行效果如图 19.32 所示。

实例运行中单击“创建 Word 文档”按钮，会将用户选择的文本文件中的内容读取到新创建的 Word 文档中。文本文件的内容如图 19.33 所示，新创建的 Word 文档的内容如图 19.34 所示。

图 19.32 读取文本文件到 Word 中

图 19.33 文本文件的内容

图 19.34 新创建的 Word 文档的内容

关键技术

本实例重点介绍怎样使用 StreamReader 读取文本文件内容，并写入 Word 文档，下面对本实例中用到的关键技术进行详细讲解。

说明：关于向 Word 文档中写入文字的详细讲解请参见实例 427 中的关键技术。

在文件操作中可以使用 StreamReader 轻松读取文本文件的内容，StreamReader 的使用方法如图 19.35 所示。

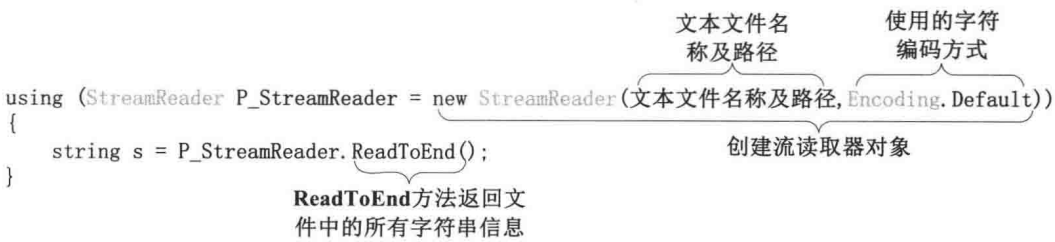


图 19.35 StreamReader 的使用方法

从图 19.35 中可以看到，首先创建文件读取器对象，然后调用该对象的 ReadToEnd 方法轻松地读取文本文件内的字符串信息。读取字符串信息后便可以将之写入 Word 文档。

注意：using 关键字后面小括号内创建的对象必须要实现 IDisposable 接口，或者此对象的基类已经实现了 IDisposable 接口。当 using 语句执行完毕后，会将小括号内创建的对象转换为 IDisposable 接口对象，并执行 Disposable 方法释放资源。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TxtToWord。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取文本文件到 Word 中”，向窗体中添加两个 TextBox 控件，分别用于显示文本文件路径和 Word 文档保存路径；添加 4 个 Button 控件，分别用于选择文本文件位置、选择 Word 文档保存位置、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                          //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                     //使用线程池
        (P_temp) =>                                                   //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                            //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                  //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;
            using (StreamReader P_StreamReader =                      //创建流读取器对象
                new StreamReader(G_OpenFileDialog.FileName,Encoding.Default))
            {
                P_Range.Text = P_StreamReader.ReadToEnd();
            }
            G_str_path = string.Format(                               //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                              //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(               //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                              //开始执行窗体线程
                (MethodInvoker)(() =>                                 //使用 Lambda 表达式
                {
                    btn_Display.Enabled = true;                       //启用显示按钮
                    MessageBox.Show("添加文档成功！", "提示！");        //弹出消息对话框
                }));
        });
}
```

秘笈心法

心法领悟 436：轻松回收 StreamReader 对象资源。

程序设计过程中，可以手动调用 StreamReader 对象的 Close 方法回收文件资源，但是使用 using 关键字则更为简洁友好。当 using 关键字下面的代码执行完毕后，系统会自动将 using 小括号中创建的 StreamReader 对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 437	读取多个文本文件到同一 Word 文档中	中级
	光盘位置：光盘\MR\19\437	趣味指数：★★★★★

实例说明

读取多个文本文件到同一个 Word 文档中的方法与读取单个文本文件到 Word 文档基本相同，首先根据用户

选择的文本文件集合使用 StreamReader 读取每一个文本文件的文本内容，然后将所有文本文件的内容写入 Word 文档。实例运行效果如图 19.36 所示。

图 19.36　读取多个文本文件到同一 Word 文档中

实例运行中单击“浏览”按钮，会将用户所选择文本文件的路径放入文件列表中，后期可以遍历文件列表使用 StreamReader 依次读取每一个文本文件的内容到 Word 文档中，多个文本文件如图 19.37～图 19.40 所示。

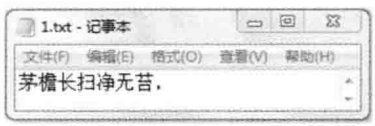
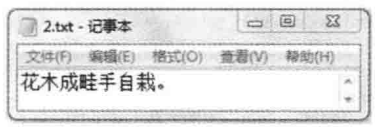


图 19.37　第一个文本文件



19.38　第二个文本文件

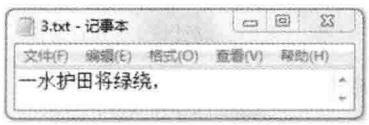


图 19.39　第三个文本文件

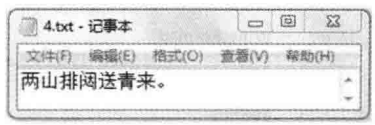


图 19.40　第 4 个文本文件

实例运行中单击“创建 Word 文档”按钮，首先会创建一个 Word 文档，然后将多个文本文件的内容写入文本文档。被写入多个文本文件内容的 Word 文档如图 19.41 所示。

图 19.41　被写入多个文本文件内容的 Word 文档

关键技术

本实例主要用到了 StreamReader 对象依次读取用户指定的文本文件集合中的字符串信息，并将字符串信息写入到 Word 文档。

说明：（1）关于读取文本文件内容的详细讲解请参见实例 436 中的关键技术。

（2）关于向 Word 文档中写入文字的详细讲解请参见实例 427 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Multi-TxtToWord。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取多个文本文件到同一 Word 文档中”，向窗体中添加一个 ListBox 控件，用于显示文本文件的集合；添加 4 个 Button 控件，分别用于向 ListBox 中添加文本文件路径、选择 Word 文档保存路径、创建 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                          //停用创建按钮
    ThreadPool.QueueUserWorkItem(                                     //使用线程池
        (P_temp) =>                                                   //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                            //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                  //创建新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;            //得到文档段落范围
            foreach (string s in G_List_FileName)                     //遍历文件集合
            {
                using (StreamReader P_StreamReader =                  //创建文件读取器对象
                    new StreamReader(s, Encoding.Default))
                {
                    P_Range.Text += P_StreamReader.ReadToEnd();       //读取文本到 Word 文档
                }
            }
            G_str_path = string.Format(                               //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                              //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(               //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                              //开始执行窗体线程
                (MethodInvoker)(() =>                                 //使用 Lambda 表达式
                {
                    btn_Display.Enabled = true;                       //启用显示按钮
                    MessageBox.Show("添加文档成功！", "提示！");        //弹出消息对话框
                }));
        });
}
```

秘笈心法

心法领悟 437：合理使用 StreamReader 对象的 ReadToEnd 方法。

使用 StreamReader 对象的 ReadToEnd 方法可以方便地一次性全部读取出文本文件的所有文本信息，适当地使用此方法会使文件操作更加便捷。

实例 438　读取 Access 数据到 Word 并对数据列进行计算　中级

光盘位置：光盘\MR\19\438　　趣味指数：★★★★★

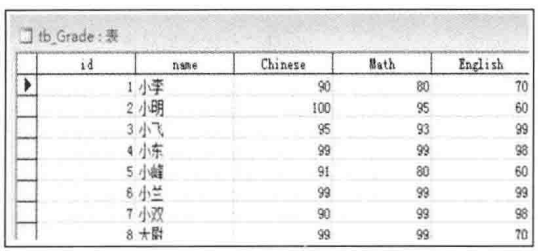
实例说明

由于 Access 数据库的使用和部署非常方便，所以在小型系统开发中经常会用到 Access。在读取 Access 中的数据时，首先会将 Access 中的数据取出，然后将取出的数据放入集合中，此时，可以使用绑定等方式在窗体中显示集合中的信息。本实例将会使用 C#得到 Access 中的数据集合，然后将集合中的数据写入到 Word 文档的

表格中。实例运行效果如图 19.42 所示。

图 19.42　读取 Access 数据到 Word 并对数据列进行计算

实例运行中单击“读取 Access 数据并写入 Word 文档”按钮，此时会读取 Access 中的数据到集合中，然后创建 Word 文档并向文档中插入表格，最后将集合中的数据依次放入表格中并统计表格中的数据。Access 中的数据如图 19.43 所示，被写入数据的 Word 文档如图 19.44 所示。


tb_Grade：表

id	name	Chinese	Math	English
1	小李	90	80	70
2	小明	100	95	60
3	小飞	95	93	99
4	小东	99	99	98
5	小峰	91	80	60
6	小兰	99	99	99
7	小双	90	99	98
8	大尉	99	99	70

图 19.43　Access 中的数据

图 19.44　被写入数据的 Word 文档

关键技术

本实例重点介绍怎样方便地读取 Access 数据库中的内容，并将内容以表格的形式轻松地写入 Word 文档。下面对本实例中用到的关键技术进行详细讲解。

（1）读取 Access 中的数据

要将 Access 中的数据以表格的形式写入到 Word 文档，首先要取出 Access 中的数据到数据集合，然后再将数据集合中的数据以表格的形式写入到 Word 文档。

注意：如果操作 Access 数据库，那么在代码中要引用 OleDb 命名空间，如 using System.Data.OleDb，下文再次涉及 OLEDB 对象的使用时，将不再提示。

在操作数据库时，可以使用 OleDbDataAdapter（数据适配器）对象填充 DataTable，然后遍历 DataTable 将其内部数据放入数据集合中，最后将此数据集合中的数据插入到 Word 文档的表格中。

（2）向 Word 文档中插入表格和数据

向 Word 文档表格中插入数据的实现方法很简单，表格对象的 Cell 方法提供了两个参数并返回一个 Range 对象。方法中的两个参数分别为表格对象中的行与列的索引。需要注意的是，参数中行与列的索引都是从 1 开始的，根据两个参数可以找到并返回表格中的单元格的 Range 对象，并可以根据 Range 对象的 Text 属性获取或设置单元格的信息。

说明：关于向 Word 文档中插入表格的详细讲解请参见实例 429 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AccessToWord。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取 Access 数据到 Word 并对数据列进行计算”，向窗体中添加一个 TextBox 控件，用于显示 Word 文档保存路径；添加 3 个 Button 控件，分别用于选择 Word 文档保存路径、读取 Access 数据到 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                                    //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                               //开始线程池
        (pp) =>                                                                 //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();             //创建应用程序对象
            object P_obj = "Normal.dot";                                        //定义文档模板
            Word.Document P_wd = G_wa.Documents.Add(                            //创建新文档
                ref P_obj, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Range(    ref G_missing, ref G_missing);  //得到文档范围
            DataTier P_DataTier = new DataTier();                               //创建数据层对象
            List<InstanceClass> P_List_InstanceClass = P_DataTier.GetMessage(); //得到数据集合
            object o1=Word.WdDefaultTableBehavior.wdWord8TableBehavior;         //设置文档中表格的格式
            object o2 = Word.WdAutoFitBehavior.wdAutoFitWindow;                 //设置文档中表格的格式
            Word.Table P_WordTable = P_Range.Tables.Add(P_Range,                //在文档中添加表格
                P_List_InstanceClass.Count + 2, 5, ref o1, ref o2);
            P_WordTable.Cell(1, 1).Range.Text = "ID";                           //向表格中添加信息
            P_WordTable.Cell(1, 2).Range.Text = "姓名";                         //向表格中添加信息
            P_WordTable.Cell(1, 3).Range.Text = "语文成绩";                     //向表格中添加信息
            P_WordTable.Cell(1, 4).Range.Text = "数学成绩";                     //向表格中添加信息
            P_WordTable.Cell(1, 5).Range.Text = "英语成绩";                     //向表格中添加信息
            for (int i = 2; i < P_List_InstanceClass.Count+2; i++)
            {
                P_WordTable.Cell(i, 1).Range.Text =                             //向表格中添加信息
                    P_List_InstanceClass[i - 2].id.ToString();
                P_WordTable.Cell(i, 2).Range.Text =                             //向表格中添加信息
                    P_List_InstanceClass[i - 2].Name;
                P_WordTable.Cell(i, 3).Range.Text =                             //向表格中添加信息
                    P_List_InstanceClass[i - 2].Chinese.ToString();
                P_WordTable.Cell(i, 4).Range.Text =                             //向表格中添加信息
                    P_List_InstanceClass[i - 2].Math.ToString();
                P_WordTable.Cell(i, 5).Range.Text =                             //向表格中添加信息
                    P_List_InstanceClass[i - 2].English.ToString();
            }
            float P_Chinese = 0;                                                //定义变量计算数据列
            float P_Math = 0;                                                   //定义变量计算数据列
            float P_English = 0;                                                //定义变量计算数据列
            P_List_InstanceClass.
                ForEach((Instance)=>                                            //使用 Lambda 表达式
                {
                    P_Chinese += ((InstanceClass)Instance).Chinese;             //计算数据列
                    P_Math += ((InstanceClass)Instance).Math;                   //计算数据列
                    P_English += ((InstanceClass)Instance).English;             //计算数据列
                });
            P_WordTable.Cell(P_List_InstanceClass.Count + 2,                    //向表格中添加信息
                1).Range.Text = "科目总成绩";
            P_WordTable.Cell(P_List_InstanceClass.Count + 2, 3).                //向表格中添加信息
                Range.Text = P_Chinese.ToString();
            P_WordTable.Cell(P_List_InstanceClass.Count + 2, 4).                //向表格中添加信息
                Range.Text = P_Math.ToString();
            P_WordTable.Cell(P_List_InstanceClass.Count + 2, 5).                //向表格中添加信息
                Range.Text = P_English.ToString();
            G_str_path = string.Format(                                         //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                                        //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
```

```
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                          //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                        //调用窗体线程
                (MethodInvoker)(() =>                                           //使用 Lambda 表达式
                {
                    MessageBox.Show("成功创建 Word 文档！", "提示！");           //弹出消息对话框
                    btn_display.Enabled = true;                                 //启用显示按钮
                }));
        });
}
```

秘笈心法

心法领悟 438：使用 Cell 方法轻松地向表格中添加信息。

表格对象的 Cell 方法提供了两个参数，分别为表格中的行与列的索引，行与列的索引都是从 1 开始的，根据行与列的索引可以找到单元格的 Range 对象，最后通过 Range 对象的 Text 属性向表格中指定的单元格添加文本信息。

实例 439　将 SQL Server 数据查询结果输出到 Word　　中级

光盘位置：光盘\MR\19\439　　趣味指数：★★★★★

实例说明

SQL Server 数据库中存储着大量的数据，用户可以使用 C#从数据库中读取出数据信息，并加以处理。本实例将会介绍一种方法，将在 SQL Server 中查询所得到的数据输出到 Word 文档。首先得到从 SQL Server 数据库中查询到的数据集合，然后将数据集合中的信息写入到 Word 文档中。实例运行效果如图 19.45 所示。

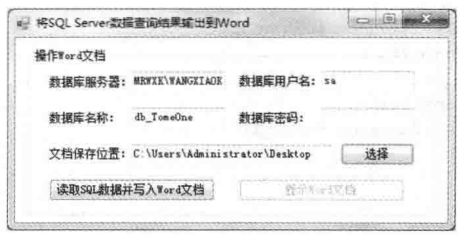


图 19.45　将 SQL Server 数据查询结果输出到 Word

实例运行中单击“读取 SQL 数据并写入 Word 文档”按钮，此时会读取 SQL Server 中的数据到新创建的 Word 文档中。SQL Server 中的数据如图 19.46 所示，被写入数据的 Word 文档如图 19.47 所示。

表 - dbo.tb_Message　摘要

	id	message
▶	1	C#程序开发范例宝典
	2	C#实战宝典
	3	C#视频学
	4	C#编程宝典
	5	C#编程词典
	6	C#开发经验技巧宝典
	7	C#入门模式

图 19.46　SQL Server 中的数据

图 19.47　被写入数据的 Word 文档

关键技术

本实例重点介绍怎样将 SQL Server 数据查询结果输出到 Word，下面对本实例中用到的关键技术进行详细讲解。

注意： 如果操作 SQL Server 数据库，那么在代码中要引用 SQL 命名空间，如 using System.Data.SqlClient，下文再次涉及 SQL Server 的使用时，将不再提示。

要将 SQL Server 中的数据插入到 Word，首先可以使用 SqlDataAdapter 对象填充 DataTable，然后遍历 DataTable 中的数据并添加到数据集合，最后将数据集合中的数据写入到 Word 文档。

注意： 关于向 Word 文档中写入文字的详细讲解请参见实例 427 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SQLToWord。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 SQL Server 数据查询结果输出到 Word”，向窗体中添加 5 个 TextBox 控件，分别用于填写 SQL 服务器信息和显示 Word 文档保存路径；添加 3 个 Button 控件，分别用于选择 Word 文档保存路径、读取 SQL 数据并写入 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                    //停用新建按钮
    ThreadPool.QueueUserWorkItem(                               //开始线程池
        (pp) =>                                                 //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();   //创建应用程序对象
            object P_obj = "Normal.dot";                        //定义文档模板
            Word.Document P_wd = G_wa.Documents.Add(            //创建 Word 文档
                ref P_obj, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Range(                    //得到文档范围
                ref G_missing, ref G_missing);
            string P_Server = string.Empty;                     //定义字符串变量用于存放服务器信息
            string P_DataBase = string.Empty;                   //定义字符串变量用于存放数据库名称
            string P_UserName = string.Empty;                   //定义字符串变量用于存放用户名
            string P_PassWord = string.Empty;                   //定义字符串变量用于存放密码
            this.Invoke(
                (MethodInvoker)(() =>
                {
                    P_Server = txt_Server.Text;                 //得到数据库服务器信息
                    P_DataBase = txt_DataBase.Text;             //得到数据库名称
                    P_UserName = txt_UserName.Text;             //得到数据库用户名
                    P_PassWord = txt_PassWord.Text;             //得到数据库密码
                }));
            DataTier P_DataTier = new DataTier(                 //创建数据层对象
                P_Server, P_DataBase, P_UserName, P_PassWord);
            List<string> P_List_Str = P_DataTier.GetMessage();  //得到数据集合
            foreach (string P_str in P_List_Str)                //遍历数据集合
            {
                P_Range.Text += P_str;                          //向文档中添加数据
            }
            G_str_path = string.Format(                         //计算文件保存路径
                @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
                DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
            P_wd.SaveAs(                                        //保存 Word 文件
                ref G_str_path,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
```

```
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                    //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                   //调用窗体线程
                (MethodInvoker)(() =>                                      //使用 Lambda 表达式
                {
                    MessageBox.Show("成功创建 Word 文档！", "提示！");      //弹出消息对话框
                    btn_display.Enabled = true;                            //启用显示按钮
                }));
        });
    }
```

秘笈心法

心法领悟 439：使用 foreach 循环方便地将字符串集合中的信息写入 Word 文档。

现在，可以使用 Range 对象的 Text 属性轻松地向 Word 文档中写入文字信息，那么，怎样将字符串集合中的字符信息写入 Word 文档呢？可以使用 foreach 遍历集合中的每一个字符串，并使用 Range 对象的 Text 属性将字符串信息添加到 Word 文档中。

实例 440	将 XML 中的数据读取到 Word 文档中	中级
	光盘位置：光盘\MR\19\440	趣味指数：★★★★★

实例说明

在程序设计过程中，经常用到 XML（eXtensible Markup Language，可扩展标记语言），XML 是一种严谨的描述数据的语言，在 C#中可以使用 XmlReader 和 XmlDocument 对象方便地读取 XML 文件的内容。在实例中将会使用 XmlDocument 对象读取指定的 XML 文件内容，并将 XML 的文本内容写入到 Word 文档中。实例运行效果如图 19.48 所示。

图 19.48　将 XML 中的数据读取到 Word 文档中

实例运行中单击“读取 XML 内容并写入 Word 文档”按钮，此时会读取 XML 文档的内容到新创建的 Word 文档中。XML 文档中的内容如图 19.49 所示，被写入 XML 信息的 Word 文档如图 19.50 所示。

```
<?xml version="1.0" standalone="yes" ?>
- <member>
  - <person>
      <name>老张</name>
      <sex>男</sex>
      <age>35</age>
      <message>玉经磨琢多成器</message>
    </person>
  - <person>
      <name>会东</name>
      <sex>男</sex>
      <age>32</age>
      <message>剑拔沉埋更倚天</message>
    </person>
</member>
```

图 19.49　XML 文档中的内容

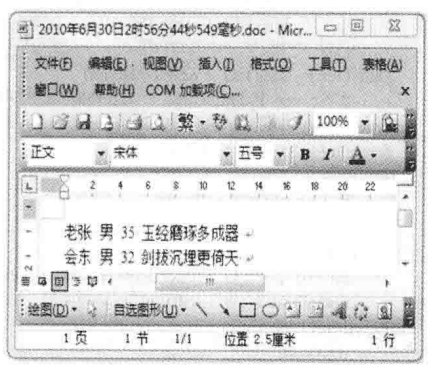


图 19.50　被写入 XML 信息的 Word 文档

关键技术

本实例重点介绍怎样使用 XmlDocument 对象读取 XML 文档，并将文档的文本内容写入 Word 文档。下面对本实例中用到的关键技术进行详细讲解。

引用了相应的命名空间后，接下来可以使用 XmlDocument 对象轻松地读取 XML 文档中的文本内容。XmlDocument 对象的使用方法如图 19.51 所示。

```
XmlDocument P_XML = new XmlDocument();                          ▶创建XmlDocument对象
P_XML.Load("test.xml");                                         ▶加载XML文档
XmlElement P_XmlElement = P_XML.DocumentElement;                ▶得到XML根节点
XmlNodeList P_XmlNodeList = P_XmlElement.ChildNodes;            ▶得到XML子节点集合
string P_Str_Message = string.Empty;                            ▶创建空字符串对象
foreach (XmlNode xn in P_XmlNodeList)                           ▶遍历所有子节点
{
    foreach (XmlNode xn2 in xn.ChildNodes)                      ▶遍历子节点中的节点
    {
        P_Str_Message += xn2.InnerText+" ";                     ▶得到节点字符串的信息
    }
    P_Str_Message += "\r\n";                                    ▶添加换行符
}
```

图 19.51　使用 XmlDocument 对象读取 XML 文档信息

从图 19.51 中可以看到，首先创建 XmlDocument 对象，通过该对象的 Load 方法载入 XML 文档，然后使用 DocumentElement 属性返回 XmlElement 根节点对象，通过根节点对象的 ChildNodes 属性得到根节点中所有子节点的集合，最后遍历子节点集合中的每一个子节点，从每一个子节点所包含的节点集合中得到相应的字符串信息。

📖 **说明**：关于向 Word 文档中写入文字的详细讲解请参见实例 427 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 XMLToWord。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 XML 中的数据读取到 Word 文档中”，向窗体中添加一个 TextBox 控件，用于显示 Word 文档保存路径；添加 3 个 Button 控件，分别用于选择 Word 文档保存路径、读取 XML 内容并写入 Word 文档和显示 Word 文档。

（4）程序的主要代码如下：

```
private void btn_New_Click(object sender, EventArgs e)
{
    btn_New.Enabled = false;                                        //停用新建按钮
    ThreadPool.QueueUserWorkItem(                                   //使用线程池
        (P_temp) =>                                                 //使用 Lambda 表达式
        {
            G_wa = new Word.Application();                          //创建 Word 应用程序对象
            Word.Document P_wd = G_wa.Documents.Add(                //建立新文档
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_wd.Paragraphs[1].Range;          //得到文档段落范围
            XmlDocument P_XML = new XmlDocument();                  //创建 XML 对象
            P_XML.Load("test.xml");                                 //打开 XML 文档
            XmlElement P_XmlElement = P_XML.DocumentElement;        //得到 XML 根节点
            XmlNodeList P_XmlNodeList = P_XmlElement.ChildNodes;    //得到子节点集合
            string P_Str_Message = string.Empty;                    //创建空字符串对象
            foreach (XmlNode xn in P_XmlNodeList)                   //遍历所有子节点
            {
                foreach (XmlNode xn2 in xn.ChildNodes)              //遍历子节点中的节点
```

```
            {
                P_Str_Message += xn2.InnerText + "\r\n";                    //得到字符串信息
            }
        }
        P_Range.Text = P_Str_Message;                                       //向 Word 中添加字符串
        G_str_path = string.Format(                                         //计算文件保存路径
            @"{0}\{1}", G_FolderBrowserDialog.SelectedPath,
            DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 s 分 m 秒 fff 毫秒") + ".doc");
        P_wd.SaveAs(                                                        //保存 Word 文件
            ref G_str_path,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing);
        ((Word._Application)G_wa.Application).Quit(                         //退出应用程序
            ref G_missing, ref G_missing, ref G_missing);
        this.Invoke(                                                        //开始执行窗体线程
            (MethodInvoker)(() =>                                           //使用 Lambda 表达式
            {
                btn_Display.Enabled = true;                                 //启用显示按钮
                MessageBox.Show("成功创建 Word 文档！", "提示！");           //弹出消息对话框
            }));
    });
}
```

秘笈心法

心法领悟 440：使用 XmlReader 可以轻松读取 XML 文档的内容。

使用 XmlDocument 可以通过节点和元素的方式轻松地读取 XML 文档中的信息，同样，也可以使用 XmlReader 通过游标的方式读取 XML 文档中的信息。使用 XmlReader 演示的代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    XmlReader P_xr = XmlReader.Create(@"XML 文档路径");                     //创建 XmlReader 对象
    string P_str = string.Empty;                                            //创建空字符串对象
    while (P_xr.Read())                                                     //读取下一个节点
    {
        if (P_xr.NodeType == XmlNodeType.Text)                              //判断节点类型
            P_str += P_xr.Value;                                            //添加字符串信息
    }
    MessageBox.Show(P_str,"");                                              //弹出消息对话框
}
```

19.3 将 Word 中的数据进行输出处理

实例 441 读取 Word 数据到文本文件

光盘位置：光盘\MR\19\441　　中级　　趣味指数：★★★★★

实例说明

前面曾经做过一个很有趣的实例，将文本文件的内容读取到 Word 文档中，本实例的操作与之相反，即将演示读取 Word 文档中的文本内容并写入文本文件。首先要得到用户所选择 Word 文档的文本内容，然后使用 StreamWriter 将文本内容写入文本文件。实例运行效果如图 19.52 所示。

图 19.52　读取 Word 数据到文本文件

实例运行中单击“读取数据到文本文件”按钮，此时会读取 Word 文档的文本信息到新创建的文本文

件中。Word 文档中的文本信息如图 19.53 所示，被写入文本信息的文本文件如图 19.54 所示。

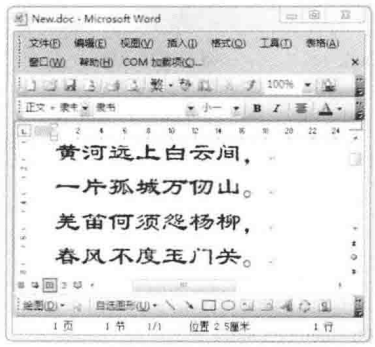


图 19.53　Word 文档中的文本信息

图 19.54　被写入文本信息的文本文件

关键技术

本实例重点介绍怎样使用 StreamWriter 对象的 Write 方法将 Word 文档中的文本内容写入文本文件，下面对本实例中用到的关键技术进行详细讲解。

使用 StreamWriter 对象的 Write 方法可以轻松地向文本文件中写入文本内容。StreamWriter 对象的使用方法如图 19.55 所示。

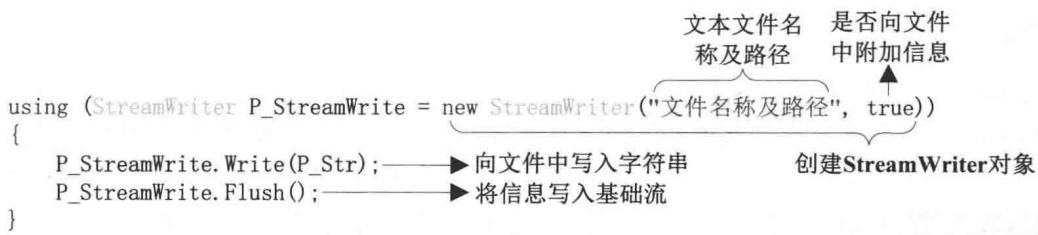


图 19.55　使用 StreamWriter 对象向文本文件中写入文本信息

从图 19.55 中可以看到，首先创建了 StreamWriter 对象，StreamWriter 构造方法中有两个参数，第一个字符串类型的参数指明了将要被写入文本信息的文件名称，第二个布尔类型的参数说明是否向文本文件中附加信息。创建 StreamWriter 对象后可以调用其 Write 方法将字符串信息写入流，最后调用 StreamWriter 对象的 Flush 方法清理缓冲区信息并将数据写入基础流。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordToTxt。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取 Word 数据到文本文件”，向窗体中添加两个 TextBox 控件，分别用于显示打开 Word 文档的路径和保存文本文件路径；添加 3 个 Button 控件，分别用于选择将要打开的 Word 文档、选择文本文件保存的路径和读取 Word 文档内的数据到文本文件。

（4）程序的主要代码如下：

```
private void btn_Get_Click(object sender, EventArgs e)
{
    btn_Get.Enabled = false;                                          //启用创建按钮
    ThreadPool.QueueUserWorkItem(                                     //开始线程池
        (pp) =>                                                       //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();   //创建应用程序对象
            object P_OpenFileDialog = G_OpenFileDialog.FileName;      //创建 object 对象
            Word.Document P_Document = G_wa.Documents.Open(           //打开 Word 文档
```

```
                ref P_OpenFileDialog, ref G_missing, ref G_missing, ref G_missing
                , ref G_missing, ref G_missing, ref G_missing, ref G_missing
                , ref G_missing, ref G_missing, ref G_missing, ref G_missing
                , ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            Word.Range P_Range = P_Document.Range(ref G_missing, ref G_missing);          //得到文档范围
            string P_Str = P_Range.Text;
            string G_str_path = string.Format(                                           //计算文件保存路径
             @"{0}\{1}", G_FolderBrowserDailog.SelectedPath,
             DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒") + ".txt");
            using (StreamWriter P_StreamWrite = new StreamWriter(                        //创建文件写入器对象
                G_str_path, true))
            {
                P_StreamWrite.Write(P_Str);
                P_StreamWrite.Flush();
            }
            ((Word._Document)P_Document).Close(ref G_missing, ref G_missing, ref G_missing);
            ((Word._Application)G_wa.Application).Quit(                                  //退出应用程序
                ref G_missing, ref G_missing, ref G_missing);
            this.Invoke(                                                                 //调用窗体线程
                (MethodInvoker)(() =>                                                    //使用 Lambda 表达式
                {
                    Clipboard.Clear();                                                   //清空剪贴板
                    MessageBox.Show("成功创建 Word 文档！", "提示！");                    //弹出消息对话框
                    btn_Get.Enabled = true;                                              //停用创建按钮
                }));
        });
    }
```

秘笈心法

心法领悟 441：方便地向文本文件中写入文本信息。

使用 StreamWriter 对象可以方便地向文本文件中写入信息，首先创建 StreamWriter 对象，然后使用 StreamWriter 对象的 Write 方法写入文本信息，写入文本信息后不要忘记调用该对象的 Close 方法释放资源。

实例 442　将 Word 中的数据分解到多个文本文件中

光盘位置：光盘\MR\19\442　　中级　　趣味指数：★★★★★

实例说明

将 Word 中的数据分解到多个文本文件的实现方法非常简单，首先要读取 Word 文档中所有的文本内容，然后根据用户选择的分解 Word 文档的方式将 Word 文档分解为多个字符串，最后，使用 StreamWriter 将多个字符串分别写入到多个文本文件中。实例运行效果如图 19.56 所示。

实例运行中单击“开始分解”按钮，此时会读取 Word 文档中的文本内容并按照用户指定的方式分解到多个文本文件中。Word 文档中的文本内容如图 19.57 所示。分解后的多个文本文件如图 19.58～图 19.61 所示。

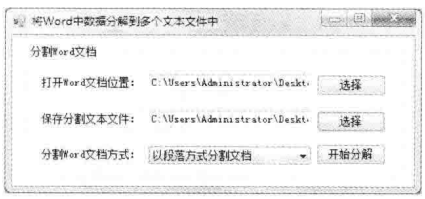


图 19.56　将 Word 中的数据分解到多个文本文件中

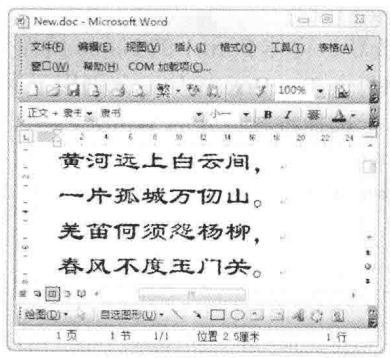


图 19.57　Word 文档中的文本内容

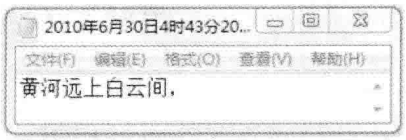


图 19.58　第一个文本文件中的文本内容

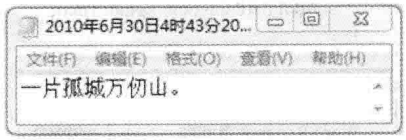


图 19.59　第二个文本文件中的文本内容

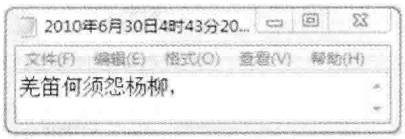


图 19.60　第三个文本文件中的文本内容

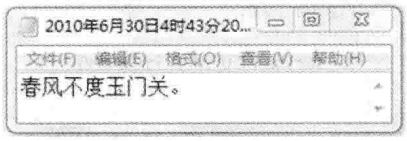


图 19.61　第 4 个文本文件中的文本内容

关键技术

本实例重点介绍怎样将 Word 中的数据分解到多个文本文件中，下面对本实例中用到的关键技术进行详细讲解。

如果要将 Word 中的文本信息分解到多个文本文件中，首先应当根据用户指定的方式提取 Word 文档中的文本信息，并将文本信息保存到多个文本文件中。

说明：（1）关于向文本文件写入内容的详细讲解请参见实例 441 中的关键技术。

（2）关于向 Word 文档中写入文字的详细讲解请参见实例 427 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordToMulti-Txt。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 Word 中数据分解到多个文本文件中”，向窗体中添加两个 TextBox 控件，分别用于显示打开 Word 文档的路径和保存文本文件路径；添加 3 个 Button 控件，分别用于选择将要打开的 Word 文档、选择文本文件保存的路径和分解 Word 文档的内容到多个文本文件。

（4）程序的主要代码如下：

```
private void btn_split_Click(object sender, EventArgs e)
{
    btn_split.Enabled = false;                                          //停用分解按钮
    ThreadPool.QueueUserWorkItem(                                       //开始线程池
        (pp) =>                                                         //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();     //创建应用程序对象
            object P_OpenFileDialog = G_OpenFileDialog.FileName;        //创建 object 对象
            Word.Document P_Document = G_wa.Documents.Open(             //打开 Word 文档
                ref P_OpenFileDialog, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            bool P_bl = false;                                          //定义布尔变量
            this.Invoke(                                                //调用窗体线程
                (MethodInvoker)(() =>                                   //使用 Lambda 表达式
                {
                    P_bl = cbox_Select.SelectedIndex == 0;
                }));
            if (P_bl)                                                   //判断怎样分割文档
            {
                foreach (Word.Paragraph Paragraph in G_wa.ActiveDocument.Paragraphs)
                {
```

```
                AddFile(Paragraph.Range.Text);                          //将文本写入文本文件
            }
        }
        else
        {
            Word.Range P_Range = G_wa.ActiveDocument.Content;           //得到文档区域
            int P_int_count = P_Range.Text.Length;                      //得到文档字符的总长度
            int P_int_i = P_int_count / 100;                            //计算循环次数
            if (P_int_i > 0)                                            //如果文字大于 100 个
            {
                for (int i = 0; i < P_int_i; i++)                       //开始循环创建文档
                {
                    object P_o1 = i == 0 ? 0 : i * 100 + 1;             //复制范围的开始部分
                    object P_o2 = i * 100 + 101;                        //复制范围的结尾部分
                    Word.Range P_Range_temp =                           //得到文档的范围
                        G_wa.ActiveDocument.Range(ref P_o1, ref P_o2);
                    AddFile(P_Range_temp.Text);                         //将文本写入文本文件
                }
                object P_o11 = P_int_i * 100 + 1;                       //复制范围的开始部分
                Word.Range P_Range_temp1 =                              //得到文档的范围
                    G_wa.ActiveDocument.Range(ref P_o11, ref G_missing);
                AddFile(P_Range_temp1.Text);                            //将文本写入文本文件
            }
            else
            {
                Word.Range P_Range2 = G_wa.ActiveDocument.Content;      //得到文档区域
                AddFile(P_Range2.Text);                                 //将文本写入文本文件
            }
        }
        ((Word._Application)G_wa.Application).Quit(                     //退出应用程序
            ref G_missing, ref G_missing, ref G_missing);
        this.Invoke(                                                    //调用窗体线程
            (MethodInvoker)(() =>                                       //使用 Lambda 表达式
            {
                Clipboard.Clear();                                      //清空剪贴板
                MessageBox.Show("成功分解 Word 文档！", "提示！");        //弹出消息对话框
                btn_split.Enabled = true;                               //启用分解按钮
            }));
    });
}
```

秘笈心法

心法领悟 442：使用 DateTime.Now 属性获取系统时间。

使用 DateTime.Now 属性可以方便地获取系统时间，而且可以使用格式化方式方便地得到系统时间的字符串表示，代码如下：

```
string P_str = DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒");
```

实例 443　读取 Word 文档中表格数据到 Access 数据库　中级

光盘位置：光盘\MR\19\443　　趣味指数：★★★★★

实例说明

使用 C#操作 Word 文档的过程中，如果准备读取 Word 文档内表格中的数据，首先要得到文档的 Document 对象，通过此对象得到文档的 Range，再通过 Range 的 Tables 属性得到 Word 文档中的表格集合，此时，可以使用索引的方法得到 Tables 中指定的表格对象，使用表格对象的 Cell 方法得到表格中指定单元格的 Range，最后可以使用 Range 的 Text 属性得到指定单元格中的数据。实例中将会读取 Word 文档中表格数据到 Access 数据库。实例运行效果如图 19.62 所示。

图 19.62　读取 Word 文档中表格数据到 Access 数据库

实例运行中单击“显示 Word 文档并手动添加数据”按钮，会打开 Word 文档，用户可以手动向文档中添加文本信息，如图 19.63 所示。单击“将 Word 文档中的数据写入 Access”按钮，可以将 Word 文档中表格的数据写入到 Access 数据库，最后单击“显示 Access 中的数据”按钮，可以显示 Access 中的数据信息，如图 19.64 所示。

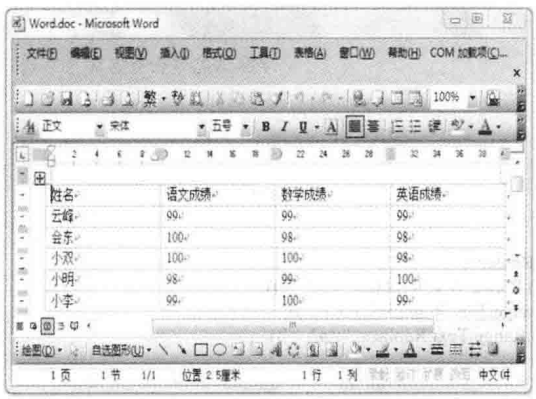


图 19.63　手动向 Word 文档中添加文本信息

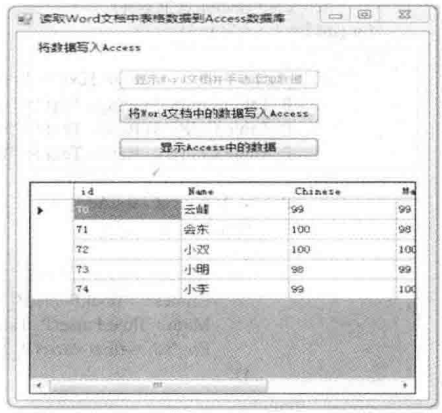


图 19.64　显示 Access 中的数据信息

关键技术

本实例重点介绍怎样读取 Word 文档表格中的数据，并将数据插入到 Access 数据库，下面对本实例中用到的关键技术进行详细讲解。

读取 Word 文档中表格信息的步骤十分简单，首先通过 Document 对象的 Tables 属性得到 Word 文档中所有表格的集合，通过使用索引器的方式可以得到指定的表格对象，表格对象的 Cell 方法提供了两个参数并返回一个 Range 对象。方法中的两个参数分别为表格对象中的行与列的索引。需要注意的是，参数中行与列的索引都是从 1 开始的，根据两个参数可以找到并返回表格中的单元格的 Range 对象，并可以根据 Range 对象的 Text 属性获取或设置单元格的信息。可以将得到表格中的信息放入数据集合，最后将数据集合中的数据插入到 Access 数据库中。

说明：本实例中 Word 文档内表格列的数据类型要与数据表中的数据类型相对应，否则在插入数据时会出现错误。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordToAccess。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取 Word 文档中表格数据到 Access 数据库”，向窗体中添加 3 个 Button 控件，分别用于打开 Word 文档、将 Word 文档中表格数据写入到 Access

和显示 Access 中的数据。

（4）程序的主要代码如下：

```
private void InsertData()
{
        G_wa = new Microsoft.Office.Interop.Word.Application();                    //创建应用程序对象
        object P_Path = Directory.GetCurrentDirectory() + @"\Word.doc";
        Word.Document P_Document = G_wa.Documents.Open(                           //打开 Word 文档
            ref P_Path, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing,
            ref G_missing, ref G_missing, ref G_missing, ref G_missing);
        try
        {
            Word.Range P_Range =                                                  //得到文档范围
                P_Document.Range(ref G_missing, ref G_missing);
            Word.Table P_Table = P_Range.Tables[1];                               //得到文档表格对象
            List<InstanceClass> P_List_InstanceClass =                            //建立集合对象
                new List<InstanceClass>();
            for (int i = 2; i < 7; i++)
            {
                if (P_Table.Cell(i, 1).Range.Text != "\r\a" &&                    //判断表格是否有信息
                    P_Table.Cell(i, 2).Range.Text != "\r\a" &&
                    P_Table.Cell(i, 3).Range.Text != "\r\a" &&
                    P_Table.Cell(i, 4).Range.Text != "\r\a")
                {
                    P_List_InstanceClass.Add(                                     //向数据集合中添加数据
                        new InstanceClass()
                        {
                            Name = P_Table.Cell(i, 1).Range.Text.Replace("\r\a", ""),
                            Chinese = float.Parse(P_Table.Cell(i, 2).Range.Text.Replace("\r\a", "")),
                            Math = float.Parse(P_Table.Cell(i, 3).Range.Text.Replace("\r\a", "")),
                            English = float.Parse(P_Table.Cell(i, 4).Range.Text.Replace("\r\a", ""))
                        });
                }
            }
            new DataTier().InsertMessage(P_List_InstanceClass);                   //向 Access 插入数据
            object P_Save = false;                                                //创建 object 对象
            ((Word._Application)G_wa.Application).Quit(                           //退出应用程序
                ref P_Save, ref G_missing, ref G_missing);
            this.Invoke(                                                          //执行窗体线程
                (MethodInvoker)(() =>                                             //使用 Lambda 表达式
                {
                    MessageBox.Show("向 Access 中插入数据成功！","提示！");          //弹出消息对话框
                }));
        }
        catch (Exception ex)
        {
            object P_Save = false;                                                //创建 object 对象
            ((Word._Application)G_wa.Application).Quit(                           //退出应用程序
                ref P_Save, ref G_missing, ref G_missing);
            throw new Exception(ex.Message);                                      //将异常抛向上一层
        }
}
```

秘笈心法

心法领悟 443：方便地使用“对象初始化器”。

从 C# 3.0 版本开始，用户即可以方便地使用“对象初始化器”对所有的对象进行初始化，对象初始化器允许用户在一条语句中为指定对象的多个字段和属性赋值。本实例中使用了对象初始化器初始化对象，并将初始化后的对象添加到集合中，下面是实例的部分代码：

```
P_List_InstanceClass.Add(                                                 //向数据集合中添加元素
    new InstanceClass()                                                   //创建对象
    {
        Name = P_Table.Cell(i, 1).Range.Text.Replace("\r\a", ""),         //通过对象初始化器为属性赋值
```

```
        Chinese = float.Parse(P_Table.Cell(i, 2).Range.Text.Replace("\r\a", "")),          //通过对象初始化器为属性赋值
        Math = float.Parse(P_Table.Cell(i, 3).Range.Text.Replace("\r\a", "")),             //通过对象初始化器为属性赋值
        English = float.Parse(P_Table.Cell(i, 4).Range.Text.Replace("\r\a", ""))           //通过对象初始化器为属性赋值
    });
```

实例 444　读取 Word 文档中表格数据到 SQL Server 数据库

中级

光盘位置：光盘\MR\19\444

趣味指数：★★★★★

实例说明

读取 Word 文档中的表格数据到 SQL Server 数据库时，首先要读取 Word 表格中的数据到数据集合中，然后将数据集合中的数据添加到 SQL Server 数据库，在填写 Word 表格中的数据时要注意，表格中数据的类型要与集合或数据库中的数据类型一致。实例运行效果如图 19.65 所示。

图 19.65　读取 Word 文档中表格数据到 SQL Server 数据库

实例运行中可以根据本地数据库的设置在 TextBox 控件中填写 SQL 服务器连接信息。单击“显示 Word 文档并手动添加数据”按钮，会打开 Word 文档，用户可以手动向文档中添加文本信息，如图 19.66 所示。单击“将 Word 文档中的数据写入 SQL”按钮，可以将 Word 文档中表格内的数据写入到 SQL 数据库，最后单击“显示 SQL 中的数据”按钮，可以显示 SQL 中的数据信息，如图 19.67 所示。

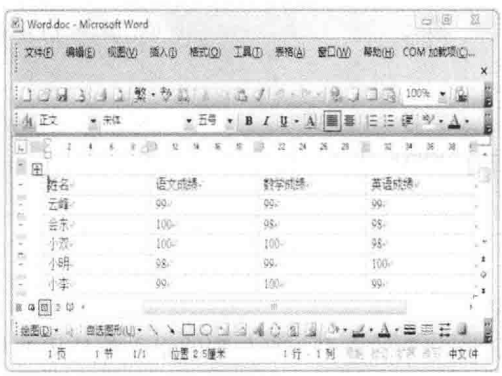


图 19.66　手动向 Word 文档中添加文本信息

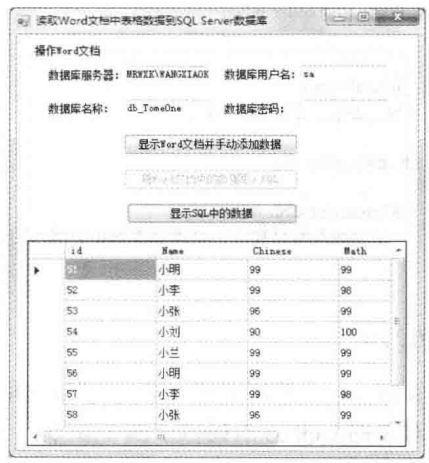


图 19.67　显示 SQL 中的数据信息

关键技术

本实例重点介绍怎样读取 Word 文档中的表格数据到 SQL Server 数据库，下面对本实例中用到的关键技术进行详细讲解。

读取 Word 文档中的表格数据到 SQL Server 数据库的实现方法与实例 443 中读取 Word 文档中的表格数据到 Access 数据库的实现方法几乎相同，只是将数据插入的数据库不同而已。首先通过 Document 对象的 Tables

属性得到 Word 文档中所有表格的集合，使用索引器的方式可以得到指定的表格对象，表格对象的 Cell 方法提供了两个参数并返回一个 Range 对象。方法中的两个参数分别为表格对象中的行与列的索引。需要注意的是，参数中行与列的索引都是从 1 开始的，根据两个参数可以找到并返回表格中的单元格的 Range 对象，并可以根据 Range 对象的 Text 属性获取或设置单元格的信息。可以将得到表格中的信息放入数据集合，最后将数据集合中的数据插入到 SQL Server 数据库中。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordToSql。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“读取 Word 文档中表格数据到 SQL Server 数据库”，向窗体中添加 4 个 TextBox 控件，用于根据本地的 SQL 设置填写 SQL 服务器连接信息；添加 3 个 Button 控件，分别用于手动向文档中添加文本信息、将 Word 文档中表格内的数据写入到 SQL 数据库和显示 SQL 数据库中的信息。

（4）程序的主要代码如下：

```
private void btn_Write_Click(object sender, EventArgs e)
{
    try
    {
        G_wa.ActiveDocument.Save();                                    //保存文档
        object P_Save = false;                                         //创建 object 对象
        ((Word._Application)G_wa.Application).Quit(                    //退出应用程序
            ref P_Save, ref G_missing, ref G_missing);
    }
    catch (Exception ex)
    {
        MessageBox.Show(                                               //弹出消息对话框
            ex.Message, "提示！ ");
    }
    btn_Write.Enabled = false;                                         //停用插入数据按钮
    try
    {
        InsertData();                                                  //执行插入数据方法
        btn_display.Enabled = true;                                    //启用显示文档按钮
    }
    catch (Exception ex)
    {
        MessageBox.Show(                                               //弹出消息对话框
            "插入信息失败，请确认 Word 中填入的信息是否正确\r\n" + ex.Message, "提示！ ");
    }
}
```

秘笈心法

心法领悟 444：使用 Cell 方法方便地读取表格中的信息。

表格对象的 Cell 方法提供了两个参数，分别为表格中的行与列的索引，行与列的索引都是从 1 开始的，根据行与列的索引可以找到单元格的 Range 对象，最后通过 Range 对象的 Text 属性可以方便地读取单元格的信息。

实例 445　实时读取 Word 文档中表格数据到 SQL Server 数据库

高级

光盘位置：光盘\MR\19\445

趣味指数：★★★★★

实例说明

在窗体编程中可以使用 Timer 组件按照指定的时间间隔重复执行相应的操作。实例中也可以使用 Timer 组

件每隔一段时间检测 Word 文档表格中的数据是否与 SQL Server 数据库中的数据一致，如果不一致，则更新数据库中的数据。但是，Timer 组件使用的是窗体线程，如果窗体线程频繁地进行文件操作，可能会导致出现窗体无响应或卡机的现象，所以，在本实例中使用了线程代替 Timer 组件，实时读取 Word 文档中的表格数据到 SQL Server 数据库。实例运行效果如图 19.68 和图 19.69 所示。

图 19.68　显示 SQL 中的数据

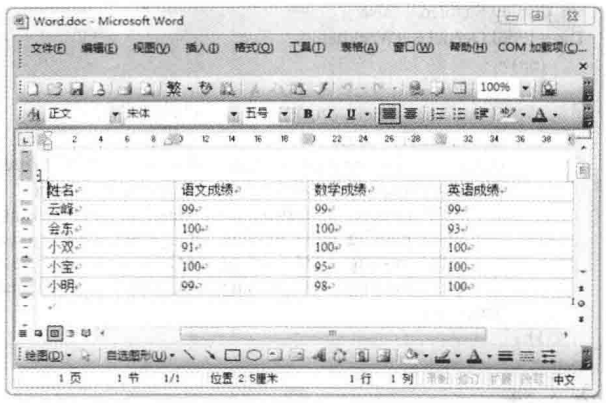


图 19.69　实时读取 Word 文档中的表格数据

关键技术

本实例重点介绍如何使用多线程技术监控 Word 文档中表格的变化，并将表格中更新的信息同步到 SQL Server 数据库。下面对本实例中用到的关键技术进行详细讲解。

说明：使用线程首先要引用命名空间“using System.Threading”。下文再次涉及线程的使用时，将不再提示。

实例中使用了多线程技术监控 Word 文档中表格的变化。多线程的使用方法如图 19.70 所示。

```
Thread P_th = new Thread(                    ——▶创建线程对象
    () =>                                    ——▶使用Lambda表达式
    {
        ……                                   ——▶执行相关操作
    });
P_th.IsBackground = true;                    ——▶设置线程为后台线程
P_th.Start();                                ——▶开始执行线程
```

图 19.70　多线程的使用方法

从图 19.70 中可以看到，创建线程非常简单，首先创建线程对象，在线程对象的构造方法中接收一个方法作为参数，实例中使用了 Lambda 表达式来代替方法，然后设置线程为后台线程，这一句是可选的，默认创建的线程为前台线程。最后使用 Start 方法开始执行线程。

注意：如果线程在执行中发现 Word 文档中表格内的数据与数据集合中的数据不一致，则会立即更新数据集合中的数据，然后将数据集合中的数据更新到 SQL Server 数据库。将数据集合中的数据更新到 SQL Server 数据库使用了非常简单的方法，首先清空数据库中指定数据表内的所有数据，然后将数据集合中的数据全部添加到数据表中。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Real-TimeToSQL。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“实时读取 Word 文档中表格数据

到 SQL Server 数据库”，向窗体中添加 4 个 TextBox 控件，用于根据本地 SQL 设置填写 SQL 服务器连接信息；添加一个 Button 控件，用于跟踪 Word 文档的内容并将更新的内容同步到 SQL 数据库。

（4）程序的主要代码如下：

```
private void btn_display_Click(object sender, EventArgs e)
{
    btn_display.Enabled = false;                                          //停用跟踪数据按钮
    ThreadPool.QueueUserWorkItem(                                         //开始线程池
        (pp) =>                                                           //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();       //创建应用程序对象
            G_wa.Visible = true;
            object P_Path = Directory.GetCurrentDirectory() + @"\Word.doc";
            Word.Document P_Document = G_wa.Documents.Open(               //打开 Word 文档
                ref P_Path, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing);
            TrackData();                                                  //将 Word 数据更新到 SQL
        });
}
```

秘笈心法

心法领悟 445：使用线程代替 Timer 组件。

Timer 组件可以按用户定义的时间间隔来引发 Tick 事件，该事件一般为周期性的，每隔若干秒或若干毫秒执行一次，Timer 组件工作在窗体线程中，如果 Timer 组件执行了较为耗时的操作，会增加窗体线程的负担，导致窗体中其他操作不能及时得到 CPU 资源，出现窗体长时间或短时间无响应的情况。在适当的情况下可以使用线程来代替 Timer 组件，这样会减少窗体线程的负担。

实例 446　每天定时读取 Word 文档中表格数据给指定数据库

高级

光盘位置：光盘\MR\19\446

趣味指数：★★★★★

实例说明

所谓的定时功能就是指系统根据指定的时间执行相应的操作，在实例中可以使用线程获取当前系统时间来实现定时执行任务的效果。当任务开始执行时，首先从 Word 文档的表格中取出数据到数据集合，然后将数据集合中的数据添加到数据库。实例运行效果如图 19.71 所示。

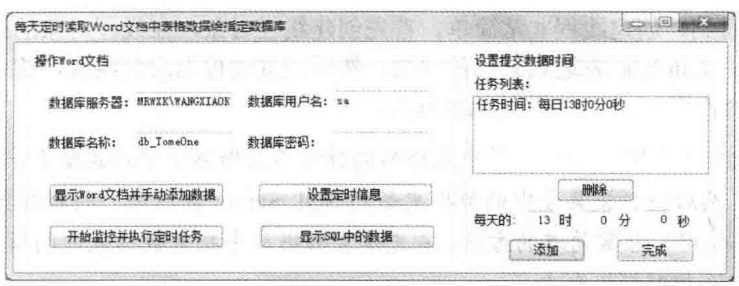


图 19.71　每天定时读取 Word 文档中表格数据给指定数据库

实例运行中可以根据本地数据库的设置在 TextBox 控件中填写 SQL 服务器连接信息。单击“显示 Word 文档并手动添加数据”按钮，会打开 Word 文档，用户可以手动向文档中添加文本信息，如图 19.72 所示。单击“设置定时信息”按钮，可以根据系统时间设置定时任务；单击“开始监控并执行定时任务”按钮，将开始执行定时任务；单击“显示 SQL 中的数据”按钮，可以显示 SQL 数据库中的信息，如图 19.73 所示。

图 19.72　手动向 Word 文档中添加文本信息

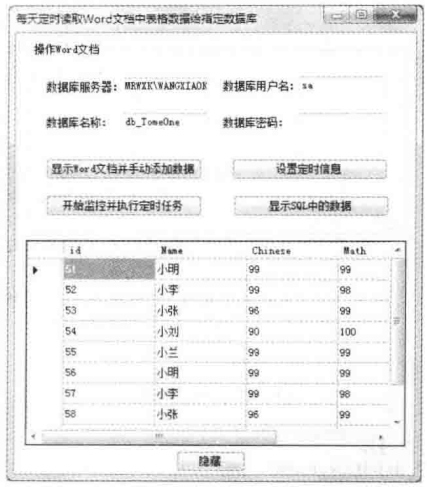


图 19.73　显示 SQL 中的信息

关键技术

本实例重点介绍怎样使用线程跟踪系统时间，并在指定时间读取 Word 文档中的表格数据并将其插入 SQL Server 数据库。下面对本实例中用到的关键技术进行详细讲解。

在指定时间向数据库中插入 Word 文档表格数据的实现逻辑很简单，首先创建一条线程，线程每隔一秒钟检测一次系统时间，如果系统时间等于用户指定的时间，那么开始执行插入数据的操作。

说明：（1）关于线程使用的详细讲解请参见实例 445 中的关键技术。

（2）关于读取 Word 文档中的表格数据到 SQL Server 数据库的详细讲解请参见实例 444 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InsertToSQL。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“每天实时读取 Word 文档中表格数据给指定数据库”，向窗体中添加 4 个 TextBox 控件，用于根据本地 SQL 设置填写 SQL 服务器连接信息；添加 4 个 Button 控件，分别用于打开 Word 文档、设置定时信息、监控定时任务和显示 SQL 数据。

（4）程序的主要代码如下：

```
private void btn_Begin_Click(object sender, EventArgs e)
{
    btn_Begin.Enabled = false;                                          //停用开始按钮
    Thread P_th = new Thread(                                           //创建线程
        () =>                                                           //使用 Lambda 表达式
        {
            while (true)                                                //开始无限循环
            {
                this.Invoke(                                            //在窗体线程中执行
                    (MethodInvoker)(() =>                               //使用 Lambda 表达式
                    {
                        foreach (object P_O in lbox_Task.Items)
                        {
                            Time P_Time = (Time)P_O;                    //转换为 Time 类型
                            if (P_Time.Hours.ToString() ==              //判断时间是否相等
                                DateTime.Now.Hour.ToString() &&P_Time.Minutes.ToString() ==
                                DateTime.Now.Minute.ToString() &&P_Time.Seconds.ToString() ==
                                DateTime.Now.Second.ToString())
                            {
                                if (P_Time.Execute)                     //判断任务是否已执行
```

```
                {
                    P_Time.Execute = false;                                    //设置任务为已执行
                    InsertData();                                              //开始插入数据
                }
            }
        }
        if ("0" == DateTime.Now.Hour.ToString() &&                             //是否重置任务
            "0" == DateTime.Now.Minute.ToString() &&"0" == DateTime.Now.Second.ToString())
        {
            foreach (object P_1 in lbox_Task.Items)                            //遍历集合
            {
                ((Time)P_1).Execute = true;                                    //重置任务
            }
        }
    }));
    Thread.Sleep(1000);                                                        //线程挂起 1 秒钟
  }
 });
 P_th.IsBackground = true;                                                     //设置线程为后台线程
 P_th.Start();                                                                 //线程开始执行
}
```

秘笈心法

心法领悟 446：使用线程还是线程池？

在窗体编程中用户可以方便地使用 Thread 类创建线程，也可以方便地使用线程池，但是在线程与线程池之间应当怎样选择呢？首先，创建线程所需的资源要比线程池所需的资源多，但是在创建线程后用户可以方便地对线程进行控制，如挂起线程、结束线程等。而线程池由 CLR 分配和管理，用户不可以过多地控制线程池，而且线程池也并不适合长时间地执行某操作。所以在程序设计中不能说线程池比线程好，只能说在特定的环境下更适合使用哪一种技术。

实例 447　将 Word 文档转换为 HTML 网页

中级

光盘位置：光盘\MR\19\447

趣味指数：★★★★★

实例说明

用户可以使用 C#方便地读取 Word 文档中的文本内容，并将文本内容保存到文本文件中，但是保存的文本内容是没有格式的，那么，有什么好的方法可以解决上面的问题吗？将 Word 文档的内容转换为 HTML 网页是一个不错的选择，本实例将介绍一种可以轻松地将 Word 文档转换为 HTML 网页的方法。实例运行效果如图 19.74 所示。

图 19.74　将 Word 文档转换为 HTML 网页

实例运行中单击“打开 Word 文档”或“创建 Word 文档”按钮，会打开 Word 文档，用户可以手动向文档中添加文本信息，如图 19.75 所示。单击“转换为 HTML”按钮，可以将 Word 文档转换为 HTML 网页，如图 19.76 所示。

图 19.75　Word 文档中的文本内容

图 19.76　HTML 网页中的文本内容

关键技术

本实例重点介绍怎样使用 Document 对象的 SaveAs 方法将 Word 文档转换为 HTML 网页，下面对本实例中用到的关键技术进行详细讲解。

在 Word 文档的操作过程中，经常可以使用 SaveAs 方法保存 Word 文档，本实例中将会使用 SaveAs 方法将 Word 文档另存为 HTML 网页文件格式。SaveAs 方法的语法格式如下：

```
public virtual void SaveAs(ref object FileName, ref object FileFormat, ref object LockComments, ref object Password, ref object AddToRecentFiles, ref object WritePassword, ref object ReadOnlyRecommended, ref object EmbedTrueTypeFonts, ref object SaveNativePictureFormat, ref object SaveFormsData, ref object SaveAsAOCELetter, ref object Encoding, ref object InsertLineBreaks, ref object AllowSubstitutions, ref object LineEnding, ref object AddBiDiMarks);
```

SaveAs 方法中的参数说明如表 19.7 所示。

表 19.7　SaveAs 方法中的参数说明

参　数	描　述
FileName	代表文件名称及路径，如果已有 FileName 指定的文档，则覆盖此文档，而且在覆盖前不会提示
FileFormat	文档的保存格式。可以是 WdSaveFormat 枚举类型值
LockComments	如果为 true，则锁定文档以进行注释。默认值为 false
Password	打开文档的口令
AddToRecentFiles	如果为 true，则将该文档添加到“文件”菜单上最近使用的文件列表中。默认值为 true
WritePassword	如果为 true，则将 TrueType 字体随文档一起保存
ReadOnlyRecommended	如果为 true，则让 Microsoft Office Word 在打开文档时建议只读状态。默认值为 false
EmbedTrueTypeFonts	如果为 true，则将 TrueType 字体随文档一起保存
SaveNativePictureFormat	如果为 true，则从其他系统平台（例如 Macintosh）导入的图形仅保存其 Windows 版本
SaveFormsData	如果为 true，则将用户在窗体中输入的数据另存为数据记录
SaveAsAOCELetter	如果文档包含一个附件，当此属性值为 true 时，将文档保存为一篇 AOCE 信函（同时保存邮件）
Encoding	要用于另存为编码文本文件的文档的代码页或字符集。默认值是系统代码页
InsertLineBreaks	如果文档另存为文本文件，则 true 表示在每行文本末尾插入分行符
AllowSubstitutions	如果文档另存为文本文件，则 true 允许 Word 将某些符号替换为外观与之类似的文本
LineEnding	参数为 WdLineEndingType 枚举类型值，Word 在另存为文本文件的文档中标记分行符和换段符
AddBiDiMarks	如果为 true，则向输出文件添加控制字符，以便保留原始文档中文本的双向布局

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordToHTML。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“将 Word 文档转换为 HTML 网页”，向窗体中添加 3 个 Button 控件，分别用于打开 Word 文档、创建 Word 文档和将 Word 文档转换为 HTML 网页。

（4）程序的主要代码如下：

```
private void btn_SaveAs_Click(object sender, EventArgs e)
{
    btn_SaveAs.Enabled = false;                                          //停用转换按钮
    try
    {
        G_wa.ActiveDocument.Save();                                      //保存文档
        ((Word._Application)G_wa.Application).Quit(                      //退出应用程序
          ref G_missing, ref G_missing, ref G_missing);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    SaveFileDialog P_SaveFileDialog = new SaveFileDialog();              //创建保存文件对话框
    P_SaveFileDialog.Filter = "*.html|*.html";                           //筛选文件扩展名
    DialogResult P_DialogResult = P_SaveFileDialog.ShowDialog();         //打开保存文件对话框
    if (P_DialogResult == DialogResult.OK)                               //判断是否保存文件
    {
        object P_str_path = P_SaveFileDialog.FileName;                   //创建 object 对象
        ThreadPool.QueueUserWorkItem(                                    //开始线程池
            (pp) =>                                                      //使用 Lambda 表达式
            {
                G_wa =                                                   //创建应用程序对象
                  new Microsoft.Office.Interop.Word.Application();
                G_wa.Visible = false;
                Word.Document P_wd = G_wa.Documents.Open(                //打开 Word 文档
                    ref G_FilePath, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing);
                object P_Format = Word.WdSaveFormat.wdFormatHTML;        //创建保存文档参数
                P_wd.SaveAs(                                             //保存 Word 文件
                    ref P_str_path,
                    ref P_Format, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                    ref G_missing, ref G_missing, ref G_missing);
                ((Word._Application)G_wa.Application).Quit(              //退出应用程序
                    ref G_missing, ref G_missing, ref G_missing);
                this.Invoke(                                             //调用窗体线程
                    (MethodInvoker)(() =>                                //使用 Lambda 表达式
                    {
                        btn_Open.Enabled = true;                         //启用打开按钮
                        btn_New.Enabled = true;                          //启用新建按钮
                        MessageBox.Show("文件已经创建", "提示！");          //提示已经创建 Word
                    }));
            });
    }
}
```

秘笈心法

心法领悟 447：将 Word 转换为 rtf 文档。

实例中详细地介绍了怎样使用 Document 对象的 SaveAs 方法将 Word 文档转换为 HTML 网页。SaveAs 方法

还可以将 Word 文档轻松地转换为 rtf 文档。只要将实例中的代码：

```
object P_Format = Word.WdSaveFormat.wdFormatHTML;
```

替换为：

```
object P_Format = Word.WdSaveFormat.wdFormatRTF;
```

并相应地更改保存文件的扩展名即可。

实例 448　在打印预览中显示 Word 文档

光盘位置：光盘\MR\19\448　　中级　　趣味指数：★★★★★

实例说明

在打印 Word 文档之前，经常会使用打印预览功能预览打印效果。本实例中将会介绍一种方法，可以方便地打开 Word 文档，打开文档的同时也会打开文档的打印预览。实例运行效果如图 19.77 所示。

实例运行中单击“打开 Word 文档并预览”按钮，会打开 Word 文档并启用预览功能，如图 19.78 所示。

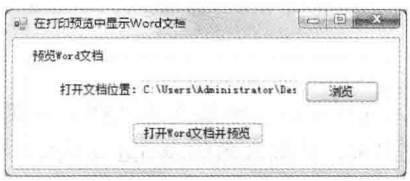


图 19.77　在打印预览中显示 Word 文档

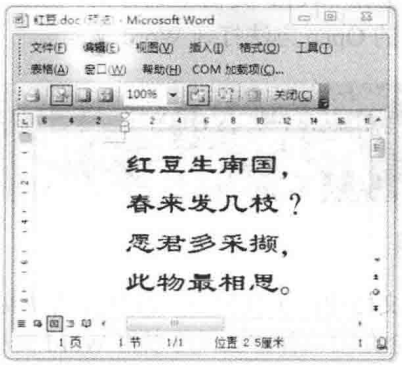


图 19.78　启用打印预览功能的 Word 文档

关键技术

本实例重点介绍如何使用 PrintPreview 方法在打印预览窗体中显示 Word 文档，下面对本实例中用到的关键技术进行详细讲解。

在打开 Word 文档后，可以使用 Document 文档对象的 PrintPreview 方法轻松地显示打印预览窗体。在打印预览窗体中，可以方便地调节显示比例，清晰直观地预览 Word 文档的打印信息。

说明：由于打印预览是在 Word 文档中运行的，所以在使用 PrintPreview 方法打开打印预览窗口前，首先要使用 Document 对象的 Open 方法打开 Word 文档。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WordPreView。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“在打印预览中显示 Word 文档”，向窗体中添加一个 TextBox 控件，用于显示打开 Word 文档的路径；添加两个 Button 控件，分别用于浏览 Word 文档和打开并预览 Word 文档。

（4）程序的主要代码如下：

```
private void btn_Open_Click(object sender, EventArgs e)
{
    btn_Open.Enabled = false;                                    //停用打开按钮
```

```
    ThreadPool.QueueUserWorkItem(                                          //开始线程池
        (pp) =>                                                            //使用 Lambda 表达式
        {
            G_wa = new Microsoft.Office.Interop.Word.Application();        //创建应用程序对象
            G_wa.Visible = true;                                           //将文档设置为可见
            object P_FileName = G_OpenFileDialog.FileName;                 //设置打开文档参数
            Word.Document P_Document = G_wa.Documents.Open(                //打开 Word 文档
                ref P_FileName, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing, ref G_missing, ref G_missing, ref G_missing, ref G_missing,
                ref G_missing);
            P_Document.PrintPreview();                                     //打开预览窗口
        });
}
```

秘笈心法

心法领悟 448：方便地使用 Word 文档的打印预览功能。

在 Word 文档的操作过程中，可以使用 Document 文档对象的 PrintPreview 方法轻松地显示打印预览页面，由于打印预览是在 Word 文档中运行的，所以在使用 PrintPreview 方法打开打印预览窗口前，首先要使用 Document 对象的 Open 方法打开 Word 文档。

实例 449　提取 Word 文档中的目录

光盘位置：光盘\MR\19\449　　高级　　趣味指数：★★★★★

实例说明

在 Word 文档的使用过程中，经常需要在文档中添加目录，这样可以清晰地描述文档中各部分所包含的内容。本实例将会介绍一种方法，可以将 Word 文档中的目录提取出来，并放入新的 Word 文档中。实例运行效果如图 19.79 所示。

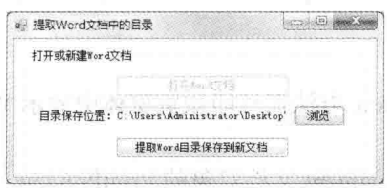


图 19.79　提取 Word 文档中的目录

实例运行中单击“打开 Word 文档”按钮，会打开 Word 文档，如图 19.80 所示。单击“提取 Word 目录保存到新文档”按钮，将提取 Word 文档中的目录并保存到新建的 Word 文档中，如图 19.81 所示。

图 19.80　Word 文档的内容

图 19.81　提取 Word 文档目录内容

关键技术

本实例重点介绍怎样使用 TablesOfContents 对象的 Add 方法提取 Word 文档中的目录，下面对本实例中用到的关键技术进行详细讲解。

实例中使用了 TablesOfContents 对象的 Add 方法提取 Word 文档中的目录。Add 方法的语法格式如下：

TableOfContents Add(Range Range, ref object UseHeadingStyles, ref object UpperHeadingLevel, ref object LowerHeadingLevel, ref object UseFields, ref object TableID, ref object RightAlignPageNumbers, ref object IncludePageNumbers, ref object AddedStyles, ref object UseHyperlinks, ref object HidePageNumbersInWeb, ref object UseOutlineLevels);

TablesOfContents 对象的 Add 方法中的参数说明如表 19.8 所示。

表 19.8　TablesOfContents 对象的 Add 方法中的参数说明

参　数	描　述
Range	指定目录存放区域，如果这个区域没有折叠，目录将覆盖区域
UseHeadingStyles	如果为 true，则使用内置样式创建目录，默认值为 true
UpperHeadingLevel	目录起始的标题级别，默认值为 1
LowerHeadingLevel	目录结束的标题级别，默认值为 9
UseFields	如果为 true，则使用 TC 域创建目录。如果要标记目录中所包含的目录项，可以使用 MarkEntry 方法。默认值为 false
TableID	单字母标识符，用于根据 TC 域创建目录
RightAlignPageNumbers	如果为 true，目录中的页码将与页面右边距对齐，默认值为 true
IncludePageNumbers	如果为 true，则在目录中包含页码，默认值为 true
AddedStyles	用于编辑目录的其他样式的字符串名称（除标题 1～标题 9 以外的其他样式），要创建新的标题样式，可以使用 HeadingStyles 对象的 Add 方法
UseHyperlinks	如果为 true，那么将该文档发布到 Web 时，目录中的目录项将转为超链接的形式，默认值为 true
HidePageNumbersInWeb	如果为 true，那么将该文档发布到 Web 时，目录中的页码将被隐藏，默认值为 false
UseOutlineLevels	如果为 true，那么文档内容下方有下划线，否则没有，默认值为 false

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetDirectry。

（2）使用 C#操作 Word 文档，需要引用相关的 COM 组件，引用 COM 组件的步骤请参照实例 426 中的设计过程。

（3）更改默认窗体 Form1 的 Name 属性为 Frm_Main，更改 Text 属性为“提取 Word 文档中的目录”，向窗体中添加一个 TextBox 控件，用于显示保存 Word 文档的路径；添加两个 Button 控件，分别用于打开 Word 文档和提取 Word 文档目录。

（4）程序的主要代码如下：

```
private void WordToWord(Word.Document P_wd, Word.Document P_document, Word.Range rg)
{
    object P_UseHeadingStyles = true;                          //是否使用内置样式创建目录
    object P_UpperHeadingLevel = 1;                            //目录起始的标题级别
    object P_LowerHeadingLevel = 9;                            //目录结束的标题级别
    object P_UseFields = false;                                //是否使用 TC（目录项）创建目录
    object P_TableID = 1;                                      //用于根据 TC 域创建目录
    object P_RightAlignPageNumbers = false;                    //目录是否右边距对齐
    object P_IncludePageNumbers = false;                       //目录中是否包含页码
    object P_AddedStyles = null;                               //目录其他样式的字符串名称
    object P_UseHyperlinks = false;                            //是否将文档发布到 Web
    object P_HidePageNumbersInWeb = false;                     //目录中的页码是否被隐藏
    P_wd.TablesOfContents.Add(rg, ref P_UseHeadingStyles,      //将提取的目录插入到文档的开始位置
        ref P_UpperHeadingLevel, ref P_LowerHeadingLevel,
```

```
            ref P_UseFields, ref P_TableID, ref P_RightAlignPageNumbers,
            ref P_IncludePageNumbers, ref P_AddedStyles, ref P_UseHyperlinks,
            ref P_HidePageNumbersInWeb, ref G_missing);
        if (P_wd.Fields.Count >= 1)
        {
            P_wd.Paragraphs[1].Range.Cut();                          //剪切文档开始位置的目录信息
            P_document.Range(ref P_start, ref p_end).Paste();        //将目录信息粘贴到新文档
        }
    }
```

秘笈心法

心法领悟 449：使用 ClipBoard.Clear 方法清空剪贴板信息。

实例中使用了 Range 对象的 Cut 方法将文档中的目录信息放入剪贴板，然后调用 Range 对象的 Paste 方法将剪贴板内的信息粘贴到新建文档中。但是，在程序执行完成后，系统剪贴板中还保存着文档中所剪切的信息，所以此时要使用 Clipboard.Clear 方法清空剪贴板中的信息。

实例 450　在 Word 文档中建立小型应用程序

光盘位置：光盘\MR\19\450　　高级　　趣味指数：★★★★★

实例说明

使用 C#可以向 Word 文档中添加文本、图片等信息，也可以向 Word 文档中添加窗体控件。本实例中将会使用 C#向 Word 文档中添加多个窗体控件，从而组合成为一个小型应用程序。实例运行效果如图 19.82 所示。

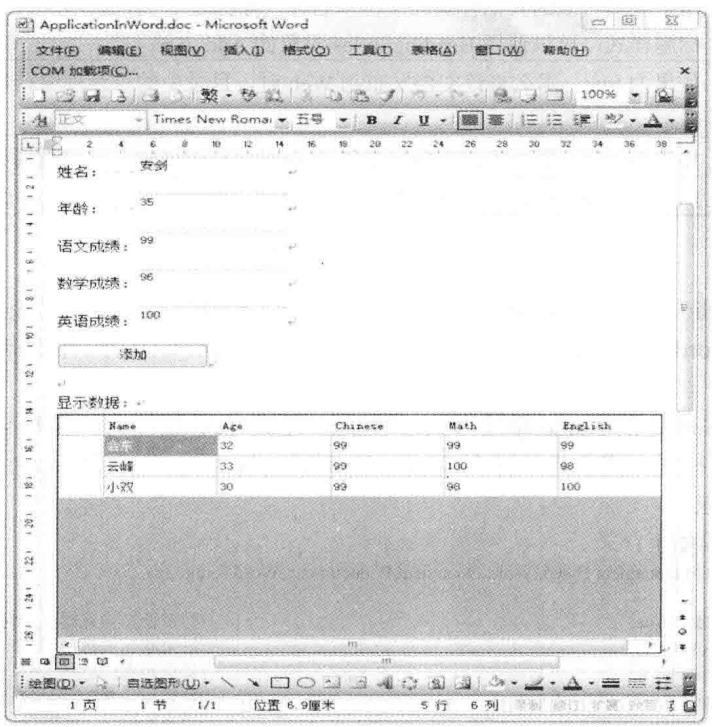


图 19.82　在 Word 文档中建立小型应用程序

关键技术

本实例重点介绍如何使用 Controls 集合的多种方法，方便地向 Word 文档中插入 Windows 控件，从而在 Word 文档中建立一个小型的应用程序。下面对本实例中用到的关键技术进行详细讲解。

（1）向 Word 文档中添加 TextBox 控件

使用 Controls 集合的 AddTextBox 方法可以方便地向 Word 文档中插入文本框控件。AddTextBox 的语法格式如下：

```
public TextBox AddTextBox(Range range, float width, float height, string name);
```

AddTextBox 方法中的参数说明如表 19.9 所示。

表 19.9　AddTextBox 方法中的参数说明

参　数	描　述
range	将要插入 Windows 控件所在的范围
width	插入 Windows 控件的宽度
height	插入 Windows 控件的高度
name	插入 Windows 控件的名称

从 AddTextBox 的语法格式中可以看到，AddTextBox 方法执行后将会向 Word 文档中添加一个 TextBox 文本框并返回此 TextBox 对象的引用。

（2）向 Word 文档中添加 Button 控件

使用 Controls 集合的 AddButton 方法可以方便地向 Word 文档中插入按钮控件。AddButton 的语法格式如下：

```
public Button AddButton(Range range, float width, float height, string name);
```

AddButton 方法中的参数说明如表 19.10 所示。

表 19.10　AddButton 方法中的参数说明

参　数	描　述
range	将要插入 Windows 控件所在的范围
width	插入 Windows 控件的宽度
height	插入 Windows 控件的高度
name	插入 Windows 控件的名称

从 AddButton 的语法格式中可以看到，AddButton 方法执行后将会向 Word 文档中添加一个 Button 按钮并返回此 Button 对象的引用。

（3）向 Word 文档中添加 DataGridView 控件

使用 Controls 集合的 AddDataGridView 方法可以方便地向 Word 文档中插入 DataGridView 控件。AddDataGridView 的语法格式如下：

```
public DataGridView AddDataGridView(Range range, float width, float height, string name);
```

AddDataGridView 方法中的参数说明如表 19.11 所示。

表 19.11　AddDataGridView 方法中的参数说明

参　数	描　述
range	将要插入 Windows 控件所在的范围
width	插入 Windows 控件的宽度
height	插入 Windows 控件的高度
name	插入 Windows 控件的名称

说明： Controls 集合中的多个添加控件的方法都会向 Word 文档中添加控件并返回相应控件对象的引用，通过方法返回的控件对象可以轻松地控制 Word 文档中的 Windows 控件。本实例中向 Word 文档添加了多个 Windows 控件，并在 Word 文档中创建了一个小型应用程序。

设计过程

（1）打开 Visual Studio 2012 开发环境，选择“文件”/“新建”/“项目”命令，弹出“新建项目”对话框，

在“项目类型”列表框中选择 Visual C#/Office 选项，再在右侧的“模板”栏中选择相应的 Word 版本选项，并在“名称”文本框中输入“ApplicationInWord”，然后选择项目保存位置，最后单击“确定”按钮新建项目，如图 19.83 所示。

图 19.83　在 Visual C# 2012 中创建 Word 文档

（2）在打开的 Word 文档中单击鼠标右键，在弹出的快捷菜单中选择“查看代码”命令，再在 Startup 事件中写入如下代码：

```
private void ThisDocument_Startup(object sender, System.EventArgs e)
{
    Word.Range P_Range =
        this.ActiveWindow.Document.Range(ref G_Missing, ref G_Missing);          //得到文档区域
    P_Range.Text = "姓名：      ";                                                //添加文本内容
    object P_Range_O = Word.WdCollapseDirection.wdCollapseEnd;                   //创建折叠参数对象
    P_Range.Collapse(ref P_Range_O);                                             //折叠 Range
    G_Txt_Name = this.Controls.AddTextBox(P_Range, 100, 20, "P_Txt_Name");       //添加控件
    P_Range = this.Paragraphs.Add(ref G_Missing).Range;                          //添加段落
    P_Range = this.Paragraphs.Add(ref G_Missing).Range;                          //添加段落
    P_Range = this.Paragraphs[this.Paragraphs.Count - 1].Range;                  //得到最后一个段落
    G_Txt_Age = this.Controls.AddTextBox(P_Range, 100, 20, "P_Txt_Age");         //添加控件
    P_Range = this.Paragraphs[this.Paragraphs.Count - 1].Range;                  //得到最后一个段落
    object P_Range_O2 = Word.WdCollapseDirection.wdCollapseStart;                //创建折叠参数对象
    P_Range.Collapse(ref P_Range_O2);                                            //折叠 Range
    P_Range.Text = "年龄：      ";                                               //添加文本
    AddControl();                                                                //向文档中添加控件
}
```

秘笈心法

心法领悟 450：怎样理解折叠 Range 的概念？

向 Word 文档中添加 Windows 控件或文本内容时，经常会使用 Collapse 方法折叠 Range 并向 Range 中继续添加信息，所谓的折叠 Range 是指起始位置和结束位置相同的 Range 对象。

第20章

高效应用 Excel

- Excel 基本操作
- 读取外部数据到 Excel
- 将 Excel 数据进行输出处理
- 利用 Excel 进行数据挖掘

20.1 Excel 基本操作

实例 451 将 Excel 文件嵌入到 WinForm 窗体中

初级

光盘位置：光盘\MR\20\451

趣味指数：★★★★★

实例说明

Excel 是微软公司提供的用于办公管理的应用软件，它在人们的日常办公应用中经常用到。本实例使用 C# 代码将 Excel 文件嵌入到了 Windows 窗体中，运行本实例，选择菜单栏中的“打开 Excel 文件”选项，即可将选择的 Excel 文件显示在 Windows 窗体中。实例运行效果如图 20.1 所示。

图 20.1 将 Excel 文件嵌入到 WinForm 窗体中

关键技术

本实例实现在 Windows 窗体中显示 Excel 文件时，主要用到了 WebBrowser 控件的 Navigate 方法，下面对其进行详细讲解。

WebBrowser 控件是 Windows 应用程序中的一种标准控件，它为 WebBrowser ActiveX 控件提供了托管包装，该托管包装可以使用户在 Windows 窗体客户端应用程序中显示网页等文件，其 Navigate 方法用来将指定位置的文档加载到 WebBrowser 控件中，该方法常用的语法格式如下：

```
public void Navigate(string urlString)
```

参数说明

urlString：要加载的文档的 URL 地址。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExcelToWinForm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 OpenFileDialog 组件，用来显示“打开文件”对话框；添加一个 WebBrowser 控件，用来显示打开的 Excel 文件。

（3）程序的主要代码如下：

```
private void 打开 Excel 文件 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog.Filter="Excel 文件|*.xls";                    //设置打开文件筛选器
    OpenFileDialog.Title = "打开 Excel 文件";                    //设置打开对话框标题
    if (OpenFileDialog.ShowDialog() == DialogResult.OK)          //判断是否选择了文件
    {
        WBrowser_Excel.Navigate(OpenFileDialog.FileName);        //在窗体中显示 Excel 文件内容
    }
}
```

秘笈心法

心法领悟 451：使用 WebBrowser 控件轻松地在窗体中浏览网页信息。

WebBrowser 控件可以在 Windows 窗体客户端应用程序中显示网页，使用 WebBrowser 控件显示网页信息非常方便，只需执行 Navigate 方法，并在方法参数中加入网址信息字符串即可。

实例 452	创建 Excel 文件并保存 光盘位置：光盘\MR\20\452	初级 趣味指数：★★★★☆

实例说明

本实例通过使用 C#语言来实现 Excel 文件的创建与保存功能，运行本实例，如图 20.2 所示。单击“选择”按钮，在弹出的对话框中选择 Excel 文件的保存位置，然后单击“创建”按钮，即可在计算机的指定位置创建一个 Excel 文件，如图 20.3 所示。

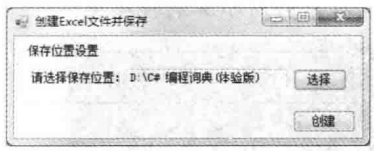


图 20.2　创建 Excel 文件并保存

图 20.3　创建完的 Excel 文件

关键技术

本实例在创建和保存 Excel 文件时，分别用到了 Excel 自动化对象模型的 Workbooks 对象的 Add 方法及 Workbook 对象的 SaveCopyAs 方法。下面对本实例中用到的关键技术进行详细讲解。

（1）Workbooks 对象的 Add 方法

该方法主要用来创建一个新的工作簿。语法格式如下：

```
Workbook Add([In, Optional] object Template);
```

参数说明

❶ Template：指定是否添加工作簿，如果为 true，表示添加；否则，表示不添加。

❷ 返回值：返回 Workbook 类型，表示一个 Excel 工作簿。

（2）Workbook 对象的 SaveCopyAs 方法

该方法主要用来将工作簿的副本保存到文件中，但不修改内存中打开的工作簿。语法格式如下：

```
public virtual void SaveCopyAs(Object Filename)
```

参数说明

Filename：指定要保存的 Excel 副本的文件名。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的 Excel 文件保存路径；添加两个 Button 控件，分别用来执行选择 Excel 文件保存路径操作和创建 Excel 文件操作。

（3）程序的主要代码如下：

```
private void btn_Create_Click(object sender, EventArgs e)
{
    string P_str_path = txt_Path.Text;                    //记录路径
```

```
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //添加新工作簿
    Microsoft.Office.Interop.Excel.Workbook newWorkBook = excel.Application.Workbooks.Add(true);
    object missing = System.Reflection.Missing.Value;                    //获取缺少的 object 类型值
    newWorkBook.Worksheets.Add(missing, missing, missing, missing);      //向 Excel 文件中增加工作表
    if (P_str_path.EndsWith("\\"))                                       //判断路径是否以\结尾
        //保存 Excel 文件
        newWorkBook.SaveCopyAs(P_str_path + DateTime.Now.ToString("yyyyMMddhhmmss") + ".xls");
    else
        //保存 Excel 文件
        newWorkBook.SaveCopyAs(P_str_path + "\\" + DateTime.Now.ToString("yyyyMMddhhmmss") + ".xls");
    //弹出提示信息
    MessageBox.Show("Excel 文件创建成功", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    //创建进程对象
    System.Diagnostics.Process[] excelProcess = System.Diagnostics.Process.GetProcessesByName("EXCEL");
    foreach (System.Diagnostics.Process p in excelProcess)
        p.Kill();                                                        //关闭进程
}
```

说明：（1）本实例在使用 Excel 对象时，首先需要添加该对象的引用，具体步骤是：选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令，弹出“引用管理器”对话框，如图 20.4 所示。在该对话框中选中要引用的 Excel 动态链接库，这里选择 Microsoft.Office.Interop.Excel 11.0.0.0，然后单击“确定”按钮，即可将 Excel 对象引用添加到程序中。

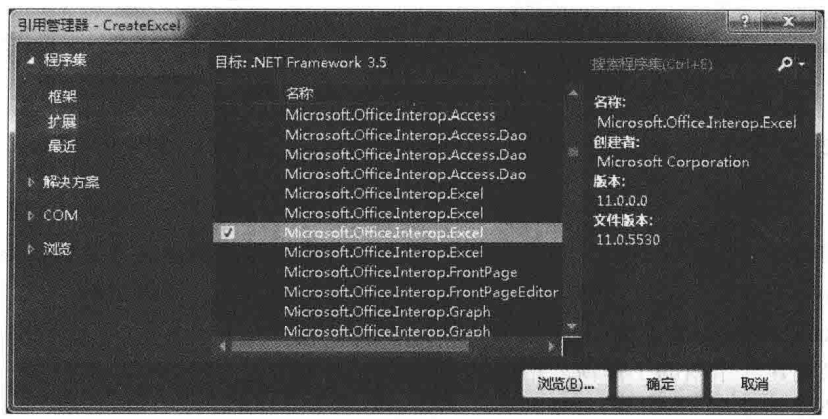


图 20.4 “引用管理器”对话框

（2）本章后面的实例中，如果需要对 Excel 进行操作，则都需要按（1）所讲的步骤添加 Excel 对象引用，由于添加的步骤是一样的，所以在后面涉及时将不再赘述。

秘笈心法

心法领悟 452：使用 Process 类杀死进程。

Process 类用来提供对本地和远程进程的访问并使用户能够启动和停止本地系统进程，其 Kill 方法用来立即停止关联的进程。

实例 453 为 Excel 工作簿设置和清除密码

初级

光盘位置：光盘\MR\20\453

趣味指数：★★★★☆

实例说明

本实例通过使用 C#语言实现为指定的 Excel 工作簿设置和清除密码的功能，运行本实例，如图 20.5 所示。

单击“选择”按钮，在弹出的对话框中选择 Excel 文件的保存位置，然后单击“设置密码”按钮，弹出“输入——输入密码”对话框，如图 20.6 所示。在其中输入密码，单击“确定”按钮，弹出“输入——确认密码”对话框，如图 20.7 所示。在其中输入确认密码，单击“确定”按钮，即可为选择的 Excel 工作簿设置密码。单击“清除密码”按钮，如果 Excel 工作簿本身没有密码，则自动清除；否则，如果有密码，则弹出如图 20.8 所示的“密码”对话框，在其中输入正确的密码，单击“确定”按钮，即可清除该 Excel 工作簿的密码。

图 20.5　为 Excel 工作簿设置和清除密码

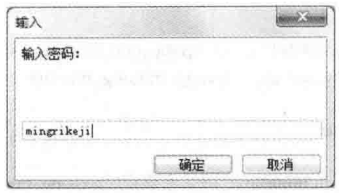


图 20.6　“输入——输入密码”对话框

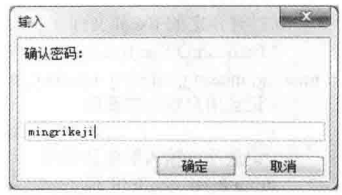


图 20.7　“输入——确认密码”对话框

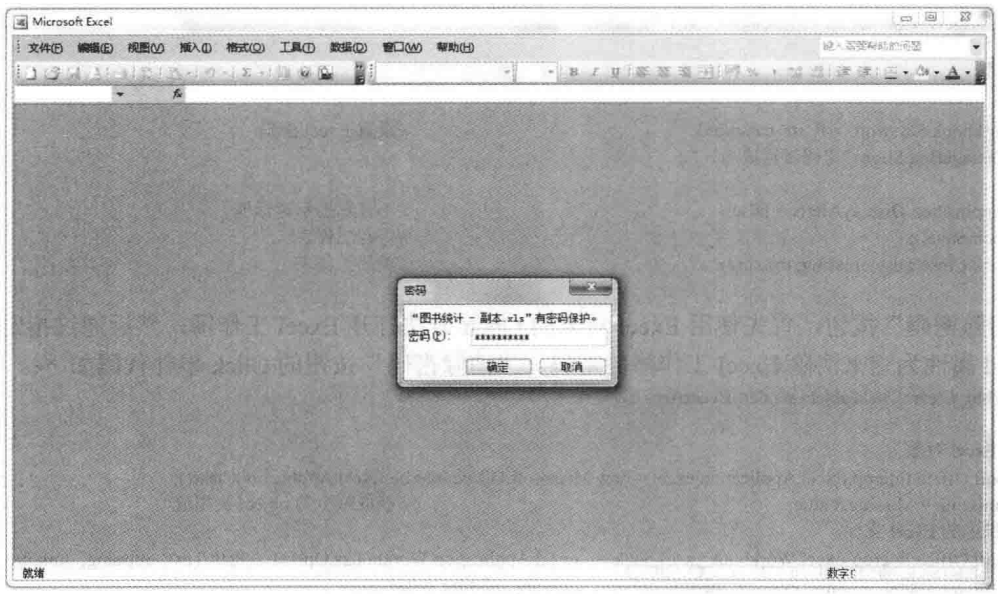


图 20.8　“密码”对话框

关键技术

本实例在为Excel工作簿设置和清除密码时，主要用到了 Excel 自动化对象模型的 Workbook 对象的 Password 属性，下面对其进行详细讲解。

Workbook 对象的 Password 属性主要用来获取或设置打开工作簿时必须提供的密码。语法格式如下：

```
[BrowsableAttribute(false)]
public virtual string Password { get; set; }
```

参数说明

属性值：打开工作簿时必须提供的密码。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetAndClearPWD。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的 Excel 文件路径；添加 3 个 Button 控件，分别用来执行选择 Excel 文件、为 Excel 文件设置密码和清除密码操作。

（3）程序的主要代码如下。

单击“设置密码”按钮，首先使用 Excel 对象的 Open 方法打开 Excel 工作簿，然后使用 Excel 对象的 InputBox

方法为打开的 Excel 工作簿输入密码和确认密码，最后通过指定 Workbook 对象的 Password 属性为 Excel 工作簿设置密码。“设置密码”按钮的 Click 事件代码如下：

```
private void btn_Set_Click(object sender, EventArgs e)
{
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.ApplicationClass();
    object missing = Missing.Value;                                    //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook=excel.Application.Workbooks.Open(txt_Path.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //记录用户输入的密码
    string P_str_password = excel.Application.InputBox("输入密码：",missing, missing, missing, missing, missing, missing, missing).ToString();
    //记录用户输入的确认密码
    string P_str_confirmPassword = excel.Application.InputBox("确认密码：", missing, missing, missing, missing, missing, missing, missing).ToString();
    if (P_str_password != P_str_confirmPassword)                       //判断密码与确认密码是否一致
    {
        MessageBox.Show("输入的密码不一致！");
    }
    else
    {
        workbook.Password = P_str_password;                            //设置 Excel 密码
        MessageBox.Show("密码设置成功！");
    }
    excel.Application.DisplayAlerts = false;                           //不显示提示对话框
    workbook.Save();                                                   //保存工作表
    workbook.Close(false, missing, missing);                           //关闭工作表
}
```

单击“清除密码”按钮，首先使用 Excel 对象的 Open 方法打开 Excel 工作簿，然后通过指定 Workbook 对象的 Password 属性为空来清除 Excel 工作簿的密码。“清除密码”按钮的 Click 事件代码如下：

```
private void btn_Clear_Click(object sender, EventArgs e)
{
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.ApplicationClass();
    object missing = Missing.Value;                                    //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Path.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    workbook.Password = "";                                            //设置 Excel 密码为空
    MessageBox.Show("密码清除成功！");
    excel.Application.DisplayAlerts = false;                           //不显示提示对话框
    workbook.Save();                                                   //保存工作表
    workbook.Close(false, missing, missing);                           //关闭工作表
}
```

秘笈心法

心法领悟 453：为 Excel 工作簿设置密码时的注意事项。

本实例使用了 Excel 对象的 InputBox 方法来为 Excel 工作簿设置密码，如果输入的密码相同，则认为输入的密码正确并更改原来的密码；如果输入的两个密码不同，则认为输入的密码不正确。

实例 454　为一个 Excel 文件创建多个工作表

初级

光盘位置：光盘\MR\20\454　　趣味指数：★★★★☆

实例说明

本实例演示如何使用 C#语言为一个 Excel 文件创建多个工作表。运行本实例，在窗体的菜单栏中选择“打

开 Excel 文件”选项，即可打开一个 Excel 文件，并显示在窗体中，如图 20.9 所示；选择“创建工作表”选项，即可在指定的 Excel 文件中创建新的工作表，如图 20.10 所示。

图 20.9　创建工作表前的 Excel 文件

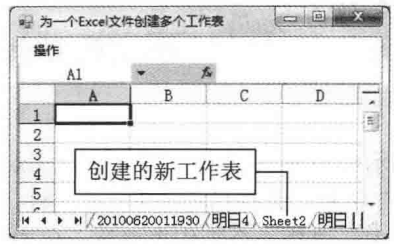


图 20.10　创建完工作表后的 Excel 文件

关键技术

本实例在为一个 Excel 文件创建多个工作表时，主要用到了 Excel 自动化对象模型的 Worksheets 对象的 Add 方法，下面对其进行详细讲解。

Worksheets 对象的 Add 方法主要用来为指定的工作簿添加工作表。语法格式如下：

```
object Add([In, Optional] object Before, [In, Optional] object After, [In, Optional] object Count, [In, Optional] object Type);
```

参数说明

❶ Before：要在其前面添加的工作表。

❷ After：要在其后面添加的工作表。

❸ Count：要添加的工作表数量。

❹ Type：工作表类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CreateMultiSheet。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏，在该菜单栏中添加两个子菜单项，分别用来执行打开 Excel 文件和为 Excel 文件创建工作表操作。

（3）程序的主要代码如下：

```
private void 创建工作表 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    CloseProcess();                                                    //关闭 Excel 进程
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    object missing = Missing.Value;                                    //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(M_str_Name, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, 1, missing);
    MessageBox.Show("添加工作表成功", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    excel.Application.DisplayAlerts = false;                           //不显示提示对话框
    workbook.Save();                                                   //保存工作表
    workbook.Close(false, missing, missing);                           //关闭工作表
    WBrowser_Excel.Navigate(M_str_Name);                               //在窗体中显示 Excel 文件内容
}
```

秘笈心法

心法领悟 454：如何在程序中打开 Excel 文件？

打开 Excel 文件时，可以使用 Workbooks 对象的 Open 方法来实现，该方法的第一个参数为要打开的 Excel 文件，其他的参数表示 Excel 的相关设置，实际使用时，直接用 Missing.Value 的值来表示即可。

实例 455 删除 Excel 文件中指定的工作表

初级

光盘位置：光盘\MR\20\455

趣味指数：★★★★☆

实例说明

本实例演示如何使用 C#语言实现删除 Excel 文件中指定工作表的功能。运行本实例，如图 20.11 所示，首先单击“选择”按钮，选择指定的 Excel 文件，这时程序自动将选中 Excel 文件所包含的所有工作表名称添加到一个下拉列表中，选择一个工作表名称，单击“删除”按钮，即可删除 Excel 文件中指定的工作表。

例如，删除 Book1.xls 工作簿中的 Sheet2 工作表，其删除前后的效果分别如图 20.12 和图 20.13 所示。

图 20.11 删除 Excel 文件中指定的工作表

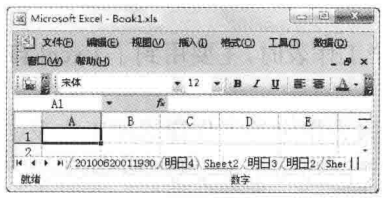


图 20.12 删除前的 Excel 文件

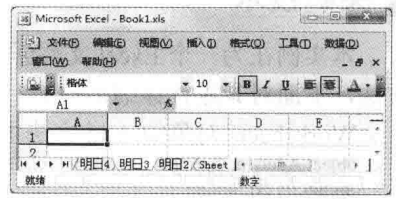


图 20.13 删除后的 Excel 文件

关键技术

本实例在删除 Excel 文件中指定的工作表时，主要用到了 Excel 自动化对象模型的 Workbook 对象的 Sheets 属性的 Delete 方法，下面对其进行详细讲解。

Workbook 对象的 Sheets 属性用来获取一个 Microsoft.Office.Interop.Excel.Sheets 集合，该集合表示工作簿中的所有工作表，该属性的 Delete 方法用来删除指定的工作表，其语法格式如下：

```
void Delete();
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DeleteSheet。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的 Excel 文件路径；添加一个 ComboBox 控件，用来显示指定 Excel 文件中的所有工作表名称；添加两个 Button 控件，分别用来执行打开 Excel 文件和删除 Excel 文件中指定工作表的操作。

（3）程序的主要代码如下：

```
private void btn_Delete_Click(object sender, EventArgs e)
{
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.ApplicationClass();
    object missing = Missing.Value;                                    //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Path.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //删除选择的工作表
    ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Sheets[cbox_SheetName.Text]).Delete();
    MessageBox.Show("工作表删除成功！");
    excel.Application.DisplayAlerts = false;                           //不显示提示对话框
    workbook.Save();                                                   //保存工作表
    CBoxBind();                                                        //对下拉列表进行数据绑定
    //创建进程对象
    System.Diagnostics.Process[] excelProcess = System.Diagnostics.Process.GetProcessesByName("EXCEL");
    foreach (System.Diagnostics.Process p in excelProcess)
        p.Kill();                                                      //关闭进程
}
```

秘笈心法

心法领悟 455：如何在保存 Excel 时不显示提示对话框？

如果要在保存 Excel 时不显示提示对话框，可以通过设置 Excel 对象的 DisplayAlerts 属性为 false 来实现。

20.2　读取外部数据到 Excel

实例 456	读取文本文件到 Excel 并对数据列进行格式化	中级
	光盘位置：光盘\MR\20\456	趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将文本文件中的内容读取到 Excel 的功能，并在 Excel 中对数据列进行格式化。运行本实例，如图 20.14 所示，单击第一个“选择”按钮，选择要读取的文本文件；单击第二个“选择”按钮，选择 Excel 文件；然后单击“读取”按钮，将文本文件中的数据读取到 Excel 文件中，并对数据列进行格式化；单击“查看”按钮，查看导入数据之后的 Excel 文件。

图 20.14　读取文本文件到 Excel 并对数据列进行格式化

例如，将如图 20.15 所示的文本文件内容读取到 Excel 文件中，则读取之后的 Excel 文件如图 20.16 所示。

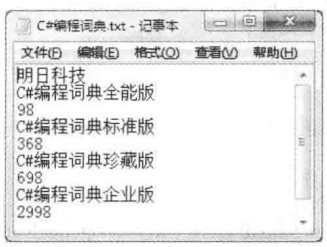


图 20.15　文本文件数据

图 20.16　读取到 Excel 中的数据

关键技术

本实例在将文本文件的内容读取到 Excel 中时，主要用到了 StreamReader 类的 ReadLine 方法和 Excel 自动化对象模型的 Excel 对象的 Cells 属性。下面对本实例中用到的关键技术进行详细讲解。

（1）StreamReader 类的 ReadLine 方法

StreamReader 类实现了 TextReader 抽象类，它主要用来使用一种特定的编码从字节流中读取字符，该类的 ReadLine 方法用来从当前流中读取一行字符并将数据作为字符串返回。语法格式如下：

```
public override string ReadLine()
```

参数说明

返回值：输入流中的下一行；如果到达了输入流的末尾，则为 null。

（2）Excel 对象的 Cells 属性

该属性用来指定 Excel 工作簿中单元格的内容。例如，下面的代码用来指定 Excel 工作簿中第一行第一列的单元格中的内容为 Excel。

```
excel.Cells[1, 1] = "Excel";
```

注意： Excel 对象的 Cells 属性的行索引和列索引都从 1 开始，这与 C#中其他的索引从 0 开始是有区别的。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TxtToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的文本文件路径和 Excel 文件路径；添加 4 个 Button 控件，分别用来执行选择文本文件、选择 Excel 文件、将文本文件中的数据导入到 Excel 文件和查看 Excel 文件操作。

（3）程序的主要代码如下：

```
private void btn_Read_Click(object sender, EventArgs e)
{
    int P_int_Count=0;                                                    //记录正在读取的行数
    string P_str_Line, P_str_Content = "";                                //记录读取行的内容及遍历到的内容
    List<string> P_str_List = new List<string>();                         //存储读取的所有内容
    StreamReader SReader = new StreamReader(txt_Txt.Text, Encoding.Default);//创建流读取对象
    while ((P_str_Line = SReader.ReadLine()) != null)                     //循环读取文本文件中的每一行
    {
        P_str_List.Add(P_str_Line);                                       //将读取到的行内容添加到泛型集合中
        P_int_Count++;                                                    //使当前读取行数加 1
    }
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    object missing = System.Reflection.Missing.Value;                     //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Excel.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    excel.Application.DisplayAlerts = false;                              //不显示提示对话框
    for (int i = 0; i < P_str_List.Count; i++)                            //遍历泛型集合
    {
        P_str_Content = P_str_List[i];                                    //记录遍历到的值
        if (Regex.IsMatch(P_str_Content, "^[0-9]*[1-9][0-9]*$"))          //判断是否是数字
            //格式化为货币格式，再添加到工作表中
            newWorksheet.Cells[i + 1, 1] = Convert.ToDecimal(P_str_Content).ToString("￥00.00");
        else
            newWorksheet.Cells[i + 1, 1] = P_str_Content;                 //直接将遍历到的内容添加到工作表中
    }
    workbook.Save();                                                      //保存工作表
    workbook.Close(false, missing, missing);                              //关闭工作表
    MessageBox.Show("已经将文本文件内容成功导入 Excel 工作表中！ ", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

说明： 本实例中由于需要使用文件流对文本文件进行读取，所以首先需要在命名空间区域添加 System.IO 命名空间。下面再次涉及对文件流的相关操作时，将不再提示。

秘笈心法

心法领悟 456：轻松回收 StreamReader 对象资源。

程序设计过程中，可以手动调用 StreamReader 对象的 Close 方法回收文件资源，但是使用 using 关键字则更为简洁友好。当 using 关键字下面的代码执行完毕后，会自动将 using 小括号中创建的 StreamReader 对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 457　将文本文件数据分解到 Excel 中的不同数据表

光盘位置：光盘\MR\20\457　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将文本文件中的内容分解到 Excel 中不同数据表的功能。运行本实例，如图 20.17 所示，单击第一个“选择”按钮，选择要读取的文本文件；单击第二个“选择”按钮，选择 Excel 文件；然后单击“读取”按钮，将文本文件中的数据分解到 Excel 文件的不同数据表中；单击“查看”按钮，查看分解完数据之后的 Excel 文件。

图 20.17　将文本文件数据分解到 Excel 中的不同数据表

例如，将如图 20.18 所示的文本文件内容分解到 Excel 文件的不同数据表中，则分解之后的 Excel 文件如图 20.19 所示。

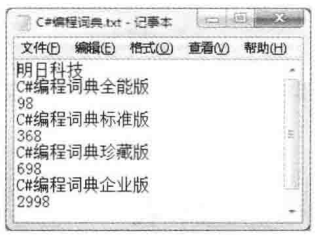


图 20.18　文本文件数据

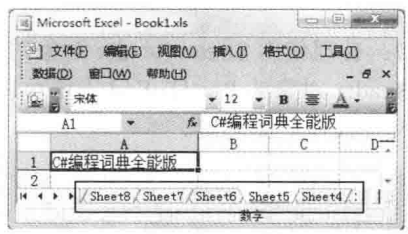


图 20.19　分解之后的 Excel 文件

关键技术

本实例在将文本文件的内容分解到 Excel 中的不同数据表中时，首先需要使用 StreamReader 类的 ReadLine 方法依次读取文本文件中的内容，然后使用 Excel 自动化对象模型的 Worksheets 对象的 Add 方法添加多个工作表，最后通过 Excel 自动化对象模型的 Excel 对象的 Cells 属性为工作表的单元格设置值。

说明：（1）关于 StreamReader 类的 ReadLine 方法和 Excel 对象的 Cells 属性的详细讲解请参见实例 456 中的关键技术。

（2）关于 Worksheets 对象的 Add 方法的详细讲解请参见实例 454 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TxtToMulSheets。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的文本文件路径和 Excel 文件路径；添加 4 个 Button 控件，分别用来执行选择文本文件、选择 Excel 文件、将文本文件中的数据导入到 Excel 文件中的不同工作表和查看 Excel 文件操作。

（3）程序的主要代码如下：

```
private void btn_Read_Click(object sender, EventArgs e)
{
    int P_int_Count = 0;                                    //记录正在读取的行数
```

```
    string P_str_Line, P_str_Content = "";                                    //记录读取行的内容及遍历到的内容
    List<string> P_str_List = new List<string>();                             //存储读取的所有内容
    StreamReader SReader = new StreamReader(txt_Txt.Text, Encoding.Default);  //创建流读取对象
    while ((P_str_Line = SReader.ReadLine()) != null)                         //循环读取文本文件中的每一行
    {
        P_str_List.Add(P_str_Line);                                           //将读取到的行内容添加到泛型集合中
        P_int_Count++;                                                        //使当前读取行数加 1
    }
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    object missing = System.Reflection.Missing.Value;                         //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Excel.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet;                    //声明工作表对象
    for (int i = 0; i < P_str_List.Count; i++)                                //遍历泛型集合
    {
        P_str_Content = P_str_List[i];                                        //记录遍历到的值
        //创建新工作表
        newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing, missing, missing, missing);
        newWorksheet.Cells[1, 1] = P_str_Content;                             //直接将遍历到的内容添加到工作表中
    }
    excel.Application.DisplayAlerts = false;                                  //不显示提示对话框
    workbook.Save();                                                          //保存工作表
    workbook.Close(false, missing, missing);                                  //关闭工作表
    MessageBox.Show("已经将文本文件的内容分解到了 Excel 的不同数据表中！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 457：合理使用 StreamReader 对象的 ReadToEnd 方法。

使用 StreamReader 对象的 ReadToEnd 方法可以方便地一次性全部读取出文本文件的所有文本信息，适当地使用此方法会使文件操作更加便捷。

实例 458　读取多个 Word 文档文件到 Excel 同一数据表

中级

光盘位置：光盘\MR\20\458

趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将多个 Word 文档中的内容读取到 Excel 同一数据表的功能。运行本实例，如图 20.20 所示，单击第一个“选择”按钮，选择要读取的多个 Word 文档；单击第二个“选择”按钮，选择 Excel 文件；然后单击“读取”按钮，将多个 Word 文档中的数据读取到 Excel 文件中；单击“查看”按钮，查看导入数据之后的 Excel 文件。

例如，将如图 20.21 和图 20.22 所示的两个 Word 文档内容读取到 Excel 文件的同一数据表中，则读取之后的 Excel 文件如图 20.23 所示。

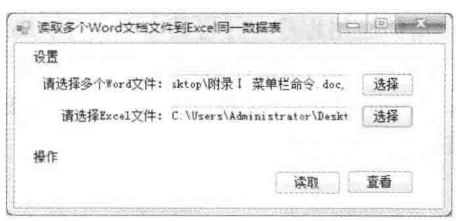


图 20.20　读取多个 Word 文档文件到 Excel 同一数据表

图 20.21　第一个 Word 文档

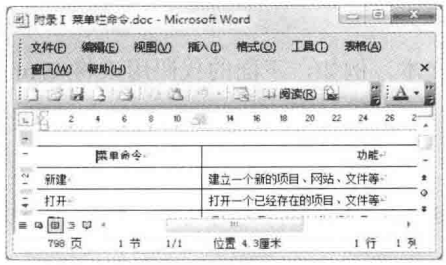


图 20.22　第二个 Word 文档

图 20.23　读取之后的 Excel 文件

说明：图 20.21 所示的 Word 文档中全部是文本，而图 20.22 所示的 Word 文档中存在表格数据，所以在将这两个 Word 文档读取到同一个 Excel 工作表中时，需要使用不同的方法。

关键技术

本实例在将多个 Word 文档中的内容读取到 Excel 的同一数据表时，首先需要使用 Microsoft Word 自动化对象模型中的 Word 对象打开 Word 文档；然后使用 Document 对象的 Content.Text 属性获取 Word 文档中的内容，另外，如果 Word 文档中存在表格，则需要遍历表格的所有行，并且使用 Table 对象的 Cell(row, column).Range.Text 属性获取遍历到的单元格内容；最后通过 Excel 自动化对象模型的 Excel 对象的 Cells 属性将 Word 文档中的内容读取到 Excel 数据表中。下面对本实例中用到的关键技术进行详细讲解。

（1）使用 Word 对象打开 Word 文档

使用 Word 对象打开 Word 文档时，需要使用 Word 对象的 Documents 对象的 Open 方法，该方法用来打开完全限定路径和文件名所指定的现有 Word 文档，它返回 Microsoft.Office.Interop.Word.Document，表示一个 Word 文档。Open 方法的语法格式如下：

```
Document Open([In] ref object FileName,[In, Optional] ref object ConfirmConversions,[In, Optional] ref object ReadOnly,[In, Optional] ref object AddToRecentFiles,[In, Optional] ref object PasswordDocument,[In, Optional] ref object PasswordTemplate,[In, Optional] ref object Revert,[In, Optional] ref object WritePasswordDocument,[In, Optional] ref object WritePasswordTemplate,[In, Optional] ref object Format,[In, Optional] ref object Encoding, [In, Optional] ref object Visible,[In, Optional] ref object OpenAndRepair,[In, Optional] ref object DocumentDirection,[In, Optional] ref object NoEncodingDialog, [In, Optional] ref object XMLTransform);
```

Open 方法中的参数说明如表 20.1 所示。

表 20.1　Open 方法中的参数说明

参　数	描　述
FileName	Word 文档名及路径
ConfirmConversions	是否显示文件转换对话框
ReadOnly	指定 Word 文档是否只读
AddToRecentFiles	是否将文档添加到最近使用的文件列表中
PasswordDocument	文档密码
PasswordTemplate	文档模板密码
Revert	是否恢复文档
WritePasswordDocument	文档保存密码
WritePasswordTemplate	文档模板保存密码
Format	指定使用的文件转换器
Encoding	指定打开文档时使用的编码格式
Visible	指定文档是否可见
OpenAndRepair	尝试修复损坏的文档
DocumentDirection	指定已打开文档中文本的水平流动方式
NoEncodingDialog	不显示编码对话框
XMLTransform	XML 转换格式

（2）Document 对象的 Content.Text 属性

Document 对象的 Content.Text 属性用来获取 Word 文档中的文本。例如，下面的代码用来将 Word 文档中的内容添加到 Excel 工作表的指定单元格中。

```
newWorksheet.Cells[i + P_int_Row + 1, 1] = document.Content.Text;
```

（3）Table 对象的 Cell(row, column).Range.Text 属性

Table 对象的 Cell(row, column).Range.Text 属性用来获取 Word 表格中指定范围的文本。例如，下面的代码用来获取 Word 表格中指定行和列的文档。

```
P_str_Content = table.Cell(row, column).Range.Text;                    //获取遍历到的单元格内容
```

说明：关于 Worksheets 对象的 Add 方法的详细讲解请参见实例 454 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiWordToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的多个 Word 文件路径和 Excel 文件路径；添加 4 个 Button 控件，分别用来执行选择多个 Word 文件、选择 Excel 文件、将多个 Word 文件中的数据导入到 Excel 文件的同一工作表和查看 Excel 文件操作。

（3）程序的主要代码如下：

```
private void btn_Read_Click(object sender, EventArgs e)
{
//创建 Word 对象
    Microsoft.Office.Interop.Word.Application word = new Microsoft.Office.Interop.Word.Application();
    Microsoft.Office.Interop.Word.Table table;                         //声明 Word 表格对象
//定义 3 个变量，分别用来存储表格数量、表格中的行数和列数
    int P_int_TableCount = 0, P_int_Row = 0, P_int_Column = 0;
    string P_str_Content;                                              //存储 Word 表格的单元格内容
    object missing = System.Reflection.Missing.Value;                 //获取缺少的 object 类型值
    string[] P_str_Names = txt_Word.Text.Split(',');                   //存储所有选择的 Word 文件名
object P_obj_Name;                                                     //存储遍历到的 Word 文件名
//创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Excel.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
//创建新工作表
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    for (int i = 0; i < P_str_Names.Length - 1; i++)                   //遍历所有选择的 Word 文件名
    {
        P_obj_Name = P_str_Names[i];                                   //记录遍历到的 Word 文件名
        //打开 Word 文档
        Microsoft.Office.Interop.Word.Document document = word.Documents.Open(ref P_obj_Name, ref missing, ref missing, ref missing, ref
missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing);
        P_int_TableCount = document.Tables.Count;                      //获取 Word 文档中表格的数量
        if (P_int_TableCount > 0)                                      //判断表格数量是否大于 0
        {
            for (int j = 1; j <= P_int_TableCount; j++)                //遍历所有表格
            {
                table = document.Tables[j];                            //记录遍历到的表格
                P_int_Row = table.Rows.Count;                          //获取表格行数
                P_int_Column = table.Columns.Count;                    //获取表格列数
                for (int row = 1; row <= P_int_Row; row++)             //遍历表格中的行
                {
                    for (int column = 1; column <= P_int_Column; column++)   //遍历表格中的列
                    {
```

```
                    P_str_Content = table.Cell(row, column).Range.Text;                    //获取遍历到的单元格内容
//将遍历到的单元格内容添加到 Excel 单元格中
                    newWorksheet.Cells[i + row, column] = P_str_Content.Substring(0, P_str_Content.Length - 2);
                }
            }
        }
    }
    else
    {
        if (P_int_Row == 0)                                                    //判断前面是否已经填充过表格
            newWorksheet.Cells[i + P_int_Row + 1, 1] = document.Content.Text;   //直接将 Word 文档内容添加到工作表中
        else
            newWorksheet.Cells[i + P_int_Row, 1] = document.Content.Text;       //直接将 Word 文档内容添加到工作表中
    }
    document.Close(ref missing, ref missing, ref missing);                     //关闭 Word 文档
}
excel.Application.DisplayAlerts = false;                                       //不显示提示对话框
workbook.Save();                                                               //保存工作表
workbook.Close(false, missing, missing);                                       //关闭工作表
word.Quit(ref missing, ref missing, ref missing);                              //退出 Word 应用程序
MessageBox.Show("已经成功将多个 Word 文档的内容合并到了 Excel 的同一个数据表中！", "提示", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}
```

说明：（1）本实例在使用 Word 对象时，首先需要添加该对象的引用，具体步骤是，选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令，弹出“引用管理器”对话框，如图 20.24 所示。在该对话框中选中要引用的 Word 动态链接库，这里选择 Microsoft.Office.Interop.Word，然后单击“确定”按钮，即可将 Word 对象引用添加到程序中。

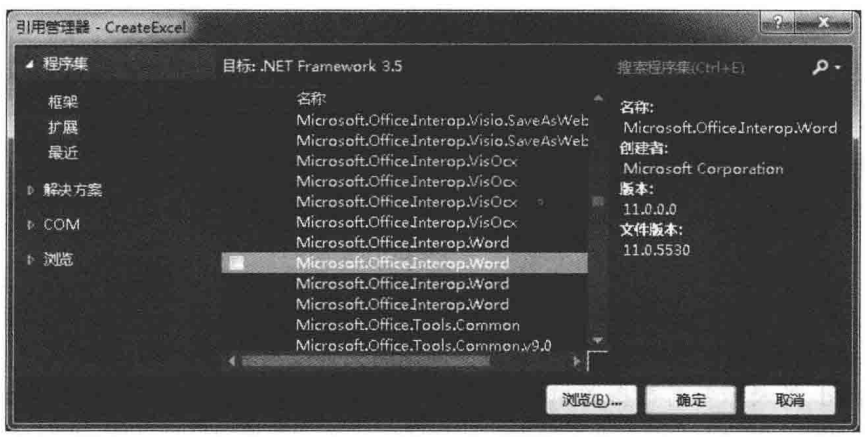


图 20.24　“引用管理器”对话框

（2）本章后面的实例中，如果需要对 Word 进行操作，则都需要按（1）所讲的步骤添加 Word 对象引用，由于添加的步骤是一样的，所以在后面涉及时将不再赘述。

秘笈心法

心法领悟 458：使用 Cell 方法轻松地获取或设置 Word 表格中的文本信息。

Word 表格对象的 Cell 方法提供了两个参数并返回一个 Range 对象，方法中的两个参数分别为表格对象中的行与列的索引。需要注意的是，参数中行与列的索引都是从 1 开始的，根据两个参数可以找到并返回表格中的单元格的 Range 对象，并可以根据 Range 对象的 Text 属性获取或设置单元格的文本内容。

实例 459 读取 Access 数据到 Excel

光盘位置：光盘\MR\20\459　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将 Access 数据库中指定表的数据读取到 Excel 的功能。运行本实例，如图 20.25 所示，单击第一个“选择”按钮，选择要读取的 Access 文件，并在下拉列表中选择要读取的 Access 数据表；单击第二个“选择”按钮，选择 Excel 文件；然后单击“执行”按钮，将 Access 数据库中的指定数据表的数据读取到 Excel 文件中；单击“查看”按钮，查看导入数据之后的 Excel 文件。

图 20.25　读取 Access 数据到 Excel

例如，将如图 20.26 所示的 Access 数据库中的 tb_PDic 数据表的内容读取到 Excel 文件中，则读取之后的 Excel 文件如图 20.27 所示。

图 20.26　Access 数据库中的 tb_PDic 数据表内容

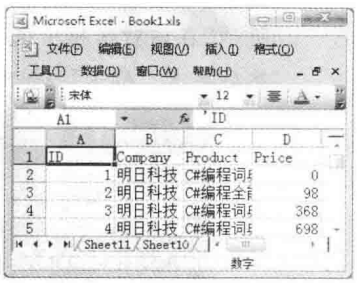


图 20.27　读取之后的 Excel 文件

关键技术

本实例在将 Access 数据库中的数据读取到 Excel 时，主要使用 OleDbCommand 类的 ExecuteNonQuery 方法执行相应的 SQL 语句来进行实现。下面对本实例中用到的关键技术进行详细讲解。

OleDbCommand 类表示要对数据源执行的 SQL 语句或存储过程，其 ExecuteNonQuery 方法用来针对 Connection 连接对象执行 SQL 语句并返回受影响的行数，该方法的语法格式如下：

```
public override int ExecuteNonQuery()
```

参数说明

返回值：受影响的行数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AccessToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的 Access 文件路径和 Excel 文件路径；添加一个 ComboBox 控件，用来显示 Access 数据库中的所有数据表名称；添加 4 个 Button 控件，分别用来执行选择 Access 文件、选择 Excel 文件、将 Access 数据库中指定数据表的数据导入到 Excel 文件和查看 Excel 文件操作。

（3）程序的主要代码如下：

```
private void btn_Read_Click(object sender, EventArgs e)
{
    try
    {
        //记录连接 Access 的语句
        string P_str_Con = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + txt_Access.Text + ";Persist Security Info=True";
        string P_str_Sql = "";                                          //存储要执行的 SQL 语句
        OleDbConnection oledbcon = new OleDbConnection(P_str_Con);      //创建 OLEDB 连接对象
        OleDbCommand oledbcom;                                          //定义 OleDbCommand 对象
        oledbcon.Open();                                                //打开数据库连接
        //向 Excel 工作表中导入数据
        P_str_Sql = @"select * into [Excel 8.0;database=" + txt_Excel.Text + "]." + "[" + cbox_Table.Text + "] from " + cbox_Table.Text + "";
        //创建 OleDbCommand 对象
        oledbcom = new System.Data.OleDb.OleDbCommand(P_str_Sql, oledbcon);
        oledbcom.ExecuteNonQuery();                                     //将数据表的内容导入到 Excel 中
        oledbcon.Close();                                               //关闭数据库连接
        oledbcon.Dispose();                                             //释放资源
        MessageBox.Show("导入成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch
    {
        MessageBox.Show(cbox_Table.Text + "工作表已经存在，请选择其他数据表！", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

说明：本实例在使用 OLEDB 对象时，首先需要在命名空间区域添加 System.Data.Oledb 命名空间。下面再次涉及 OLEDB 对象的使用时，将不再提示。

秘笈心法

心法领悟 459：向 Excel 工作表中导入数据的 SQL 语句。

将 Access 数据表中的数据导入到 Excel 时，可以直接使用 SQL 语句实现。实现的 SQL 语句如下：

```
P_str_Sql = @"select * into [Excel 8.0;database=Excel 文件名].[Excel 工作表名] from Access 数据表名";
```

实例 460　将 SQL Server 数据查询结果输出到 Excel

光盘位置：光盘\MR\20\460　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将 SQL Server 数据查询结果输出到 Excel 的功能。运行本实例，如图 20.28 所示，首先选择查询条件，并输入查询关键字，单击“查询”按钮，在 SQL Server 数据库中执行查询，并将查询结果显示在数据表格控件中；然后单击“导出 Excel”按钮，将数据表格控件中的查询结果输出到 Excel 文件中。

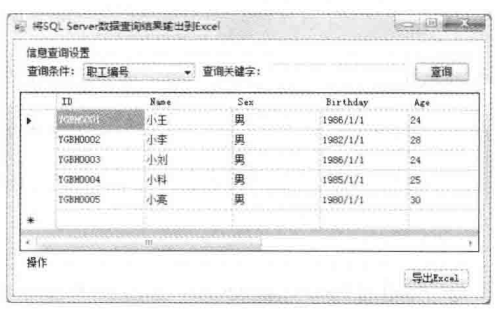


图 20.28　将 SQL Server 数据查询结果输出到 Excel

例如，在如图 20.29 所示的 SQL Server 数据库中查询性别为“男”的数据，并将查询之后的数据输出到 Excel

文件中，则输入之后的 Excel 文件如图 20.30 所示。

图 20.29　SQL Server 数据库中的数据

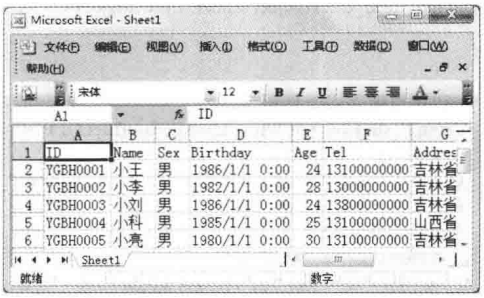


图 20.30　输入数据之后的 Excel 文件

关键技术

本实例在将 SQL Server 数据库的查询结果输出到 Excel 时，首先需要将 SQL Server 数据库的查询结果显示在 DataGridView 控件中，然后遍历 DataGridView 控件中的数据，并通过设置 Excel 对象的 Cells 属性来将遍历到的值填充到 Excel 的单元格中。

说明：关于 Excel 对象的 Cells 属性的详细讲解请参见实例 456 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SqlToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，用来选择查询条件；添加一个 TextBox 控件，用来输入查询关键字；添加一个 DataGridView 控件，用来显示 SQL Server 数据库中的数据；添加两个 Button 控件，分别用来执行数据库查询和将数据库查询结果输出到 Excel 文件中的操作。

（3）程序的主要代码如下：

```
private void btn_Excel_Click(object sender, EventArgs e)
{
    if (dgv_Info.Rows.Count == 0)                                    //判断是否有数据
        return;                                                      //返回
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    excel.Application.Workbooks.Add(true);                           //在 Excel 中添加一个工作簿
    excel.Visible = true;                                            //设置 Excel 显示
    //生成字段名称
    for (int i = 0; i < dgv_Info.ColumnCount; i++)
    {
        excel.Cells[1, i + 1] = dgv_Info.Columns[i].HeaderText;      //将数据表格控件中的列表头填充到 Excel 中
    }
    //填充数据
    for (int i = 0; i < dgv_Info.RowCount - 1; i++)                  //遍历数据表格控件的所有行
    {
        for (int j = 0; j < dgv_Info.ColumnCount; j++)               //遍历数据表格控件的所有列
        {
            if (dgv_Info[j, i].ValueType == typeof(string))          //判断遍历到的数据是否是字符串类型
            {
                excel.Cells[i + 2, j + 1] = "'" + dgv_Info[j, i].Value.ToString();   //填充 Excel 表格
            }
            else
            {
                excel.Cells[i + 2, j + 1] = dgv_Info[j, i].Value.ToString();         //填充 Excel 表格
            }
        }
    }
}
```

秘笈心法

心法领悟 460：如何连接 SQL Server 数据库？

C#中连接 SQL Server 数据库时，需要使用 SqlConnection 类，该类位于 System.Data.SqlClient 命名空间下。

实例 461　实时读取不同数据到 Excel 进行汇总处理　高级

光盘位置：光盘\MR\20\461　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了实时读取不同数据到 Excel 进行汇总处理的功能。运行本实例，如图 20.31 所示，单击第一个“选择”按钮，选择要读取的文本文件；单击第二个“选择”按钮，选择要读取的 Word 文件；单击第三个“选择”按钮，选择要读取的 Access 数据库文件；单击第 4 个“选择”按钮，选择 Excel 文件；单击“开始”按钮，启动 Timer 计时器，并实时将各种不同的数据读取到 Excel 文件进行汇总处理；单击“停止”按钮，停止 Timer 计时器，同时停止向 Excel 文件中读取数据；单击“查看”按钮，查看导入数据之后的 Excel 文件。

图 20.31　实时读取不同数据到 Excel 进行汇总处理

关键技术

本实例主要实现将 3 种不同的数据（文本文件数据、Word 文档数据和 Access 数据库数据）实时汇总到 Excel 中的功能，在将这 3 种数据汇总到 Excel 时，分别需要用到文本文件的流读取、Word 文档内容的获取和 OLEDB 获取 Access 数据等技术，另外，还需要使用 Timer 计时器来控制数据的实时汇总。下面对本实例中用到的关键技术进行详细讲解。

Timer 计时器可以按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，其 Interval 属性用来获取或设置在相对于上一次发生的 Tick 事件引发 Tick 事件之前的时间（以毫秒为单位）。Interval 属性的语法格式如下：

```
public int Interval { get; set; }
```

参数说明

属性值：指定在相对于上一次发生的 Tick 事件引发 Tick 事件之前的毫秒数，该值不能小于 1。

说明：（1）关于文本文件的流读取的详细讲解请参见实例 456 中的关键技术。

（2）关于 Word 文档内容的获取的详细讲解请参见实例 458 中的关键技术。

（3）关于使用 OLEDB 获取 Access 数据的详细讲解请参见实例 459 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DisDataToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 TextBox 控件，分别用来显示

选择的文本文件路径、Word 文档路径、Access 数据库文件路径和 Excel 文件路径；添加 7 个 Button 控件，分别用来执行选择文本文件、选择 Word 文档、选择 Access 数据库文件、选择 Excel 文件、开始和停止将不同数据导入到 Excel 文件中、查看 Excel 文件操作；添加一个 Timer 组件，用来实时将各种不同的数据导入到 Excel 中进行汇总处理。

（3）程序的主要代码如下。

在 timer1 组件的 Tick 事件中，首先判断是否选择了 Excel 文件，如果已经选择，则分别判断是否选择了文本文件、Word 文件及 Access 数据库文件，并调用相应的自定义方法将各种不同的数据导入到选择的 Excel 文件中。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (txt_Excel.Text != "")                                          //判断是否选择了 Excel 文件
    {
        if (txt_Txt.Text != "")                                        //判断是否选择了文本文件
            TxtToExcel(txt_Txt.Text, txt_Excel.Text);                  //将文本文件数据导入 Excel
        if (txt_Word.Text != "")                                       //判断是否选择了 Word 文件
            WordToExcel(txt_Word.Text, txt_Excel.Text);                //将 Word 文件数据导入 Excel
        if (txt_Access.Text != "")                                     //判断是否选择了 Access 文件
        {
            List<string> P_list_Tables = GetTable(txt_Access.Text);    //获取 Access 中的所有表
            for (int i = 0; i < P_list_Tables.Count; i++)              //遍历所有表
            {
                AccessToExcel(txt_Access.Text, P_list_Tables[i], txt_Excel.Text);   //将表中的数据导入 Excel
            }
        }
    }
}
```

上面的代码中用到了 TxtToExcel、WordToExcel 和 AccessToExcel 方法，这 3 种方法是用户自定义的方法，下面分别进行介绍。TxtToExcel 方法用来将文本文件的数据导入到 Excel 文件中，该方法中主要是通过文件流读取文本文件中的数据，并将读取到的数据填充到指定的 Excel 单元格中。TxtToExcel 方法的实现代码如下：

```
private void TxtToExcel(string P_str_Txt,string P_str_Excel)
{
    int P_int_Count = 0;                                               //记录正在读取的行数
    string P_str_Line;                                                 //记录读取行的内容
    List<string> P_str_List = new List<string>();                      //存储读取的所有内容
    StreamReader SReader = new StreamReader(P_str_Txt, Encoding.Default);   //创建流读取对象
    while ((P_str_Line = SReader.ReadLine()) != null)                  //循环读取文本文件中的每一行
    {
        P_str_List.Add(P_str_Line);                                    //将读取到的行内容添加到泛型集合中
        P_int_Count++;                                                 //使当前读取行数加 1
    }
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    object missing = System.Reflection.Missing.Value;                  //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing, missing,
missing, missing);
    excel.Application.DisplayAlerts = false;                           //不显示提示对话框
    for (int i = 0; i < P_str_List.Count; i++)                         //遍历泛型集合
    {
        newWorksheet.Cells[i + 1, 1] = P_str_List[i];                  //直接将遍历到的内容添加到工作表中
    }
    workbook.Save();                                                   //保存工作表
    workbook.Close(false, missing, missing);                           //关闭工作表
}
```

WordToExcel 方法用来将 Word 文件中的数据导入到 Excel 文件中，该方法中首先需要判断 Word 文件中是否有表格存在，如果有表格存在，则将 Word 表格中的数据一一插入到 Excel 文件的对应单元格中；如果没有表格存在，则将 Word 文件中的所有内容整体插入到 Excel 文件的一个单元格中。WordToExcel 方法的实现代码如下：

```
private void WordToExcel(string P_str_Word, string P_str_Excel)
{
    //创建 Word 对象
    Microsoft.Office.Interop.Word.Application word = new Microsoft.Office.Interop.Word.Application();
    Microsoft.Office.Interop.Word.Table table;                                    //声明 Word 表格对象
    //定义 3 个变量，分别用来存储表格数量、表格中的行数和列数
    int P_int_TableCount = 0, P_int_Row = 0, P_int_Column = 0;
    string P_str_Content;                                                        //存储 Word 表格的单元格内容
    object missing = System.Reflection.Missing.Value;                           //获取缺少的 object 类型值
    object P_obj_Name;                                                           //存储遍历到的 Word 文件名
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开指定的 Excel 文件
Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //创建新工作表
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    P_obj_Name = (object)P_str_Word;                                             //记录遍历到的 Word 文件名
    //打开 Word 文档
    Microsoft.Office.Interop.Word.Document document = word.Documents.Open(ref P_obj_Name, ref missing, ref missing, ref missing, ref missing,
ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing);
    word.DisplayAlerts = Microsoft.Office.Interop.Word.WdAlertLevel.wdAlertsNone;//设置操作 Word 时不显示任何对话框
    P_int_TableCount = document.Tables.Count;                                    //获取 Word 文档中表格的数量
    if (P_int_TableCount > 0)                                                    //判断表格数量是否大于 0
    {
        for (int j = 1; j <= P_int_TableCount; j++)                              //遍历所有表格
        {
            table = document.Tables[j];                                          //记录遍历到的表格
            P_int_Row = table.Rows.Count;                                        //获取表格行数
            P_int_Column = table.Columns.Count;                                  //获取表格列数
            for (int row = 1; row <= P_int_Row; row++)                           //遍历表格中的行
            {
                for (int column = 1; column <= P_int_Column; column++)           //遍历表格中的列
                {
                    P_str_Content = table.Cell(row, column).Range.Text;          //获取遍历到的单元格内容
                    //将遍历到的单元格内容添加到 Excel 单元格中
                    newWorksheet.Cells[row, column] = P_str_Content.Substring(0, P_str_Content.Length - 2);
                }
            }
        }
    }
    else
    {
        if (P_int_Row == 0)                                                      //判断前面是否已经填充过表格
            //直接将 Word 文档内容添加到工作表中
            newWorksheet.Cells[P_int_Row + 1, 1] = document.Content.Text;
        else
            //直接将 Word 文档内容添加到工作表中
            newWorksheet.Cells[P_int_Row, 1] = document.Content.Text;
    }
    document.Close(ref missing, ref missing, ref missing);                       //关闭 Word 文档
    excel.Application.DisplayAlerts = false;                                     //不显示提示对话框
    workbook.Save();                                                             //保存工作表
    workbook.Close(false, missing, missing);                                     //关闭工作表
    word.Quit(ref missing, ref missing, ref missing);                            //退出 Word 应用程序
}
```

AccessToExcel 方法用来将 Access 数据库中的指定数据表的内容导入到 Excel 文件中，该方法主要通过使用“select * into from 表名” SQL 语句来进行实现。AccessToExcel 方法的实现代码如下：

```
private void AccessToExcel(string P_str_Access,string P_str_Table,string P_str_Excel)
{
    string P_str_Con = @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + P_str_Access + ";Persist Security Info=True";
    //记录连接 Access 的语句
    string P_str_Sql = "";                                                       //存储要执行的 SQL 语句
    OleDbConnection oledbcon = new OleDbConnection(P_str_Con);                   //创建 OLEDB 连接对象
```

```
    OleDbCommand oledbcom;                                          //定义 OleDbCommand 对象
    oledbcon.Open();                                                //打开数据库连接
P_str_Sql = @"select * into [Excel 8.0;database=" + P_str_Excel + "]." + "[" + P_str_Table + "] from " + P_str_Table + "";
    //记录连接 Excel 的语句
    //创建 OleDbCommand 对象
    oledbcom = new System.Data.OleDb.OleDbCommand(P_str_Sql, oledbcon);
    oledbcom.ExecuteNonQuery();                                     //将数据表的内容导入到 Excel 中
    oledbcon.Close();                                               //关闭数据库连接
    oledbcon.Dispose();                                             //释放资源
}
```

秘笈心法

心法领悟 461：使用线程代替 Timer 组件。

Timer 组件可以按用户定义的时间间隔来引发 Tick 事件，Tick 事件一般为周期性的，每隔若干秒或若干毫秒执行一次，Timer 组件工作在窗体线程中，如果 Timer 组件中执行了较为耗时的操作，会增加窗体线程的负担，导致窗体中其他操作不能及时得到 CPU 资源，出现窗体长时间或短时间无响应的情况。在适当的情况下可以使用线程来代替 Timer 组件，这样会减少窗体线程的负担。

实例 462　每天定时将各地上报结果处理到 Excel

高级

光盘位置：光盘\MR\20\462

趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了每天定时将各地上报结果处理到 Excel 的功能。运行本实例，如图 20.32 所示，单击第一个“选择”按钮，选择上报结果文件夹；单击第二个“选择”按钮，选择 Excel 文件；然后在“定时设置”栏中设置定时时间，单击“设置”按钮，启动 Timer 计时器，在 Timer 计时器中实时判断当前时间是否与设置的定时时间相同，如果相同，则循环遍历上报结果文件夹中的所有文件，并将这些文件的内容处理到 Excel 文件中。

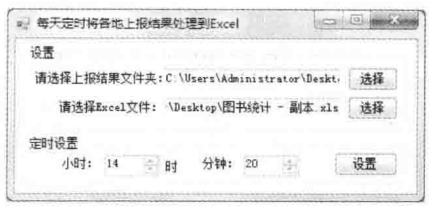


图 20.32　每天定时将各地上报结果处理到 Excel

关键技术

本实例实现每天定时将各地上报结果处理到 Excel 时，首先需要把用户的设置写入到 INI 文件中，当程序启动时，在 Timer 计时器中判断系统当前时间是否与用户设置的时间相同，如果相同，则将上报结果文件夹中的所有文件都处理到 Excel。在实现本实例的过程中，主要用到文件夹的遍历、INI 文件的读写以及将文本文件数据写入到 Excel 等技术，下面对本实例中用到的关键技术进行详细讲解。

（1）文件夹的遍历

在对文件夹进行遍历时，需要用到 Directory 类的 GetFiles 方法。Directory 类公开用于创建、移动和枚举通过文件夹和子文件夹的静态方法，其 GetFiles 方法用来返回指定文件夹中的文件名称，该方法常用的语法格式如下：

```
public static string[] GetFiles(string path)
```

参数说明

❶ path：将从其检索文件的文件夹。

❷ 返回值：指定文件夹中文件名的字符串数组。

（2）INI 文件的读写

C#中对 INI 文件进行读写时，分别需要用到 GetPrivateProfileString 和 WritePrivateProfileString 两个 API 函数，其中 GetPrivateProfileString 函数用于读取 INI 文件的内容。语法格式如下：

```
[DllImport("kernel32")]
```

```
private static extern int GetPrivateProfileString(string lpAppName,string lpKeyName,string lpDefault,StringBuilder lpReturnedString,int nSize,string lpFileName);
```

GetPrivateProfileString 函数中的参数说明如表 20.2 所示。

表 20.2　GetPrivateProfileString 函数中的参数说明

参　数	描　述	参　数	描　述
lpAppName	INI 文件内部根节点的值	lpReturnedString	要读取节点的值
lpKeyName	根节点下子标记的值	nSize	读取的节点内容的最大容量
lpDefault	当标记值未设定或不存在时的默认值	lpFileName	文件的全路径

WritePrivateProfileString 函数用于对 INI 文件进行写入和修改操作。语法格式如下：

```
[DllImport("kernel32")]
private static extern long WritePrivateProfileString(string mpAppName,string mpKeyName,string mpDefault,string mpFileName);
```

WritePrivateProfileString 函数中的参数说明如表 20.3 所示。

表 20.3　WritePrivateProfileString 函数中的参数说明

参　数	描　述	参　数	描　述
mpAppName	INI 文件内部根节点的值	mpDefault	想要修改的内容
mpKeyName	将要修改的标记名称	mpFileName	INI 文件的全路径

说明：（1）对系统 API 函数进行重写时，需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

（2）关于如何将文本文件数据写入到 Excel 的详细讲解请参见实例 456。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TimeReportResultToExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的上报结果文件夹和 Excel 文件路径；添加两个 NumericUpDown 控件，分别用来设置小时和分钟；添加 3 个 Button 控件，分别用来执行选择上报结果文件夹、选择 Excel 文件和定时设置操作；添加一个 Timer 组件，用来实时检测当前时间是否与设置的时间相同，如果相同，则将上报结果处理到 Excel。

（3）程序的主要代码如下。

单击“设置”按钮，把用户设置的信息写入到系统 INI 文件中，并启动计时器。“设置”按钮的 Click 事件代码如下：

```
private void btn_Set_Click(object sender, EventArgs e)
{
    WritePrivateProfileString("Set", "Floder", txt_Floder.Text, M_str_Name);                    //设置默认文件夹路径
    WritePrivateProfileString("Set", "Excel", txt_Path.Text, M_str_Name);                       //设置 Excel 文件路径
    WritePrivateProfileString("Set", "Hour", nudown_Hour.Value.ToString(), M_str_Name);         //设置小时
    WritePrivateProfileString("Set", "Min", nudown_Min.Value.ToString(), M_str_Name);           //设置分钟
    MessageBox.Show("配置文件设置成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    timer1.Start();                                                                             //启动计时器
}
```

上面的代码中用到了 WritePrivateProfileString 函数，该函数为系统 API 函数，主要用来对 INI 文件进行写入和修改操作。代码如下：

```
#region 修改 INI 文件中的内容
/// <summary>
///修改 INI 文件中的内容
/// </summary>
/// <param name="lpApplicationName">欲在其中写入的节点名称</param>
/// <param name="lpKeyName">欲设置的项名</param>
/// <param name="lpString">要写入的新字符串</param>
/// <param name="lpFileName">INI 文件名</param>
/// <returns>非零表示成功，零表示失败</returns>
[DllImport("kernel32")]
```

```
public static extern int WritePrivateProfileString(string lpApplicationName,string lpKeyName,string lpString,string lpFileName);
#endregion
```

在 timer1 组件的 Tick 事件中，首先判断当前时间是否与设置的时间相同，如果相同，则遍历上报文件夹中的所有文件，并调用 ReadDataToExcel 方法将遍历到的文件内容处理到 Excel 中。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (DateTime.Now.Hour == nudown_Hour.Value && DateTime.Now.Minute == nudown_Min.Value)
    {
        string P_str_Floder = txt_Floder.Text;                          //记录文件夹路径
        string P_str_Excel = txt_Path.Text;                             //记录 Excel 文件路径
        List<string> P_list_FileNames = new List<string>();             //创建泛型集合对象，用来存储所有上报的文件
        P_list_FileNames = GetAllFile(P_str_Floder);                    //获取所有指定类型的文件
        for (int j = 0; j < P_list_FileNames.Count; j++)                //遍历所有上报的文件
        {
            ReadDataToExcel(P_list_FileNames[j], P_str_Excel);          //将上报的文件数据处理到 Excel
        }
        CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
        MessageBox.Show("程序在" + DateTime.Now.ToShortTimeString() + "分时自动将各地上报的结果处理到了 Excel 中！", "提示",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

上面的代码中用到了 ReadDataToExcel 方法，该方法为用户自定义的方法，主要用来将文本文件的数据导入到 Excel 文件中。该方法中主要是通过文件流读取文本文件中的数据，并将读取到的数据填充到指定的 Excel 单元格中。ReadDataToExcel 方法的实现代码如下：

```
private void ReadDataToExcel(string P_str_File, string P_str_Excel)
{
    int P_int_Count = 0;                                                //记录正在读取的行数
    string P_str_Line;                                                  //记录读取行的内容
    List<string> P_str_List = new List<string>();                       //存储读取的所有内容
    StreamReader SReader = new StreamReader(P_str_File, Encoding.Default);//创建流读取对象
    while ((P_str_Line = SReader.ReadLine()) != null)                   //循环读取文本文件中的每一行
    {
        P_str_List.Add(P_str_Line);                                     //将读取到的行内容添加到泛型集合中
        P_int_Count++;                                                  //使当前读取行数加 1
    }
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    object missing = System.Reflection.Missing.Value;                   //获取缺少的 object 类型值
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(P_str_Excel, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //添加一个新的工作表
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    //以当前时间作为工作表名称
    newWorksheet.Name = DateTime.Now.ToString("yyyyMMddhhmmss")+DateTime.Now.Millisecond;
    excel.Application.DisplayAlerts = false;                            //不显示提示对话框
    for (int i = 0; i < P_str_List.Count; i++)                          //遍历泛型集合
    {
        newWorksheet.Cells[i + 1, 1] = P_str_List[i];                   //直接将遍历到的内容添加到工作表中
    }
    workbook.Save();                                                    //保存工作表
    workbook.Close(false, missing, missing);                            //关闭工作表
}
```

秘笈心法

心法领悟 462：使用 DateTime.Now 属性获取系统时间。

使用 DateTime.Now 属性可以方便地获取系统时间，而且可以使用格式化方式方便地得到系统时间的字符串表示形式。代码如下：

```
string P_str = DateTime.Now.ToString("yyyy 年 M 月 d 日 h 时 m 分 s 秒 fff 毫秒");
```

20.3　将 Excel 数据进行输出处理

实例 463	读取 Excel 指定表数据到文本文件 光盘位置：光盘\MR\20\463	中级 趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将 Excel 文件中的数据读取到文本文件的功能。运行本实例，如图 20.33 所示，单击“选择”按钮，选择要读取的 Excel 文件；然后在下拉列表中选择要读取的工作表，单击“导出到文本文件”按钮，将 Excel 文件中指定工作表的内容读取到文本文件中，如图 20.34 所示。

图 20.33　读取 Excel 指定表数据到文本文件

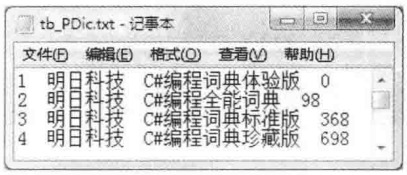


图 20.34　读取之后的文本文件

关键技术

本实例在将 Excel 中的指定表数据读取到文本文件中时，主要用到了 StreamWriter 类的 Write 方法。下面对本实例中用到的关键技术进行详细讲解。

StreamWriter 类实现了 TextWriter 抽象类，它主要使用一种特定的编码向流中写入字符，该类的 Write 方法用来以指定的编码方式向指定文件中写入内容。Write 方法为重载方法，其最常用的重载形式如下：

```
public override void Write(string value)
```

参数说明

value：要写入流的字符串。如果 value 为 null，则不写入任何内容。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExcelToTxt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的 Excel 文件；添加一个 ComboBox 控件，用来显示指定 Excel 文件中的所有工作表名称；添加两个 Button 控件，分别用来执行选择 Excel 文件操作和将 Excel 文件中指定表的数据导出到文本文件的操作。

（3）程序的主要代码如下：

```
private void btn_Txt_Click(object sender, EventArgs e)
{
    OleDbConnection olecon = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + txt_Path.Text + ";Extended
Properties=Excel 8.0");                                          //连接 Excel 数据库
    olecon.Open();                                               //打开数据库连接
    //从工作表中查询数据
    OleDbDataAdapter oledbda = new OleDbDataAdapter("select * from [" + cbox_SheetName.Text + "$]", olecon);
    DataSet myds = new DataSet();                                //创建数据集对象
    oledbda.Fill(myds);                                          //填充数据集
    //创建写入流对象
    StreamWriter SWriter = new StreamWriter(cbox_SheetName.Text + ".txt", false, Encoding.Default);
    string P_str_Content = "";                                   //存储读取的内容
    for (int i = 0; i < myds.Tables[0].Rows.Count; i++)          //遍历数据集中表的行数
    {
        for (int j = 0; j < myds.Tables[0].Columns.Count; j++)   //遍历数据集中表的列数
        {
```

```
                P_str_Content += myds.Tables[0].Rows[i][j].ToString() + "   ";      //记录当前遍历到的内容
            }
            P_str_Content += Environment.NewLine;                                  //字符串换行
        }
        SWriter.Write(P_str_Content);                                              //向文本文件中写入内容
        SWriter.Close();                                                           //关闭写入流对象
        SWriter.Dispose();                                                         //释放写入流所占用的资源
        MessageBox.Show("已经将" + cbox_SheetName.Text + "工作表中的数据成功写入到了文本文件中", "提示", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
    }
```

秘笈心法

心法领悟 463：方便地向文本文件中写入文本信息。

使用 StreamWriter 对象可以方便地向文本文件中写入信息，首先创建 StreamWriter 对象，然后使用 StreamWriter 对象的 Write 方法写入文本信息，写入文本信息后不要忘记调用该对象的 Close 方法释放资源。

实例 464　将 Excel 数据分解到多个文本文件

光盘位置：光盘\MR\20\464　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将 Excel 文件中的数据分解到多个文本文件中的功能。运行本实例，如图 20.35 所示，单击“选择”按钮，选择要分解的 Excel 文件；然后在下拉列表中分别选择要分解的工作表和要分解的文本文件个数，单击“导出到文本文件”按钮，将 Excel 文件中指定工作表的内容分解到指定个数的文本文件中，如图 20.36 所示。

图 20.35　将 Excel 数据分解到多个文本文件

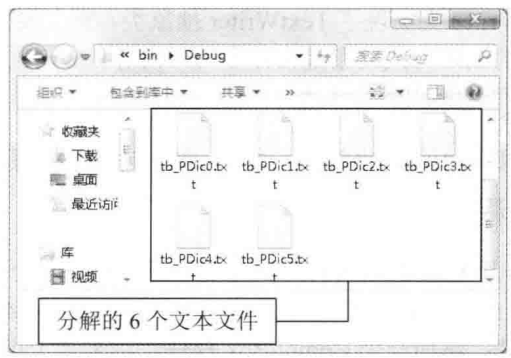


图 20.36　分解之后的文本文件

关键技术

本实例在将 Excel 中的数据分解到多个文本文件时，首先需要循环读取出 Excel 文件中的内容，然后根据要分解的文本文件个数，使用 StreamWriter 类的 Write 方法向各个文件中写入不同的内容即可。

说明：关于 StreamWriter 类的 Write 方法的详细讲解请参见实例 463 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExcelToMultiTxt。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示选择的 Excel 文件；添加两个 ComboBox 控件，分别用来显示指定 Excel 文件中的所有工作表名称和要分解为的文本文件个数；添加两个 Button 控件，分别用来执行选择 Excel 文件操作和将 Excel 文件中指定表的数据分解到多

个文本文件的操作。

（3）程序的主要代码如下：

```
//将 Excel 数据分解到多个文本文件中
private void WriteContent()
{
    int P_int_Counts = CBoxShowCount().Tables[0].Rows.Count;                    //获取记录总数
    int P_int_Page = Convert.ToInt32(cbox_Count.Text);                          //记录要分解为几个文件
    int P_int_PageRow = Convert.ToInt32(P_int_Counts / P_int_Page);             //记录每个文件的记录数
    for (int i = 0; i < P_int_Page; i++)                                        //循环访问每个文件
    {
        //创建写入流对象
        using (StreamWriter SWriter = new StreamWriter(cbox_SheetName.Text + i + ".txt", false, Encoding.Default))
        {
            string P_str_Content = "";                                          //存储读取的内容
            for (int r = i * P_int_PageRow; r < (i + 1) * P_int_PageRow; r++)   //遍历数据集中表的行数
            {
                if (r < P_int_Counts)                                           //判断遍历到的行数是否小于总行数
                {
                    //遍历数据集中表的列数
                    for (int c = 0; c < CBoxShowCount().Tables[0].Columns.Count; c++)
                    {
                        //记录当前遍历到的内容
                        P_str_Content += CBoxShowCount().Tables[0].Rows[r][c].ToString() + "   ";
                    }
                    P_str_Content += Environment.NewLine;                       //字符串换行
                }
            }
            SWriter.Write(P_str_Content);                                       //向文本文件中写入内容
            SWriter.Close();                                                    //关闭写入流对象
        }
    }
}
```

秘笈心法

心法领悟 464：轻松回收 StreamWriter 对象资源。

程序设计过程中，可以手动调用 StreamWriter 对象的 Close 方法回收文件资源，但是使用 using 关键字则更为简洁友好。当 using 关键字下面的代码执行完毕后，会自动将 using 小括号中创建的 StreamWriter 对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 465　读取多个 Excel 文件数据到 Word 文档　中级

光盘位置：光盘\MR\20\465　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将多个 Excel 文件中的数据读取到同一 Word 文档中的功能。运行本实例，如图 20.37 所示，单击第一个“选择”按钮，选择要读取的多个 Excel 文件；单击第二个“选择”按钮，选择 Word 文档的存放路径；然后单击“读取”按钮，将多个 Excel 文件中的数据读取到同一 Word 文档中；单击“查看”按钮，查看读入数据之后的 Word 文档，读入数据之后的 Word 文档如图 20.38 所示。

关键技术

本实例在实现将多个 Excel 文件中的数据读取到 Word 文档时，首先需要使用 Microsoft Word 自动化对象模型中的 Documents 对象的 Add 方法创建一个新的 Word 文档；然后遍历所有的 Excel 文件，并根据遍历到的 Excel 工作表中的数据在 Word 文档中创建指定行数和列数的表格；最后使用 Cell 对象的 InsertAfter 方法向 Word 表格中插入数据，并调用 Document 对象的 SaveAs 方法保存 Word 文档。下面对本实例中用到的关键技术进行详细讲解。

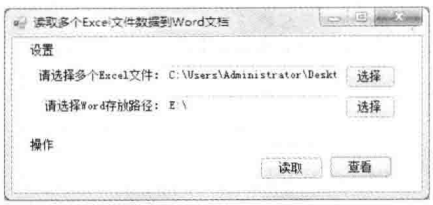


图 20.37　读取多个 Excel 文件数据到 Word 文档

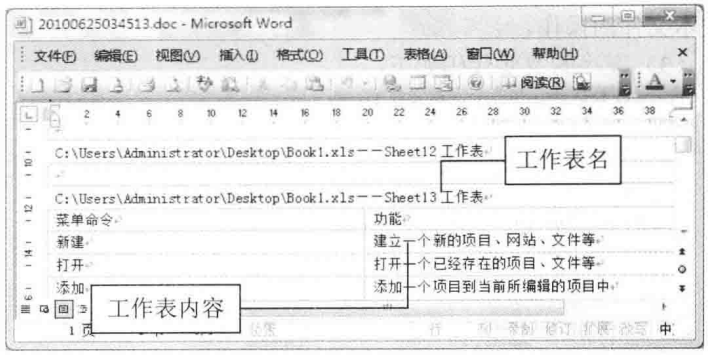


图 20.38　读取之后的 Word 文档

（1）Documents 对象的 Add 方法

Documents 对象表示 Word 文档集合，其 Add 方法用来建立新的 Word 文档，该方法的语法格式如下：

```
Document Add([In, Optional] ref object Template,[In, Optional] ref object NewTemplate,[In, Optional] ref object DocumentType,[In, Optional] ref object Visible);
```

Documents 对象的 Add 方法中的参数说明如表 20.4 所示。

表 20.4　Documents 对象的 Add 方法中的参数说明

参　数	描　述
Template	模板名称
NewTemplate	指定是否需要将文档以模板形式打开
DocumentType	指定 Word 文档类型
Visible	指定是否在可见窗口中打开 Word 文档

（2）在 Word 文档中创建表格

在 Word 文档中创建表格时，需要使用 Tables 对象的 Add 方法。Tables 对象表示 Word 表格集合，其 Add 方法用来在 Word 文档中创建一个新的表格，该方法的语法格式如下：

```
Table Add([In] Range Range,[In] int NumRows,[In] int NumColumns,[In, Optional] ref object DefaultTableBehavior,[In, Optional] ref object AutoFitBehavior);
```

Tables 对象的 Add 方法中的参数说明如表 20.5 所示。

表 20.5　Tables 对象的 Add 方法中的参数说明

参　数	描　述
Range	要创建表格的区域
NumRows	指定表格的行数
NumColumns	指定表格的列数
DefaultTableBehavior	默认 Word 表格自适应方式
AutoFitBehavior	Word 表格的自适应方式

（3）Cell 对象的 InsertAfter 方法

Cell 对象的 InsertAfter 方法用来将指定文本插入某区域或选定内容的后面。语法格式如下：

```
public void InsertAfter(string Text)
```

参数说明

Text：要插入的文本。

（4）Document 对象的 SaveAs 方法

Document 对象的 SaveAs 方法用来使用新名称或新格式保存 Word 文档。语法格式如下：

```
public virtual void SaveAs(ref Object FileName,ref Object FileFormat,ref Object LockComments,ref Object Password,ref Object AddToRecentFiles,ref Object WritePassword,ref Object ReadOnlyRecommended,ref Object EmbedTrueTypeFonts,ref Object SaveNativePictureFormat,ref Object SaveFormsData,ref Object SaveAsAOCELetter,ref Object Encoding,ref Object InsertLineBreaks,ref Object AllowSubstitutions,ref Object LineEnding,ref Object AddBiDiMarks)
```

Document 对象的 SaveAs 方法中的参数说明如表 20.6 所示。

表 20.6　Document 对象的 SaveAs 方法中的参数说明

参　数	描　述
FileName	文档的名称。默认值是当前文件夹名和文件名。如果文档在以前没有保存过，则使用默认名称（如 Doc1.doc）。如果已经存在具有指定文件名的文档，则会在不先提示用户的情况下覆盖文档
FileFormat	文档的保存格式
LockComments	如果为 true，则锁定文档以进行注释。默认值为 false
Password	用来打开文档的密码字符串
AddToRecentFiles	如果为 true，则将该文档添加到“文件”菜单上最近使用的文件列表中。默认值为 true
WritePassword	用来保存对文件所做更改的密码字符串
ReadOnlyRecommended	如果为 true，则让 Microsoft Office Word 在打开文档时建议设置为只读状态。默认值为 false
EmbedTrueTypeFonts	如果为 true，则将 TrueType 字体随文档一起保存
SaveNativePictureFormat	如果图形是从另一个平台导入的，则 true 表示仅保存导入图形的 Windows 版本
SaveFormsData	如果为 true，则将用户在窗体中输入的数据另存为数据记录
SaveAsAOCELetter	如果文档附加了邮件程序，则 true 表示会将文档另存为 AOCE 信函（邮件程序会进行保存）
Encoding	要用于另存为编码文本文件的文档的代码页或字符集。默认值为系统代码页
InsertLineBreaks	如果文档另存为文本文件，则 true 表示在每行文本末尾插入分行符
AllowSubstitutions	如果文档另存为文本文件，则 true 允许 Word 将某些符号替换为外观与之类似的文本
LineEnding	Word 在另存为文本文件的文档中标记分行符和换段符
AddBiDiMarks	如果为 true，则向输出文件添加控制字符，以便保留原始文档中文本的双向布局

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiExcelToWord。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的多个 Excel 文件路径和 Word 文档路径；添加 4 个 Button 控件，分别用来执行选择多个 Excel 文件、选择 Word 文档文件、将多个 Excel 文件中的数据导出到 Word 文档和查看 Word 文档内容操作。

（3）程序的主要代码如下：

```
private void btn_Read_Click(object sender, EventArgs e)
{
    object missing = System.Reflection.Missing.Value;                          //获取缺少的 object 类型值
    string[] P_str_Names = txt_Excel.Text.Split(',');                          //存储所有选择的 Excel 文件名
     object P_obj_Name;                                                        //存储遍历到的 Excel 文件名
     //创建 Word 对象
    Microsoft.Office.Interop.Word.Application word = new Microsoft.Office.Interop.Word.Application();
    if (txt_Word.Text.EndsWith("\\"))                                          //判断路径是否以\结尾
        P_obj_WordName = txt_Word.Text + DateTime.Now.ToString("yyyyMMddhhmmss") + ".doc";
    else
        //记录 Word 文件路径及名称
        P_obj_WordName = txt_Word.Text + "\\" + DateTime.Now.ToString("yyyyMMddhhmmss") + ".doc";
    Microsoft.Office.Interop.Word.Table table;                                 //声明 Word 表格对象
    //声明 Word 文档对象
    Microsoft.Office.Interop.Word.Document document = new Microsoft.Office.Interop.Word.Document();
    document = word.Documents.Add(ref missing, ref missing, ref missing, ref missing);   //新建 Word 文档
    Microsoft.Office.Interop.Word.Range range ;                                //声明范围对象
    int P_int_Rows = 0, P_int_Columns = 0;                                     //存储工作表中数据的行数和列数
    object P_obj_start = 0, P_obj_end = 0;                                     //分别记录创建表格的开始和结束范围
    //定义要合并的范围位置
    object P_obj_Range = Microsoft.Office.Interop.Word.WdCollapseDirection.wdCollapseEnd;
    for (int i = 0; i < P_str_Names.Length - 1; i++)                           //遍历所有选择的 Excel 文件名
    {
```

```
            P_obj_Name = P_str_Names[i];                                    //记录遍历到的 Excel 文件名
            //获取指定 Excel 中的所有工作表
            List<string> P_list_SheetName = CBoxBind(P_obj_Name.ToString());
            for (int j = 0; j < P_list_SheetName.Count; j++)                //遍历所有工作表
            {
                range = document.Range(ref missing, ref missing);           //获取 Word 范围
                range.InsertAfter(P_obj_Name + "——" + P_list_SheetName[j] + "工作表");  //插入文本
                range.Font.Name = "宋体";                                   //设置字体
                range.Font.Size = 10;                                       //设置字体大小
                //获取工作表中的所有数据
                DataSet myds = CBoxShowCount(P_obj_Name.ToString(), P_list_SheetName[j]);
                P_int_Rows = myds.Tables[0].Rows.Count;                     //记录工作表的行数
                P_int_Columns = myds.Tables[0].Columns.Count;               //记录工作表的列数
                range.Collapse(ref P_obj_Range);                            //合并范围
                if (P_int_Rows > 0 && P_int_Columns > 0)                    //判断如果工作表中有记录
                {
                    //在指定范围处添加一个指定行数和列数的表格
                    table = range.Tables.Add(range, P_int_Rows, P_int_Columns, ref missing, ref missing);
                    for (int r = 0; r < P_int_Rows; r++)                    //遍历行
                    {
                        for (int c = 0; c < P_int_Columns; c++)             //遍历列
                        {
                            //将遍历到的数据添加到 Word 表格中
                            table.Cell(r + 1, c + 1).Range.InsertAfter(myds.Tables[0].Rows[r][c].ToString());
                        }
                    }
                }
                //定义 Word 文档的保存格式
                object P_obj_Format = Microsoft.Office.Interop.Word.WdSaveFormat.wdFormatDocument;
                //设置保存时不显示对话框
                word.DisplayAlerts = Microsoft.Office.Interop.Word.WdAlertLevel.wdAlertsNone;
                //保存 Word 文档
                document.SaveAs(ref P_obj_WordName, ref P_obj_Format, ref missing, ref missing, ref missing, ref missing, ref missing, ref
missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing, ref missing);
            }
        }
        document.Close(ref missing, ref missing, ref missing);             //关闭 Word 文档
        word.Quit(ref missing, ref missing, ref missing);                   //退出 Word 应用程序
        MessageBox.Show("已经成功将多个 Excel 文件的内容读取到了一个 Word 文档中！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
```

秘笈心法

心法领悟 465：如何理解折叠 Range 的概念？

向 Word 文档中添加 Windows 控件或文本内容时，经常会使用 Collapse 方法折叠 Range 并向 Range 中继续添加信息，所谓的折叠 Range 是指起始位置和结束位置相同的 Range 对象。

实例 466　读取 Excel 查询结果到 Access 数据库

光盘位置：光盘\MR\20\466　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将 Excel 文件的查询结果读取到 Access 数据库中的功能。运行本实例，如图 20.39 所示，单击第一个“选择”按钮，选择要读取的 Excel 文件，并在下拉列表中选择要读取的工作表；单击第二个“选择”按钮，选择 Access 数据库文件；然后单击“导出”按钮，将 Excel 中的指定工作表数据读取到 Access 数据库文件中；单击“查看”按钮，查看读

图 20.39　读取 Excel 查询结果到 Access 数据库

入数据之后的 Access 数据库文件。

关键技术

本实例在将 Excel 查询结果读取到 Access 数据库时，主要用到了 Microsoft Access 自动化对象模型的 DoCmd 对象的 TransferSpreadsheet 方法，下面对其进行详细讲解。

使用 DoCmd 对象可以对 Access 数据库文件执行诸如打开、关闭和设置控件值等任务，其 TransferSpreadsheet 方法用来将 Excel 工作表中的内容读取到 Access 数据表中，该方法的语法格式如下：

```
TransferSpreadsheet(object TransferType, object SpreadsheetType, object TableName, object FileName, object HasFieldNames, object Range, object UseOA)
```

TransferSpreadsheet 方法中的参数说明如表 20.7 所示。

表 20.7　TransferSpreadsheet 方法中的参数说明

参　数	描　述
TransferType	指定要执行的数据转换操作
SpreadsheetType	指定要操作的 Excel 工作表类型
TableName	要向其中导入 Excel 数据、从中导出 Excel 数据或链接 Excel 数据的 Access 数据表的名称
FileName	要从其中导入、导出到或链接到的 Excel 的名称与路径
HasFieldNames	指定是否将 Excel 中的第一行用作字段名
Range	Excel 中单元格的有效范围或范围名称
UseOA	默认值，一般取值为 Missing.Value

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExcelToAccess。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的 Excel 文件路径和 Access 文件路径；添加一个 ComboBox 控件，用来显示指定 Excel 文件中的所有工作表名称；添加 4 个 Button 控件，分别用来执行选择 Excel 文件、选择 Access 文件、将 Excel 指定表中的数据导出到 Access 数据库和查看 Access 数据库文件内容操作。

（3）程序的主要代码如下：

```
private void btn_Export_Click(object sender, EventArgs e)
{
    object missing = System.Reflection.Missing.Value;                          //声明 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(txt_Path.Text, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing);
    Microsoft.Office.Interop.Excel.Worksheet worksheet;                         //声明工作表
    //创建 Access 对象
    Microsoft.Office.Interop.Access.Application access = new Microsoft.Office.Interop.Access.Application();
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[cbox_SheetName.Text]);
    worksheet.Move(workbook.Sheets[1], missing);                               //将选择的工作表作为第一个工作表
    object P_obj_Name = (object)worksheet.Name;                                //获取工作表名称
    excel.DisplayAlerts = false;                                               //设置 Excel 保存时不显示对话框
    workbook.Save();                                                           //保存工作簿
    CloseProcess("EXCEL");                                                     //关闭所有 Excel 进程
    object P_obj_Excel = (object)txt_Path.Text;                                //记录 Excel 文件路径
    try
    {
        access.OpenCurrentDatabase(txt_Access.Text, true, "");                 //打开 Access 数据库
        //将 Excel 指定工作表中的数据导入到 Access 中
        access.DoCmd.TransferSpreadsheet(Microsoft.Office.Interop.Access.AcDataTransferType.acImport,
```

```
Microsoft.Office.Interop.Access.AcSpreadSheetType.acSpreadsheetTypeExcel97, P_obj_Name, P_obj_Excel, true, missing, missing);
            access.Quit(Microsoft.Office.Interop.Access.AcQuitOption.acQuitSaveAll);      //关闭并保存 Access 数据库文件
            CloseProcess("MSACCESS");                                                     //关闭所有 Access 数据库进程
            MessageBox.Show("已经将 Excel 的" + cbox_SheetName.Text + "工作表中的数据导入到 Access 数据库中!", "提示", MessageBoxButtons.
OK, MessageBoxIcon.Information);
        }
        catch
        {
            MessageBox.Show("Access 数据库中已经存在该表！", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }
```

说明：（1）本实例在使用 Access 对象时，首先需要添加该对象的引用，具体步骤是，选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令，弹出“引用管理器”对话框，如图 20.40 所示。在该对话框中选中要引用的 Access 动态链接库，这里选择 Microsoft.Office.Interop.Access，然后单击“确定”按钮，即可将 Access 对象引用添加到程序中。

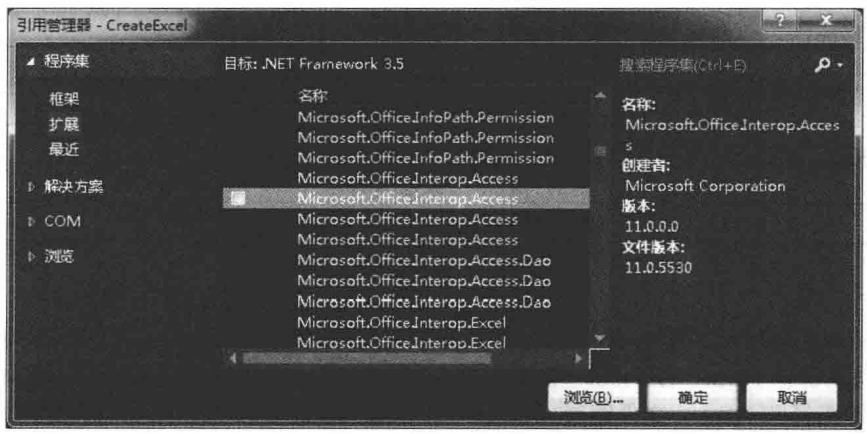


图 20.40 “引用管理器”对话框

（2）本章后面的实例中，如果需要对 Access 数据库进行操作，则都需要按（1）所讲的步骤添加 Access 对象引用，由于添加的步骤是一样的，所以在后面涉及时将不再赘述。

秘笈心法

心法领悟 466：有效使用 Access 对象。

程序中遇到对 Access 数据库的相关操作时，可以使用 Access 对象简化操作。例如，本实例使用 Access 对象模型的 DoCmd 对象的 TransferSpreadsheet 方法将 Excel 工作表中的内容读取到了 Access 数据表中。代码如下：

```
access.DoCmd.TransferSpreadsheet(Microsoft.Office.Interop.Access.AcDataTransferType.acImport,
Microsoft.Office.Interop.Access.AcSpreadSheetType.acSpreadsheetTypeExcel97, P_obj_Name, P_obj_Excel, true, missing, missing);
```

实例 467 将多个 Excel 文件输出到同一个 SQL Server 数据库

光盘位置：光盘\MR\20\467　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将多个 Excel 文件中的数据输出到同一个 SQL Server 数据库中的功能。运行本实例，如图 20.41 所示，单击“选择”按钮，选择要读取的多个 Excel 文件；然后在“SQL Server 数据库连接设置”栏中对数据库连接信息进行设置；最后单击“导出”按钮，将多个 Excel 中的数据输出到同一 SQL Server 数据库中。

图 20.41　将多个 Excel 文件输出到同一个 SQL Server 数据库

关键技术

本实例在将多个 Excel 文件中的数据输出到同一个 SQL Server 数据表时，主要用到了 SqlBulkCopy 类的 WriteToServer 方法，下面对其进行详细讲解。

SqlBulkCopy 类使用户可以用其他源的数据有效批量加载到 SQL Server 数据表中，其 WriteToServer 方法用来将所有行从数据源复制到 SqlBulkCopy 对象的 DestinationTableName 属性（该属性用来指定服务器上目标表的名称）指定的目标表中，该方法有 4 种重载形式，本实例中用到的该方法的重载形式语法如下：

```
public void WriteToServer(DataTable table)
```

参数说明

table：一个 DataTable 对象，它的行将被复制到目标表中。

说明：（1）Microsoft SQL Server 提供了一个称为 bcp 的流行的命令提示符实用工具，用于将数据从一个表移动到另一个表（表既可以在同一个服务器上，也可以在不同服务器上）。SqlBulkCopy 类允许编写提供类似功能的托管代码解决方案。

（2）使用 SqlBulkCopy 类只能向 SQL Server 表写入数据，但是，数据源不限于 SQL Server；可以使用任何数据源，只要数据可以加载到 DataTable 实例或者可以使用 IDataReader 对象读取数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiExcelToSql。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 TextBox 控件，分别用来显示选择的多个 Excel 文件路径、输入 SQL Server 服务器名、登录用户名和登录密码；添加两个 CheckBox 控件，用来选择 SQL Server 数据库的登录方式；添加一个 ComboBox 控件，用来选择要导入到的 SQL Server 数据库；添加 3 个 Button 控件，分别用来执行选择多个 Excel 文件、获取指定 SQL Server 服务器上的所有数据库和将多个 Excel 文件内容导入到同一个 SQL Server 数据库的操作。

（3）程序的主要代码如下。

选择完多个 Excel 文件，并对 SQL Server 服务器进行设置后，单击“导出”按钮，调用自定义的 ImportDataToSql 方法循环将多个 Excel 文件中的内容导出到同一个 SQL Server 数据库中。“导出”按钮的 Click 事件代码如下：

```
private void btn_Export_Click(object sender, EventArgs e)
{
    string[] P_str_Names = txt_Path.Text.Split(',');                    //存储所有选择的 Excel 文件名
    string P_str_Name = "";                                             //存储遍历到的 Excel 文件名
    List<string> P_list_SheetNames = new List<string>();                //创建泛型集合对象，用来存储工作表名称
    for (int i = 0; i < P_str_Names.Length - 1; i++)                    //遍历所有选择的 Excel 文件名
    {
        P_str_Name = P_str_Names[i];                                    //记录遍历到的 Excel 文件名
        P_list_SheetNames = GetSheetName(P_str_Name);                   //获取 Excel 文件中的所有工作表名
        for (int j = 0; j < P_list_SheetNames.Count; j++)               //遍历所有工作表
```

```
        {
            if (ckbox_Windows.Checked)                                          //如果用 Windows 身份验证登录 SQL Server
                ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Initial Catalog =" + cbox_Server.Text +
";Integrated Security=SSPI;");                                                  //将工作表内容导出到 SQL Server
            else if (ckbox_SQL.Checked)                                         //如果用 SQL Server 身份验证登录 SQL Server
                //将工作表内容导出到 SQL Server
                ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Database=" + cbox_Server.Text +
";Uid=" + txt_Name.Text + ";Pwd=" + txt_Pwd.Text + ";");
        }
    }
    MessageBox.Show("已经将所有选择的 Excel 工作表导入到了 SQL Server 数据库中!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

上面的代码中用到了 ImportDataToSql 方法，该方法主要用来将 Excel 中指定工作表的内容导出到 SQL Server 数据库中。该方法中主要使用 SqlBulkCopy 对象的 WriteToServer 方法对 SQL Server 数据库进行数据写入操作。ImportDataToSql 方法的实现代码如下：

```
public void ImportDataToSql(string P_str_Excel, string P_str_SheetName, string P_str_SqlCon)
{
    DataSet myds = new DataSet();                                               //创建数据集对象
    try
    {
        //获得全部数据
        string P_str_OledbCon = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + P_str_Excel + ";Extended Properties=Excel 8.0;";
        OleDbConnection oledbcon = new OleDbConnection(P_str_OledbCon);         //创建 Oledb 数据库连接对象
        string P_str_ExcelSql = "";                                             //记录要执行的 Excel 查询语句
        OleDbDataAdapter oledbda = null;                                        //创建 Oledb 数据桥接器对象
        P_str_ExcelSql = string.Format("select * from [{0}$]", P_str_SheetName);//记录要执行的 Excel 查询语句
        oledbda = new OleDbDataAdapter(P_str_ExcelSql, P_str_OledbCon);         //使用数据桥接器执行 Excel 查询
        oledbda.Fill(myds, P_str_SheetName);                                    //填充数据
        //定义变量，用来记录创建表的 SQL 语句
        string P_str_CreateSql = string.Format("create table {0}(", P_str_SheetName);
        foreach (DataColumn c in myds.Tables[0].Columns)                        //遍历数据集中的所有行
        {
            P_str_CreateSql += string.Format("[{0}] text,", c.ColumnName);      //在表中创建字段
        }
        P_str_CreateSql = P_str_CreateSql + ")";                                //完善创建表的 SQL 语句
        using (SqlConnection sqlcon = new SqlConnection(P_str_SqlCon))          //创建 SQL 数据库连接对象
        {
            sqlcon.Open();                                                      //打开数据库连接
            SqlCommand sqlcmd = sqlcon.CreateCommand();                         //创建执行命令对象
            sqlcmd.CommandText = P_str_CreateSql;                               //指定要执行的 SQL 语句
            sqlcmd.ExecuteNonQuery();                                           //执行操作
            sqlcon.Close();                                                     //关闭数据连接
        }
        using (SqlBulkCopy bcp = new SqlBulkCopy(P_str_SqlCon))                 //用 bcp 导入数据
        {
            bcp.BatchSize = 100;                                                //每次传输的行数
            bcp.DestinationTableName = P_str_SheetName;                         //定义目标表
            bcp.WriteToServer(myds.Tables[0]);                                  //将数据写入 SQL Server 数据表
        }
    }
    catch
    {
        MessageBox.Show("SQL Server 数据库中已经存在" + P_str_SheetName + "表!", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

秘笈心法

心法领悟 467：程序中尽量不要使用太多嵌套 for 语句。

由于嵌套 for 语句将消耗很大的资源，所以在实际开发项目时，能不使用嵌套 for 语句尽量不要使用。

实例 468　实时读取 Excel 数据到 SQL Server 数据库

光盘位置：光盘\MR\20\468　　中级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了实时将 Excel 文件中的数据读取到 SQL Server 数据库中的功能。运行本实例，如图 20.42 所示，单击“选择”按钮，选择要读取的 Excel 文件；然后在“SQL Server 数据库连接设置”栏中对数据库连接信息进行设置；最后根据用户的意愿确定是否选中“是否自动执行”复选框，单击“设置”按钮，启动 Timer 计时器，在 Timer 计时器中实时将 Excel 中的数据读取到 SQL Server 数据库中。

图 20.42　实时读取 Excel 数据到 SQL Server 数据库

关键技术

本实例在实现时，首先需要将用户配置信息写入到 INI 文件中，然后在程序启动时，判断程序是否自动运行，如果是，则在 Timer 计时器中实时将 Excel 中的数据读取到 SQL Server 数据库中，在读取过程中，主要使用 SqlBulkCopy 类的 WriteToServer 方法来进行实现。

说明：（1）关于 INI 文件读写操作的详细讲解请参见实例 462 中的关键技术。

（2）关于 Timer 计时器的详细讲解请参见实例 461 中的关键技术。

（3）关于 SqlBulkCopy 类的 WriteToServer 方法的详细讲解请参见实例 467 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RealTimeExcelToSql。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 TextBox 控件，分别用来显示选择的 Excel 文件路径、输入 SQL Server 服务器名、登录用户名和登录密码；添加 3 个 CheckBox 控件，分别用来选择 SQL Server 数据库的登录方式和设置程序是否自动执行；添加一个 ComboBox 控件，用来选择要导入到的 SQL Server 数据库；添加 3 个 Button 控件，分别用来执行选择 Excel 文件、获取指定 SQL Server 服务器上的所有数据库和将设置写入到系统配置文件的操作；添加一个 Timer 组件，用来实时将 Excel 文件内容写入到 SQL Server 数据库中。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    string P_str_Name = "";                                          //存储遍历到的 Excel 文件名
    List<string> P_list_SheetNames = new List<string>();             //创建泛型集合对象，用来存储工作表名称
    P_str_Name = txt_Path.Text;                                      //记录遍历到的 Excel 文件名
    P_list_SheetNames = GetSheetName(P_str_Name);                    //获取 Excel 文件中的所有工作表名
    for (int j = 0; j < P_list_SheetNames.Count; j++)                //遍历所有工作表
    {
        if (ckbox_Windows.Checked)                                   //如果用 Windows 身份验证登录 SQL Server
            ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Initial Catalog =" + cbox_Server.Text +
";Integrated Security=SSPI;");                                       //将工作表内容导出到 SQL Server
        else if (ckbox_SQL.Checked)                                  //如果用 SQL Server 身份验证登录 SQL Server
            ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Database=" + cbox_Server.Text + ";Uid=" +
txt_Name.Text + ";Pwd=" + txt_Pwd.Text + ";");                       //将工作表内容导出到 SQL Server
    }
    System.Threading.Thread.Sleep(60000);                            //使线程休眠 1 分钟
}
```

说明：上面的代码中用到了 ImportDataToSql 方法，该方法主要用来将 Excel 工作表中的数据导入到 SQL Server 数据库中，关于该方法的详细代码请参见实例 467 中的设计过程。

秘笈心法

心法领悟 468：Timer 计时器组件的使用。

Timer 组件是一个计时器组件，它主要用来按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次。

实例 469　批量读取 Excel 文件给指定数据库

高级

光盘位置：光盘\MR\20\469

趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了批量读取 Excel 文件数据给指定数据库的功能。运行本实例，如图 20.43 所示，单击第一个“选择”按钮，选择要读取的多个 Excel 文件；如果选中“Access 数据库设置”单选按钮，则该区域的所有控件可用，这时选择 Access 数据库文件的路径；如果选中“SQL Server 数据库连接设置”单选按钮，则该区域的所有控件可用，在该区域中对 SQL Server 数据库连接信息进行设置；最后单击“导出”按钮，批量将选择的所有 Excel 文件的数据读取到指定的数据库中。

图 20.43　批量读取 Excel 文件给指定数据库

关键技术

本实例主要实现将 Excel 数据批量读取到 Access 和 SQL Server 两种不同的数据库中，实现该功能时，首先需要循环遍历每个 Excel 文件，然后根据用户的设置，把遍历到的 Excel 文件中的每个工作表中的数据读取到指定的数据库中，这其中需要用到的技术主要有 Microsoft Access 自动化对象模型的 DoCmd 对象的 TransferSpreadsheet 方法和 SqlBulkCopy 类的 WriteToServer 方法等。

说明：（1）关于 DoCmd 对象的 TransferSpreadsheet 方法的详细讲解请参见实例 466 中的关键技术。

（2）关于 SqlBulkCopy 类的 WriteToServer 方法的详细讲解请参见实例 467 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BatchExcelToDatabase。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 5 个 TextBox 控件，分别用来显示选择的多个 Excel 文件路径、显示选择的 Access 文件路径、输入 SQL Server 服务器名、登录用户名和登录密码；添加两个 RadioButton 控件，用来选择要将 Excel 文件内容导入到何种数据库中；添加两个 CheckBox 控件，用

来选择 SQL Server 数据库的登录方式；添加一个 ComboBox 控件，用来选择要导入到的 SQL Server 数据库；添加 4 个 Button 控件，分别用来执行选择多个 Excel 文件、选择 Access 文件、获取指定 SQL Server 服务器上的所有数据库和将多个 Excel 文件内容导入到指定数据库的操作。

（3）程序的主要代码如下。

对导出信息设置完成后，单击“导出”按钮，首先使用 List 泛型获取 Excel 文件中的所有工作表名称，然后遍历每个工作表，并根据用户选择的操作方式，调用不同的自定义方法批量将 Excel 文件中的数据导入到指定数据库中。“导出”按钮的 Click 事件代码如下：

```
private void btn_Export_Click(object sender, EventArgs e)
{
    string[] P_str_Names = txt_Path.Text.Split(',');                          //存储所有选择的 Excel 文件名
    string P_str_Name = "";                                                   //存储遍历到的 Excel 文件名
    List<string> P_list_SheetNames = new List<string>();                      //创建泛型集合对象，用来存储工作表名称
    for (int i = 0; i < P_str_Names.Length - 1; i++)                          //遍历所有选择的 Excel 文件名
    {
        P_str_Name = P_str_Names[i];                                          //记录遍历到的 Excel 文件名
        P_list_SheetNames = GetSheetName(P_str_Name);                         //获取 Excel 文件中的所有工作表名
        for (int j = 0; j < P_list_SheetNames.Count; j++)                     //遍历所有工作表
        {
            if (rbtn_Access.Checked)                                          //判断 Access 数据库连接设置单选按钮选中
            {
                ImportDataToAccess(P_str_Name, P_list_SheetNames[j]);         //将工作表内容导出到 Access
            }
            else if (rbtn_Sql.Checked)                                        //判断 SQL Server 数据库连接设置单选按钮选中
            {
                if (ckbox_Windows.Checked)                                    //如果用 Windows 身份验证登录 SQL Server
                    ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Initial Catalog =" +
cbox_Server.Text + ";Integrated Security=SSPI;");                             //将工作表内容导出到 SQL Server
                else if (ckbox_SQL.Checked)                                   //如果用 SQL Server 身份验证登录 SQL Server
                    //将工作表内容导出到 SQL Server
                    ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Database=" + cbox_Server.Text +
";Uid=" + txt_Name.Text + ";Pwd=" + txt_Pwd.Text + ";");
            }
        }
    }
    MessageBox.Show("已经将所有选择的 Excel 工作表导入到了指定的数据库中！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

上面的代码中用到了 GetSheetName、ImportDataToAccess 和 ImportDataToSql 方法，下面分别对它们进行详细讲解。GetSheetName 方法为自定义的、返回值类型为 List 泛型的一个方法，主要用来获取指定 Excel 文件中的所有工作表名称，将获取到的工作表名称存储到一个 List 泛型集合中。GetSheetName 方法的实现代码如下：

```
private List<string> GetSheetName(string P_str_Excel)                        //获取所有工作表名称
{
    List<string> P_list_SheetName = new List<string>();                      //创建泛型集合对象
    OleDbConnection olecon = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + P_str_Excel + ";Extended Properties=
Excel 8.0");                                                                  //连接 Excel 数据库
    olecon.Open();                                                            //打开数据库连接
    System.Data.DataTable DTable = olecon.GetSchema("Tables");                //创建表对象
    DataTableReader DTReader = new DataTableReader(DTable);                   //创建表读取对象
    while (DTReader.Read())                                                   //循环读取
    {
        string P_str_Name = DTReader["Table_Name"].ToString().Replace('$', ' ').Trim();   //记录工作表名称
        if (!P_list_SheetName.Contains(P_str_Name))                           //判断泛型集合中是否已经存在该工作表名称
            P_list_SheetName.Add(P_str_Name);                                 //将工作表名添加到泛型集合中
    }
    DTable = null;                                                            //清空表对象
    DTReader = null;                                                          //清空表读取对象
    olecon.Close();                                                           //关闭数据库连接
    return P_list_SheetName;                                                  //返回得到的泛型集合
}
```

ImportDataToAccess 方法主要用来将 Excel 工作表中的数据导入到指定的 Access 数据库中，该方法中在对 Access 数据库执行数据导入操作时，主要使用 Access 对象的 DoCmd 属性的 TransferSpreadsheet 方法来实现。

ImportDataToAccess 方法的实现代码如下：

```
private void ImportDataToAccess(string P_str_Excel, string P_str_SheetName)
{
    object missing = System.Reflection.Missing.Value;                          //声明 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                             //声明工作表
    //创建 Access 对象
    Microsoft.Office.Interop.Access.Application access = new Microsoft.Office.Interop.Access.Application();
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[P_str_SheetName]);
    worksheet.Move(workbook.Sheets[1], missing);                                //将选择的工作表作为第一个工作表
    object P_obj_Name = (object)worksheet.Name;                                 //获取工作表名称
    excel.DisplayAlerts = false;                                                //设置 Excel 保存时不显示对话框
    workbook.Save();                                                            //保存工作簿
    CloseProcess("EXCEL");                                                      //关闭所有 Excel 进程
    try
    {
        access.OpenCurrentDatabase(txt_Access.Text, true, "");                  //打开 Access 数据库
        //将 Excel 指定工作表中的数据导入到 Access 中
        access.DoCmd.TransferSpreadsheet(Microsoft.Office.Interop.Access.AcDataTransferType.acImport,
Microsoft.Office.Interop.Access.AcSpreadSheetType.acSpreadsheetTypeExcel97, P_obj_Name, P_str_Excel, true, missing, missing);
        access.Quit(Microsoft.Office.Interop.Access.AcQuitOption.acQuitSaveAll);//关闭并保存 Access 数据库文件
        CloseProcess("MSACCESS");                                               //关闭所有 Access 数据库进程
    }
    catch
    {
        MessageBox.Show("Access 数据库中已经存在" + P_str_SheetName + "表！ ", "警告", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        CloseProcess("MSACCESS");                                               //关闭所有 Access 数据库进程
    }
}
```

说明：上面的代码中用到了 ImportDataToSql 方法，该方法主要用来将 Excel 工作表中的数据导入到 SQL Server 数据库中，关于该方法的详细代码请参见实例 467 中的设计过程。

秘笈心法

心法领悟 469：List<T>泛型集合的使用。

List<T>泛型集合表示可通过索引访问的对象的强类型列表，它是 ArrayList 类的泛型等效类，提供用于对列表进行搜索、排序和操作的方法。

实例 470　每天定时读取 Excel 文件给指定数据库

光盘位置：光盘\MR\20\470　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了每天定时读取 Excel 文件数据给指定数据库的功能。运行本实例，如图 20.44 所示，单击第一个“选择”按钮，选择要读取的 Excel 文件；如果选中“Access 数据库设置”单选按钮，则该区域的所有控件可用，这时选择 Access 数据库文件的路径；如果选中“SQL Server 数据库连接设置”单选按钮，则该区域的所有控件可用，在该区域中对 SQL Server 数据库连接信息进行设置；最后在“定时设置”栏中设置定时时间，单击“设置”按钮，启动 Timer 计时器，在该计时器中实时判断当前时间是否与设置的定时时间相同，如果相同，则将选择的 Excel 文件的数据读取到指定的数据库中。

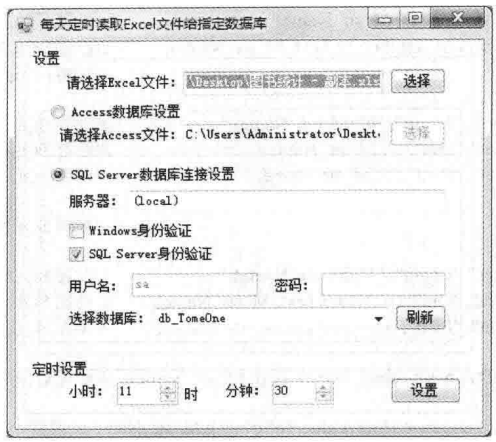


图 20.44　每天定时读取 Excel 文件给指定数据库

关键技术

本实例主要实现每天定时将 Excel 数据读取到 Access 和 SQL Server 两种不同的数据库中，实现该功能时，首先需要把用户的设置写入到 INI 文件中，当程序启动时，在 Timer 计时器中判断系统当前时间是否与用户设置的时间相同，如果相同，则将 Excel 文件中的每个工作表中的数据读取到用户设置的数据库中，在实现过程中需要用到的技术主要有 INI 文件的读写、Timer 计时器的使用、Microsoft Access 自动化对象模型的 DoCmd 对象的 TransferSpreadsheet 方法和 SqlBulkCopy 类的 WriteToServer 方法等。

说明：（1）关于 INI 文件读写操作的详细讲解请参见实例 462 中的关键技术。

（2）关于 Timer 计时器的详细讲解请参见实例 461 中的关键技术。

（3）关于 DoCmd 对象的 TransferSpreadsheet 方法的详细讲解请参见实例 466 中的关键技术。

（4）关于 SqlBulkCopy 类的 WriteToServer 方法的详细讲解请参见实例 467 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TimeExcelToDatabase。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 5 个 TextBox 控件，分别用来显示选择的 Excel 文件路径、显示选择的 Access 文件路径、输入 SQL Server 服务器名、登录用户名和登录密码；添加两个 RadioButton 控件，用来选择要将 Excel 文件内容导入到何种数据库中；添加两个 CheckBox 控件，用来选择 SQL Server 数据库的登录方式；添加一个 ComboBox 控件，用来选择要导入到的 SQL Server 数据库；添加两个 NumericUpDown 控件，分别用来设置小时和分钟；添加 4 个 Button 控件，分别用来执行选择 Excel 文件、选择 Access 文件、获取指定 SQL Server 服务器上的所有数据库和将设置写入到系统配置文件的操作；添加一个 Timer 组件，用来实时检测当前时间是否与设置的时间相同，如果相同，则将 Excel 文件内容导入到指定数据库中。

（3）程序的主要代码如下。

单击“设置”按钮，把用户设置的信息写入到系统 INI 文件中，并启动计时器。“设置”按钮的 Click 事件代码如下：

```
private void btn_Set_Click(object sender, EventArgs e)
{
    WritePrivateProfileString("Set", "Excel", txt_Path.Text, M_str_Name);          //设置 Excel 文件路径
    if (rbtn_Access.Checked)
    {
        WritePrivateProfileString("Set", "Mode", "0", M_str_Name);                 //设置要导出的数据库为 Access
        WritePrivateProfileString("Set", "Access", txt_Access.Text, M_str_Name);   //设置 Access 数据库路径
    }
    else if (rbtn_Sql.Checked)
    {
```

```
        WritePrivateProfileString("Set", "Mode", "1", M_str_Name);                    //设置要导出的数据库为 SQL Server
        WritePrivateProfileString("Set", "Server", txt_Server.Text, M_str_Name);      //设置 SQL 服务器名
        if (ckbox_Windows.Checked)                                                    //判断 Windows 身份验证复选框是否选中
        {
            WritePrivateProfileString("Set", "Login", "0", M_str_Name);               //设置 SQL 登录模式的值为 0
            WritePrivateProfileString("Set", "Uid", "", M_str_Name);                  //设置 SQL 登录用户名为空
            WritePrivateProfileString("Set", "Pwd", "", M_str_Name);                  //设置 SQL 登录密码为空
        }
        else if (ckbox_SQL.Checked)                                                   //判断 SQL Server 身份验证复选框是否选中
        {
            WritePrivateProfileString("Set", "Login", "1", M_str_Name);               //设置 SQL 登录模式的值为 1
            WritePrivateProfileString("Set", "Uid", txt_Name.Text, M_str_Name);       //设置 SQL 登录用户名
            WritePrivateProfileString("Set", "Pwd", txt_Pwd.Text, M_str_Name);        //设置 SQL 登录密码
        }
        WritePrivateProfileString("Set", "Database", cbox_Server.Text, M_str_Name);   //设置数据库名
    }
    WritePrivateProfileString("Set", "Hour", nudown_Hour.Value.ToString(), M_str_Name);//设置小时
    WritePrivateProfileString("Set", "Min", nudown_Min.Value.ToString(), M_str_Name);//设置分钟
    MessageBox.Show("配置文件设置成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    timer1.Start();                                                                   //启动计时器
}
```

在 timer1 组件的 Tick 事件中，首先判断当前时间是否与设置的时间相同，如果相同，则根据用户的设置将指定的 Excel 文件中数据导入到指定数据库中。timer1 组件的 Tick 事件代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (DateTime.Now.Hour == nudown_Hour.Value && DateTime.Now.Minute == nudown_Min.Value)
    {
        string P_str_Name = txt_Path.Text;                                            //记录 Excel 文件名
        List<string> P_list_SheetNames = new List<string>();                         //创建泛型集合对象，用来存储工作表名称
        P_list_SheetNames = GetSheetName(P_str_Name);                                 //获取 Excel 文件中的所有工作表名
        for (int j = 0; j < P_list_SheetNames.Count; j++)                             //遍历所有工作表
        {
            if (rbtn_Access.Checked)                                                  //判断 Access 数据库连接设置单选按钮选中
            {
                ImportDataToAccess(P_str_Name, P_list_SheetNames[j]);                 //将工作表内容导出到 Access
            }
            else if (rbtn_Sql.Checked)                                                //判断 SQL Server 数据库连接设置单选按钮选中
            {
                if (ckbox_Windows.Checked)                                            //如果用 Windows 身份验证登录 SQL Server
                    ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Initial Catalog =" + cbox_Server.
Text + ";Integrated Security=SSPI;");                                                 //将工作表内容导出到 SQL Server
                else if (ckbox_SQL.Checked)                                           //如果用 SQL Server 身份验证登录 SQL Server
                    //将工作表内容导出到 SQL Server
                    ImportDataToSql(P_str_Name, P_list_SheetNames[j], "Data Source=" + txt_Server.Text + ";Database=" + cbox_Server.Text +
";Uid=" + txt_Name.Text + ";Pwd=" + txt_Pwd.Text + ";");
            }
        }
        MessageBox.Show("程序在" + DateTime.Now.ToShortTimeString() + "分时自动将 Excel 工作表中数据导入到了指定的数据库中！", "提
示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

说明：上面的代码中用到了 ImportDataToAccess 和 ImportDataToSql 两个方法，其中 ImportDataToAccess 方法用来将 Excel 工作表中的数据导入到 Access 数据库中，ImportDataToSql 方法用来将 Excel 工作表中的数据导入到 SQL Server 数据库中。关于 ImportDataToAccess 方法的详细代码请参见实例 469 中的设计过程，关于 ImportDataToSql 方法的详细代码请参见实例 467 中的设计过程。

秘笈心法

心法领悟 470：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

20.4　利用 Excel 进行数据挖掘

实例 471	在 Excel 的指定范围内搜索文本	高级
	光盘位置：光盘\MR\20\471	趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了在 Excel 的指定范围内搜索文本的功能。运行本实例，如图 20.45 所示，首先单击上面工具栏中的“打开 Excel 文件”按钮，打开指定的 Excel 文件，并且选择要操作的工作表；然后在下面的工具栏中输入查找的范围及文本，单击“搜索”按钮，即可在指定工作表的指定范围内搜索文本，并将搜索到的文本以红色字体显示。

图 20.45　在 Excel 的指定范围内搜索文本

关键技术

本实例实现在 Excel 的指定范围内搜索文本时，主要用到了 Microsoft Excel 自动化对象模型的 Range 对象的 Find 方法，下面对其进行详细讲解。

Range 对象表示 Excel 数据表格中一定区域的单元格，该区域中可以包括一个单元格、多个连续的单元格以及多个不连续的单元格，其 Find 方法用来在 Range 对象中查找文本，并返回包含查找文本的单元格，该方法的语法格式如下：

```
Range Find([In] object What, [In, Optional] object After, [In, Optional] object LookIn, [In, Optional] object LookAt, [In, Optional] object SearchOrder, [In, Optional] XlSearchDirection SearchDirection, [In, Optional] object MatchCase, [In, Optional] object MatchByte, [In, Optional] object SearchFormat);
```

Find 方法中的参数说明如表 20.8 所示。

表 20.8　Find 方法中的参数说明

参　数	描　述
What	要搜索的文本
After	单个单元格，搜索区域将从该单元格之后开始
LookIn	要搜索的数据类型
LookAt	指定是否匹配整个单元格的内容
SearchOrder	指定按行或者列方式搜索
SearchDirection	指定向前或向后搜索
MatchCase	指定搜索时是否区分大小写
MatchByte	指定是否对双字节进行完全匹配
SearchFormat	指定搜索文本是否格式化

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SearchTextInRange。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 ToolStrip 控件，分别用来作为程序的两个工具栏，其中第一个工具栏用来选择 Excel 文件及其工作表，第二个工具栏用来输入查找范围、要查找的文本及执行查找操作；添加一个 WebBrowser 控件，用来显示 Excel 文件。

（3）程序的主要代码如下：

```
private void tsbtn_Query_Click(object sender, EventArgs e)
{
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    string P_str_Excel = tstxt_Excel.Text;                          //记录 Excel 文件路径
    string P_str_SheetName = tscbox_Sheet.Text;                     //记录选择的工作表名称
    object P_obj_Start = tstxt_Start.Text;                          //记录开始单元格
    object P_obj_End = tstxt_End.Text;                              //记录结束单元格
    object missing = System.Reflection.Missing.Value;              //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                 //声明工作表
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[P_str_SheetName]);
    //定义查找范围
    Microsoft.Office.Interop.Excel.Range searchRange = worksheet.get_Range(P_obj_Start, P_obj_End);
    Microsoft.Office.Interop.Excel.Range currentRange = null;       //定义当前找到的范围
    Microsoft.Office.Interop.Excel.Range firstRange = null;         //定义找到的第一个范围
    object P_obj_Text = tstxt_Text.Text;                            //记录要搜索的文本
    //搜索第一个匹配项，指定从其后开始搜索的单元格以外的所有参数
    currentRange = searchRange.Find(P_obj_Text, missing, Microsoft.Office.Interop.Excel.XlFindLookIn.xlValues, Microsoft.Office.Interop.Excel.XlLookAt.
xlPart, Microsoft.Office.Interop.Excel.XlSearchOrder.xlByRows, Microsoft.Office.Interop.Excel.XlSearchDirection.xlNext, false, missing, missing);
    while (currentRange != null)                                    //一直搜索，直到没有匹配项
    {
        if (firstRange == null)                                     //如果第一个范围不包含任何值
        {
            firstRange = currentRange;                              //记录当前范围
        }
        //如果查找范围的地址与第一个查找范围的地址匹配
        else if (currentRange.get_Address(missing, missing, Microsoft.Office.Interop.Excel.XlReferenceStyle.xlA1, missing, missing) == firstRange.
get_Address(missing, missing, Microsoft.Office.Interop.Excel.XlReferenceStyle.xlA1, missing, missing))
        {
            break;                                                  //退出
        }
        //为单元格添加边框
        currentRange.BorderAround(Microsoft.Office.Interop.Excel.XlLineStyle.xlContinuous,
Microsoft.Office.Interop.Excel.XlBorderWeight.xlThin, Microsoft.Office.Interop.Excel.XlColorIndex.xlColorIndexAutomatic, Color.Black.ToArgb());
        currentRange.Font.Color = System.Drawing.ColorTranslator.ToOle(Color.Red);  //设置搜索到的文本颜色
        currentRange.Font.Bold = true;                              //设置搜索到的文本为粗体
        currentRange = searchRange.FindNext(currentRange);          //查找下一处
    }
    MessageBox.Show("搜索完毕！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    excel.DisplayAlerts = false;                                    //设置保存 Excel 时不显示对话框
    workbook.Save();                                                //保存工作表
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    WBrowser_Excel.Navigate(P_str_Excel);                           //在窗体中重新显示 Excel 文件内容
}
```

秘笈心法

心法领悟 471：Font 对象的妙用。

如果理解了实例中 Font 对象的使用方法，那么就可以轻松地设置其 Name 属性更改 Excel 文件中字体的名称，设置 Size 属性更改 Excel 文件中字体的大小，设置 Color 属性更改 Excel 文件中字体的颜色。

实例 472　利用 Excel 对数据进行多列排序

光盘位置：光盘\MR\20\472　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了在 Excel 中对数据进行多列排序的功能。运行本实例，首先单击上面工具栏中的“打开 Excel 文件”按钮，打开指定的 Excel 文件，并且选择要操作的工作表；然后在下面的工具栏中输入要排序的范围，单击“升序排序”按钮，对输入的多列范围进行升序排序，如图 20.46 所示。

单击“降序排序”按钮，对输入的多列范围进行降序排序，如图 20.47 所示。

图 20.46　多列升序排序

图 20.47　多列降序排序

关键技术

本实例实现在 Excel 中对数据进行多列排序时，主要用到了 Microsoft Excel 自动化对象模型的 Range 对象的 Sort 方法，下面对其进行详细讲解。

Sort 方法用来对 Excel 文件中的列进行排序，该方法的语法格式如下：

```
object Sort([In, Optional] object Key1,[In, Optional] XlSortOrder Order1,[In, Optional] object Key2,[In, Optional] object Type,[In, Optional] XlSortOrder Order2,[In, Optional] object Key3,[In, Optional] XlSortOrder Order3,[In, Optional] XlYesNoGuess Header,[In, Optional] object OrderCustom,[In, Optional] object MatchCase,[In, Optional] XlSortOrientation Orientation,[In, Optional] XlSortMethod SortMethod,[In, Optional] XlSortDataOption DataOption1, [In, Optional] XlSortDataOption DataOption2, [In, Optional] XlSortDataOption DataOption3);
```

Sort 方法中的参数说明如表 20.9 所示。

表 20.9　Sort 方法中的参数说明

参　数	描　述
Key1	指定要排序的第一列
Order1	指定第一列的排序方式
Key2	指定要排序的第二列
Type	指定对数据透视表进行排序时对哪些元素进行排序
Order2	指定第二列的排序方式
Key3	指定要排序的第 3 列
Order3	指定第 3 列的排序方式
Header	指定第一行是否包含页眉信息
OrderCustom	指定一个在自定义排序顺序列表中的从 1 开始的偏移量
MatchCase	指定排序时是否区分大小写
Orientation	指定排序方向
SortMethod	指定用于中文语言的排序方法
DataOption1	指定如何对 Key1 中指定的范围内的文本进行排序
DataOption2	指定如何对 Key2 中指定的范围内的文本进行排序
DataOption3	指定如何对 Key3 中指定的范围内的文本进行排序

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SortMultiColumns。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 ToolStrip 控件，分别用来作为程序的两个工具栏，其中第一个工具栏用来选择 Excel 文件及其工作表，第二个工具栏用来输入排序范围及执行升序排序、降序排序操作；添加一个 WebBrowser 控件，用来显示 Excel 文件。

（3）程序的主要代码如下。

单击“升序排序”按钮，使用 Excel.Range 对象的 Sort 方法对指定的列进行升序排序。代码如下：

```
private void tsbtn_ASCSort_Click(object sender, EventArgs e)
{
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    string P_str_Excel = tstxt_Excel.Text;                          //记录 Excel 文件路径
    string P_str_SheetName = tscbox_Sheet.Text;                     //记录选择的工作表名称
    object P_obj_Start = tstxt_Start.Text;                          //记录开始单元格
    object P_obj_End = tstxt_End.Text;                              //记录结束单元格
    object missing = System.Reflection.Missing.Value;              //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                 //声明工作表
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[P_str_SheetName]);
    //定义查找范围
    Microsoft.Office.Interop.Excel.Range searchRange = worksheet.get_Range(P_obj_Start, P_obj_End);
    switch (searchRange.Columns.Count)
    {
        case 1:
            //按 1 列升序排序
            searchRange.Sort(
     searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     missing, missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
     Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
     Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
            break;
        case 2:
            //按 2 列升序排序
            searchRange.Sort(
     searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     searchRange.Columns[2, missing], missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
     Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
     Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
            break;
        case 3:
        default:
            //按 3 列升序排序
            searchRange.Sort(
     searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     searchRange.Columns[2, missing], missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     searchRange.Columns[3, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlAscending,
     Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
     Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
     Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
```

```
        Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
        Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
        Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
                break;
        }
        excel.DisplayAlerts = false;                                    //设置保存 Excel 时不显示对话框
        workbook.Save();                                                //保存工作表
        CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
        WBrowser_Excel.Navigate(P_str_Excel);                           //在窗体中重新显示 Excel 文件内容
}
```

单击“降序排序”按钮，使用 Excel.Range 对象的 Sort 方法对指定的列进行降序排序。代码如下：

```
private void tsbtn_DESCSort_Click(object sender, EventArgs e)
{
    CloseProcess("EXCEL");                                              //关闭所有 Excel 进程
    string P_str_Excel = tstxt_Excel.Text;                              //记录 Excel 文件路径
    string P_str_SheetName = tscbox_Sheet.Text;                         //记录选择的工作表名称
    object P_obj_Start = tstxt_Start.Text;                              //记录开始单元格
    object P_obj_End = tstxt_End.Text;                                  //记录结束单元格
    object missing = System.Reflection.Missing.Value;                  //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                     //声明工作表
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[P_str_SheetName]);
    //定义查找范围
    Microsoft.Office.Interop.Excel.Range searchRange = worksheet.get_Range(P_obj_Start, P_obj_End);
    switch (searchRange.Columns.Count)
    {
        case 1:
            //按 1 列降序排序
            searchRange.Sort(
     searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
     missing, missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
     missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
     Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
     Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
     Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
     Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
            break;
        case 2:
            //按 2 列降序排序
            searchRange.Sort(
    searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    searchRange.Columns[2, missing], missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
    Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
    Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
    Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
    Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
    Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
            break;
        case 3:
        default:
            //按 3 列降序排序
            searchRange.Sort(
    searchRange.Columns[1, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    searchRange.Columns[2, missing], missing, Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    searchRange.Columns[3, missing], Microsoft.Office.Interop.Excel.XlSortOrder.xlDescending,
    Microsoft.Office.Interop.Excel.XlYesNoGuess.xlNo, missing, missing,
    Microsoft.Office.Interop.Excel.XlSortOrientation.xlSortColumns,
    Microsoft.Office.Interop.Excel.XlSortMethod.xlPinYin,
```

```
            Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
            Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal,
            Microsoft.Office.Interop.Excel.XlSortDataOption.xlSortNormal);
                    break;
            }
            excel.DisplayAlerts = false;                                    //设置保存 Excel 时不显示对话框
            workbook.Save();                                                //保存工作表
            CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
            WBrowser_Excel.Navigate(P_str_Excel);                           //在窗体中重新显示 Excel 文件内容
        }
```

秘笈心法

心法领悟 472：巧用 Excel 对象的 Sort 方法对列进行排序。

Excel 对象的 Sort 方法主要用来对 Excel 文件中的列进行排序，它可以分别对一列、两列或者 3 列数据进行排序。

实例 473	利用 Excel 将数据生成图表	高级
	光盘位置：光盘\MR\20\473	趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了在 Excel 中生成图表的功能。运行本实例，首先单击工具栏中的“打开 Excel 文件”按钮，打开指定的 Excel 文件，并且选择要操作的工作表；然后单击“生成图表”按钮，即可在指定的工作表中生成一个柱形图，如图 20.48 所示。

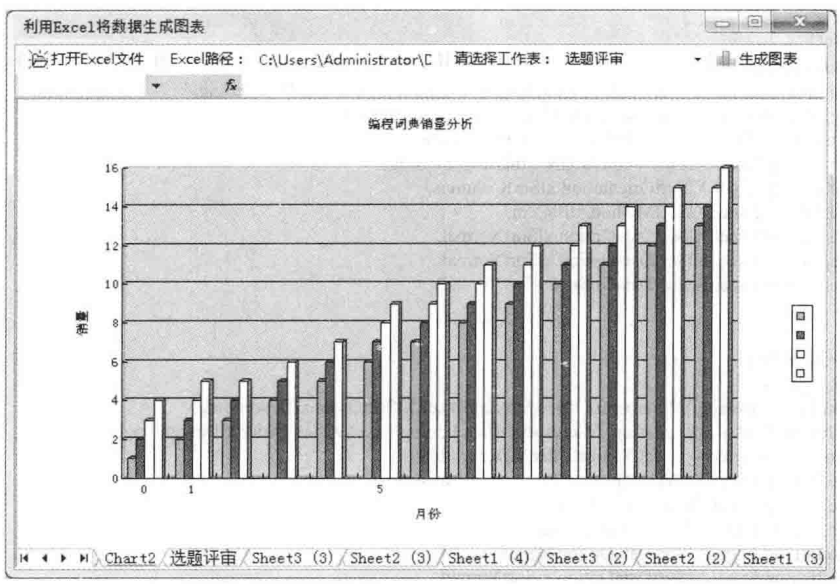


图 20.48　利用 Excel 将数据生成图表

关键技术

本实例实现在 Excel 中生成图表时，主要用到了 Microsoft Excel 自动化对象模型的 Chart 对象的 ChartWizard 方法，下面对其进行详细讲解。

Chart 对象是公开事件的图表对象，它可以绑定到数据，其 ChartWizard 方法用来修改给定 Chart 控件的属性，以便绘制一个新的图表，该方法的语法格式如下：

```
public void ChartWizard(Object Source,Object Gallery,Object Format,Object PlotBy,Object CategoryLabels,Object SeriesLabels,Object HasLegend,
Object Title,Object CategoryTitle,Object ValueTitle,Object ExtraTitle)
```

ChartWizard 方法中的参数说明如表 20.10 所示。

表 20.10　ChartWizard 方法中的参数说明

参　数	描　述
Source	包含新图表的源数据的范围
Gallery	图表类型
Format	内置自动套用格式的选项编号
PlotBy	指定每个系列的数据是按行绘制还是按列绘制
CategoryLabels	指定源范围中包含类别标签的行数或列数
SeriesLabels	指定源范围中包含系列标签的行数或列数
HasLegend	指定是否包含图例
Title	指定标题文本
CategoryTitle	分类轴标题文本
ValueTitle	数值轴标题文本
ExtraTitle	三维图表的系列轴标题或二维图表的第二个数值轴标题

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ExcelChart。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为程序的工具栏，该工具栏用来选择 Excel 文件及其工作表，并执行生成图表操作；添加一个 WebBrowser 控件，用来显示 Excel 文件及生成的图表。

（3）程序的主要代码如下：

```
private void tsbtn_Build_Click(object sender, EventArgs e)
{
    CloseProcess("EXCEL");                                                          //关闭所有 Excel 进程
    object missing = System.Reflection.Missing.Value;                               //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(tstxt_Excel.Text, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                                 //声明工作表
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[tscbox_Sheet.Text]);
    Microsoft.Office.Interop.Excel.Range searchRange = worksheet.get_Range("A1", "E1");          //定义标题范围
    object[] P_obj_Items = { "编程词典", "VC 编程词典", "Java 编程词典", "ASP.NET 编程词典", "C#编程词典" };
    searchRange.set_Value(missing, P_obj_Items);                                    //绘制标题
    searchRange.Font.Bold = true;                                                   //设置字体加粗
    searchRange.Font.Name = "宋体";                                                 //设置字体样式
    searchRange.Font.Size = 10;                                                     //设置字体大小
    searchRange.HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;   //设置标题对齐方式
    //获得要生成图表的数据
    for (int i = 0; i < 13; i++)
    {
        worksheet.Cells[2 + i, 1] = i;
        worksheet.Cells[2 + i, 2] = i + 1;
        worksheet.Cells[2 + i, 3] = i + 2;
        worksheet.Cells[2 + i, 4] = i + 3;
        worksheet.Cells[2 + i, 5] = i + 4;
    }
    //创建 Excel 绘图对象
    Microsoft.Office.Interop.Excel.Chart chart = (Microsoft.Office.Interop.Excel.Chart)workbook.Charts.Add(missing, missing, missing, missing);
    //定义绘制图表范围
    Microsoft.Office.Interop.Excel.Range chartRange = worksheet.get_Range("A1:A14", "B1:E14");
    //在指定范围绘制图表
    chart.ChartWizard(chartRange, Microsoft.Office.Interop.Excel.XlChartType.xl3DColumn, missing, Microsoft.Office.Interop.Excel.XlRowCol.xlColumns, 1,
```

```
1, true, "编程词典销量分析", "月份", "销量", missing);
    excel.DisplayAlerts = false;                          //设置保存 Excel 时不显示对话框
    workbook.Save();                                      //保存工作簿
    workbook.Close(false, missing, missing);              //关闭工作簿
    CloseProcess("EXCEL");                                //关闭所有 Excel 进程
    WBrowser_Excel.Navigate(tstxt_Excel.Text);            //在窗体中重新显示 Excel 文件内容
}
```

秘笈心法

心法领悟 473：通过 Cells 属性为单元格赋值。

Excel 对象的 Cells 属性用来指定 Excel 工作簿中单元格的内容。例如，下面的代码用来指定 Excel 工作簿中第一行第一列的单元格中的内容为 Excel。

```
excel.Cells[1, 1] = "Excel";
```

实例 474　利用 Excel 对数据进行多表计算

光盘位置：光盘\MR\20\474　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了利用 Excel 对数据进行多表计算的功能。运行本实例，如图 20.49 所示，首先单击上面工具栏中的“打开 Excel 模板”按钮，打开指定的 Excel 模板文件，并且选择要操作的工作表；然后在下面的工具栏中输入要计算的列和行，单击“合并计算”按钮，程序自动对 Debug 文件夹中的 3 个 Excel 文件进行合并计算，并体现在 Excel 模板文件中。

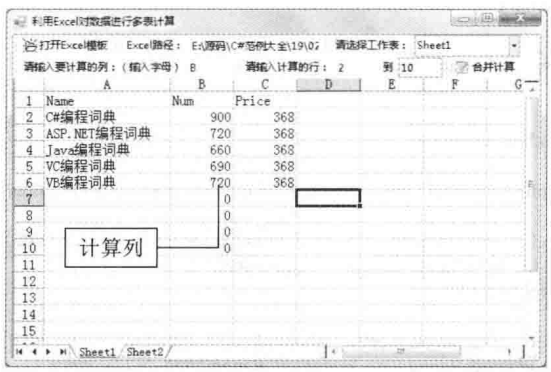


图 20.49　利用 Excel 对数据进行多表计算

关键技术

本实例在实现对多个 Excel 工作表中的内容进行计算时，首先需要有一个固定的模板，以便使用户知道要对哪行哪列数据进行计算，然后需要使用 Microsoft Excel 自动化对象模型中的 Range 对象（使用 Worksheet 对象的 Cells 属性获得）的 Text 属性获取 Excel 工作表中指定单元格的内容，最后将获取到的内容进行相加，并且写入汇总 Excel 文件的相应单元格中即可。下面对本实例中用到的关键技术进行详细讲解。

（1）Worksheet 对象的 Cells 属性

Worksheet 对象是公开事件并用作控件的容器的 Excel 工作表对象，其 Cells 属性用来获取一个 Range 对象，该对象表示工作表上的所有单元格（不仅是当前正在使用的单元格）。Cells 属性的语法格式如下：

```
[BrowsableAttribute(false)]
public virtual Range Cells { get; }
```

参数说明

属性值：一个 Range 对象，表示工作表上的所有单元格（不仅是当前正在使用的单元格）。

（2）Range 对象的 Text 属性

Range 对象的 Text 属性用来获取指定单元格中的内容。语法格式如下：

```
object Text {get;}
```

参数说明

属性值：指定单元格的内容。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CalMultiSheet。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 ToolStrip 控件，分别用来作为程序的两个工具栏，其中第一个工具栏用来选择 Excel 模板文件及其工作表，第二个工具栏用来输入要计算的列、行及执行合并计算操作；添加一个 WebBrowser 控件，用来显示 Excel 文件。

（3）程序的主要代码如下：

```
private void CalcData(int P_int_Row, string P_str_Column)
{
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    string P_str_workBook1 = Application.StartupPath + "\\Excel1.xls";   //记录第一个要计算的工作簿路径
    string P_str_workBook2 = Application.StartupPath + "\\Excel2.xls";   //记录第二个要计算的工作簿路径
    string P_str_workBook3 = Application.StartupPath + "\\Excel3.xls";   //记录第三个要计算的工作簿路径
    string P_str_workBook4 = Application.StartupPath + "\\Sum.xls";      //记录存储计算结果的工作簿路径
    object missing = System.Reflection.Missing.Value;               //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.ApplicationClass excel1 = new Microsoft.Office.Interop.Excel.ApplicationClass();
    excel1.Visible = false;                                         //设置 Excel 文件隐藏
    //打开第一个计算的工作簿
    Microsoft.Office.Interop.Excel.Workbook workbook1 = excel1.Workbooks.Open(P_str_workBook1, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //记录要计算的第一个工作表
    Microsoft.Office.Interop.Excel._Worksheet worksheet1 = (Microsoft.Office.Interop.Excel._Worksheet)workbook1.Worksheets.get_Item(tscbox_Sheet.Text);
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.ApplicationClass excel2 = new Microsoft.Office.Interop.Excel.ApplicationClass();
    excel2.Visible = false;                                         //设置 Excel 文件隐藏
    //打开第二个计算的工作簿
    Microsoft.Office.Interop.Excel.Workbook workbook2 = excel2.Workbooks.Open(P_str_workBook2, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //记录要计算的第二个工作表
    Microsoft.Office.Interop.Excel._Worksheet worksheet2 = (Microsoft.Office.Interop.Excel._Worksheet)workbook2.Worksheets.get_Item(tscbox_Sheet.Text);
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.ApplicationClass excel3 = new Microsoft.Office.Interop.Excel.ApplicationClass();
    excel3.Visible = false;                                         //设置 Excel 文件隐藏
    //打开第三个计算的工作簿
    Microsoft.Office.Interop.Excel.Workbook workbook3 = excel3.Workbooks.Open(P_str_workBook3, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //记录要计算的第三个工作表
    Microsoft.Office.Interop.Excel._Worksheet worksheet3 = (Microsoft.Office.Interop.Excel._Worksheet)workbook3.Worksheets.get_Item(tscbox_Sheet.Text);
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.ApplicationClass excel4 = new Microsoft.Office.Interop.Excel.ApplicationClass();
    excel4.Visible = false;                                         //设置 Excel 文件隐藏
    //打开存储计算结果的工作簿
    Microsoft.Office.Interop.Excel.Workbook workbook4 = excel4.Workbooks.Open(P_str_workBook4, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //记录要存储计算结果的工作表
    Microsoft.Office.Interop.Excel._Worksheet worksheet4 = (Microsoft.Office.Interop.Excel._Worksheet)workbook4.Worksheets.get_Item(tscbox_Sheet.Text);
    excel4.DisplayAlerts = false;                                   //设置保存 Excel 时不显示对话框
    worksheet4.Activate();                                          //激活工作表
    Decimal P_dml_NumOne = 0;                                       //获取第一个工作表的值
    Decimal P_dml_NumTwo = 0;                                       //获取第二个工作表的值
    Decimal P_dml_NumThree = 0;                                     //获取第三个工作表的值
    //判断指定单元格内容是否为空
    if (((Microsoft.Office.Interop.Excel.Range)worksheet1.Cells[P_int_Row, P_str_Column]).Text.ToString() == "" || ((Microsoft.Office.Interop.Excel.
```

```
Range)worksheet1.Cells[P_int_Row, P_str_Column]).Text == null)
    {
        P_dml_NumOne = 0;                                          //将第一个值设置为 0
    }
    else
    {
        P_dml_NumOne = Convert.ToDecimal(((Microsoft.Office.Interop.Excel.Range)worksheet1.Cells[P_int_Row, P_str_Column]).Text);
        //记录第一个值
    }
    //判断指定单元格内容是否为空
    if (((Microsoft.Office.Interop.Excel.Range)worksheet2.Cells[P_int_Row, P_str_Column]).Text.ToString() == "" || ((Microsoft.Office.Interop.Excel.
Range)worksheet2.Cells[P_int_Row, P_str_Column]).Text == null)
    {
        P_dml_NumTwo = 0;                                          //将第二个值设置为 0
    }
    else
    {
        P_dml_NumTwo = Convert.ToDecimal(((Microsoft.Office.Interop.Excel.Range)worksheet2.Cells[P_int_Row, P_str_Column]).Text);
        //记录第二个值
    }
    //判断指定单元格内容是否为空
    if (((Microsoft.Office.Interop.Excel.Range)worksheet3.Cells[P_int_Row, P_str_Column]).Text.ToString() == "" || ((Microsoft.Office.Interop.Excel.
Range)worksheet3.Cells[P_int_Row, P_str_Column]).Text == null)
    {
        P_dml_NumThree = 0;                                        //将第三个值设置为 0
    }
    else
    {
        //记录第三个值
        P_dml_NumThree = Convert.ToDecimal(((Microsoft.Office.Interop.Excel.Range)worksheet3.Cells[P_int_Row, P_str_Column]).Text);
    }
    Decimal P_dml_Sum = P_dml_NumOne + P_dml_NumTwo + P_dml_NumThree;//计算总和
    try
    {
        //向工作簿的指定单元格中写入计算后的数据
        worksheet4.Cells[P_int_Row, P_str_Column] = P_dml_Sum.ToString();
        workbook4.Save();                                          //保存 Excel 文件
    }
    catch
    {
        MessageBox.Show("写入第" + P_int_Row.ToString() + "行，第" + P_str_Column + "列时出错！");
    }
    finally
    {
        workbook4.Save();                                          //保存 Excel 文件
        //退出打开的 4 个 Excel 文件
        excel1.Quit();
        excel2.Quit();
        excel3.Quit();
        excel4.Quit();
    }
    WBrowser_Excel.Navigate(P_str_workBook4);                      //在窗体中显示多表计算后的 Excel 文件
}
```

秘笈心法

心法领悟 474：如何在程序中打开 Excel 文件？

打开 Excel 文件时，可以使用 Workbooks 对象的 Open 方法来实现，该方法的第一个参数为要打开的 Excel 文件，其他的参数表示 Excel 的相关设置，实际使用时，直接用 Missing.Value 的值来表示即可。例如，本实例中打开指定 Excel 工作表的实现代码如下：

```
Microsoft.Office.Interop.Excel.Workbook workbook1 = excel1.Workbooks.Open(P_str_workBook1, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing);
```

实例 475　利用 Excel 对数据进行格式转换

光盘位置：光盘\MR\20\475　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了利用 Excel 对数据进行格式转换的功能。运行本实例，如图 20.50 所示，首先选择查询条件，并输入查询关键字，单击“查询”按钮，在 SQL Server 数据库中执行查询，并将查询结果显示在数据表格控件中；然后单击“Excel 格式转换”按钮，将数据表格控件中的查询结果输出到 Excel 文件中，并对 Excel 文件中的数据进行格式转换，这里将日期类型转换为长日期格式，数字转换为“000,000.00”格式，字符串格式转化为使用双引号括起来，如图 20.51 所示。

图 20.50　利用 Excel 对数据进行格式转换

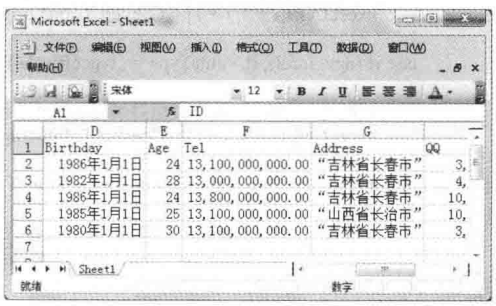


图 20.51　格式转换之后的 Excel 文件

关键技术

本实例在对数据进行格式转换时，主要是通过 DataGridViewCell 对象的 ValueType 属性判断 DataGridView 单元格中的数据类型来实现的，下面对其进行详细讲解。

DataGridViewCell 对象表示 DataGridView 控件中的单个单元格，其 ValueType 属性用来获取或设置单元格中值的数据类型，该属性的语法格式如下：

```
[BrowsableAttribute(false)]
public virtual Type ValueType { get; set; }
```

参数说明

属性值：System.Type 类型，表示单元格中值的数据类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FormatDataInExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，用来选择查询条件；添加一个 TextBox 控件，用来输入查询关键字；添加一个 DataGridView 控件，用来显示 SQL Server 数据库中的数据；添加两个 Button 控件，分别用来执行数据库查询操作和将数据库查询结果输出到 Excel 文件并对不同的列进行格式转换的操作。

（3）程序的主要代码如下：

```
private void btn_Excel_Click(object sender, EventArgs e)
{
    if (dgv_Info.Rows.Count == 0)                                    //判断是否有数据
        return;                                                      //返回
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    excel.Application.Workbooks.Add(true);                           //在 Excel 中添加一个工作簿
    excel.Visible = true;                                            //设置 Excel 显示
    //生成字段名称
    for (int i = 0; i < dgv_Info.ColumnCount; i++)
```

```
    {
        excel.Cells[1, i + 1] = dgv_Info.Columns[i].HeaderText;          //将数据表格控件中的列表头填充到 Excel 中
    }
    //填充数据
    for (int i = 0; i < dgv_Info.RowCount - 1; i++)                      //遍历数据表格控件的所有行
    {
        for (int j = 0; j < dgv_Info.ColumnCount; j++)                   //遍历数据表格控件的所有列
        {
            if (dgv_Info[j, i].ValueType == typeof(string))              //判断遍历到的数据是否是字符串类型
            {
                //格式化为字符串类型并填充 Excel 表格
                excel.Cells[i + 2, j + 1] = "'" + dgv_Info[j, i].Value.ToString().Trim() + "'";
            }
            //判断遍历到的数据是否是 16 位或者 32 位数字类型
            else if (dgv_Info[j, i].ValueType == typeof(Int16) || dgv_Info[j, i].ValueType == typeof(Int32))
            {
                //格式化为 32 位数字类型并填充 Excel 表格
                excel.Cells[i + 2, j + 1] = Convert.ToInt32(dgv_Info[j, i].Value).ToString("N");
            }
            else if (dgv_Info[j, i].ValueType == typeof(Int64))          //判断遍历到的数据是否是 64 位数字类型
            {
                //格式化为 64 位数字类型并填充 Excel 表格
                excel.Cells[i + 2, j + 1] = Convert.ToInt64(dgv_Info[j, i].Value).ToString("N");
            }
            else if (dgv_Info[j, i].ValueType == typeof(DateTime))       //判断遍历到的数据是否是日期类型
            {
                //格式化为长日期形式并填充 Excel 表格
                excel.Cells[i + 2, j + 1] = Convert.ToDateTime(dgv_Info[j, i].Value).ToLongDateString();
            }
            else
            {
                excel.Cells[i + 2, j + 1] = dgv_Info[j, i].Value.ToString();//填充 Excel 表格
            }
        }
    }
}
```

秘笈心法

心法领悟 475：方便地使用“对象初始化器”。

从 C# 3.0 版本开始，用户即可以方便地使用“对象初始化器”对所有的对象进行初始化，对象初始化器允许用户在一条语句中为指定对象的多个字段和属性赋值。实例中使用了对象初始化器初始化对象，并将初始化后的对象添加到集合中。

实例 476　将多个 Excel 文件汇总到一个 Excel 文件

光盘位置：光盘\MR\20\476　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将多个 Excel 文件汇总到一个 Excel 文件的功能。运行本实例，如图 20.52 所示，单击第一个“选择”按钮，选择要汇总的多个 Excel 文件；单击第二个“选择”按钮，选择存放汇总数据的 Excel 文件；然后单击“汇总”按钮，将多个 Excel 文件汇总到一个 Excel 文件中；单击“查看”按钮，查看汇总数据之后的 Excel 文件。

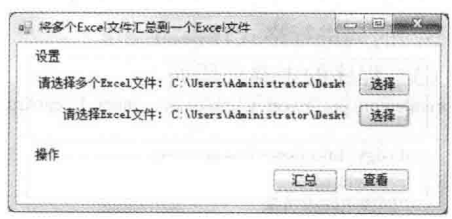


图 20.52　将多个 Excel 文件汇总到一个 Excel 文件

关键技术

本实例在实现将多个 Excel 文件的内容汇总到一个 Excel 文件时，主要用到了 Microsoft Excel 自动化对象模

型的 Worksheet 对象的 Copy 方法，下面对其进行详细讲解。

Worksheet 对象的 Copy 方法用来将工作表内容复制到工作簿中的另一个位置，也可以将工作表内容复制到其他的工作表中。语法格式如下：

```
public virtual void Copy(Object Before,Object After)
```

参数说明

❶ Before：复制工作表的放置位置之后的工作表。如果指定 After，则不能指定 Before。

❷ After：复制工作表的放置位置之前的工作表。如果指定 Before，则不能指定 After。

📖 **说明**：如果 Copy 方法中既未指定 Before，也未指定 After，则 Excel 将创建一个包含复制工作表的新工作簿。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiExcelToOneExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的多个 Excel 文件路径和单个 Excel 文件路径；添加 4 个 Button 控件，分别用来执行选择多个 Excel 文件、选择单个 Excel 文件、将多个 Excel 文件汇总到一个 Excel 文件和查看汇总的 Excel 文件的操作。

（3）程序的主要代码如下：

```
private void btn_Gather_Click(object sender, EventArgs e)
{
    object missing = System.Reflection.Missing.Value;                                    //定义 object 默认值
    string[] P_str_Names = txt_MultiExcel.Text.Split(',');                               //存储所有选择的 Excel 文件名
    string P_str_Name = "";                                                              //存储遍历到的 Excel 文件名
    List<string> P_list_SheetNames = new List<string>();                                 //创建泛型集合对象，用来存储工作表名称
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开指定的 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Excel.Text, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //创建新工作表
    Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    for (int i = 0; i < P_str_Names.Length - 1; i++)                                     //遍历所有选择的 Excel 文件名
    {
        P_str_Name = P_str_Names[i];                                                     //记录遍历到的 Excel 文件名
        //指定要复制的工作簿
        Microsoft.Office.Interop.Excel.Workbook Tempworkbook = excel.Application.Workbooks.Open(P_str_Name, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
        P_list_SheetNames = GetSheetName(P_str_Name);                                    //获取 Excel 文件中的所有工作表名
        for (int j = 0; j < P_list_SheetNames.Count; j++)                                //遍历所有工作表
        {
            //指定要复制的工作表
            Microsoft.Office.Interop.Excel.Worksheet TempWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)Tempworkbook.Sheets[P_list_
SheetNames[j]];                                                                          //创建新工作表
            TempWorksheet.Copy(missing, newWorksheet);                                   //将工作表内容复制到目标工作表中
        }
        Tempworkbook.Close(false, missing, missing);                                     //关闭临时工作簿
    }
    workbook.Save();                                                                     //保存目标工作簿
    workbook.Close(false, missing, missing);                                             //关闭目标工作簿
    MessageBox.Show("已经将所有选择的 Excel 工作表汇总到了一个 Excel 工作表中！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    CloseProcess("EXCEL");                                                               //关闭所有 Excel 进程
}
```

秘笈心法

心法领悟 476：在 Excel 文件中新增工作表。

在 Excel 文件中新增工作表时，可以使用 Excel 自动化对象模型的 Worksheets 对象的 Add 方法来实现，该方法主要用来为指定的工作簿添加工作表。例如，本实例中创建新工作表的实现代码如下：

```
Microsoft.Office.Interop.Excel.Worksheet newWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing, missing,
missing, missing);
```

实例 477 将多个 Excel 文件进行自动汇总

光盘位置：光盘\MR\20\477　　高级　　趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了将多个 Excel 文件进行自动汇总的功能。运行本实例，如图 20.53 所示，单击第一个“选择”按钮，选择要汇总的多个 Excel 文件；单击第二个“选择”按钮，选择存放汇总数据的 Excel 文件；然后在“自动汇总设置”栏中设置自动汇总时间，单击“设置”按钮，启动 Timer 计时器，在该计时器中实时判断当前时间是否与设置的自动汇总时间相同，如果相同，则自动将多个 Excel 文件汇总到一个 Excel 文件中；单击“查看”按钮，查看汇总数据之后的 Excel 文件。

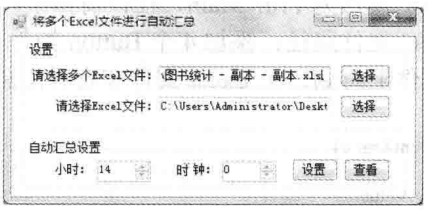


图 20.53　将多个 Excel 文件进行自动汇总

关键技术

本实例主要实现自动将多个 Excel 文件数据汇总到一个 Excel 文件中，实现该功能时，首先需要把用户的设置写入到 INI 文件中，当程序启动时，在 Timer 计时器中判断系统当前时间是否与用户设置的时间相同，如果相同，则调用 Microsoft Excel 自动化对象模型的 Worksheet 对象的 Copy 方法自动将多个 Excel 文件的数据汇总到一个 Excel 文件中。

说明：（1）关于 INI 文件读写操作的详细讲解请参见实例 462 中的关键技术。

（2）关于 Worksheet 对象的 Copy 方法的详细讲解请参见实例 476 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AutoMultiExcelToOneExcel。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示选择的多个 Excel 文件路径和单个 Excel 文件路径；添加两个 NumericUpDown 控件，分别用来设置小时和分钟；添加 4 个 Button 控件，分别用来执行选择多个 Excel 文件、选择单个 Excel 文件、将定时设置写入到系统配置文件中和查看汇总的 Excel 文件的操作；添加一个 Timer 组件，用来实时检测当前时间是否与设置的时间相同，如果相同，则自动将多个 Excel 文件的内容汇总到一个 Excel 中。

（3）程序的主要代码如下。

单击“设置”按钮，把用户设置的信息写入到系统 INI 文件中，并启动计时器。“设置”按钮的 Click 事件代码如下：

```
private void btn_Set_Click(object sender, EventArgs e)
{
    WritePrivateProfileString("Set", "MultiExcel", txt_MultiExcel.Text, M_str_Name);          //设置多个 Excel 文件路径
    WritePrivateProfileString("Set", "Excel", txt_Excel.Text, M_str_Name);                    //设置目标 Excel 文件路径
    WritePrivateProfileString("Set", "Hour", nudown_Hour.Value.ToString(), M_str_Name);       //设置小时
    WritePrivateProfileString("Set", "Min", nudown_Min.Value.ToString(), M_str_Name);         //设置分钟
    MessageBox.Show("配置文件设置成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    timer1.Start();                                                                           //启动计时器
}
```

说明：上面的代码中用到了 WritePrivateProfileString 函数，该函数主要用来对 INI 文件进行写入和修改操作，关于该函数的详细代码请参见实例 462 中的设计过程。

在 timer1 组件的 Tick 事件中，首先判断当前时间是否与设置的时间相同，如果相同，则自动将多个 Excel 文件的数据汇总到一个 Excel 文件中。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    object missing = System.Reflection.Missing.Value;                                    //定义 object 默认值
    string[] P_str_Names = txt_MultiExcel.Text.Split(',');                               //存储所有选择的 Excel 文件名
    string P_str_Name = "";                                                              //存储遍历到的 Excel 文件名
    List<string> P_list_SheetNames = new List<string>();                                 //创建泛型集合对象，用来存储工作表名称
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开指定的 Excel 文件
Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Open(txt_Excel.Text, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);
    //创建新工作表
    Microsoft.Office.Interop.Excel.Worksheet  newWorksheet  =  (Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets.Add(missing,
missing, missing, missing);
    if (DateTime.Now.Hour == nudown_Hour.Value && DateTime.Now.Minute == nudown_Min.Value)
    {
        for (int i = 0; i < P_str_Names.Length - 1; i++)                                 //遍历所有选择的 Excel 文件名
        {
            P_str_Name = P_str_Names[i];                                                 //记录遍历到的 Excel 文件名
            Microsoft.Office.Interop.Excel.Workbook Tempworkbook = excel.Application.Workbooks.Open(P_str_Name, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing, missing);//指定要复制的工作簿
            P_list_SheetNames = GetSheetName(P_str_Name);                                //获取 Excel 文件中的所有工作表名
            for (int j = 0; j < P_list_SheetNames.Count; j++)                            //遍历所有工作表
            {
                //指定要复制的工作表
                Microsoft.Office.Interop.Excel.Worksheet TempWorksheet = (Microsoft.Office.Interop.Excel.Worksheet)Tempworkbook.Sheets[P_list_
SheetNames[j]];                                                                          //创建新工作表
                TempWorksheet.Copy(missing, newWorksheet);                               //将工作表内容复制到目标工作表中
            }
            Tempworkbook.Close(false, missing, missing);                                 //关闭临时工作簿
        }
    }
    workbook.Save();                                                                     //保存目标工作簿
    workbook.Close(false, missing, missing);                                             //关闭目标工作簿
    MessageBox.Show("程序在" + DateTime.Now.ToShortTimeString() + "分时自动汇总了多个 Excel 文件！", "提示", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    CloseProcess("EXCEL");                                                               //关闭所有 Excel 进程
}
```

秘笈心法

心法领悟 477：使用文件流对 INI 文件进行读写。

本实例使用系统 API 函数实现了对 INI 文件进行读写的功能，但是开发人员还可以使用文件流来对 INI 文件进行读写，其中主要用到了 StreamReader 类和 StreamWrite 类。

实例 478	自动用递增变化的数据填充 Excel	高级
	光盘位置：光盘\MR\20\478	趣味指数：★★★★★

实例说明

本实例使用 C#代码实现了自动用递增变化的数据填充 Excel 的功能。运行本实例，如图 20.54 所示，首先单击上面工具栏中的“打开 Excel 文件”按钮，打开指定的 Excel 文件，并且选择要操作的工作表；然后在下面的工具栏中输入要填充的范围，并选择要填充的数据类型，单击“确定”按钮，程序即可自动使用指定的数据

填充 Excel 中指定的范围。

图 20.54　自动用递增变化的数据填充 Excel

注意：自动填充 Excel 文件中的指定范围时，一定要确保每个范围的第一个单元格中有相应的内容。例如，如果要使用“工作日”类型自动填充 Excel 工作表中的 A1～A10 范围，则 A1 单元格中必须有与“工作日”类型对应的数据，如 Monday 或 Mon 等。

关键技术

本实例实现用递增变化的数据自动填充 Excel 时，主要用到了 Microsoft Excel 自动化对象模型的 Range 对象的 AutoFill 方法，下面对其进行详细讲解。

AutoFill 方法用来自动填充 Excel 指定范围内的单元格。语法格式如下：

```
public Object AutoFill(Range Destination,XlAutoFillType Type)
```

参数说明

❶ Destination：要填充的单元格。

❷ Type：XlAutoFillType 枚举值之一，表示填充类型。XlAutoFillType 枚举的枚举值及说明如表 20.11 所示。

表 20.11　XlAutoFillType 枚举的枚举值及说明

枚 举 值	描　述
xlFillDays	按照一周的天的名称自动填充单元格
xlFillFormats	只复制格式从源区域到目标范围内
xlFillSeries	自动按照指定序列填充单元格（如“1，2”后面是“3，4，5……”延长）
xlFillWeekdays	按照一周的工作日填充单元格
xlGrowthTrend	扩大从源到目标范围内的数值范围，假设在源范围内的数字之间的关系是乘法（如“1，2”扩展为“4，8，16”）
xlFillCopy	从源复制到目标范围内的值的范围和形式
xlFillDefault	Excel 中确定的值，用来填充目标范围格式
xlFillMonths	按照每月的名称自动填充单元格
xlFillValues	从源范围复制值到目标范围内
xlFillYears	按照年份自动填充单元格
xlLinearTrend	扩大从源到目标范围内的数值范围，假设数字之间的关系是添加剂（如“1，2”扩展为“3，4，5”）

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 FillExcelByIncrementalData。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 ToolStrip 控件，分别用来作为

程序的两个工具栏，其中第一个工具栏用来选择 Excel 文件及其工作表，第二个工具栏用来输入填充范围、选择填充数据类型及执行 Excel 自动填充操作；添加一个 WebBrowser 控件，用来显示 Excel 文件。

（3）程序的主要代码如下：

```
private void tsbtn_Query_Click(object sender, EventArgs e)
{
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    string P_str_Excel = tstxt_Excel.Text;                          //记录 Excel 文件路径
    string P_str_SheetName = tscbox_Sheet.Text;                     //记录选择的工作表名称
    object P_obj_Start = tstxt_Start.Text;                          //记录开始单元格
    object P_obj_End = tstxt_End.Text;                              //记录结束单元格
    object missing = System.Reflection.Missing.Value;              //定义 object 默认值
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    //打开 Excel 文件
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Workbooks.Open(P_str_Excel, missing, missing, missing, missing, missing, missing,
missing, missing, missing, missing, missing, missing, missing, missing);
Microsoft.Office.Interop.Excel.Worksheet worksheet;                 //声明工作表
    //获取选择的工作表
    worksheet = ((Microsoft.Office.Interop.Excel.Worksheet)workbook.Worksheets[P_str_SheetName]);
    //定义查找范围
    Microsoft.Office.Interop.Excel.Range searchRange = worksheet.get_Range(P_obj_Start, missing);
    switch (tscbox_Type.Text)                                       //按照下拉列表中的文本进行条件判断
    {
        case "数字":                                                //以数字序列填充
            searchRange.AutoFill(worksheet.get_Range(P_obj_Start, P_obj_End), Microsoft.Office.Interop.Excel.XlAutoFillType.xlFillSeries);
            break;
        case "工作日":                                              //以工作日填充
            searchRange.AutoFill(worksheet.get_Range(P_obj_Start, P_obj_End), Microsoft.Office.Interop.Excel.XlAutoFillType.xlFillWeekdays);
            break;
        case "月份":                                                //以月份填充
            searchRange.AutoFill(worksheet.get_Range(P_obj_Start, P_obj_End), Microsoft.Office.Interop.Excel.XlAutoFillType.xlFillMonths);
            break;
        case "年份":                                                //以年份填充
            searchRange.AutoFill(worksheet.get_Range(P_obj_Start, P_obj_End), Microsoft.Office.Interop.Excel.XlAutoFillType.xlFillYears);
            break;
        default:                                                    //以默认方式填充
            searchRange.AutoFill(worksheet.get_Range(P_obj_Start, P_obj_End), Microsoft.Office.Interop.Excel.XlAutoFillType.xlFillDefault);
            break;
    }
    excel.DisplayAlerts = false;                                    //设置保存 Excel 时不显示对话框
    workbook.Save();                                                //保存工作簿
    workbook.Close(false, missing, missing);                        //关闭工作簿
    CloseProcess("EXCEL");                                          //关闭所有 Excel 进程
    WBrowser_Excel.Navigate(P_str_Excel);                           //在窗体中重新显示 Excel 文件内容
}
```

秘笈心法

心法领悟 478：Range 属性的妙用。

通过 Worksheet 对象的 get_Range 方法可以返回一个 Range 对象，如果 Range 方法中的参数是默认参数，此时的 Range 是 Excel 工作表中所有内容的范围，可以通过 Range.Text 属性获取或设置 Range 对象范围内的文本内容。

图形图像与多媒体篇

- 第 21 章　基本图形绘制
- 第 22 章　图像处理技术
- 第 23 章　常用图表应用
- 第 24 章　动画处理技术
- 第 25 章　音频与视频控制

第21章

基本图形绘制

- 绘制基本图形
- 绘制组合图形

21.1　绘制基本图形

实例 479	绘制直线	初级
	光盘位置：光盘\MR\21\479	趣味指数：★★★☆

实例说明

本实例演示如何在 C#中绘制直线，实例运行效果如图 21.1 所示。

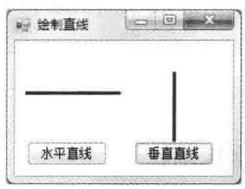


图 21.1　绘制直线

关键技术

本实例在绘制直线时主要用到了 Graphics 类中的 DrawLine 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制一条连接两个 Point 结构的线。

语法如下：

```
public void DrawLine (Pen pen,Point pt1,Point pt2)
```

参数说明

❶ pen：Pen 对象，它确定线条的颜色、宽度和样式。

❷ pt1：Point 结构，它表示要连接的第一个点。

❸ pt2：Point 结构，它表示要连接的第二个点。

（2）第二种用于绘制一条连接由坐标指定的两个点的线条。

语法如下：

```
public void DrawLine (Pen pen,int x1,int y1,int x2,int y2)
```

DrawLine 方法中的参数说明如表 21.1 所示。

表 21.1　DrawLine 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定线条的颜色、宽度和样式
x1	第一个点的 x 坐标
y1	第一个点的 y 坐标
x2	第二个点的 x 坐标
y2	第二个点的 y 坐标

说明：DrawLine 方法有多种重载形式，这里只介绍了它的两种最常用的重载形式。下面介绍绘制其他图形的方法时，它们也有多种重载形式，但限于本书的篇幅，遇到时都只介绍两种。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDLine。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来绘制横

向和纵向的直线。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Pen blackPen = new Pen(Color.Black, 3);                    //创建 Pen 对象
    Point point1 = new Point(10, 50);                          //创建一个 Point 对象
    Point point2 = new Point(100, 50);                         //再创建一个 Point 对象
    Graphics g = this.CreateGraphics();                        //创建一个 Graphics 对象
    g.DrawLine(blackPen, point1, point2);                      //调用 DrawLine 方法绘制直线
}
private void button2_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                 //创建 Graphics 对象
    Pen myPen = new Pen(Color.Black, 3);                       //创建 Pen 对象
    graphics.DrawLine(myPen, 150, 30, 150, 100);               //调用 DrawLine 方法绘制直线
}
```

秘笈心法

心法领悟 479：使用 DrawLine 方法绘制坐标轴。

一般在绘制坐标轴时，都需要用到 DrawLine 方法。

实例 480　绘制矩形

光盘位置：光盘\MR\21\480　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制矩形，实例运行效果如图 21.2 所示。

图 21.2　绘制矩形

关键技术

本实例在绘制矩形时主要用到了 Graphics 类中的 DrawRectangle 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制由 Rectangle 结构指定的矩形。

语法如下：

```
public void DrawRectangle (Pen pen,Rectangle rect)
```

参数说明

❶ pen：Pen 对象，它确定矩形的颜色、宽度和样式。

❷ rect：表示要绘制矩形的 Rectangle 结构。

（2）第二种用于绘制由坐标对、宽度和高度指定的矩形。

语法如下：

```
public void DrawRectangle (Pen pen,int x,int y,int width,int height)
```

DrawRectangle 方法中的参数说明如表 21.2 所示。

表 21.2　DrawRectangle 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定矩形的颜色、宽度和样式
x	要绘制矩形的左上角的 x 坐标
y	要绘制矩形的左上角的 y 坐标
width	要绘制矩形的宽度
height	要绘制矩形的高度

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDRectangle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawRectangle 方法绘制一个线条宽度为 8、颜色为蓝色的矩形。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //声明一个 Graphics 对象
    Pen myPen = new Pen(Color.Blue, 8);                           //创建 Pen 对象
    graphics.DrawRectangle(myPen, 10, 10, 150, 100);              //调用 Graphics 对象的 DrawRectangle 方法
}
```

秘笈心法

心法领悟 480：使用 DrawRectangle 方法绘制正方形。

使用 DrawRectangle 方法绘制矩形时，如果指定的矩形的宽度和高度相同，则可以绘制一个正方形。

实例 481	绘制正方形	初级
	光盘位置：光盘\MR\21\481	趣味指数：★★★☆

实例说明

本实例演示如何在 C#中绘制正方形，实例运行效果如图 21.3 所示。

图 21.3　绘制正方形

关键技术

本实例在绘制正方形时，主要是通过将 Graphics 类中的 DrawRectangle 方法的宽和高两个参数设置为相同的值来实现的。

说明：关于 Graphics 类中的 DrawRectangle 方法的详细讲解请参见实例 480 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawSquare。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawRectangle 方法绘制一个线条宽度为 8、颜色为蓝色的正方形。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //声明一个 Graphics 对象
    Pen myPen = new Pen(Color.Blue, 8);                           //创建 Pen 对象
    graphics.DrawRectangle(myPen, 30, 10, 100, 100);              //调用 Graphics 对象的 DrawRectangle 方法
}
```

秘笈心法

心法领悟 481：使用 DrawRectangle 方法绘制柱形图。

在实际项目中绘制柱形图表时，可以使用 DrawRectangle 方法绘制柱形图。

实例 482 绘制椭圆

光盘位置：光盘\MR\21\482　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制椭圆，实例运行效果如图 21.4 所示。

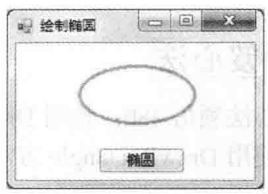


图 21.4　绘制椭圆

关键技术

本实例在绘制椭圆时主要用到了 Graphics 类中的 DrawEllipse 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制边界由 Rectangle 结构指定的椭圆。

语法如下：

```
public void DrawEllipse (Pen pen,Rectangle rect)
```

参数说明

❶ pen：Pen 对象，它确定曲线的颜色、宽度和样式。

❷ rect：Rectangle 结构，它定义椭圆的边界。

（2）第二种用于绘制一个由边框（该边框由一对坐标、高度和宽度指定）指定的椭圆。

语法如下：

```
public void DrawEllipse (Pen pen,int x,int y,int width,int height)
```

DrawEllipse 方法中的参数说明如表 21.3 所示。

表 21.3　DrawEllipse 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定曲线的颜色、宽度和样式
x	定义椭圆边框的左上角的 x 坐标
y	定义椭圆边框的左上角的 y 坐标
width	定义椭圆边框的宽度
height	定义椭圆边框的高度

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDEllipse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawEllipse 方法绘制一个线条宽度为 3 的橘黄色椭圆。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //创建 Graphics 对象
    Pen myPen = new Pen(Color.DarkOrange, 3);                     //创建 Pen 对象
    graphics.DrawEllipse(myPen, 55, 20, 100, 50);                 //绘制椭圆
}
```

秘笈心法

心法领悟 482：使用 DrawEllipse 方法绘制圆形。

使用 DrawEllipse 方法绘制椭圆时，如果指定的椭圆的宽度和高度相同，则可以绘制一个圆形。

实例 483　绘制圆弧

光盘位置：光盘\MR\21\483　　初级　　趣味指数：★★★☆

实例说明

本实例演示如何在 C#中绘制圆弧，实例运行效果如图 21.5 所示。

图 21.5　绘制圆弧

关键技术

本实例在绘制圆弧时主要用到了 Graphics 类中的 DrawArc 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制一段弧线，它表示由 Rectangle 结构指定的椭圆的一部分。

语法如下：

```
public void DrawArc (Pen pen,Rectangle rect,float startAngle,float sweepAngle)
```

DrawArc 方法中的参数说明如表 21.4 所示。

表 21.4　DrawArc 方法中的参数说明（一）

参　数	说　明
pen	Pen 对象，它确定弧线的颜色、宽度和样式
rect	Rectangle 结构，它定义椭圆的边界
startAngle	从 x 轴到弧线的起始点沿顺时针方向度量的角（以度为单位）
sweepAngle	从 startAngle 参数到弧线的结束点沿顺时针方向度量的角（以度为单位）

（2）第二种用于绘制一段弧线，它表示由一对坐标、宽度和高度指定的椭圆部分。

语法如下：

```
public void DrawArc (Pen pen,int x,int y,int width,int height,int startAngle,int sweepAngle)
```

DrawArc 方法中的参数说明如表 21.5 所示。

表 21.5　DrawArc 方法中的参数说明（二）

参　数	说　明
pen	Pen 对象，它确定弧线的颜色、宽度和样式
x	定义椭圆边框的左上角的 x 坐标
y	定义椭圆边框的左上角的 y 坐标
width	定义椭圆边框的宽度
height	定义椭圆边框的高度
startAngle	从 x 轴到弧线的起始点沿顺时针方向度量的角（以度为单位）
sweepAngle	从 startAngle 参数到弧线的结束点沿顺时针方向度量的角（以度为单位）

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDArc。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawArc 方法绘制一条线条宽度为 3 的暗红色圆弧。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics ghs = this.CreateGraphics();                    //创建 Graphics 对象
```

```
    Pen myPen = new Pen(Color.DarkRed, 3);                    //创建 Pen 对象
    Rectangle myRectangle = new Rectangle(50, 20, 100, 60);   //定义一个 Rectangle 结构
    ghs.DrawArc(myPen, myRectangle, 180, 180);                //绘制圆弧
}
```

秘笈心法

心法领悟 483：Rectangle 结构的使用。

Rectangle 结构用来存储一组整数，共 4 个，表示一个矩形的位置和大小，该结构位于 System.Drawing 命名空间下。

实例 484　绘制指定角度的扇形

光盘位置：光盘\MR\21\484　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制一个指定角度的扇形，实例运行效果如图 21.6 所示。

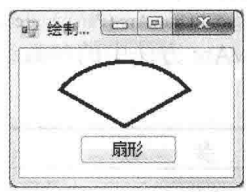


图 21.6　绘制指定角度的扇形

关键技术

本实例在绘制扇形时主要用到了 Graphics 类中的 DrawPie 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制由一个 Rectangle 结构和两条射线所指定的椭圆定义的扇形。

语法如下：

```
public void DrawPie(Pen pen,Rectangle rect,float startAngle,float sweepAngle)
```

DrawPie 方法中的参数说明如表 21.6 所示。

表 21.6　DrawPie 方法中的参数说明（一）

参　数	说　明
pen	Pen，它确定扇形的颜色、宽度和样式
rect	Rectangle 结构，它表示定义该扇形所属的椭圆的边框
startAngle	从 x 轴到扇形的第一条边沿顺时针方向度量的角（以度为单位）
sweepAngle	从 startAngle 参数到扇形的第二条边沿顺时针方向度量的角（以度为单位）

（2）第二种用于绘制一个扇形，该形状由一个坐标对、宽度、高度以及两条射线所指定的椭圆定义。

语法如下：

```
public void DrawPie(Pen pen,int x,int y,int width,int height,int startAngle,int sweepAngle)
```

DrawPie 方法中的参数说明如表 21.7 所示。

表 21.7　DrawPie 方法中的参数说明（二）

参　数	说　明
pen	Pen，它确定扇形的颜色、宽度和样式
x	边框的左上角的 x 坐标，该边框定义扇形所属的椭圆
y	边框的左上角的 y 坐标，该边框定义扇形所属的椭圆
width	边框的宽度，该边框定义扇形所属的椭圆
height	边框的高度，该边框定义扇形所属的椭圆
startAngle	从 x 轴到扇形的第一条边沿顺时针方向度量的角（以度为单位）
sweepAngle	从 startAngle 参数到扇形的第二条边沿顺时针方向度量的角（以度为单位）

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDPie。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawPie 方法绘制一个线条宽度为 3 的 120° 黑色扇形。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics ghs = this.CreateGraphics();                    //创建 Graphics 对象
    Pen mypen = new Pen(Color.Black, 3);                     //创建 Pen 对象
    ghs.DrawPie(mypen, 20, 10, 120, 100, 210, 120);          //绘制扇形
}
```

秘笈心法

心法领悟 484：Pen 类的使用。

Pen 类定义用于绘制直线和曲线的对象，使用其 DashStyle 属性可以绘制几种虚线。

实例 485　绘制贝塞尔曲线

光盘位置：光盘\MR\21\485　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制贝塞尔曲线，实例运行效果如图 21.7 所示。

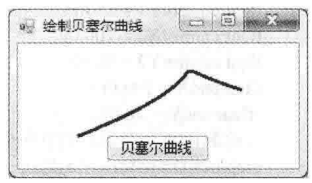


图 21.7　绘制贝塞尔曲线

关键技术

本实例在绘制贝塞尔曲线时主要用到了 Graphics 类中的 DrawBezier 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制由 4 个 Point 结构定义的贝塞尔曲线。

语法如下：

```
public void DrawBezier (Pen pen,Point pt1,Point pt2,Point pt3,Point pt4)
```

DrawBezier 方法中的参数说明如表 21.8 所示。

表 21.8　DrawBezier 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定曲线的颜色、宽度和样式
pt1	Point 结构，它表示曲线的起始点
pt2	Point 结构，它表示曲线的第一个控制点
pt3	Point 结构，它表示曲线的第二个控制点
pt4	Point 结构，它表示曲线的结束点

（2）第二种用于绘制由 4 个点的有序坐标对定义的贝塞尔曲线。

语法如下：

```
public void DrawBezier (Pen pen,float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
```

DrawRectangle 方法中的参数说明如表 21.9 所示。

表 21.9　DrawRectangle 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定曲线的颜色、宽度和样式
x1	曲线起始点的 x 坐标

续表

参　数	说　明
y1	曲线起始点的 y 坐标
x2	曲线的第一个控制点的 x 坐标
y2	曲线的第一个控制点的 y 坐标
x3	曲线的第二个控制点的 x 坐标
y3	曲线的第二个控制点的 y 坐标
x4	曲线的结束点的 x 坐标
y4	曲线的结束点的 y 坐标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDBezier。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawBezier 方法绘制一段贝塞尔曲线。

（3）程序的主要代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //创建一个 GDI+绘图图面对象
    Pen myPen = new Pen(Color.Blue, 3);                           //创建一个用于绘制直线和曲线的对象
    float startX = 50.0F;                                         //初始化起始点的 x 坐标
    float startY = 80.0F;                                         //初始化起始点的 y 坐标
    float controlX1 = 200.0F;                                     //初始化第一个控制点的 x 坐标
    float controlY1 = 21.0F;                                      //初始化第一个控制点的 y 坐标
    float controlX2 = 100.0F;                                     //初始化第二个控制点的 x 坐标
    float controlY2 = 10.0F;                                      //初始化第二个控制点的 y 坐标
    float endX = 190.0F;                                          //初始化结束点的 x 坐标
     float endY = 40.0F;                                          //初始化结束点的 y 坐标
     //绘制由 4 个表示点的有序坐标对定义的贝塞尔样条
    graphics.DrawBezier(myPen, startX, startY, controlX1, controlY1, controlX2, controlY2, endX, endY);
}
```

秘笈心法

心法领悟 485：使用 DrawBezier 方法绘制波形图。

绘制波形图时，可以根据实际情况，使用不同的贝塞尔曲线来组成。

实例 486　绘制多边形

光盘位置：光盘\MR\21\486　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制多边形，实例运行效果如图 21.8 所示。

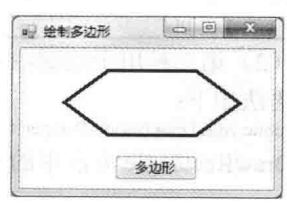


图 21.8　绘制多边形

关键技术

本实例在绘制多边形时主要用到了 Graphics 类中的 DrawPolygon 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制由一组 PointF 结构定义的多边形。

语法如下：

```
public void DrawPolygon (Pen pen,PointF[] points)
```

参数说明

❶ pen：Pen 对象，它确定多边形的颜色、宽度和样式。

❷ points：PointF 结构数组，这些结构表示多边形的顶点。

（2）第二种用于绘制由一组 Point 结构定义的多边形。

语法如下：

```
public void DrawPolygon (Pen pen,Point[] points)
```

参数说明

❶ pen：Pen 对象，用于确定多边形的颜色、宽度和样式。

❷ points：Point 结构数组，这些结构表示多边形的顶点。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDPolygon。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawPolygon 方法绘制一个线条宽度为 3 的黑色多边形。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics ghs = this.CreateGraphics();                          //创建 Graphics 对象
    Pen myPen = new Pen(Color.Black, 3);                           //创建 Pen 对象
    Point point1 = new Point(80, 20);                              //创建 Point 对象，表示第一个点
    Point point2 = new Point(40, 50);                              //创建 Point 对象，表示第二个点
    Point point3 = new Point(80, 80);                              //创建 Point 对象，表示第三个点
    Point point4 = new Point(160, 80);                             //创建 Point 对象，表示第 4 个点
    Point point5 = new Point(200, 50);                             //创建 Point 对象，表示第 5 个点
    Point point6 = new Point(160, 20);                             //创建 Point 对象，表示第 6 个点
    Point[] myPoints ={ point1, point2, point3, point4, point5, point6 };  //创建 Point 结构数组
    ghs.DrawPolygon(myPen, myPoints);                              //调用 Graphics 对象的 DrawPolygon 方法绘制一个多边形
}
```

秘笈心法

心法领悟 486：多边形在雷达图中的应用。

对于现在比较流行的一个雷达图，就可以根据需要，绘制不同边的多边形，例如，一个学生有 6 门课程，就可以绘制一个 6 个边的雷达图，表示该学生各科的成绩，及其与班级平均分的对比。

实例 487　绘制文本

光盘位置：光盘\MR\21\487　　初级　　趣味指数：★★★★☆

实例说明

本实例演示如何在 C#中绘制文本，实例运行效果如图 21.9 所示。

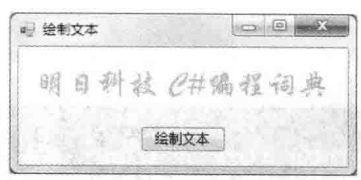


图 21.9　绘制文本

关键技术

本实例在绘制文本时主要用到了 Graphics 类中的 DrawString 方法，该方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于在指定矩形并且用指定的 Brush 和 Font 对象绘制指定的文本字符串。

语法如下：

```
public void DrawString(string s,Font font,Brush brush,RectangleF layoutRectangle)
```

DrawString 方法中的参数说明如表 21.10 所示。

表 21.10　DrawString 方法中的参数说明（一）

参　数	说　明
s	要绘制的字符串
font	定义字符串的文本格式
brush	确定所绘制文本的颜色和纹理
layoutRectangle	RectangleF 结构，它指定所绘制文本的位置

（2）第二种用于在指定位置并且用指定的 Brush 和 Font 对象绘制指定的文本字符串。

语法如下：

```
public void DrawString(string s,Font font,Brush brush,float x,float y)
```

DrawString 方法中的参数说明如表 21.11 所示。

表 21.11　DrawString 方法中的参数说明（二）

参　数	说　明
s	要绘制的字符串
font	Font，它定义字符串的文本格式
brush	Brush，它确定所绘制文本的颜色和纹理
x	所绘制文本的左上角的 x 坐标
y	所绘制文本的左上角的 y 坐标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UseDString。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用于调用 Graphics 类中的 DrawString 方法绘制一段文本字符串。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string str = "明日科技 C#编程词典";                          //定义绘制的字符串
    Font myFont = new Font("华文行楷", 20);                      //创建 Font 对象
    SolidBrush myBrush = new SolidBrush(Color.DarkOrange);       //创建画刷对象
    Graphics myGraphics = this.CreateGraphics();                 /创建 Graphics 对象
    myGraphics.DrawString(str, myFont, myBrush,10,20);           //绘制文本
}
```

秘笈心法

心法领悟 487：水印文字的实现。

对于一些有可能侵权的图片，或者要作为宣传的图片，可以通过使用 DrawString 方法将用到的文字绘制到图片上，从而得到所谓的“水印文字”。

21.2　绘制组合图形

实例 488　简单画图程序　　中级

光盘位置：光盘\MR\21\488　　趣味指数：★★★☆

实例说明

画图工具是大家经常用到的，那么，如何在程序中通过鼠标的单击和移动绘制线条和图形呢？通过本实例的

介绍，读者将了解如何在窗体上绘制各种图形，如矩形、椭圆、线条和文字等。实例运行效果如图 21.10 所示。

图 21.10　简单画图程序

关键技术

本实例在制作画图程序时，主要用到了 Graphics 类的 DrawLine、DrawRectangle、DrawEllipse 和 DrawString 方法。

说明：（1）关于 DrawLine 方法的详细讲解请参见实例 479 中的关键技术。

（2）关于 DrawRectangle 方法的详细讲解请参见实例 480 中的关键技术。

（3）关于 DrawEllipse 方法的详细讲解请参见实例 482 中的关键技术。

（4）关于 DrawString 方法的详细讲解请参见实例 487 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawTool。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 ToolStrip 控件，用来作为窗体的工具栏。

（3）在项目中添加一个新的 Windows 窗体，并将其命名为 Frm_Text.cs，用来作为文字输入窗体；在该窗体中添加一个 TextBox 控件，用来输入文本。

（4）程序的主要代码如下。

menuStrip1 菜单栏的“打开”菜单项的 Click 事件用于打开“打开文件”对话框，并选择相应的图片，将图片绘制在窗体上。代码如下：

```
private void 打开 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "Image Files(*.bmp;*.wmf;*.ico;*.cur;*.jgp)|*.bmp;*.wmf;*.ico;*.cur;*.jpg";
    openFileDialog1.Multiselect = false;                    //只能选择单个文件
    if (openFileDialog1.ShowDialog( ) == DialogResult.OK)   //打开文件对话框
    {
        this.Text = "MyDraw\t" + openFileDialog1.FileName;  //修改窗口标题
        editFileName = openFileDialog1.FileName;            //获取打开文件的路径
        theImage = Image.FromFile(openFileDialog1.FileName); //根据文件的路径创建 Image 对象
        Graphics g = this.CreateGraphics( );                //创建窗体的 Graphics 对象
        g.DrawImage(theImage, this.ClientRectangle);        //在窗体上绘制图片
        ig = Graphics.FromImage(theImage);                  //创建 Graphics 对象
        ig.DrawImage(theImage, this.ClientRectangle);       //在窗体上绘制图片
        toolStrip1.Enabled = true;                          //该控件可用
    }
}
```

menuStrip1 菜单栏的“新建”菜单项的 Click 事件主要用白色清除窗体的背景，从而实现“文件新建”功能。代码如下：

```
private void 新建 ToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    Graphics g = this.CreateGraphics( );                                    //创建窗体的 Graphics 对象
    g.Clear(backColor);                                                     //以指定的颜色清除
    toolStrip1.Enabled = true;                                              //该命令项可用
    theImage = new Bitmap(this.ClientRectangle.Width, this.ClientRectangle.Height);  //创建一个 Bitmap 对象
    editFileName = "新建文件";
this.Text = "MyDraw\t" + editFileName;                                      //修改窗口标题
    ig = Graphics.FromImage(theImage);                                      //创建 Graphics 对象
    ig.Clear(backColor);                                                    //以指定的颜色清除
}
```

menuStrip1 菜单栏的“保存”菜单项的 Click 事件用于将窗体背景保存为 BMP 格式的图片。代码如下：

```
private void 保存 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "图像(*.bmp)|*.bmp";                           //设置保存图片的类型
    saveFileDialog1.FileName = editFileName;                                //设置保存图片的名称
    if (saveFileDialog1.ShowDialog( ) == DialogResult.OK)                   //打开“另存为”对话框
    {
        theImage.Save(saveFileDialog1.FileName, ImageFormat.Bmp);           //将图片另存为
        this.Text = "MyDraw\t" + saveFileDialog1.FileName;                  //显示另存为图片的路径
        editFileName = saveFileDialog1.FileName;
    }
}
```

menuStrip1 菜单栏的“颜色”菜单项的 Click 事件用于打开“颜色”对话框，以选择画笔的颜色。代码如下：

```
private void 颜色 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog( ) == DialogResult.OK)                      //打开“颜色”对话框
    {
        foreColor = colorDialog1.Color;                                     //获取画笔的颜色
    }
}
```

在窗体的 MouseDown 事件中，如果当前绘制的是字符串，在鼠标的当前位置显示文本框，在文本框中输入要绘制的字符串，并将字符串绘在窗体的指定位置；如果绘制的是图形，设置图形的起始位置。代码如下：

```
private void Frm_Main_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)                                      //如果按下的是鼠标左键
    {
        if (drawTool == drawTools.String)                                   //如果选择文字输入，则打开 strInput 窗体
        {
            text inputBox = new text( );                                    //创建 text 对象
            inputBox.StartPosition = FormStartPosition.CenterParent;        //窗体居中
            if (inputBox.ShowDialog( ) == DialogResult.OK)                  //如果对话框的返回值为 OK
            {
                Graphics g = this.CreateGraphics( );                        //创建窗体的 Graphics 对象
                Font theFont = this.Font;                                   //获取当前窗体的文本样式
                g.DrawString(inputBox.textBox1.Text, theFont, new SolidBrush(foreColor), e.X, e.Y);//绘制字符串
                ig.DrawString(inputBox.textBox1.Text, theFont, new SolidBrush(foreColor), e.X, e.Y);
            }
        }
        else if ((isDrawing = !isDrawing) == true)                          //如果开始绘制，则记录鼠标位置
        {
            startPoint = new Point(e.X, e.Y);                               //获取鼠标的起始位置
            oldPoint = new Point(e.X, e.Y);
        }
    }
}
```

在窗体的 MouseMove 事件中，根据鼠标的移动大小绘制指定的图形。代码如下：

```
private void Frm_Main_MouseMove(object sender, MouseEventArgs e)
{
    Graphics g;
    g = this.CreateGraphics( );
    if (isDrawing)
    {
        switch (drawTool)
        {
            case drawTools.None:
```

```
                break;
            case drawTools.Pen:
                //从上一个点到当前点绘制线段
                g.DrawLine(new Pen(foreColor, 1), oldPoint, new Point(e.X, e.Y));
                ig.DrawLine(new Pen(foreColor, 1), oldPoint, new Point(e.X, e.Y));
                oldPoint.X = e.X;
                oldPoint.Y = e.Y;
                break;
            case drawTools.Line:
                //首先恢复此次操作之前的图像，然后再添加 Line
                this. Frm_Main_Paint(this, new PaintEventArgs(this.CreateGraphics( ), this.ClientRectangle));
                g.DrawLine(new Pen(foreColor, 1), startPoint, new Point(e.X, e.Y));
                break;
            case drawTools.Ellipse:
                //首先恢复此次操作之前的图像，然后再添加 Ellipse
                this. Frm_Main_Paint(this, new PaintEventArgs(this.CreateGraphics( ), this.ClientRectangle));
                g.DrawEllipse(new Pen(foreColor, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
                break;
            case drawTools.Rectangle:
                //首先恢复此次操作之前的图像，然后再添加 Rectangle
                this. Frm_Main_Paint(this, new PaintEventArgs(this.CreateGraphics( ), this.ClientRectangle));
                g.DrawRectangle(new Pen(foreColor, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
                break;
            case drawTools.String:
                break;
            case drawTools.Rubber:
                //用背景色绘制宽线段
                g.DrawLine(new Pen(backColor, 20), oldPoint, new Point(e.X, e.Y));
                ig.DrawLine(new Pen(backColor, 20), oldPoint, new Point(e.X, e.Y));
                oldPoint.X = e.X;
                oldPoint.Y = e.Y;
                break;
        }
    }
}
```

在窗体的 MouseUp 事件中，根据用户选择的画笔，绘制直线、椭圆或者矩形等指定的图形。代码如下：

```
private void Frm_Main_MouseUp(object sender, MouseEventArgs e)
{
    isDrawing = false;
    switch (drawTool)
    {
        case drawTools.Line:                                    //绘制直线
            ig.DrawLine(new Pen(foreColor, 1), startPoint, new Point(e.X, e.Y));
            break;
        case drawTools.Ellipse:                                 //绘制椭圆
            ig.DrawEllipse(new Pen(foreColor, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
            break;
        case drawTools.Rectangle:                               //绘制矩形
            ig.DrawRectangle(new Pen(foreColor, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
            break;
    }
}
```

秘笈心法

心法领悟 488：创建 Graphics 对象的 3 种方法。

创建 Graphics 对象有以下 3 种方法：

- 在窗体或控件的 Paint 事件中创建，将其作为 PaintEventArgs 的一部分。
- 调用控件或窗体的 CreateGraphics 方法以获取对 Graphics 对象的引用，该对象表示控件或窗体的绘图画面。
- 由从 Image 继承的任何对象创建 Graphics 对象。

实例 489 绘制公章

光盘位置：光盘\MR\21\489

中级

趣味指数：★★★☆

实例说明

在现代化企业中，公章的应用是非常普遍的，它代表了一个企业的身份，本实例将用 GDI+技术制作一个简单的公章。实例运行效果如图 21.11 所示。

图 21.11 绘制公章

关键技术

本实例实现时主要用到了 Graphics 类的 DrawString、DrawRectangle 和 MeasureString 方法，它们分别用于在画布中绘制字符串、椭圆，以及对字符串进行测量。下面对本实例中用到的关键技术进行详细讲解。

MeasureString 方法用于对指定的字符串进行测量。语法格式如下：

```
public SizeF MeasureString (string text,Font font)
```

参数说明

❶ text：要测量的字符串。

❷ font：Font，它定义字符串的文本格式。

说明：（1）关于 DrawString 方法的详细讲解请参见实例 487 中的关键技术。

（2）关于 DrawRectangle 方法的详细讲解请参见实例 480 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Cachet。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来在窗体上绘制公章。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int tem_Line = 0;                                                    //记录圆的直径
    int circularity_W = 4;                                               //设置圆画笔的粗细
    if (panel1.Width >= panel1.Height)                                   //如果 panel1 控件的宽度大于等于高度
        tem_Line = panel1.Height;                                        //设置高度为圆的直径
    else
        tem_Line = panel1.Width;                                         //设置宽度为圆的直径
    //设置圆的绘制区域
    rect = new Rectangle(circularity_W, circularity_W, tem_Line - circularity_W * 2, tem_Line - circularity_W * 2);
    Font star_Font = new Font("Arial", 30, FontStyle.Regular);           //设置星号的字体样式
    string star_Str = "★";
    Graphics g = this.panel1.CreateGraphics();                           //创建 Graphics 对象
    g.SmoothingMode = SmoothingMode.AntiAlias;                           //消除绘制图形的锯齿
    g.Clear(Color.White);                                                //以白色清空 panel1 控件的背景
    Pen myPen = new Pen(Color.Red, circularity_W);                       //设置画笔的颜色
    g.DrawEllipse(myPen, rect);                                          //绘制圆
    SizeF Var_Size = new SizeF(rect.Width, rect.Width);                  //创建 SizeF 对象
    Var_Size = g.MeasureString(star_Str, star_Font);                     //对指定字符串进行测量
    g.DrawString(star_Str, star_Font, myPen.Brush, new PointF((rect.Width / 2F) + circularity_W - Var_Size.Width / 2F, rect.Height / 2F -
Var_Size.Width / 2F));                                                   //在指定的位置绘制星号
    Var_Size = g.MeasureString("专用章", Var_Font);                      //对指定字符串进行测量
    g.DrawString("专用章", Var_Font, myPen.Brush, new PointF((rect.Width / 2F) + circularity_W - Var_Size.Width / 2F, rect.Height / 2F +
Var_Size.Height * 2));                                                   //绘制文字
    string tempStr = "吉林省明日科技有限公司";
```

```
    int len = tempStr.Length;                                          //获取字符串的长度
    float angle = 180 + (180 - len * 20) / 2;                          //设置文字的旋转角度
    for (int i = 0; i < len; i++)                                      //将文字以指定的弧度进行绘制
    {
        //将指定的平移添加到 g 的变换矩阵前
        g.TranslateTransform((tem_Line + circularity_W / 2) / 2, (tem_Line + circularity_W / 2) / 2);
        g.RotateTransform(angle);                                      //将指定的旋转用于 g 的变换矩阵
        Brush myBrush = Brushes.Red;                                   //定义画刷
        g.DrawString(tempStr.Substring(i, 1), Var_Font, myBrush, 60, 0);  //显示旋转文字
        g.ResetTransform();                                            //将 g 的全局变换矩阵重置为单位矩阵
        angle += 20;                                                   //设置下一个文字的角度
    }
}
```

秘笈心法

心法领悟 489：SmoothingMode 枚举的使用。

SmoothingMode 枚举主要用于指定是否将平滑处理（消除锯齿）应用于直线、曲线和已填充区域的边缘，它一般用来消除图像的锯齿。

实例 490　波形图的绘制

光盘位置：光盘\MR\21\490　　中级　　趣味指数：★★★★☆

实例说明

波形图是一种特殊的图形，是按照特定的规律绘制出的曲线，在许多工程或有关计算方面的软件中都会用到这类图形。本实例设计了一个波形图绘图软件，运行程序，单击“绘图”按钮即可在窗体中绘制出一段波形图。实例运行效果如图 21.12 所示。

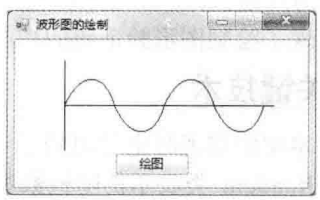


图 21.12　波形图的绘制

关键技术

本实例在绘制波形图时主要用到了 Graphics 类的 DrawBezier 方法。

说明：关于 DrawBezier 方法的详细讲解请参见实例 485 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WaveImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来在窗体上绘制一段波形图。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics( );                        //创建窗体的 Graphics 对象
    Pen myPen=new Pen(Color.Black, 1);                                 //设置画笔
    int beginX = 50;                                                   //定义变量
    int beginY = 65;
    int height = 35;
    int width = 50;
    Point pointX1 = new Point(beginX, beginY);
    Point pointY1 = new Point(beginX + 210, beginY);
    Point pointX2 = new Point(beginX, beginY - 45);
    Point pointY2 = new Point(beginX, beginY + 45);
    //调用 DrawLine 方法绘制两条垂直相交的直线，用来作为波形图的横纵坐标
    graphics.DrawLine(myPen, pointX1, pointY1);
    graphics.DrawLine(myPen, pointX2, pointY2);
    //绘制上半区域交错连接的贝塞尔曲线
```

```
    graphics.DrawBezier(myPen, beginX, beginY, beginX + 15, beginY - height, beginX + 40, beginY - height, beginX + width, beginY);
    graphics.DrawBezier(myPen, beginX + width, beginY, beginX + width + 15, beginY + height,  beginX + width + 40, beginY + height, beginX +
width * 2, beginY);                                          //绘制下半区域交错连接的贝塞尔曲线
    graphics.DrawBezier(myPen, beginX + width * 2, beginY, beginX + width * 2 + 15, beginY - height,  beginX + width * 2 + 40, beginY - height,
beginX + width * 3, beginY);                                 //绘制上半区域交错连接的贝塞尔曲线
    graphics.DrawBezier(myPen, beginX + width * 3, beginY, beginX + width * 3 + 15, beginY + height,  beginX + width * 3 + 40, beginY + height,
beginX + width * 4, beginY);                                 //绘制下半区域交错连接的贝塞尔曲线
}
```

秘笈心法

心法领悟 490：调用窗体的 CreateGraphics 方法创建 Graphics 对象。

在程序中绘图时，可以调用窗体的 CreateGraphics 方法来获取对 Graphics 对象的引用，该对象表示窗体的绘图画面。例如，本实例中创建 Graphics 对象的代码如下：

```
Graphics graphics = this.CreateGraphics( );                  //创建窗体的 Graphics 对象
```

实例 491　绘制图形验证码

光盘位置：光盘\MR\21\491　　中级　　趣味指数：★★★★★

实例说明

经常上网的读者对验证码一定不陌生，每当用户注册或登录一个网络程序时大多数情况下都要求输入指定位数的验证码，所有信息经过验证无误时方可进入系统，那么如何生成类似的图形验证码呢？本实例使用 C#语言实现了绘制图形验证码的功能，实例运行效果如图 21.13 所示。

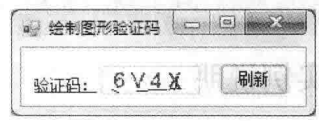


图 21.13　绘制图形验证码

关键技术

本实例实现时主要用到了 Random 的 Next 方法，下面对其进行详细讲解。

Random 类表示伪随机数生成器，它是一种能够产生满足某些随机性统计要求的数字序列的设备，其 Next 方法用来返回一个指定范围内的随机数，该方法的语法格式如下：

```
public virtual int Next(int minValue,int maxValue)
```

参数说明

❶ minValue：返回的随机数的下界。

❷ maxValue：返回的随机数的上界。maxValue 必须大于等于 minValue。

❸ 返回值：一个大于等于 minValue 且小于 maxValue 的 32 位带符号整数。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawValidateCode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图形验证码；添加一个 Button 控件，用来生成图形验证码。

（3）程序的主要代码如下：

```
private string CheckCode()                                   //此方法生成
{
    int number;
    char code;
    string checkCode = String.Empty;                         //声明变量存储随机生成的 4 位英文或数字
    Random random = new Random();                            //生成随机数
    for (int i = 0; i < 4; i++)
    {
        number = random.Next();                              //返回非负随机数
        if (number % 2 == 0)                                 //判断数字是否为偶数
            code = (char)('0' + (char)(number % 10));
        else                                                 //如果不是偶数
```

```
                code = (char)('A' + (char)(number % 26));
            checkCode += " " + code.ToString();                    //累加字符串
        }
        return checkCode;                                          //返回生成的字符串
    }
    private void CodeImage(string checkCode)
    {
        if (checkCode == null || checkCode.Trim() == String.Empty)
            return;
        System.Drawing.Bitmap image = new System.Drawing.Bitmap((int)Math.Ceiling((checkCode.Length * 9.5)), 22);
        Graphics g = Graphics.FromImage(image);                    //创建 Graphics 对象
        try
        {
            Random random = new Random();                          //生成随机生成器
            g.Clear(Color.White);                                  //清空图片背景色
            for (int i = 0; i < 3; i++)                            //画图片的背景噪音线
            {
                int x1 = random.Next(image.Width);
                int x2 = random.Next(image.Width);
                int y1 = random.Next(image.Height);
                int y2 = random.Next(image.Height);
                g.DrawLine(new Pen(Color.Black), x1, y1, x2, y2);
            }
            Font font = new System.Drawing.Font("Arial", 12, (System.Drawing.FontStyle.Bold));
            g.DrawString(checkCode, font, new SolidBrush(Color.Red), 2, 2);
            for (int i = 0; i < 150; i++)                          //画图片的前景噪音点
            {
                int x = random.Next(image.Width);
                int y = random.Next(image.Height);
                image.SetPixel(x, y, Color.FromArgb(random.Next()));
            }
            //画图片的边框线
            g.DrawRectangle(new Pen(Color.Silver), 0, 0, image.Width - 1, image.Height - 1);
            this.pictureBox1.Width = image.Width;                  //设置 PictureBox 的宽度
            this.pictureBox1.Height = image.Height;                //设置 PictureBox 的高度
            this.pictureBox1.BackgroundImage = image;              //设置 PictureBox 的背景图像
        }
        catch
        { }
    }
```

秘笈心法

心法领悟 491：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加载图像，或者使用 LoadAsync 方法异步加载图像。

实例 492	绘制中文验证码	高级
	光盘位置：光盘\MR\21\492	趣味指数：★★★★★

实例说明

上网浏览网页时，有些网站提供会员注册或者发布帖子的功能。在注册或者发布帖子时很多都是只有输入正确验证码后才能提交数据。验证码主要用于防止恶意提交数据，验证码大体可分为数字验证码、数字与字母混合、字母验证码以及中文验证码。本实例可以随机生成中文验证码，并且可以验证输入的验证码是否正确。实例运行效果如图 21.14 所示。

图 21.14　绘制中文验证码

关键技术

本实例在实现时主要用到了 Random 类和 Encoding 类，它们分别用于生成随机区位码和根据区位码获取相应的汉字。下面对本实例中用到的关键技术进行介绍。

Encoding 类表示字符编码，在此类中重点讲解 GetString 方法，此方法将指定字节数组中的所有字节解码为一个字符串。语法格式如下：

```
public virtual string GetString(byte[] bytes)
```

参数说明

❶ bytes：包含要解码的字节序列的字节数组。

❷ 返回值：包含指定字节序列解码结果的字符串。

说明：关于 Random 类的详细讲解请参见实例 491 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChineseCode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示生成的中文验证码；添加一个 TextBox 控件，用来输入验证码；添加两个 Button 控件，分别用来生成验证码和判断输入的验证码是否正确。

（3）程序的主要代码如下：

```
private void CreateImage()
{
    Encoding gb = Encoding.GetEncoding("gb2312");                                   //获取 GB2312 编码页（表）
    object[] bytes = CreateCode(4);                                                 //调用函数产生 4 个随机中文汉字编码
    //根据汉字编码的字节数组解码出中文汉字
    string str1 = gb.GetString((byte[])Convert.ChangeType(bytes[0], typeof(byte[])));
    string str2 = gb.GetString((byte[])Convert.ChangeType(bytes[1], typeof(byte[])));
    string str3 = gb.GetString((byte[])Convert.ChangeType(bytes[2], typeof(byte[])));
    string str4 = gb.GetString((byte[])Convert.ChangeType(bytes[3], typeof(byte[])));
    txt = str1 + str2 + str3 + str4;                                                //获取随机生成的 4 个汉字
    if (txt == null || txt == String.Empty)                                         //如果没有汉字
    {
        return;                                                                     //返回
    }
    Bitmap image = new Bitmap((int)Math.Ceiling((txt.Length * 21.5)), 22);          //创建 Bitmap 实例用于绘制验证码
    Graphics g = Graphics.FromImage(image);                                         //创建 Graphics 对象
    try
    {
        Random random = new Random();                                               //生成随机生成器
        g.Clear(Color.White);                                                       //清空图片背景色
        for (int i = 0; i < 2; i++)                                                 //画图片的背景噪音线
        {
            Point tem_Point_1 = new Point(random.Next(image.Width), random.Next(image.Height));
            Point tem_Point_2 = new Point(random.Next(image.Width), random.Next(image.Height));
            g.DrawLine(new Pen(Color.Black), tem_Point_1, tem_Point_2);
        }
        Font font = new Font("宋体", 12, (FontStyle.Bold));
        LinearGradientBrush brush = new LinearGradientBrush(new Rectangle(0, 0, image.Width, image.Height),
Color.Blue, Color.DarkRed, 1.2f, true);
        g.DrawString(txt, font, brush, 2, 2);
        for (int i = 0; i < 100; i++)                                               //画图片的前景噪音点
        {
            Point tem_point = new Point(random.Next(image.Width),random.Next(image.Height));
            image.SetPixel(tem_point.X,tem_point.Y, Color.FromArgb(random.Next()));
        }
        //画图片的边框线
        g.DrawRectangle(new Pen(Color.Silver), 0, 0, image.Width - 1, image.Height - 1);
        pictureBox1.Image = image;                                                  //显示生成的中文验证码
    }
    catch { }
}
```

秘笈心法

心法领悟 492：使用 LinearGradientBrush 类绘制渐变色彩。

LinearGradientBrush 类提供一种渐变色彩的特效，填满图形的内部区域，它位于 System.Drawing.Drawing2D 命名空间下。本实例中使用 LinearGradientBrush 类绘制渐变区域的代码如下：

```
LinearGradientBrush brush = new LinearGradientBrush(new Rectangle(0, 0, image.Width, image.Height),
Color.Blue, Color.DarkRed, 1.2f, true);
```

实例 493　使用双缓冲技术绘图

光盘位置：光盘\MR\21\493　　高级　　趣味指数：★★★★★

实例说明

在窗体中使用 GDI+技术绘图时，有时会发现绘制出的图形线条不够流畅，或者在改变窗体大小时会出现不断闪烁的现象。绘制的图形线条不流畅，是因为窗体在重绘时其自身的重绘与图形的重绘之间存在时间差，从而导致这两者之间的图像显示不协调；改变窗体大小出现的闪烁现象，是因为窗体在重绘时其自身的背景颜色与图形颜色频繁交替，从而造成人们视觉上的闪烁现象。若使用双缓冲技术绘制图形，则可以解决上述绘图中出现的若干问题，本实体使用双缓冲技术绘制 4 个图形，分别是贝塞尔曲线、圆形、矩形及一个不规则图形区域。实例运行效果如图 21.15 所示。

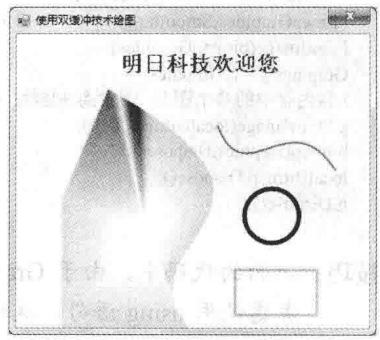


图 21.15　使用双缓冲技术绘图

关键技术

本实例在实现过程中，首先通过 Bitmap 类的构造函数创建一个位图实例，然后通过调用 Graphics 类的 FromImage 方法创建画布对象，最后调用 Graphics 类的 DrawImage 方法实现在窗体上绘制图形。下面对本实例中用到的关键技术进行详细讲解。

（1）Bitmap 类的构造函数

该构造函数用来在内存中创建位图实例，其重载方式有多种，本实例中用到的重载形式如下：

```
public Bitmap(int width, int height);
```

参数说明

❶ width：定义位图的宽度。

❷ height：定义位图的高度。

（2）FromImage 方法

该方法通过加载 Image 实例创建画布对象。语法格式如下：

```
public static Graphics FromImage(Image image);
```

参数说明

image：Image 类或其子类的实例引用。

（3）DrawImage 方法

该方法实现在画布上绘制图像，其重载方式有多种，本实例中用到的重载形式如下：

```
public void DrawImage(Image image, int x, int y);
```

参数说明

❶ image：要绘制的图像。

❷ x：所绘制图像的左上角的 x 坐标。

❸ y：所绘制图像的左上角的 y 坐标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DoubleBuffer。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下。

窗体的 Paint 事件在窗体被绘制时触发，如在窗体开始运行时或窗体大小改变时等情况下都将触发该事件。本例通过在窗体的 Paint 事件中编写绘图程序来实现，首先在窗体的 Paint 事件中创建一个 Bitmap 位图实例，然后通过加载该位图实例生成一个 Graphics 画布对象，接着调用自定义方法 PaintImage 实现绘制图形，最后通过调用 Graphics 类的 DrawImage 方法将内存中的多个图形一次性绘制到窗体上。代码如下：

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Bitmap localBitmap = new Bitmap(ClientRectangle.Width, ClientRectangle.Height);      //创建位图对象
    Graphics bitmapGraphics = Graphics.FromImage(localBitmap);                           //创建的画布
    bitmapGraphics.Clear(BackColor);                                                     //清空画布
    bitmapGraphics.SmoothingMode = SmoothingMode.AntiAlias;                              //消除画布锯齿
    PaintImage(bitmapGraphics);                                                          //绘制多个图形
    Graphics g = e.Graphics;                                                             //获取窗体画布
    //将内存中的多个图形一次性绘制到窗体上
    g.DrawImage(localBitmap, 0, 0);
    bitmapGraphics.Dispose();                                                            //销毁画布对象
    localBitmap.Dispose();                                                               //销毁位图对象
    g.Dispose();                                                                         //销毁画布对象
}
```

技巧：上面的代码中，由于 Graphics 类实现了 IDisposable 接口，所以在创建 Graphics 类的实例时，可以考虑使用 using 语句，这样当 using 语句块运行结束后，程序会自动调用 IDisposable 接口的 Dispose 方法来销毁实例。

上面的代码中用到了 PaintImage 方法，该方法为自定义的无返回值类型方法，主要用来绘制 4 个图形，它有一个参数，表示绘图的画布。PaintImage 方法的实现代码如下：

```
private void PaintImage(Graphics g)
{
    //绘制不规则图形区域
    GraphicsPath path = new GraphicsPath(new Point[]{ new Point(100,60),new Point(350,200),new
Point(105,225),new  Point(190,ClientRectangle.Bottom),new  Point(50,ClientRectangle.Bottom),new  Point(50,180)}, new byte[]{(byte)PathPointType.
Start,(byte)PathPointType.Bezier,(byte)PathPointType.Bezier,(byte)PathPointType.Bezier,(byte)PathPointType.Line, (byte)PathPointType.Line});
    PathGradientBrush pgb = new PathGradientBrush(path);                                 //创建 PathGradientBrush 对象
    pgb.SurroundColors = new Color[] { Color.Green, Color.Yellow, Color.Red, Color.Blue, Color.Orange,
Color.LightBlue };                                                                       //设置填充区域的颜色数组
    g.FillPath(pgb, path);                                                               //使用指定颜色填充不规则图形区域
    g.DrawString("明日科技欢迎您", new Font("宋体", 18, FontStyle.Bold), new SolidBrush(Color.Red), new
PointF(110, 20));                                                                        //在画布上绘制字符串
    g.DrawBeziers(new Pen(new SolidBrush(Color.Green),2),new Point[] {new Point(220,100),new
Point(250,180),new Point(300,70),new Point(350,150)});                                   //在画布上绘制贝塞尔曲线
    g.DrawArc(new Pen(new SolidBrush(Color.Blue), 5), new Rectangle(new Point(250, 170), new Size(60,
60)), 0, 360);                                                                           //在画布上绘制圆形
    g.DrawRectangle(new Pen(new SolidBrush(Color.Orange), 3), new Rectangle(new Point(240, 260), new
Size(90, 50)));                                                                          //在画布上绘制长方形
}
```

秘笈心法

心法领悟 493：有效避免窗体纵向拖放过程中的图像褶皱现象。

本实例通过使用双缓冲技术绘制图形，从而有效减少绘制图形时的闪烁现象，并使图形的线条更加平滑和流畅，但在调整窗体纵向大小的过程中，有时图像会出现褶皱现象，出现这种现象主要是由于窗体的有效区域没有完全重绘造成的，这时通过调用窗体的 SetStyle 方法来设置控件的 ResizeRedraw 属性值为 true，可以实现窗体有效区域的完全重绘，这时再纵向拉伸窗体，图像就不会出现褶皱现象。

第22章

图像处理技术

- 图像格式转换
- 图像预览
- 图像的缩放与变换
- 图像的特殊效果
- 特殊效果的文字
- 图像识别
- 常用的图像工具
- 图像应用

22.1 图像格式转换

实例 494 BMP 转换为 JPG 格式

光盘位置：光盘\MR\22\494

初级

趣味指数：★★★★

实例说明

BMP 文件是一种位图格式文件，这种格式的文件中包含文件头、位图信息头、调色板和数据区 4 个部分，而且它将图像中的像素值以矩阵的形式存储在文件中，因此这类文件非常大；而 JPG 格式文件是另一种存储图像的文件格式，该类文件支持高级压缩，但不要过度压缩，因为在压缩的比例较大时可能会产生失真。随着 Internet 技术的发展，JPG 文件已经逐渐成为主流的图像格式。本实例实现了将 BMP 位图格式文件转换为 JPG 格式文件的功能，以便节省更多的空间。实例运行效果如图 22.1 所示。

图 22.1 BMP 转换为 JPG 格式

关键技术

本实例在将 BMP 格式图片转换为 JPG 格式时，需要使用 ImageFormat 类，该类主要用来指定图像的格式，其常用属性及说明如表 22.1 所示。

表 22.1 ImageFormat 类的常用属性及说明

属　性	说　明
Bmp	获取位图图像格式（BMP）
Emf	获取增强型 Windows 图元文件图像格式（EMF）
Exif	获取可交换图像文件（Exif）格式
Gif	获取图形交换（GIF）图像格式
Icon	获取 Windows 图标图像格式
Jpeg	获取联合图像专家组（JPEG）图像格式
MemoryBmp	获取内存位图图像格式
Png	获取 W3C 可移植网络图形（PNG）图像格式
Tiff	获取标签图像文件（TIFF）图像格式
Wmf	获取 Windows 图元文件（WMF）图像格式

说明：程序中使用 ImageFormat 类时，首先需要在命名空间区域添加 System.Drawing.Imaging 命名空间，下面再遇到类似情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BMPChangeJPG。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 ComboBox 控件，用来选择要转换成的图片格式；添加两个 Button 控件，分别用来执行打开 BMP 格式图片和转换图片格式操作。

（3）程序的主要代码如下：

```
private void buttonConvert_Click(object sender, EventArgs e)
{
```

```
    if (comboBox.SelectedItem == null)                          //如果没有选择项
    {
        return;                                                 //退出本次操作
    }
    else
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();   //创建 SaveFileDialog 对象
        saveFileDialog.Title = "转化为:";                        //设置标题
        saveFileDialog.OverwritePrompt = true;                  //如果文件名存在则提示
        saveFileDialog.CheckPathExists = true;                  //如果文件的路径不存在则提示
        saveFileDialog.Filter = comboBox.Text + "|" + comboBox.Text;  //设置文件类型
        if (saveFileDialog.ShowDialog() == DialogResult.OK)     //打开“另存为”对话框
        {
            string fileName = saveFileDialog.FileName;          //获取另存为的文件路径和文件名
            bitmap.Save(fileName, ImageFormat.Jpeg);            //调用 Save 方法将图片保存为 Jpeg 格式
            FileInfo f = new FileInfo(fileName);                //创建 FileInfo 对象
            this.Text = "图像转换:" + f.Name;                    //在窗体标题栏中显示转换的文件名
            label1.Text = f.Name;
        }
    }
}
```

秘笈心法

心法领悟 494：BMP 格式的图片。

BMP 是 Windows 操作系统使用的一种标准格式，它用于存储与设备无关和与应用程序无关的图像。由于 BMP 文件通常不能压缩，因此，它不太适合通过 Internet 传输。

实例 495　JPG 转换成 BMP 格式

光盘位置：光盘\MR\22\495　　初级　　趣味指数：★★★★

实例说明

随着 Internet 的逐渐发展，信息不断膨胀，网络使日常生活发生了巨大变化，JPG 格式的图像文件是 Internet 中传输的主要图像格式，此种文件体积小、质量高，所以加快了传输速度；而 BMP 文件是一种简单的文件格式，大多数操作软件中都直接支持此种格式。本实例实现了将 JPG 文件转换为 BMP 格式文件的功能。实例运行效果如图 22.2 所示。

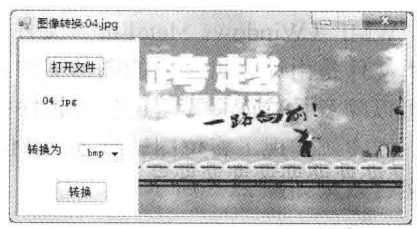


图 22.2　JPG 转换成 BMP 格式

关键技术

本实例在将 JPG 格式图片转换为 BMP 格式时，主要是通过 ImageFormat 类来实现的。

说明：关于 ImageFormat 类的详细讲解请参见实例 494 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 JPGChangeBMP。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 ComboBox 控件，用来选择要转换成的图片格式；添加两个 Button 控件，分别用来执行打开 JPG 格式图片和转换图片格式操作。

（3）程序的主要代码如下：

```
private void buttonConvert_Click(object sender, EventArgs e)
{
    if (comboBox.SelectedItem == null)                          //如果没有选定项
```

```
    {
        return;                                                    //退出本次操作
    }
    else
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();      //创建 SaveFileDialog 对象
        saveFileDialog.Title = "转化为:";                          //设置“另存为”对话框的标题
        saveFileDialog.OverwritePrompt = true;                     //如果文件名存在则提示
        saveFileDialog.CheckPathExists = true;                     //如果文件的路径不存在则提示
        saveFileDialog.Filter = comboBox.Text + "|" + comboBox.Text; //设置文件类型
        if (saveFileDialog.ShowDialog() == DialogResult.OK)        //打开“另存为”对话框
        {
            string fileName = saveFileDialog.FileName;             //获取另存为文件的路径及名称
            bitmap.Save(fileName, ImageFormat.Bmp);                //调用 Save 方法将图片保存为 Bmp 格式
            FileInfo f = new FileInfo(fileName);                   //创建 FileInfo 对象
            this.Text = "图像转换:" + f.Name;                      //设置窗体标题栏
            label1.Text = f.Name;
        }
    }
}
```

秘笈心法

心法领悟 495：JPEG 格式的图片。

JPEG 是一种适应于自然景观（如扫描的照片）的压缩方案，它的一些信息会在压缩过程中丢失，但这些丢失人眼是察觉不到的。JPEG 文件每像素存储 24 位，因此它们能够显示超过 16000000 种颜色。JPEG 文件不支持透明或动画。Web 浏览器显示的 JPEG 文件使用.jpg 扩展名。

实例 496 位图转化为 WMF

光盘位置：光盘\MR\22\496　　初级　　趣味指数：★★★★

实例说明

WMF（Windows MetaFile）文件是一种矢量图片格式，这类文件是由计算机生成的图形文件，它比图像文件小，而且可以随意拉伸变形，Word 等软件中直接支持此种图像格式。本实例实现了将位图转换为 WMF 格式文件的功能。实例运行效果如图 22.3 所示。

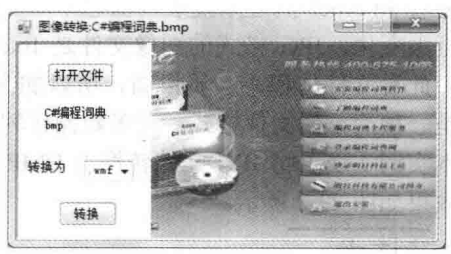


图 22.3　位图转化为 WMF

关键技术

本实例在将位图转换为 WMF 格式时，主要是通过 ImageFormat 类来实现的。

说明：关于 ImageFormat 类的详细讲解请参见实例 494 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BMPChangeWMF。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 ComboBox 控件，用来选择要转换成的图片格式；添加两个 Button 控件，分别用来执行打开位图（BMP）格式图片和转换图片格式操作。

（3）程序的主要代码如下：

```
private void buttonConvert_Click(object sender, EventArgs e)
{
```

```
    if (comboBox.SelectedItem == null)                          //如果没有选择项
    {
        return;                                                 //退出本次操作
    }
    else
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();   //创建 SaveFileDialog 对象
        saveFileDialog.Title = "转化为:";                        //设置“另存为”对话框的标题
        saveFileDialog.OverwritePrompt = true;                  //如果文件名存在则提示
        saveFileDialog.CheckPathExists = true;                  //如果文件的路径不存在则提示
        saveFileDialog.Filter = comboBox.Text + "|" + comboBox.Text;  //设置文件类型
        if (saveFileDialog.ShowDialog() == DialogResult.OK)     //打开“另存为”对话框
        {
            string fileName = saveFileDialog.FileName;          //获取另存为文件的路径及名称
            bitmap.Save(fileName, ImageFormat.Wmf);             //调用 Save 方法将图片保存为 Wmf 格式
            FileInfo f = new FileInfo(fileName);                //创建 FileInfo 对象
            this.Text = "图像转换:" + f.Name;                    //设置窗体标题栏
            label1.Text = f.Name;
        }
    }
}
```

秘笈心法

心法领悟 496：WMF 格式的图片。

WMF 表示 Windows 图元文件，图元文件也称为矢量图像，它是一种存储为一系列绘图命令和设置的图像。

实例 497　ICO 文件转化为位图

光盘位置：光盘\MR\22\497　　初级　　趣味指数：★★★★

实例说明

有些程序中要求只能用 BMP 格式的图片，但一般小图标都是 ICO 格式的，这就需要将其转换为 BMP 格式。本实例实现了将 ICO 格式图标转化为位图的功能。实例运行效果如图 22.4 所示。

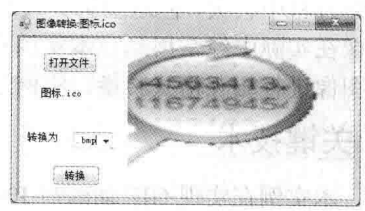


图 22.4　ICO 文件转化为位图

关键技术

本实例在将 ICO 格式图标转换为位图时，主要是通过 ImageFormat 类来实现的。

说明：关于 ImageFormat 类的详细讲解请参见实例 494 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ICOChangeBMP。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 ComboBox 控件，用来选择要转换成的图片格式；添加两个 Button 控件，分别用来执行打开 ICO 格式图标和转换图片格式操作。

（3）程序的主要代码如下：

```
private void buttonConvert_Click(object sender, EventArgs e)
{
    if (comboBox.SelectedItem == null)                          //如果没有选定项
    {
        return;                                                 //退出本次操作
    }
    else
```

```
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();                    //创建 SaveFileDialog 对象
        saveFileDialog.Title = "转化为:";                                        //设置“另存为”对话框的标题
        saveFileDialog.OverwritePrompt = true;                                   //如果文件名存在则提示
        saveFileDialog.CheckPathExists = true;                                   //如果文件的路径不存在则提示
        saveFileDialog.Filter = comboBox.Text + "|" + comboBox.Text;             //设置文件类型
        if (saveFileDialog.ShowDialog() == DialogResult.OK)                      //打开“另存为”对话框
        {
            string fileName = saveFileDialog.FileName;                           //获取另存为文件的路径及名称
            bitmap.Save(fileName, ImageFormat.Bmp);                              //调用 Save 方法将图片保存为 Bmp 格式
            FileInfo f = new FileInfo(fileName);                                 //创建 FileInfo 对象
            this.Text = "图像转换:" + f.Name;                                    //设置窗体标题栏
            label1.Text = f.Name;
        }
    }
}
```

秘笈心法

心法领悟 497：ICO 格式的图片。

ICO 格式的文件一般为图标文件，它常用的大小主要有 16×16、32×32、48×48 以及 96×96。

实例 498　GIF 动画与 JPG 图片的转换　初级

光盘位置：光盘\MR\22\498　　趣味指数：★★★★

实例说明

GIF 动画是一种用于在网页中显示图像的通用格式，它可以压缩，而且在压缩过程中没有信息丢失；而 JPG 图片是一种适应于自然景观（如扫描的照片）的压缩方案，它的一些信息会在压缩过程中丢失，但这些丢失人眼是察觉不到的，这两种格式的图像在实际中经常用到。本实例将讲解如何通过 C#语言对这两种图像进行相互格式转换。实例运行效果如图 22.5 所示。

图 22.5　GIF 动画与 JPG 图片的转换

关键技术

本实例在实现 GIF 动画与 JPG 格式图片的相互转换时，主要用到了 ImageFormat 类。

说明：关于 ImageFormat 类的详细讲解请参见实例 494 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GIFChangeJPG。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 ComboBox 控件，用来选择要转换成的图片格式；添加两个 Button 控件，分别用来执行打开 GIF、JPG 格式图像和转换图像格式操作。

（3）程序的主要代码如下：

```
private void buttonConvert_Click(object sender, EventArgs e)
{
    if (comboBox.SelectedItem == null)                                           //如果没有选择项
    {
        return;                                                                  //退出本次操作
    }
    else
    {
        SaveFileDialog saveFileDialog = new SaveFileDialog();                    //创建 SaveFileDialog 对象
```

```
        saveFileDialog.Title = "转化为:";                          //设置“另存为”对话框的标题
        saveFileDialog.OverwritePrompt = true;                     //如果文件名存在则提示
        saveFileDialog.CheckPathExists = true;                     //如果文件的路径不存在则提示
        saveFileDialog.Filter = comboBox.Text + "|" + comboBox.Text;  //设置文件类型
        if (saveFileDialog.ShowDialog() == DialogResult.OK)        //打开“另存为”对话框
        {
            string fileName = saveFileDialog.FileName;             //获取另存为文件的路径及名称
            if (comboBox.SelectedIndex == 0)                       //如果选择了第一项
                bitmap.Save(fileName, ImageFormat.Gif);            //调用 Save 方法将图片保存为 Gif 格式
            else if (comboBox.SelectedIndex == 1)                  //如果选择了第二项
                bitmap.Save(fileName, ImageFormat.Jpeg);           //调用 Save 方法将图片保存为 Jpeg 格式
            FileInfo f = new FileInfo(fileName);                   //创建 FileInfo 对象
            this.Text = "图像转换:" + f.Name;                      //设置窗体标题栏
            label1.Text = f.Name;
        }
    }
}
```

秘笈心法

心法领悟 498：GIF 格式的图片。

GIF 是一种用于在网页中显示图像的通用格式，它主要适用于画线、有纯色块的图片和在颜色之间有清晰边界的图片。GIF 文件可以压缩，而且在压缩过程中没有信息丢失。

实例 499	批量图像格式转换	高级
	光盘位置：光盘\MR\22\499	趣味指数：★★★★★

实例说明

计算机支持的图像格式有很多种，但是如果想将某种格式转换成另一种格式就需要用到图像格式转换工具。由于需要转换的图片可能会有很多，所以开发出批量图像格式转换工具。通过本实例可以将选择的多个图片转换成指定的图像格式，实例运行效果如图 22.6 所示。

图 22.6　批量图像格式转换

关键技术

本实例在实现批量图像格式转换时主要用到了 Thread 类和 ImageFormat 类，其中，由于本实例实现的是批量图像格式转换，所以有可能会执行得比较慢，从而出现程序“假死”状态，这时如果使用 Thread 类新开一个线程去执行，就会避免这种情况的发生；而在执行图像格式转换功能时，主要是通过 ImageFormat 类实现的。下面对本实例中用到的关键技术进行详细讲解。

Thread 类位于 System.Threading 命名空间下，它主要用于创建并控制线程、设置线程优先级并获取其状态。

一个进程可以创建一个或多个线程以执行与该进程关联的部分程序代码，线程执行的程序代码由 ThreadStart 委托或 ParameterizedThreadStart 委托指定。其 Start 方法用来使线程被安排执行，而 Abort 方法用来终止线程。例如，本实例中启动批量图像格式转换线程的代码如下：

```
td = new Thread(new ThreadStart(ConvertImage));                    //通过线程调用 ConvertImage 方法进行转换
td.Start();                                                        //开启线程
```

本实例中终止批量图像格式转换线程的代码如下：

```
if (td != null)                                                    //判断是否存在线程
{
    if (td.ThreadState == ThreadState.Running)                     //判断线程是否正在运行
    {
        td.Abort();                                                //终止线程
    }
}
```

说明：关于 ImageFormat 类的详细讲解请参见实例 494 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PictureBatchConversion。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为窗体的工具栏；添加一个 ListView 控件，用来显示待转换的图片信息及转换的状态；添加一个 OpenFileDialog 控件，用来选择待转换的图片；添加一个 FolderBrowserDialog 控件，用来选择保存路径；添加一个 StatusStrip 控件，用来显示状态信息。

（3）程序的主要代码如下。

Frm_Main 窗体的后台代码中，创建一个 ConvertImage 方法用于进行图像格式转换，该方法中使用 switch 语句判断要转换的类型，如果类型 Imgtype1 为 0，则将图像转换为 BMP 类型。代码如下：

```
case 0:                                                            //如果选择第一项则转换为 BMP
    for (int i = 0; i < path1.Length; i++)                         //遍历图片集合
    {
        string ImgName = path1[i].Substring(path1[i].LastIndexOf("\\") + 1,
path1[i].Length - path1[i].LastIndexOf("\\") - 1);                 //获取图片名称（带扩展名）
        ImgName = ImgName.Remove(ImgName.LastIndexOf("."));        //获取图片名称（不带扩展名）
        OlePath = path1[i].ToString();                             //获取图片所在路径
        bt = new Bitmap(OlePath);                                  //创建 Bitmap 对象
        path = path2 + "\\" + ImgName + ".bmp";                    //设置保存路径
        bt.Save(path, System.Drawing.Imaging.ImageFormat.Bmp);     //保存
        listView1.Items[flags - 1].SubItems[6].Text = "已转换";     //显示转换的状态
        tsslPlan.Text = "正在转换"+flags*100/path1.Length+"%";      //显示转换的进度
        if (flags == path1.Length)                                 //如果转换的数量等于图片总数量
        {
            toolStrip1.Enabled = true;
            tsslPlan.Text = "图片转换全部完成";                     //提示转换完成
        }
        flags++;
    }
    break;
```

如果类型 Imgtype1 为 1，则将图像转换为 JPEG 格式。代码如下：

```
case 1:                                                            //如果选择第二项则转换成 JPEG 格式
    for (int i = 0; i < path1.Length; i++)                         //遍历图片集合
    {
        string ImgName = path1[i].Substring(path1[i].LastIndexOf("\\") + 1,
 path1[i].Length - path1[i].LastIndexOf("\\") - 1);                //获取图片名称（带扩展名）
        ImgName = ImgName.Remove(ImgName.LastIndexOf("."));        //获取图片名称（不带扩展名）
        OlePath = path1[i].ToString();                             //获取图片所在路径
        bt = new Bitmap(OlePath);                                  //创建 Bitmap 对象
        path = path2 + "\\" + ImgName + ".jpeg";                   //设置保存路径
        bt.Save(path, System.Drawing.Imaging.ImageFormat.Jpeg);    //保存
        listView1.Items[flags - 1].SubItems[6].Text = "已转换";     //显示转换的状态
        tsslPlan.Text = "正在转换" + flags * 100 / path1.Length + "%";  //显示转换的进度
```

```
        if (flags == path1.Length)                                    //如果转换的数量等于图片总数量
        {
            toolStrip1.Enabled = true;
            tsslPlan.Text = "图片转换全部完成";                        //提示转换完成
                }
                flags++;
    }
break;
```

如果类型 Imgtype1 为 2，则将图像转换为 PNG 格式。代码如下：

```
case 2:                                                               //如果选择第三项则转换成 PNG 格式
    for (int i = 0; i < path1.Length; i++)                            //遍历图片集合
    {
        string ImgName = path1[i].Substring(path1[i].LastIndexOf("\\") + 1,
path1[i].Length - path1[i].LastIndexOf("\\") - 1);                    //获取图片名称（带扩展名）
        ImgName = ImgName.Remove(ImgName.LastIndexOf("."));           //获取图片名称（不带扩展名）
        OlePath = path1[i].ToString();                                //获取图片所在路径
        bt = new Bitmap(OlePath);                                     //创建 Bitmap 对象
        path = path2 + "\\" + ImgName + ".png";                       //设置保存路径
        bt.Save(path, System.Drawing.Imaging.ImageFormat.Png);        //保存
        listView1.Items[flags - 1].SubItems[6].Text = "已转换";       //显示转换的状态
        tsslPlan.Text = "正在转换" + flags * 100 / path1.Length + "%";//显示转换的进度
        if (flags == path1.Length)                                    //如果转换的数量等于图片总数量
        {
            toolStrip1.Enabled = true;
            tsslPlan.Text = "图片转换全部完成";                        //提示转换完成
        }
        flags++;
    }
    break;
```

如果类型 Imgtype1 为 3，则将图像转换为 GIF 格式。代码如下：

```
case 3:                                                               //如果选择第 4 项则转换成 GIF 格式
      for (int i = 0; i < path1.Length; i++)                          //遍历图片集合
      {
          string ImgName = path1[i].Substring(path1[i].LastIndexOf("\\") + 1,
path1[i].Length - path1[i].LastIndexOf("\\") - 1);                    //获取图片名称（带扩展名）
          ImgName = ImgName.Remove(ImgName.LastIndexOf("."));         //获取图片名称（不带扩展名）
          OlePath = path1[i].ToString();                              //获取图片所在路径
          bt = new Bitmap(OlePath);                                   //创建 Bitmap 对象
          path = path2 + "\\" + ImgName + ".gif";                     //设置保存路径
          bt.Save(path, System.Drawing.Imaging.ImageFormat.Gif);      //保存
          listView1.Items[flags - 1].SubItems[6].Text = "已转换";     //显示转换的状态
          tsslPlan.Text = "正在转换" + flags * 100 / path1.Length + "%";//显示转换的进度
          if (flags == path1.Length)
          {
              toolStrip1.Enabled = true;
              tsslPlan.Text = "图片转换全部完成";                      //提示转换完成
          }
          flags++;
      }
       break;
```

说明：在进行图像格式转换时，如果原图是 GIF 动画，转换成其他格式的图片后只能保留一帧。而如果将其他格式的图片转换成 GIF 格式也只能是静态的图片。

秘笈心法

心法领悟 499：使用线程代替 Timer 组件。

Timer 组件可以按用户定义的时间间隔来引发 Tick 事件，Tick 事件一般为周期性的，每隔若干秒或若干毫秒执行一次，Timer 组件工作在窗体线程中，如果 Timer 组件中执行了较为耗时的操作，会增加窗体线程的负担，导致窗体中其他操作不能及时得到 CPU 资源，出现窗体长时间或短时间无响应的情况。在适当的情况下可以使用线程来代替 Timer 组件，这样会减少窗体线程的负担。

22.2 图像预览

实例 500	局部图像放大	中级
	光盘位置：光盘\MR\22\500	趣味指数：★★★★★

实例说明

浏览图片时，如果随着鼠标的移动可以将鼠标周围的区域放大显示，将会显得特别方便。本实例实现了图像局部放大的功能，当鼠标在图像上移动时即放大其周围的图像。实例运行效果如图 22.7 所示。

图 22.7 局部图像放大

关键技术

本实例通过使用 Graphics 类的 DrawImage 方法将图像的局部进行放大。DrawImage 方法为可重载方法，主要用来在指定位置绘制指定的 Image 图像，常用格式有以下 3 种。

（1）在指定的位置按原始大小绘制指定的 Image 图像。语法格式如下：

```
public void DrawImage (Image image, Point point)
```

参数说明

❶ image：要绘制的 Image 图像。

❷ point：Point 结构，表示所绘制图像的左上角的位置。

（2）在指定位置按指定大小绘制指定的 Image 图像。语法格式如下：

```
public void DrawImage (Image image, Rectangle rect)
```

参数说明

❶ image：要绘制的 Image 图像。

❷ rect：Rectangle 结构，指定所绘制图像的位置和大小。

（3）在指定位置按指定大小绘制指定 Image 图像的指定部分。语法格式如下：

```
public void DrawImage ( Image image, Point[] destPoints, Rectangle srcRect, GraphicsUnit srcUnit)
```

DrawImage 方法中的参数说明如表 22.2 所示。

表 22.2 DrawImage 方法中的参数说明

参　数	描　述
image	要绘制的 Image 图像
destPoints	由 3 个 Point 结构组成的数组，这 3 个结构定义一个平行四边形
srcRect	Rectangle 结构，指定 image 图像中要绘制的部分
srcUnit	GraphicsUnit 枚举成员，指定 srcRect 参数所用的度量单位

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImageBlowUp。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，并将其 AutoScroll 属性设置为 true，以便显示图片的全部；添加一个 PictureBox 控件，用来显示选择的图片；添加一个 Button 控件，用来执行打开图片操作。

（3）程序的主要代码如下：

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
```

```
    try
    {
        Cursor.Current = myCursor;                                          //定义鼠标
        Graphics graphics = pictureBox1.CreateGraphics();                   //创建 pictureBox1 控件的 Graphics 对象
        //声明两个 Rectangle 对象，分别用来指定要放大的区域和放大后的区域
        Rectangle sourceRectangle = new Rectangle(e.X - 10, e.Y - 10, 20, 20);   //要放大的区域
        Rectangle destRectangle = new Rectangle(e.X - 20, e.Y - 20, 40, 40);
        //调用 DrawImage 方法对选定区域进行重新绘制，以放大该部分
        graphics.DrawImage(myImage, destRectangle, sourceRectangle, GraphicsUnit.Pixel);
    }
    catch { }
}
```

秘笈心法

心法领悟 500：使用 try…catch 语句捕获异常。

在大部分情况下，开发人员不希望异常状况发生导致程序结束，因此，可以使用 try…catch 程序语句块捕捉程序中的 Exception 对象，再使用自定义的程序逻辑处理异常状况。如果有需要，用户也可以使用多重 try…catch 语句块（如一个 try 块后跟一个或多个 catch 子句构成）。

实例 501　浏览大图片

光盘位置：光盘\MR\22\501　　中级　　趣味指数：★★★☆

实例说明

当在程序中浏览大图片时，如果通过调整窗体大小或控件的大小查看其实际大小，将会使窗体或控件看起来不十分规整，那么如何在一个固定大小的窗体上浏览大的图片呢？运行本实例，单击“打开图片”按钮，如果打开的图片特别大，可以通过显示滚动条进行浏览。实例运行效果如图 22.8 所示。

图 22.8　浏览大图片

关键技术

本实例实现浏览大图片功能的方法是，将 PictureBox 控件的大小设置为与图片大小相同，然后将该 PictureBox 控件放置在 Panel 控件内部，并设置 Panel 控件的 AutoScroll 属性为 true。

Panel 控件的 AutoScroll 属性用来获取或设置一个值，以指定容器是否允许用户滚动到任何放置在其可见边界之外的控件。如果容器允许自动滚动，则为 true；否则为 false。默认值为 false。

另外，将 PictureBox 控件的大小设置为与图片大小相同的实现代码如下：

```
PictureBox.Height = Image.Height;
PictureBox.Width = Image.Width;
```

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BrowseImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，并将其 AutoScroll 属性设置为 true，以便显示图片的全部；添加一个 PictureBox 控件，用来显示选择的图片；添加一个 Button 控件，用来执行打开图片操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "*.jpg,*.jpeg,*.bmp,*.gif,*.ico,*.png,*.tif,*.wmf|*.jpg;*.jpeg;*.bmp;*.gif;*.ico;*.png;*.tif;*.wmf";
    openFileDialog1.ShowDialog();                                           //打开文件对话框
    Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);   //根据文件的路径和名称创建 Image 对象
```

```
    pictureBox1.Image = myImage;                    //显示图片
    pictureBox1.Height = myImage.Height;            //设置控件的高度
    pictureBox1.Width = myImage.Width;              //设置控件的宽度
}
```

秘笈心法

心法领悟 501：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加载图像，或者使用 LoadAsync 方法异步加载图像。

实例 502　剪切图片

光盘位置：光盘\MR\22\502　　初级　　趣味指数：★★★★

实例说明

使用 Photoshop 软件时，可以对打开的图片进行剪切操作，那么如何利用 C#实现该功能呢？运行本实例，单击“打开图片”按钮，打开一幅图片，用鼠标选择需要剪切的部分，然后在窗体的空白部分单击鼠标右键，即可实现图片剪切的功能。实例运行效果如图 22.9 所示。

图 22.9　剪切图片

关键技术

本实例实现剪切图片的实现过程是，在显示图片的 PictureBox 控件中按下鼠标左键时，记录鼠标所在位置；然后在 PictureBox 控件中移动鼠标，并根据鼠标移动的位置，调用 Graphics 类的 DrawRectangle 方法绘制一个矩形；最后释放鼠标左键，在 PictureBox 控件中绘制一个指定大小的矩形，将选择的图片区域包含在内。当在窗体中单击鼠标时，使用 Bitmap 类的 Clone 方法复制选择区域的内容，并通过 DrawImage 方法将其绘制在窗体上，同时调用 Graphics 类的 FillRectangle 方法将 pictureBox1 控件中选择的区域用白色覆盖。从以上过程可以看出，本实例用到的技术主要有 DrawRectangle、FillRectangle 和 Clone 方法。下面对本实例中用到的关键技术进行详细讲解。

（1）DrawRectangle 方法

该方法用来绘制由坐标对、宽度和高度指定的矩形。语法格式如下：

```
public void DrawRectangle ( Pen pen, int x, int y, int width, int height)
```

DrawRectangle 方法中的参数说明如表 22.3 所示。

表 22.3　DrawRectangle 方法中的参数说明

参　数	描　述
pen	Pen 对象，确定矩形的颜色、宽度和样式
x	要绘制矩形的左上角的 x 坐标
y	要绘制矩形的左上角的 y 坐标
width	要绘制矩形的宽度
height	要绘制矩形的高度

（2）FillRectangle 方法

该方法用来填充 Rectangle 结构指定的矩形的内部。语法格式如下：

```
public void FillRectangle ( Brush brush, Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ rect：Rectangle 结构，表示要填充的矩形。

（3）Clone 方法

该方法用来创建 Bitmap 对象（由 Rectangle 结构和指定的 PixelFormat 枚举定义）的某个部分的副本。语法格式如下：

```
public Bitmap Clone ( Rectangle rect, PixelFormat format)
```

参数说明

❶ rect：定义 Bitmap 对象中要复制的部分，坐标相对于该 Bitmap 对象。

❷ format：为目标 Bitmap 对象指定 PixelFormat 枚举。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CutImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行打开图片操作；添加一个 PictureBox 控件，用来显示选择的图片，并在其上对图片进行剪切。

（3）程序的主要代码如下：

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        if ((isDrag = !isDrag) == true)                          //如果开始绘制，则记录鼠标位置
        {
            startPoint = new Point(e.X, e.Y);                    //记录鼠标的当前位置
            oldPoint = new Point(e.X, e.Y);
        }
    }
}
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    isDrag = false;
    ig = pictureBox1.CreateGraphics();                           //创建 pictureBox1 控件的 Graphics 类
    //绘制矩形框
    ig.DrawRectangle(new Pen(Color.Black, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
    //获得矩形框的区域
    theRectangle = new Rectangle(startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
}
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    Graphics g = this.CreateGraphics();                          //为当前窗体创建 Graphics 类
    if (isDrag)                                                  //如果鼠标已按下
    {
        //绘制一个矩形框
        g.DrawRectangle(new Pen(Color.Black, 1), startPoint.X, startPoint.Y, e.X - startPoint.X, e.Y - startPoint.Y);
    }
}
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    try
    {
        Graphics graphics = this.CreateGraphics();               //为当前窗体创建 Graphics 类
        Bitmap bitmap = new Bitmap(pictureBox1.Image);           //创建 Bitmap 对象
        Bitmap cloneBitmap = bitmap.Clone(theRectangle, PixelFormat.DontCare);//通过剪切图片的大小创建 Bitmap 对象
        graphics.DrawImage(cloneBitmap, e.X, e.Y);               //绘制剪切的图片
        Graphics g = pictureBox1.CreateGraphics();
        SolidBrush myBrush = new SolidBrush(Color.White);        //定义画刷
        g.FillRectangle(myBrush, theRectangle);                  //绘制剪切后的图片
    }
    catch { }
}
```

秘笈心法

心法领悟 502：创建 Graphics 对象的 3 种方法。

创建 Graphics 对象有以下 3 种方法：

- 在窗体或控件的 Paint 事件中创建，将其作为 PaintEventArgs 的一部分。
- 调用控件或窗体的 CreateGraphics 方法以获取对 Graphics 对象的引用，该对象表示控件或窗体的绘图画面。
- 由从 Image 继承的任何对象创建 Graphics 对象。

实例 503　鼠标拖拽图像

光盘位置：光盘\MR\22\503　　初级　　趣味指数：★★★☆

实例说明

鼠标拖拽功能可以大大简化用户的各种操作步骤，在 Word 中和播放影音文件时都会用到。本实例讲解如何使用鼠标拖拽图像，实例运行效果如图 22.10 所示。

图 22.10　鼠标拖拽图像

关键技术

本实例主要是通过触发 PictureBox 控件的 MouseEnter 事件和 Frm_Main 窗体的 MouseMove 事件来实现鼠标拖拽功能的。PictureBox 控件的 MouseEnter 事件用来在鼠标进入到该控件时发生，而其 MouseMove 事件则在鼠标移到该控件时发生。当鼠标进入到 PictureBox 控件上时，触发该控件的 MouseEnter 事件，将 bool 类型的全局变量设置为 true，然后在 Frm_Main 窗体中移动鼠标，触发该窗体的 MouseMove 事件，该事件中，通过控制 PictureBox 控件的 Location 属性来实现拖拽图像效果。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MouseDelayImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加一个 Button 控件，用来执行打开图像操作；添加一个 PictureBox 控件，用来显示打开的图像，并在其上对图像进行拖拽。

（3）程序的主要代码如下：

```
bool flag = false;                                                    //定义变量，用来标识鼠标是否移动
private void pictureBox1_MouseEnter(object sender, EventArgs e)
{
    flag = true;                                                      //将标识设置为 true
}
private void Frm_Main_MouseMove(object sender, MouseEventArgs e)
{
    if (flag)                                                         //如果鼠标在窗体中移动
        pictureBox1.Location = new System.Drawing.Point(e.X, e.Y);    //定义 PictureBox 控件的坐标位置
}
```

秘笈心法

心法领悟 503：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加

载图像，或者使用 LoadAsync 方法异步加载图像。

22.3　图像的缩放与变换

实例 504	放大和缩小图像 光盘位置：光盘\MR\22\504	初级 趣味指数：★★★★

实例说明

在很多的图片浏览器中，都可以对图片进行放大和缩小操作，从而在观察图片时能更清晰和全面。本实例中通过控制 PictureBox 控件的大小来实现图片的放大与缩小功能。实例运行效果如图 22.11 所示。

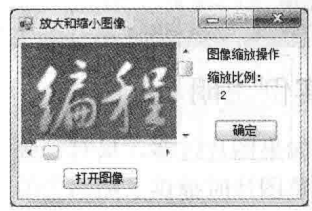


图 22.11　放大和缩小图像

关键技术

对图像进行放大和缩小，其实就是对 PictureBox 控件的 Height 和 Width 属性进行设置。另外，还应该将 PictureBox 控件的 SizeMode 属性设置为 Zoom，以便能够显示图片的全部。

SizeMode 属性用来指定如何在 PictureBox 控件中显示图像。语法格式如下：

```
public PictureBoxSizeMode SizeMode { get; set; }
```

参数说明

属性值：PictureBoxSizeMode 枚举值之一，默认为 Normal。PictureBoxSizeMode 枚举值及说明如表 22.4 所示。

表 22.4　PictureBoxSizeMode 枚举值及说明

枚 举 值	描　述
AutoSize	调整 PictureBox 控件大小，使其等于所包含的图像大小
CenterImage	如果 PictureBox 控件比图像大，则图像居中显示；如果图像比 PictureBox 控件大，则图像居于 PictureBox 控件中心，而外边缘将被剪裁掉
Normal	图像被置于 PictureBox 控件的左上角。如果图像比包含它的 PictureBox 控件大，则该图像将被剪裁掉
StretchImage	PictureBox 控件中的图像被拉伸或收缩，以适合 PictureBox 控件的大小
Zoom	图像大小按其原有的大小比例被增加或减小

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MaxMinImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加一个 PictureBox 控件，用来显示打开的图像；添加一个 Panel 控件，用来控制 PictureBox 控件的大小；添加一个 TextBox 控件，用来输入缩放比例系数；添加两个 Button 控件，分别用来打开图像和对指定图像进行缩放操作。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        pictureBox1.Height = Convert.ToInt32(myImage.Height * Convert.ToSingle(textBox1.Text.Trim()));        //设置高度
        pictureBox1.Width = Convert.ToInt32(myImage.Width * Convert.ToSingle(textBox1.Text.Trim()));         //设置宽度
    }
```

```
        catch { }
    }
```

秘笈心法

心法领悟 504：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加载图像，或者使用 LoadAsync 方法异步加载图像。

实例 505	生成图片缩略图	中级
	光盘位置：光盘\MR\22\505	趣味指数：★★★★☆

实例说明

如果图片过多，只有一张张地打开图片才能知道图片的内容，显然这样浏览起来非常不便。Windows 系统在浏览图片时提供了缩略图的功能，这样大大地方便了浏览者了解每张图片的内容，本实例实现了与 Windows 系统缩略图相同的功能。实例运行效果如图 22.12 所示。

图 22.12　生成图片缩略图

关键技术

本实例实现时主要用到了 Image 类的 GetThumbnailImage 方法，下面对其进行详细讲解。

Image 类的 GetThumbnailImage 方法主要用于返回指定 Image 的缩略图。语法格式如下：

```
public Image GetThumbnailImage(int thumbWidth,int thumbHeight, Image.GetThumbnailImageAbort callback,IntPtr callbackData)
```

GetThumbnailImage 方法中的参数说明如表 22.5 所示。

表 22.5　GetThumbnailImage 方法中的参数说明

参　数	描　述
thumbWidth	请求的缩略图的宽度（以像素为单位）
thumbHeight	请求的缩略图的高度（以像素为单位）
callback	一个 Image.GetThumbnailImageAbort 委托
callbackData	必须为 Zero
返回值	表示缩略图的 Image

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImgMicroimage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为窗体的菜单栏；添加一个 FolderBrowserDialog 控件，用来选择图片文件夹；添加一个 ImageList 组件，用来存储生成的缩略图；添加一个 Panel 控件，用来显示生成的缩略图。

（3）程序的主要代码如下：

```
public bool GetReducedImage(double Percent, string targetFilePath)
{
    try
    {
        Bitmap bt = new Bitmap(120,120);                                    //创建 Bitmap 实例
        Graphics g = Graphics.FromImage(bt);                                //创建 Graphics 实例
        g.Clear(Color.White);                                               //设置画布背景颜色为白色
        Image ReducedImage;                                                 //缩略图
        Image.GetThumbnailImageAbort callb = new Image.GetThumbnailImageAbort(ThumbnailCallback);
        ImageWidth = Convert.ToInt32(ResourceImage.Width * Percent);        //设置宽度
        ImageHeight = Convert.ToInt32(ResourceImage.Height * Percent);      //设置高度
        //获取缩略图
        ReducedImage = ResourceImage.GetThumbnailImage(ImageWidth, ImageHeight, callb, IntPtr.Zero);
        if (ImageWidth > ImageHeight)                                       //如果原图宽度大于高度
        {
            //缩放图片
            g.DrawImage(ReducedImage,0,(int)(120-ImageHeight)/2,ImageWidth,ImageHeight);
        }
        else
        {
            g.DrawImage(ReducedImage, (int)(120 - ImageWidth)/2, 0, ImageWidth, ImageHeight);
        }
        g.DrawRectangle(new Pen(Color.Gray), 0, 0,119,119);                 //绘制缩略图的边框
        bt.Save(@targetFilePath, ImageFormat.Jpeg);                         //保存缩略图
        bt.Dispose();                                                       //释放对象
        ReducedImage.Dispose();                                             //释放对象
        return true;
    }
    catch (Exception e)
    {
        ErrMessage = e.Message;
        return false;
    }
}
```

秘笈心法

心法领悟 505：避免生成缩略图时的闪烁现象。

开发本实例过程中，将生成的缩略图添加到 ListView 控件时，会出现非常严重的闪烁现象，影响了整体的浏览效果，所以，本程序中通过重写 ListView 控件消除了闪烁现象。

实例 506　旋转 JPG 图像

初级

光盘位置：光盘\MR\22\506

趣味指数：★★★★

实例说明

很多组合图片都是以不同的方向进行组合的，为了更方便地从各个角度浏览图片，本实例实现了将 JPG 图像以 90° 进行顺时针旋转功能。实例运行效果如图 22.13 所示。

图 22.13　旋转 JPG 图像

关键技术

本实例在实现旋转 JPG 图像功能时，主要用到了 Bitmap 类的 RotateFlip 方法，下面对其进行详细讲解。

RotateFlip 方法用来旋转、翻转或者同时旋转和翻转 Image 图像或图形。语法格式如下：

```
public void RotateFlip ( RotateFlipType rotateFlipType)
```

参数说明

属性值： RotateFlipType 枚举值之一，指定应用于图像的旋转或翻转的类型。RotateFlipType 枚举值及说明如表 22.6 所示。

表 22.6　RotateFlipType 枚举值及说明

枚 举 值	描　述
Rotate180FlipNone	指定不进行翻转的 180° 旋转
Rotate180FlipX	指定后接水平翻转的 180° 旋转
Rotate180FlipXY	指定后接水平翻转和垂直翻转的 180° 旋转
Rotate180FlipY	指定后接垂直翻转的 180° 旋转
Rotate270FlipNone	指定不进行翻转的 270° 旋转
Rotate270FlipX	指定后接水平翻转的 270° 旋转
Rotate270FlipXY	指定后接水平翻转和垂直翻转的 270° 旋转
Rotate270FlipY	指定后接垂直翻转的 270° 旋转
Rotate90FlipNone	指定不进行翻转的 90° 旋转
Rotate90FlipX	指定后接水平翻转的 90° 旋转
Rotate90FlipXY	指定后接水平翻转和垂直翻转的 90° 旋转
Rotate90FlipY	指定后接垂直翻转的 90° 旋转
RotateNoneFlipNone	指定不进行旋转和翻转
RotateNoneFlipX	指定没有后跟水平翻转的旋转
RotateNoneFlipXY	指定没有后跟水平翻转和垂直翻转的旋转

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RevolveJPG。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开和旋转 JPG 图像操作；添加一个 PictureBox 控件，用来显示打开的 JPG 图像和经过旋转之后的 JPG 图像。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    Image myImage = pictureBox1.Image;                          //创建 Image 对象
    myImage.RotateFlip(RotateFlipType.Rotate90FlipXY);          //调用 RotateFlip 方法将 JPG 格式图像进行旋转
    pictureBox1.Image = myImage;                                //显示旋转后的图片
}
```

秘笈心法

心法领悟 506：Image 类的使用。

Image 类为源自 Bitmap 和 Metafile 的类提供功能的抽象基类，它位于 System.Drawing 命名空间下。

实例 507　图像的翻转

光盘位置：光盘\MR\22\507　　初级　　趣味指数：★★★★

实例说明

在很多的图片浏览器中，都可以实现图片翻转的功能，将图片以水平方式或垂直方式进行翻转，可以使其以多种方式显示，用户还可以根据自己的喜好将翻转后的图片设置为墙纸。运行本实例，单击“打开图像”按钮，在窗体中显示一个图片，然后单击“翻转”按钮，使图片进行垂直翻转。实例运行效果如图 22.14 所示。

图 22.14　图像的翻转

关键技术

本实例主要通过调用 Bitmap 类的 RotateFlip 方法实现图像的翻转功能。

说明：关于 Bitmap 类的 RotateFlip 方法的详细讲解请参见实例 506 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CrossImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开和翻转图像操作；添加一个 PictureBox 控件，用来显示打开的图像和经过翻转之后的图像。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    Image myImage = pictureBox1.Image;                          //创建 Image 对象
    myImage.RotateFlip(RotateFlipType.Rotate180FlipX);          //调用 RotateFlip 方法实现图形翻转
    pictureBox1.Image = myImage;                                //显示旋转后的图片
}
```

秘笈心法

心法领悟 507：使用 Bitmap 类的 RotateFlip 方法实现图像旋转。

在实现图像旋转功能时，可以通过将 Bitmap 类的 RotateFlip 方法的参数指定为 RotateFlipType 枚举值中与 90° 相关的枚举值来实现。

实例 508　以任意角度旋转图像

光盘位置：光盘\MR\22\508　　初级　　趣味指数：★★★★☆

实例说明

任意角度旋转图像主要是将图像旋转一周，也就是 360°，在旋转过程中，如果图像不能完全覆盖背景，则用图像的不显示部分覆盖。运行本实例，单击“打开图像”按钮，选中要旋转的图像，然后单击“以任意角

度旋转显示图像”按钮，即可实现图像的旋转。实例运行效果如图 22.15 所示。

图 22.15　以任意角度旋转图像

关键技术

本实例实现时主要使用 TextureBrush 类的 RotateTransform 方法对图像进行旋转，然后使用 Graphics 类的 FillRectangle 方法对旋转的图像进行绘制。下面对本实例中用到的关键技术进行详细讲解。

（1）RotateTransform 方法

TextureBrush 类表示使用图像来填充形状的内部，它的每个属性都是一个 Brush 对象，其 RotateTransform 方法将指定旋转应用于 Graphics 对象的变换矩阵。语法格式如下：

```
public void RotateTransform (float angle)
```

参数说明

angle：旋转角度（以度为单位）。

（2）FillRectangle 方法

该方法填充由一对坐标、一个宽度和一个高度指定的矩形的内部。语法格式如下：

```
public void FillRectangle (Brush brush,int x,int y,int width,int height)
```

FillRectangle 方法中的参数说明如表 22.7 所示。

表 22.7　FillRectangle 方法中的参数说明

参　数	描　述
brush	确定填充特性的 Brush
x	要填充的矩形的左上角的 x 坐标
y	要填充的矩形的左上角的 y 坐标
width	要填充的矩形的宽度
height	要填充的矩形的高度

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RevolveImageByAngle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以任意角度旋转图像操作；添加两个 PictureBox 控件，分别用来显示打开的图像和以任意角度旋转的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.panel1.CreateGraphics();                    //创建绘图对象
```

```
    float MyAngle = 0;                                                                    //旋转的角度
    while (MyAngle < 360)
    {
        TextureBrush MyBrush = new TextureBrush(MyBitmap);                                //创建 TextureBrush 对象
        this.panel1.Refresh();                                                            //使工作区无效
        MyBrush.RotateTransform(MyAngle);                                                 //以指定角度旋转图像
        g.FillRectangle(MyBrush, 0, 0, this.ClientRectangle.Width, this.ClientRectangle.Height);  //绘制旋转后的图像
        MyAngle += 0.5f;                                                                  //增加旋转的角度
        System.Threading.Thread.Sleep(50);                                                //使线程休眠 50 毫秒
    }
}
```

秘笈心法

心法领悟 508：do while 循环与 while 循环有什么不同吗？

do while 与 while 语句的运行方式基本相同，只是 do while 语句要先执行一次循环体的内容，然后再判断布尔条件，如果满足条件则执行循环体，不满足布尔条件时退出循环体。

实例 509	以椭圆形显示图像	初级
	光盘位置：光盘\MR\22\509	趣味指数：★★★★★

实例说明

以椭圆形显示图像，即使用椭圆截取原图像中的一部分进行显示。本实例实现了以椭圆形显示图像的功能，运行本实例，单击“打开图像”按钮，选择指定的图像，单击“以椭圆形显示图像”按钮，即可在窗体中以椭圆形显示图像。实例运行效果如图 22.16 所示。

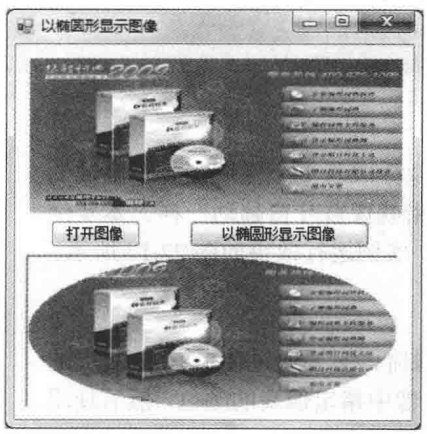


图 22.16　以椭圆形显示图像

关键技术

本实例实现时主要使用 Graphics 类的 FillEllipse 方法将打开的图像以椭圆形进行显示，下面对 Graphics 类的 FillEllipse 方法进行详细讲解。

FillEllipse 方法用来填充 Rectangle 结构指定的边框所定义的椭圆的内部，该方法有多种重载形式，本实例中用到的重载形式如下：

```
public void FillEllipse(Brush brush,Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush 对象。

❷ rect：Rectangle 结构，它表示定义椭圆的边框。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShowImageByEllipse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以椭圆形显示图像操作；添加两个 PictureBox 控件，分别用来显示打开的图像和椭圆形的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.panel1.Refresh();                                          //刷新 Panel 控件
    Graphics g = this.panel1.CreateGraphics();                      //创建绘图对象
    TextureBrush MyBrush = new TextureBrush(MyBitmap);              //创建 TextureBrush 对象
    g.FillEllipse(MyBrush, this.panel1.ClientRectangle);            //绘制椭圆图像
}
```

秘笈心法

心法领悟 509：TextureBrush 类的使用。

TextureBrush 类表示使用图像来填充形状的内部，它的每个属性都是一个 Brush 对象。例如，开发人员可以使用该类的 RotateTransform 方法将指定旋转应用于 Graphics 对象的变换矩阵，从而实现图像以任意角度旋转的功能。

22.4 图像的特殊效果

实例 510	百叶窗效果显示图像	中级
	光盘位置：光盘\MR\22\510	趣味指数：★★★★☆

实例说明

百叶窗效果显示图像，就是将图像分成若干个区域，各个区域的图形以一种渐进的方式逐渐显示，效果就像百叶窗翻动一样。本实例实现了以百叶窗效果显示图像的功能，实例运行效果如图 22.17 所示。

图 22.17 百叶窗效果显示图像

关键技术

本实例在实现百叶窗效果显示图像时，主要通过使用 Bitmap 类的 GetPixel 和 SetPixel 方法获取和设置图像中指定像素的颜色，然后使用 Refresh 重新刷新窗体背景，使其以百叶窗效果显示出来。下面对 Bitmap 类的 GetPixel 和 SetPixel 方法分别进行介绍。

（1）GetPixel 方法

该方法主要用来获取 Bitmap 图像中指定像素的颜色。语法格式如下：

```
public Color GetPixel ( int x, int y)
```

参数说明

❶ x：要检索的像素的 x 坐标。

❷ y：要检索的像素的 y 坐标。

❸ 返回值：Color 结构，表示指定像素的颜色。

（2）SetPixel 方法

该方法主要用来设置 Bitmap 图像中指定像素的颜色。语法格式如下：

```
public void SetPixel ( int x, int y, Color color)
```

参数说明

❶ x：要设置的像素的 x 坐标。

❷ y：要设置的像素的 y 坐标。

❸ color：Color 结构，表示要分配给指定像素的颜色。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 HundredWindow。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以百叶窗效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        Bitmap myBitmap = (Bitmap)this.BackgroundImage.Clone( );                    //用窗体背景的副本创建 Bitmap 对象
        int intWidth = myBitmap.Width;                                              //记录图片的宽度
        int intHeight = myBitmap.Height / 20;                                       //记录图片的指定高度
        Graphics myGraphics = this.CreateGraphics( );                               //创建窗体的 Graphics 对象
        myGraphics.Clear(Color.WhiteSmoke);                                         //用指定的颜色清除窗体背景
        Point[] myPoint = new Point[30];                                            //定义数组
        for (int i = 0; i < 30; i++)                                                //记录百叶窗各节点的位置
        {
            myPoint[i].X = 0;
            myPoint[i].Y = i * intHeight;
        }
        Bitmap bitmap = new Bitmap(myBitmap.Width, myBitmap.Height);                //创建 Bitmap 对象
        //通过调用 Bitmap 对象的 SetPixel 方法重新设置图像的像素点颜色，从而实现百叶窗效果
        for (int m = 0; m < intHeight; m++)
        {
            for (int n = 0; n < 20; n++)
            {
                for (int j = 0; j < intWidth; j++)
                {
                    bitmap.SetPixel(myPoint[n].X + j, myPoint[n].Y + m, myBitmap.GetPixel(myPoint[n].X + j,
myPoint[n].Y + m));                                                                 //获取当前像素颜色值
                }
            }
            this.Refresh( );                                                        //绘制无效
            this.BackgroundImage = bitmap;                                          //显示百叶窗体的效果
            System.Threading.Thread.Sleep(100);                                     //线程挂起
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 510：使用 Thread 类的 Sleep 方法实现线程休眠。

Thread 类是一个线程类，其 Sleep 方法用来将当前线程阻塞指定的毫秒数。例如，本实例中使用 Thread 类的 Sleep 方法使线程休眠 100 毫秒，代码如下：

```
System.Threading.Thread.Sleep(100);                                             //线程挂起
```

实例 511　推拉效果显示图像

光盘位置：光盘\MR\22\511　　中级　　趣味指数：★★★★★

实例说明

推拉效果，顾名思义就是将要显示的图像由上、下、左或右方向在屏幕内显示，以这种效果显示图像可

分为4种不同的方式，即由上至下、由左至右、由下至上或由右至左。本实例实现了由上至下显示一幅图像，实例运行效果如图22.18所示。

图22.18 推拉效果显示图像

关键技术

本实例实现推拉效果显示图像时，首先将图像作为窗体的背景显示出来，然后将该图像中的第一条水平线上的图像复制到将要显示的窗体的最顶端，然后再将该图像中的前两条线上的图像复制到将要显示的窗体的最顶端，接着是第三条，这样依次复制，直到将图像中的所有部分都复制到窗体中为止。在以上实现过程中主要使用Bitmap类的Clone方法来复制图像。

说明：关于Bitmap类的Clone方法的详细讲解请参见实例502中的关键技术。

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为BuntForms。

（2）更改默认窗体Form1的Name属性为Frm_Main，在该窗体中添加一个OpenFileDialog控件，用来显示“打开文件”对话框；添加两个Button控件，分别用来执行打开图像和以推拉效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        Graphics myGraphics = this.CreateGraphics();                          //创建窗体的 Graphics 对象
        Bitmap myBitmap = new Bitmap(myImage);                                //创建 Bitmap 对象
        myGraphics.Clear(this.BackColor);                                     //以指定的颜色清除
        for (int i = 0; i < this.Height; i++)                                 //从上到下遍历图片
        {
            Rectangle rectangle1 = new Rectangle(0, 0, this.Width, i);        //获取图片指定行的像素
            Rectangle rectangle2 = new Rectangle(0, this.Height - i, this.Width, i + 1); //获取图片当前行区域的以下区域
            //通过调用 Bitmap 对象的 Clone 和 DrawImage 方法实现图像的推拉效果
            Bitmap cloneBitmap = myBitmap.Clone(rectangle2, PixelFormat.DontCare);
            myGraphics.DrawImage(cloneBitmap, rectangle1);                    //绘制指定大小的图片
            this.Show();
        }
    }
    catch { }
}
```

秘笈心法

心法领悟511：使用Bitmap类的Clone方法复制图像。

Clone方法用来创建Bitmap对象（由Rectangle结构和指定的PixelFormat枚举定义）的某个部分的副本，也就是人们常说的克隆图像。例如，本实例中使用Clone方法的代码如下：

```
Bitmap cloneBitmap = myBitmap.Clone(rectangle2, PixelFormat.DontCare);
```

实例512 水平交错效果显示图像

光盘位置：光盘\MR\22\512

中级

趣味指数：★★★★☆

实例说明

水平交错效果就是将一幅图像分成左右两部分，然后使它们分别从左右两个方向向窗体中间移动，最终合

并形成一幅图像。本实例实现了水平交错效果显示图像的功能，实例运行效果如图 22.19 所示。

图 22.19　水平交错效果显示图像

关键技术

本实例实现水平交错效果显示图像时，首先将图像分为左右两部分，然后调用 Bitmap 类的 GetPixel 和 SetPixel 方法将这两部分从左右两边同时向窗体输入。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 LevelInterleaving。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以水平交错效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        int intWidth = this.BackgroundImage.Width;                                    //获取背景图片的宽度
        int intHeight = this.BackgroundImage.Height;                                  //获取背景图片的高度
        Graphics myGraphics = this.CreateGraphics( );                                 //创建窗体的 Graphics 对象
        myGraphics.Clear(Color.WhiteSmoke);                                           //以指定的颜色清除
        Bitmap bitmap = new Bitmap(intWidth, intHeight);                              //创建 Bitmap 对象
        int i = 0;
        //通过调用 Bitmap 对象的 SetPixel 方法实现水平交错效果显示图像
        while (i <= intWidth / 2)
        {
            for (int m = 0; m <= intHeight - 1; m++)
            {
                bitmap.SetPixel(i, m, myBitmap.GetPixel(i, m));                       //设置当前像素的颜色值
            }
            for (int n = 0; n <= intHeight - 1; n++)
            {
                bitmap.SetPixel(intWidth - i - 1, n, myBitmap.GetPixel(intWidth - i - 1, n));   //设置当前像素的颜色值
            }
            i++;
            this.Refresh( );                                                          //刷新当前窗体
            this.BackgroundImage = bitmap;                                            //显示水平交错的图像
            System.Threading.Thread.Sleep(10);                                        //线程挂起
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 512：如何清空窗体的背景色？

清空窗体背景色时，可以使用 Graphics 类的 Clear 方法来实现。例如，本实例中清空 GroupBox 控件的背景

色的实现代码如下：

```
Graphics myGraphics = this.CreateGraphics( );                    //创建窗体的 Graphics 对象
myGraphics.Clear(Color.WhiteSmoke);                              //以指定的颜色清除
```

实例 513　垂直交错效果显示图像

光盘位置：光盘\MR\22\513　　中级　　趣味指数：★★★★★

实例说明

垂直交错效果显示图像与水平交错效果一样，只是将一幅图像分成上下两部分，然后使它们分别从上下两个方向向窗体中间移动，最终合并形成一幅图像。本实例实现了垂直交错效果显示图像的功能，实例运行效果如图 22.20 所示。

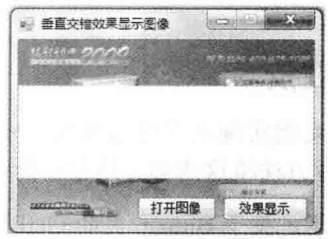


图 22.20　垂直交错效果显示图像

关键技术

本实例实现垂直交错效果显示图像时，首先将图像分为上下两部分，然后调用 Bitmap 类的 GetPixel 和 SetPixel 方法将这两部分从上下两边同时向窗体输入。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 UprightnessInterleaving。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以垂直交错效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        int intWidth = this.BackgroundImage.Width;                                   //获取背景图片的宽度
        int intHeight = this.BackgroundImage.Height;                                 //获取背景图片的高度
        Graphics myGraphics = this.CreateGraphics( );                                //创建窗体的 Graphics 对象
        myGraphics.Clear(Color.WhiteSmoke);                                          //以指定的颜色清除
        Bitmap bitmap = new Bitmap(intWidth, intHeight);                             //创建 Bitmap 对象
        int i = 0;
        //通过调用 Bitmap 对象的 SetPixel 方法实现垂直交错效果显示图像
        while (i <= intHeight / 2)
        {
            for (int m = 0; m <= intWidth - 1; m++)
            {
                bitmap.SetPixel(m, i, myBitmap.GetPixel(m, i));                      //设置当前像素的颜色值
            }
            for (int n = 0; n <= intWidth - 1; n++)
            {
                bitmap.SetPixel(n, intHeight - i - 1, myBitmap.GetPixel(n, intHeight - i - 1));   //设置当前像素的颜色值
            }
            i++;
            this.Refresh( );                                                         //刷新当前窗体
            this.BackgroundImage = bitmap;                                           //显示垂直交错的图像
            System.Threading.Thread.Sleep(10);                                       //线程挂起
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 513：使用 Refresh 方法刷新当前窗体。

Refresh 方法用来强制控件使其工作区无效并立即重绘自己和任何子控件，从而达到刷新窗体的目的，利用该功能，可以实时地在窗体上获取最新的数据。

实例 514	纹理效果显示图像	中级
	光盘位置：光盘\MR\22\514	趣味指数：★★★★☆

实例说明

纹理是一种常见的图像处理技术，用该技术绘制的图像会呈现一些比较有规律的波纹，很多图像处理软件中都提供了此功能。纹理的效果多种多样，变化无常，有很多种算法都能够绘制出纹理，本实例介绍一种简单常用的绘制纹理效果的方法。运行本实例，单击“效果显示”按钮，即可在窗体中以纹理效果显示图像。实例运行效果如图 22.21 所示。

图 22.21　纹理效果显示图像

关键技术

本实例实现纹理效果显示图像时，首先通过使用 Bitmap 对象的 LockBits 方法将图像锁定到内存中，然后通过 BitmapData 对象的 Scan0 属性获得图像中第一个像素数据的地址，最后通过使用 System.Runtime.InteropServices.Marshal 类的 Copy 方法为指定图像的像素点着色，并用变换后的图像作为窗体的新背景。下面对本实例中用到的关键技术进行详细介绍。

（1）Bitmap 对象的 LockBits 方法

该方法主要用来将 Bitmap 图像锁定到系统内存中。语法格式如下：

```
public BitmapData LockBits ( Rectangle rect, ImageLockMode flags, PixelFormat format)
```

LockBits 方法中的参数说明如表 22.8 所示。

表 22.8　LockBits 方法中的参数说明

参　数	描　述
rect	Rectangle 结构，指定要锁定的 Bitmap 图像部分
flags	ImageLockMode 枚举，指定 Bitmap 图像的访问级别（读/写）
format	PixelFormat 枚举，指定 Bitmap 图像的数据格式
返回值	BitmapData，包含有关此锁定操作的信息

（2）BitmapData 对象的 Scan0 属性

该属性主要用来获取或设置位图中第一个像素数据的地址。语法格式如下：

```
public IntPtr Scan0 { get; set; }
```

参数说明

属性值：位图中第一个像素数据的地址。

（3）System.Runtime.InteropServices.Marshal 类的 Copy 方法

System.Runtime.InteropServices.Marshal 类提供了一个方法集，这些方法用于分配非托管内存、复制非托管

内存块和将托管类型转换为非托管类型。另外，还提供了在与非托管代码交互时使用的其他杂项方法，该类的Copy方法用来将一维的托管8位无符号整数数组中的数据复制到非托管内存指针。Copy方法的语法格式如下：

```
public static void Copy ( byte[] source, int startIndex, IntPtr destination, int length)
```

Copy方法中的参数说明如表22.9所示。

表22.9 Copy方法中的参数说明

参　数	描　述
source	从中进行复制的一维数组
startIndex	数组中Copy开始位置的从零开始的索引
destination	要复制到的内存指针
length	要复制的数组元素的数目

设计过程

（1）打开Visual Studio 2012开发环境，新建一个Windows窗体应用程序，并将其命名为ImageGrain。

（2）更改默认窗体Form1的Name属性为Frm_Main，在该窗体中添加一个OpenFileDialog控件，用来显示“打开文件”对话框；添加两个Button控件，分别用来执行打开图像和以纹理效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);          //创建 Image 对象
        myBitmap = new Bitmap(myImage);                                                    //创建 Bitmap 对象
        Rectangle rect = new Rectangle(0, 0, myBitmap.Width, myBitmap.Height);             //创建 Rectangle 对象
        System.Drawing.Imaging.BitmapData bmpData = myBitmap.LockBits(rect,
        System.Drawing.Imaging.ImageLockMode.ReadWrite, myBitmap.PixelFormat);            //将指定图像锁定到内存中
        IntPtr ptr = bmpData.Scan0;                                                        //获得图像中第一个像素数据的地址
        int bytes = myBitmap.Width * myBitmap.Height * 3;                                  //设置大小
        byte[] rgbValues = new byte[bytes];                                                //创建 byte 数组
        //使用 RGB 值为声明的 rgbValues 数组赋值
        System.Runtime.InteropServices.Marshal.Copy(ptr, rgbValues, 0, bytes);
        for (int counter = 0; counter < rgbValues.Length; counter += 3)                    //初始化大小
            rgbValues[counter] = 255;                                                      //使用 RGB 值为图像的像素点着色
        System.Runtime.InteropServices.Marshal.Copy(rgbValues, 0, ptr, bytes);
        myBitmap.UnlockBits(bmpData);                                                      //从内存中解锁图像
        this.BackgroundImage = myBitmap;                                                   //显示设置后的图片
    }
    catch { }
}
```

秘笈心法

心法领悟514：使用BackgroundImage属性设置窗体的背景图片。

BackgroundImage属性用来获取或设置在窗体中显示的背景图像，例如，本实例中通过该属性设置窗体背景图片的代码如下：

```
this.BackgroundImage = myBitmap;                                                           //显示设置后的图片
```

实例515 浮雕效果显示图像

光盘位置：光盘\MR\22\515　　中级　　趣味指数：★★★★☆

实例说明

浮雕效果是一种特殊的图像处理效果，在许多图像处理软件中都可以看到，Photoshop软件中就有处理浮雕

效果的滤镜。经过浮雕效果处理的图像会呈现出一种刻画在石碑上的效果，图像中物体的边缘会呈现一种立体感。本实例实现了以浮雕效果显示图像的功能，实例运行效果如图 22.22 所示。

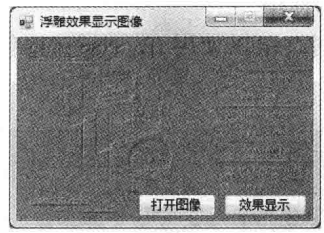


图 22.22　浮雕效果显示图像

关键技术

本实例实现浮雕效果显示图像时，首先通过 Bitmap 对象的 GetPixel 方法获取各像素点的颜色；然后分别使用 Color 对象的 R、G、B 属性获得各像素点的 R、G、B 元素值，并使用这些值减去相邻像素点的元素值再加上 128；最后通过使用 Bitmap 对象的 SetPixel 方法重新为图像的像素点着色，以使得呈现的图像有阶梯感。下面对本实例中用到的关键技术进行详细讲解。

Color 对象的 R 属性用来获取 Color 结构的红色分量值，G 属性用来获取 Color 结构的绿色分量值，B 属性用来获取 Color 结构的蓝色分量值。

说明： *关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解，请参见实例 510 中的关键技术。*

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BassoRelievo。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以浮雕效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        myBitmap = new Bitmap(myImage);                          //创建 Bitmap 对象
        for (int i = 0; i < myBitmap.Width - 1; i++)             //遍历图片中的所有像素
        {
            for (int j = 0; j < myBitmap.Height - 1; j++)
            {
                Color Color1 = myBitmap.GetPixel(i, j);          //获取当前像素的颜色值
                Color Color2 = myBitmap.GetPixel(i + 1, j + 1);  //获取斜点下像素的颜色值
                int red = Math.Abs(Color1.R - Color2.R + 128);   //设置 R 色值
                int green = Math.Abs(Color1.G - Color2.G + 128); //设置 G 色值
                int blue = Math.Abs(Color1.B - Color2.B + 128);  //设置 B 色值
                //颜色处理
                if (red > 255) red = 255;                        //如果 R 色值大于 255，将 R 色值设为 255
                if (red < 0) red = 0;                            //如果 R 色值小于 0，将 R 色值设为 0
                if (green > 255) green = 255;                    //如果 G 色值大于 255，将 R 色值设为 255
                if (green < 0) green = 0;                        //如果 G 色值小于 0，将 R 色值设为 0
                if (blue > 255) blue = 255;                      //如果 B 色值大于 255，将 R 色值设为 255
                if (blue < 0) blue = 0;                          //如果 B 色值小于 0，将 R 色值设为 0
                //通过调用 Bitmap 对象的 SetPixel 方法为图像的像素点重新着色
                myBitmap.SetPixel(i, j, Color.FromArgb(red, green, blue));
            }
        }
        this.BackgroundImage = myBitmap;                         //显示处理后的图片
    }
    catch { }
}
```

秘笈心法

心法领悟 515：多条件判断在程序中的使用。

程序中遇到需要判断多个条件的情况时，可以使用 if...else if 语句，也可以使用 switch 语句，相比较来讲，switch 语句的执行效率更高。

实例 516　底片效果显示图像

光盘位置：光盘\MR\22\516　　中级　　趣味指数：★★★★☆

实例说明

底片效果显示图像其实就是对原有图像取反色。本实例将介绍如何以底片效果显示图像。运行本实例，单击“以底片效果显示图像”按钮，即可在窗体中以底片效果显示图像。实例运行效果如图 22.23 所示。

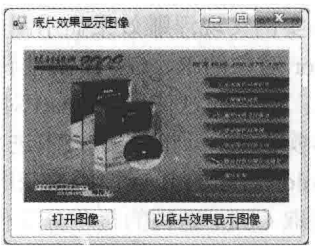


图 22.23　底片效果显示图像

关键技术

本实例实现以底片效果显示图像时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 NegativeImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以底片效果显示图像的功能；添加一个 PictureBox 控件，用来显示原图像和底片效果的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int Height = this.pictureBox1.Image.Height;                    //获取图片高度
        int Width = this.pictureBox1.Image.Width;                      //获取图片宽度
        Bitmap newbitmap = new Bitmap(Width, Height);                  //创建位图对象
        Bitmap oldbitmap = (Bitmap)this.pictureBox1.Image;             //获取原图
        Color pixel;                                                   //定义一个 Color 结构
        for (int x = 1; x < Width; x++)                                //遍历图片的每个位置
        {
            for (int y = 1; y < Height; y++)
            {
                int r, g, b;                                           //定义 3 个变量，用来记录指定点的 R、G、B 值
                pixel = oldbitmap.GetPixel(x, y);                      //获取指定点的像素值
                r = 255 - pixel.R;                                     //记录 R 值
                g = 255 - pixel.G;                                     //记录 G 值
                b = 255 - pixel.B;                                     //记录 B 值
                newbitmap.SetPixel(x, y, Color.FromArgb(r, g, b));     //为指定点重新着色
            }
        }
        this.pictureBox1.Image = newbitmap;                            //显示底片效果的图像
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

秘笈心法

心法领悟 516：使用 Exception 类捕捉异常。

所有.NET 中的异常都可被称为 Exception。虽然说异常多种多样，但基本上它们都是 Exception。这个说法可以打一个简单比喻，就像是无论黑猫、白猫、花猫，它们都是“猫”一样。

实例 517	积木效果显示图像 光盘位置：光盘\MR\22\517	中级 趣味指数：★★★★☆

实例说明

图片的显示方式多种多样，那么如何将一幅图片以积木的效果显示在窗体上呢？本实例通过对图片像素点的明暗度进行数值处理，实现了以积木效果显示图像的功能。实例运行效果如图 22.24 所示。

图 22.24　积木效果显示图像

关键技术

本实例中实现积木效果显示图像时，首先调用 Color 类的 FromArgb 方法定义一组颜色，然后使用 Bitmap 对象的 SetPixel 方法为图像中的各像素点重新着色。下面对本实例中用到的关键技术进行详细介绍。

Color 类的 FromArgb 方法用来从 4 个 ARGB 分量（alpha、红色、绿色和蓝色）值创建 Color 结构。语法格式如下：

```
public static Color FromArgb ( int alpha, int red, int green, int blue)
```

FromArgb 方法中的参数说明如表 22.10 所示。

表 22.10　FromArgb 方法中的参数说明

参　数	描　述
alpha	alpha 分量。有效值为从 0~255
red	红色分量。有效值为从 0~255
green	绿色分量。有效值为从 0~255
blue	蓝色分量。有效值为从 0~255
返回值	从 4 个 ARGB 分量（alpha、红色、绿色和蓝色）值创建的 Color 结构

说明：关于 Bitmap 类的 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BuildingBlock。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以积木效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    Graphics myGraphics = this.CreateGraphics( );                    //创建窗体的 Graphics 对象
    Bitmap myBitmap = new Bitmap(this.BackgroundImage);              //创建 Bitmap 对象
    int myWidth, myHeight, i, j, iAvg, iPixel;                       //定义变量
    Color myColor, myNewColor;                                       //定义颜色变量
    RectangleF myRect;
    myWidth = myBitmap.Width;                                        //获取背景图片的宽度
    myHeight = myBitmap.Height;                                      //获取背景图片的高度
```

```
    myRect = new RectangleF(0, 0, myWidth, myHeight);                                    //获取图片的区域
    Bitmap bitmap = myBitmap.Clone(myRect, System.Drawing.Imaging.PixelFormat.DontCare); //创建 Bitmap 对象
    i = 0;
    while (i < myWidth - 1)                                                              //遍历图片的所有像素
    {
        j = 0;
        while (j < myHeight - 1)
        {
            myColor = bitmap.GetPixel(i, j);                                             //获取当前像素的颜色值
            iAvg = (myColor.R + myColor.G + myColor.B) / 3;                              //平均法
            iPixel = 0;
            if (iAvg >= 128)                                                             //如果颜色值大于等于 128
                iPixel = 255;                                                            //设置为 255
            else
                iPixel = 0;
            //通过调用 Color 对象的 FromArgb 方法获得图像各像素点的颜色
            myNewColor = Color.FromArgb(255, iPixel, iPixel, iPixel);
            bitmap.SetPixel(i, j, myNewColor);                                           //设置颜色值
            j = j + 1;
        }
        i = i + 1;
    }
    myGraphics.Clear(Color.WhiteSmoke);                                                  //以指定的颜色清除
    myGraphics.DrawImage(bitmap, new Rectangle(0, 0, myWidth, myHeight));                //绘制处理后的图片
}
```

秘笈心法

心法领悟 517：使用 Color 结构获取颜色。

Color 结构表示一种 ARGB 颜色（alpha、红色、绿色、蓝色），现在已经命名的颜色都可以使用 Color 结构的属性来表示，例如，蓝色可以用 Color.Blue 来表示。

实例 518　马赛克效果显示图像

中级

光盘位置：光盘\MR\22\518　　趣味指数：★★★★☆

实例说明

马赛克效果显示图像就是随机小块地显示图像，最后合并成一整幅图像。使用马赛克效果显示图像时，关键要注意随机位置点的确定和图像块大小的确定。本实例将介绍如何以马赛克效果显示图像。运行本实例，单击“效果显示”按钮，即可在窗体中以马赛克效果显示图像。实例运行效果如图 22.25 所示。

图 22.25　马赛克效果显示图像

关键技术

本实例通过使用 Bitmap 对象的 GetPixel 和 SetPixel 方法以描点的方式实现了以马赛克效果显示图像的功能。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Mosaic。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加两个 Button 控件，分别用来执行打开图像和以马赛克效果显示图像的功能。

（3）程序的主要代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    Bitmap myBitmap = new Bitmap(this.BackgroundImage);                    //根据窗体的背景创建 Bitmap 对象
    int intWidth = myBitmap.Width / 50;                                    //获取图片的指定宽度
    int intHeight = myBitmap.Height / 50;                                  //获取图片的指定高度
    Graphics myGraphics = this.CreateGraphics();                           //创建窗体的 Graphics 对象
    myGraphics.Clear(Color.WhiteSmoke);                                    //以指定的颜色清除
    Point[] myPoint = new Point[2500];                                     //定义数组
    for (int i = 0; i < 50; i++)                                           //获取指定区域图片的位置
    {
        for (int j = 0; j < 50; j++)
        {
            myPoint[i * 50 + j].X = i * intWidth;
            myPoint[i * 50 + j].Y = j * intHeight;
        }
    }
    Bitmap bitmap = new Bitmap(myBitmap.Width, myBitmap.Height);           //创建 Bitmap 对象
    for (int i = 0; i < 10000; i++)
    {
        Random rand = new Random();                                        //创建 Random 对象
        int intPos = rand.Next(2500);                                      //获取一个随机数
        for (int m = 0; m < intWidth; m++)
        {
            for (int n = 0; n < intHeight; n++)
            {
                //通过调用 Bitmap 对象的 SetPixel 方法为图像的各像素点重新着色
                bitmap.SetPixel(myPoint[intPos].X + m, myPoint[intPos].Y + n, myBitmap.GetPixel(myPoint[intPos].X + m,myPoint[intPos].Y + n));
            }
        }
        this.Refresh();                                                    //工作区无效
        this.BackgroundImage = bitmap;                                     //显示处理后的图片
        for (int k = 0; k < 2500; k++)
        {
            for (int m = 0; m < intWidth; m++)
            {
                for (int n = 0; n < intHeight; n++)
                {
                    //通过调用 Bitmap 对象的 SetPixel 方法为图像的各像素点重新着色
                    bitmap.SetPixel(myPoint[k].X + m, myPoint[k].Y + n, myBitmap.GetPixel(myPoint[k].X + m,myPoint[k].Y + n));
                }
            }
            this.Refresh();                                                //工作区无效
            this.BackgroundImage = bitmap;                                 //显示处理后的图片
        }
    }
}
```

秘笈心法

心法领悟 518：Point 结构的使用。

Point 结构表示在二维平面中定义点的、整数 X 和 Y 坐标的有序对，它位于 System.Drawing 命名空间下。

实例 519　柔化效果显示图像

光盘位置：光盘\MR\22\519　　中级　　趣味指数：★★★★☆

实例说明

柔化效果显示图像时，主要是对当前像素点和周围像素点的颜色进行比较，如果颜色差距较大，则取其平均值，否则取原来的值，然后使用获取的值重新为图像着色。本实例将介绍如何以柔化效果显示图像。运行本实例，单击“以柔化效果显示图像”按钮，即可在窗体中以柔化效果显示图像。实例运行效果如图 22.26 所示。

图 22.26　柔化效果显示图像

关键技术

本实例在实现图像的柔化效果时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SuavityImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以柔化效果显示图像的功能；添加一个 PictureBox 控件，用来显示原图像和柔化效果的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int Height = this.pictureBox1.Image.Height;                    //获取图像的高度
        int Width = this.pictureBox1.Image.Width;                      //获取图像的宽度
        Bitmap bitmap = new Bitmap(Width, Height);                     //创建新的位图对象
        Bitmap MyBitmap = (Bitmap)this.pictureBox1.Image;              //记录原图
        Color pixel;                                                   //定义一个 Color 结构
        int[] Gauss ={ 1, 2, 1, 2, 4, 2, 1, 2, 1 };                    //定义高斯模板值
        for (int x = 1; x < Width - 1; x++)                            //遍历原图的每个位置
            for (int y = 1; y < Height - 1; y++)
            {
                int r = 0, g = 0, b = 0;                               //声明 3 个变量，用来记录 R、G、B 值
                int Index = 0;                                         //声明一个变量，用来记录位置
                for (int col = -1; col <= 1; col++)
                    for (int row = -1; row <= 1; row++)
                    {
                        pixel = MyBitmap.GetPixel(x + row, y + col);   //获取指定点的像素
                        r += pixel.R * Gauss[Index];                   //记录 R 值
                        g += pixel.G * Gauss[Index];                   //记录 G 值
                        b += pixel.B * Gauss[Index];                   //记录 B 值
                        Index++;
                    }
                r /= 16;                                               //为 R 重新赋值
                g /= 16;                                               //为 G 重新赋值
                b /= 16;                                               //为 B 重新赋值
                //处理颜色值溢出
                r = r > 255 ? 255 : r;
                r = r < 0 ? 0 : r;
                g = g > 255 ? 255 : g;
                g = g < 0 ? 0 : g;
                b = b > 255 ? 255 : b;
                b = b < 0 ? 0 : b;
                bitmap.SetPixel(x - 1, y - 1, Color.FromArgb(r, g, b));//重新为指定点赋颜色值
            }
        this.pictureBox1.Image = bitmap;                               //显示柔化效果的图像
    }
    catch (Exception ex)
    {
```

```
            MessageBox.Show(ex.Message, "信息提示");
        }
    }
```

秘笈心法

心法领悟 519：程序中尽量不要使用太多嵌套 for 语句。

由于嵌套 for 语句将消耗很大的资源，所以在实际开发项目时，能不使用嵌套 for 语句尽量不要使用。

实例 520	雾化效果显示图像 光盘位置：光盘\MR\22\520	中级 趣味指数：★★★★☆

实例说明

雾化效果处理的不是基于图像中像素点之间的运算，而是在图像中引入一个随机性，按照指定的像素块大小，使图片具有类似于玻璃上的水雾效果。本实例将介绍如何以雾化效果显示图像。运行本实例，单击“效果”按钮，即可在窗体中以雾化效果显示图像。实例运行效果如图 22.27 所示。

图 22.27　雾化效果显示图像

关键技术

本实例在实现图像的雾化效果时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AtomizationImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行以雾化效果显示图像的功能；添加一个 PictureBox 控件，用来显示雾化效果的图像。

（3）程序的主要代码如下：

```
public Image AtomizationEffect(PictureBox Pict, int effect)
{
    int Var_W = Pict.Width;                                          //获取图片的宽度
    int Var_H = Pict.Height;                                         //获取图片的高度
    Bitmap Var_bmp = new Bitmap(Var_W, Var_H);                       //根据图片的大小创建 Bitmap 对象
    Bitmap Var_SaveBmp = (Bitmap)Pict.Image;                         //根据图片创建 Bitmap 对象
    for (int i = 0; i < Var_W; i++)                                  //遍历图片中的各像素
    {
        for (int j = 0; j < Var_H; j++)
        {
            Random Var_random = new Random();                        //创建 Random 对象
            int k = Var_random.Next(200000);                         //获取随机数
            //获取像素块
            int tem_w = i + k % effect;
            int tem_h = j + k % effect;
            //获取的像素不超出图片的大小
            if (tem_w >= Var_W)
                tem_w = Var_W - 1;
            if (tem_h >= Var_H)
                tem_h = Var_H - 1;
            Color tem_Color = Var_SaveBmp.GetPixel(tem_w, tem_h);    //获取当前像素的颜色
            Var_bmp.SetPixel(i, j, tem_Color);                       //设置当前像素的颜色
        }
    }
```

```
    return Var_bmp;
}
```

秘笈心法

心法领悟 520：使用 Random 类生成随机数。

Random 类表示伪随机数生成器，它是一种能够产生满足某些随机性统计要求的数字序列的设备，其 Next 方法用来返回一个指定范围内的随机数。例如，本实例中生成随机数的实现代码如下：

```
Random Var_random = new Random();                    //创建 Random 对象
int k = Var_random.Next(200000);                     //获取随机数
```

实例 521	锐化效果显示图像	中级
	光盘位置：光盘\MR\22\521	趣味指数：★★★★☆

实例说明

锐化效果显示图像就是改变图像像素点中颜色值变化较大的地方，在颜色值变化较小的地方，颜色值改变不大，而在图像的边缘，由于相邻两个像素的颜色值变化较大，所以在锐化后，颜色值变化比较大。本实例将介绍如何以锐化效果显示图像。运行本实例，单击“效果”按钮，即可在窗体中以锐化效果显示图像。实例运行效果如图 22.28 所示。

图 22.28　锐化效果显示图像

关键技术

本实例在实现图像的锐化效果时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AcuteImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行以锐化效果显示图像的功能；添加一个 PictureBox 控件，用来显示锐化效果的图像。

（3）程序的主要代码如下：

```
public Image AcuteEffect(PictureBox Pict)
{
    int Var_W = Pict.Width;                                          //获取图片的宽度
    int Var_H = Pict.Height;                                         //获取图片的高度
    Bitmap Var_bmp = new Bitmap(Var_W, Var_H);                       //根据图片的大小创建 Bitmap 对象
    Bitmap Var_SaveBmp = (Bitmap)Pict.Image;                         //根据图片创建 Bitmap 对象
    int[] Laplacian = { -1, -1, -1, -1, 9, -1, -1, -1, -1 };         //拉普拉斯模板
    //遍历图像中的各像素
    for (int i = 1; i < Var_W-1; i++)
        for (int j = 1; j < Var_H-1; j++)
        {
            int tem_r = 0, tem_g = 0, tem_b = 0, tem_index = 0;      //定义变量
            for (int c=-1;c<=1;c++)
                for (int r = -1; r <= 1; r++)
                {
                    Color tem_color = Var_SaveBmp.GetPixel(i + r, j + c);   //获取指定像素的颜色值
                    tem_r += tem_color.R * Laplacian[tem_index];            //设置 R 色值
                    tem_g += tem_color.G * Laplacian[tem_index];            //设置 G 色值
                    tem_b += tem_color.B * Laplacian[tem_index];            //设置 B 色值
                    tem_index++;
```

```
            }
            tem_r = tem_r > 255 ? 255 : tem_r;                    //如果 R 色值大于 255，将 R 色值设为 255，否则不变
            tem_r = tem_r < 0 ? 0 : tem_r;                        //如果 R 色值小于 0，将 R 色值设为 0，否则不变
            tem_g = tem_g > 255 ? 255 : tem_g;                    //如果 G 色值大于 255，将 R 色值设为 255，否则不变
            tem_g = tem_g < 0 ? 0 : tem_g;                        //如果 G 色值小于 0，将 R 色值设为 0，否则不变
            tem_b = tem_b > 255 ? 255 : tem_b;                    //如果 B 色值大于 255，将 R 色值设为 255，否则不变
            tem_b = tem_b < 0 ? 0 : tem_b;                        //如果 B 色值小于 0，将 R 色值设为 0，否则不变
            Var_bmp.SetPixel(i - 1, j - 1, Color.FromArgb(tem_r, tem_g, tem_b));   //设置指定像素的颜色
        }
    return Var_bmp;
}
```

秘笈心法

心法领悟 521：合理使用三元运算符。

三元运算符又叫条件运算符（ ?: ），它会根据布尔型表达式的值返回两个值中的一个。如果条件为 true，则计算第一表达式并以它的计算结果为准；如果为 false，则计算第二表达式并以它的计算结果为准。在程序的开发过程中，开发人员可以适当地使用三元运算符来代替 if 语句，这样会使代码更加简洁清晰。

实例 522　黑白效果显示图像

光盘位置：光盘\MR\22\522　　中级　　趣味指数：★★★★☆

实例说明

黑白效果显示图像就是将一个彩色图片，按照各像素的 RGB 颜色值，通过平均值法，将其改变成比较柔和的黑白图像。本实例将介绍如何以黑白效果显示图像。运行本实例，单击“效果”按钮，即可在窗体中以黑白效果显示图像。实例运行效果如图 22.29 所示。

图 22.29　黑白效果显示图像

关键技术

本实例在实现图像的黑白效果时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明： 关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BlackandWhiteImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行以黑白效果显示图像的功能；添加一个 PictureBox 控件，用来显示黑白效果的图像。

（3）程序的主要代码如下：

```
public Image BlackandWhiteEffect(PictureBox Pict)
{
    int Var_H = Pict.Image.Height;                        //获取图像的高度
    int Var_W = Pict.Image.Width;                         //获取图像的宽度
    Bitmap Var_bmp = new Bitmap(Var_W, Var_H);            //根据图像的大小创建 Bitmap 对象
    Bitmap Var_SaveBmp = (Bitmap)Pict.Image;              //根据图像创建 Bitmap 对象
    //遍历图像的像素
    for (int i = 0; i < Var_W; i++)
        for (int j = 0; j < Var_H; j++)
        {
            Color tem_color = Var_SaveBmp.GetPixel(i, j);     //获取当前像素的颜色值
            int tem_r, tem_g, tem_b, tem_Value = 0;           //定义变量
```

```
            tem_r = tem_color.R;                                                    //获取 R 色值
            tem_g = tem_color.G;                                                    //获取 G 色值
            tem_b = tem_color.B;                                                    //获取 B 色值
            tem_Value = ((tem_r + tem_g + tem_b) / 3);                              //用平均值法产生黑白图像
            Var_bmp.SetPixel(i, j, Color.FromArgb(tem_Value, tem_Value, tem_Value)); //改变当前像素的颜色值
        }
    return Var_bmp;
}
```

秘笈心法

心法领悟 522：return 语句的使用。

return 语句用于退出类的方法，并控制返回方法的调用者，如果方法有返回类型，return 语句必须返回这个类型的值；如果方法没有返回类型，应使用没有表达式的 return 语句。

实例 523　光晕效果显示图像

光盘位置：光盘\MR\22\523　　中级　　趣味指数：★★★★☆

实例说明

光晕效果显示图像就是在图片中以指定的中心点，设置一个圆，并按设置好的亮度值将圆中各像素的颜色值变亮，以形成光晕效果。本实例将介绍如何以光晕效果显示图像。运行本实例，单击“效果”按钮，即可在窗体中以光晕效果显示图像。实例运行效果如图 22.30 所示。

图 22.30　光晕效果显示图像

关键技术

本实例在实现图像的光晕效果时，主要用到了 Bitmap 类的 GetPixel 和 SetPixel 方法。

说明：关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WatermarkImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来执行以光晕效果显示图像的功能；添加一个 PictureBox 控件，用来显示光晕效果的图像。

（3）程序的主要代码如下：

```
public Image pp(PictureBox Pict, int x, int y, int R, float better)          //R 强光照射面的半径，即“光晕”
{
    Bitmap Var_Bmp = new Bitmap(Pict.Image, Pict.Width, Pict.Height);        //根据图像创建 Bitmap 对象
    int tem_W = Var_Bmp.Width;                                               //获取图像的宽度
    int tem_H = Var_Bmp.Height;                                              //获取图像的高度
    //定义一个 Bitmap 类的副本
```

```
    Bitmap Var_SaveBmp = Var_Bmp.Clone(new RectangleF(0, 0, tem_W, tem_H), System.Drawing.Imaging.PixelFormat.DontCare);
    Point Var_Center = new Point(x, y);                                    //光晕的中心点
    //遍历图像中的各像素
    for (int i = tem_W - 1; i >= 1; i--)
    {
        for (int j = tem_H - 1; j >= 1; j--)
        {
            //设置光晕的范围
            float Var_Length = (float)Math.Sqrt(Math.Pow((i - Var_Center.X), 2) + Math.Pow((j - Var_Center.Y), 2));
            if (Var_Length < R)                                            //如果像素位于“光晕”之内
            {
                Color Var_Color = Var_SaveBmp.GetPixel(i, j);
                int r, g, b;
                float Var_Pixel = better * (1.0f - Var_Length / R);        //设置光亮度的强弱
                r = Var_Color.R + (int)Var_Pixel;                          //设置加强后的 R 值
                r = Math.Max(0, Math.Min(r, 255));                         //如果 R 值不在颜色值的范围内，对 R 值进行设置
                g = Var_Color.G + (int)Var_Pixel;                          //设置加强后的 G 值
                g = Math.Max(0, Math.Min(g, 255));                         //如果 G 值不在颜色值的范围内，对 G 值进行设置
                b = Var_Color.B + (int)Var_Pixel;                          //设置加强后的 B 值
                b = Math.Max(0, Math.Min(b, 255));                         //如果 B 值不在颜色值的范围内，对 B 值进行设置
                Var_SaveBmp.SetPixel(i, j, Color.FromArgb(255, r, g, b));  //将增亮后的像素值回写到位图
            }
        }
    }
    return Var_SaveBmp;
}
```

秘笈心法

心法领悟 523：巧用 Math 类。

Math 类为三角函数、对数函数和其他通用数学函数提供常数和静态方法，通过调用该类中的方法，可以很方便地实现各种数学计算。例如，本实例中通过调用该类的 Sqrt 和 Pow 方法计算光晕范围。代码如下：

```
float Var_Length = (float)Math.Sqrt(Math.Pow((i - Var_Center.X), 2) + Math.Pow((j - Var_Center.Y), 2));
```

实例 524　分块效果显示图像

光盘位置：光盘\MR\22\524　　中级　　趣味指数：★★★★★

实例说明

分块效果显示图像其实就是将图像分成几部分，然后逐部分显示出来的过程。本实例将介绍如何以分块效果显示图像。运行本实例，单击“以分块效果显示图像”按钮，即可在窗体中以分块效果显示图像。实例运行效果如图 22.31 所示。

图 22.31　分块效果显示图像

关键技术

本实例在实现图像的分块效果时，主要用到了 Bitmap 类的 Clone 方法和 Graphics 类的 DrawImage 方法。

说明：（1）关于 Bitmap 类的 Clone 方法的详细讲解请参见实例 502 中的关键技术。

（2）关于 Graphics 类的 DrawImage 方法的详细讲解请参见实例 500 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PartsImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以分块效果显示图像操作；添加两个 PictureBox 控件，分别用来显示打开的图像和分块效果的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.panel1.CreateGraphics();                    //创建绘图对象
    g.Clear(Color.White);                                         //清空背景色
    int width = MyBitmap.Width;                                   //获取图片的宽度
    int height = MyBitmap.Height;                                 //获取图片的高度
    //定义将图片切分成 4 个部分的区域
    RectangleF[] block ={
      new RectangleF(0,0,width/2,height/2),
      new RectangleF(width/2,0,width/2,height/2),
      new RectangleF(0,height/2,width/2,height/2),
      new RectangleF(width/2,height/2,width/2,height/2)};
    //分别克隆图片的 4 个部分
    Bitmap[] MyBitmapBlack ={
        MyBitmap.Clone(block[0],System.Drawing.Imaging.PixelFormat.DontCare),
        MyBitmap.Clone(block[1],System.Drawing.Imaging.PixelFormat.DontCare),
        MyBitmap.Clone(block[2],System.Drawing.Imaging.PixelFormat.DontCare),
        MyBitmap.Clone(block[3],System.Drawing.Imaging.PixelFormat.DontCare)};
    //绘制图片的 4 个部分，各部分绘制时间间隔为 0.5 秒
    g.DrawImage(MyBitmapBlack[0], 0, 0);
    System.Threading.Thread.Sleep(500);
    g.DrawImage(MyBitmapBlack[1], width / 2, 0);
    System.Threading.Thread.Sleep(500);
    g.DrawImage(MyBitmapBlack[3], width / 2, height / 2);
    System.Threading.Thread.Sleep(500);
    g.DrawImage(MyBitmapBlack[2], 0, height / 2);
}
```

秘笈心法

心法领悟 524：使用 Thread 类的 Sleep 方法实现线程休眠。

Thread 类用来创建并控制线程，设置其优先级并获取其状态，其 Sleep 方法用来将当前线程阻塞指定的毫秒数。例如，本实例中使用 Thread 类的 Sleep 方法使线程休眠 500 毫秒。代码如下：

```
System.Threading.Thread.Sleep(500);
```

实例 525 油画效果显示图像

光盘位置：光盘\MR\22\525

中级

趣味指数：★★★★☆

实例说明

当人们对一张彩色图像的某一范围内的像素进行随机处理时，就会产生油画效果。本实例将介绍如何以油画效果显示图像。运行本实例，单击“以油画效果显示图像”按钮，即可在窗体中以油画效果显示图像。实例运行效果如图 22.32 所示。

图 22.32　油画效果显示图像

关键技术

本实例在实现图像的油画效果时，主要用到了 Bitmap 类的 Clone、GetPixel 和 SetPixel 方法以及 Graphics 类的 DrawImage 方法。

说明：（1）关于 Bitmap 类的 Clone 方法的详细讲解请参见实例 502 中的关键技术。

（2）关于 Bitmap 类的 GetPixel 和 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

（3）关于 Graphics 类的 DrawImage 方法的详细讲解请参见实例 500 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CanvasImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以油画效果显示图像操作；添加两个 PictureBox 控件，分别用来显示打开的图像和油画效果的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.panel1.CreateGraphics();                          //创建绘图对象
    int width = MyBitmap.Width;                                         //获取图像的宽度
    int height = MyBitmap.Height;                                       //获取图像的高度
    RectangleF rect = new RectangleF(0, 0, width, height);              //定义一个 RectangleF 结构
    Bitmap img = MyBitmap.Clone(rect, System.Drawing.Imaging.PixelFormat.DontCare);
    Random rnd = new Random();                                          //产生随机数序列
    //取不同的值决定油画效果的不同程度
    int iModel = 2;
    int i = width - iModel;
    while (i > 1)
    {
        int j = height - iModel;
        while (j > 1)
        {
            int iPos = rnd.Next(100000) % iModel;
            //将该点的 RGB 值设置成附近 iModel 点之内的任一点
            Color color = img.GetPixel(i + iPos, j + iPos);
            img.SetPixel(i, j, color);                                  //重新为指定像素点着色
            j = j - 1;
        }
        i = i - 1;
    }
    g.Clear(Color.White);                                               //清空背景
    g.DrawImage(img, new Rectangle(0, 0, width, height));               //重新绘制图像
}
```

秘笈心法

心法领悟 525：do while 循环与 while 循环有什么不同吗？

do while 与 while 语句的运行方式基本相同，只是 do while 语句要先执行一次循环体的内容，然后再判断布尔条件，如果满足条件则执行循环体，不满足布尔条件时退出循环体。

实例 526　四周扩散形式显示图像　　中级

光盘位置：光盘\MR\22\526　　趣味指数：★★★★☆

实例说明

以四周扩散形式显示图像主要是按一定的比例将图片进行缩放，运行本实例，单击“打开图像”按钮，选择指定的图像，单击“以四周扩散方式显示图像”按钮，将选择的图像以四周扩散形式进行显示。实例运行效果如图 22.33 所示。

图 22.33　四周扩散形式显示图像

关键技术

本实例在实现以四周扩散形式显示图像时，主要用到了 Graphics 类的 DrawImage 方法。

说明：关于 Graphics 类的 DrawImage 方法的详细讲解请参见实例 500 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DiffuseImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开图像和以四周扩散形式显示图像操作；添加两个 PictureBox 控件，分别用来显示打开的图像和以四周扩散形式显示的图像。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        int width = this.MyBitmap.Width;                                          //图像宽度
        int height = this.MyBitmap.Height;                                        //图像高度
        Graphics g = this.panel1.CreateGraphics();                                //创建 Graphics 对象
        g.Clear(Color.Gray);                                                      //初始为全灰色
        for (int i = 0; i <= width / 2; i++)
        {
            int j = Convert.ToInt32(i * (Convert.ToSingle(height) / Convert.ToSingle(width)));   //获取高和宽的比例
            Rectangle DestRect = new Rectangle(width / 2 - i, height / 2 - j, 2 * i, 2 * j);     //设置缩小后图片的大小
            Rectangle SrcRect = new Rectangle(0, 0, MyBitmap.Width, MyBitmap.Height);            //获取原图片大小
            g.DrawImage(MyBitmap, DestRect, SrcRect, GraphicsUnit.Pixel);         //按照指定的大小绘制原图片
            System.Threading.Thread.Sleep(10);                                    //线程挂起
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "信息提示");
    }
}
```

秘笈心法

心法领悟 526：Rectangle 结构的使用。

Rectangle 结构用来存储一组整数，共 4 个，表示一个矩形的位置和大小，该结构位于 System.Drawing 命名空间下。

22.5　特殊效果的文字

实例 527	倒影效果的文字 光盘位置：光盘\MR\22\527	中级 趣味指数：★★★★☆

实例说明

倒影效果文字就是在文字的下面显示其文字的倒影效果。运行本实例，单击“倒影效果”按钮，绘制指定的文字，并在文字的下面绘制其倒影效果。实例运行效果如图 22.34 所示。

图 22.34　倒影效果的文字

关键技术

本实例主要通过使用 Graphics 对象的 MeasureString 和 ScaleTransform 方法来绘制倒影效果的文字。下面对本实例中用到的关键技术进行详细讲解。

（1）MeasureString 方法

该方法用于测量指定的 Font 格式绘制的字符串。语法格式如下：

```
public SizeF MeasureString (string text, Font font)
```

参数说明

❶ text：要测量的字符串。

❷ font：Font，它定义字符串的文本格式。

❸ 返回值：SizeF 结构，该结构表示 text 参数指定的、使用 font 参数绘制的字符串的大小，单位由 PageUnit 属性指定。

（2）ScaleTransform 方法

该方法将指定的缩放操作（将该对象的变换矩阵左乘该缩放矩阵）应用于 Graphics 对象的变换矩阵。语法格式如下：

```
public void ScaleTransform (float sx,float sy)
```

参数说明

❶ sx：x 方向的比例因子。

❷ sy：y 方向的比例因子。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InvertedImageCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制倒影效果的文字；添加一个 Panel 控件，用来显示绘制的倒影效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
```

```
    Graphics g = panel1.CreateGraphics();                                    //创建控件的 Graphics 类
    g.Clear(Color.White);                                                    //以指定的颜色清除控件背景
    Brush Var_Brush_Back = Brushes.Gray;                                     //设置前景色
    Brush Var_Brush_Fore = Brushes.Black;                                    //设置背景色
    Font Var_Font = new Font("宋体", 40);                                    //设置字体样式
    string Var_Str = "倒影效果的文字";                                       //设置字符串
    SizeF Var_Size = g.MeasureString(Var_Str, Var_Font);                     //获取字符串的大小
    g.DrawString(Var_Str, Var_Font, Var_Brush_Fore, 0, 0);                   //绘制文本
    g.ScaleTransform(1, -1.0F);                                              //缩放变换矩阵
    g.DrawString(Var_Str, Var_Font, Var_Brush_Back, 0, -Var_Size.Height * 1.6F);  //绘制倒影文本
}
```

秘笈心法

心法领悟 527：使用 DrawString 方法绘制水印文字。

对于一些有可能侵权的图片，或者要作为宣传的图片，可以通过使用 DrawString 方法将用到的文字绘制到图片上，从而得到所谓的“水印文字”。

实例 528　投影效果的文字　　中级

光盘位置：光盘\MR\22\528　　趣味指数：★★★★☆

实例说明

投影效果文字就是在文字的后面显示其文字的投影效果。运行本实例，单击“投影效果”按钮，绘制指定的文字，并在文字的后面绘制其投影效果。实例运行效果如图 22.35 所示。

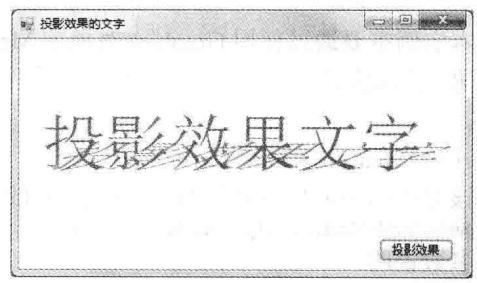


图 22.35　投影效果的文字

关键技术

本实例主要使用 Matrix 对象的 Scale、Shear 和 Translate 方法实现文字的投影效果。下面对本实例中用到的关键技术进行详细讲解。

（1）Scale 方法

该方法通过预先计算比例向量，将指定的比例向量应用到 Matrix 对象。语法格式如下：

```
public void Scale (float scaleX,float scaleY)
```

参数说明

❶ scaleX：Matrix 对象在 X 轴方向缩放的值。

❷ scaleY：Matrix 对象在 Y 轴方向缩放的值。

（2）Shear 方法

该方法通过预先计算切变变换，将指定的切变向量应用到 Matrix 对象。语法格式如下：

```
public void Shear (float shearX,float shearY)
```

参数说明

❶ shearX：水平切变因子。

❷ shearY：垂直切变因子。

（3）Translate 方法

该方法通过预先计算转换向量，将指定的转换向量（offsetX 和 offsetY）应用到 Matrix 对象。语法格式如下：

```
public void Translate (float offsetX,float offsetY)
```

参数说明

❶ offsetX：x 值，通过它转换 Matrix 对象。

❷ offsetY：y 值，通过它转换 Matrix 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ProjectionCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制投影效果的文字；添加一个 Panel 控件，用来显示绘制的投影效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                                    //创建 Graphics 对象
    g.Clear(Color.White);                                                    //以白色清空 panel1 的背景
    g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.ClearTypeGridFit;  //设置文本输出的质量
    g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;      //消除绘制时出现的锯齿
    Font Var_Font = new Font("宋体", 48);                                    //定义文字的字体
    Matrix Var_Matrix = new Matrix();                                        //创建 Matrix 对象
    Var_Matrix.Shear(-1.4F, 0.0F);                                           //设置投影
    Var_Matrix.Scale(1, 0.5F);                                               //设置缩放
    Var_Matrix.Translate(168, 118);                                          //设置平移
    g.Transform = Var_Matrix;                                                //设置坐标平面变换
    SolidBrush Var_Brush_1 = new SolidBrush(Color.Gray);                     //设置文字的画刷
    SolidBrush Var_Brush_2 = new SolidBrush(Color.SlateBlue);                //设置投影的画刷
    string Var_Str = "投影效果文字";                                         //设置文字
    g.DrawString(Var_Str, Var_Font, Var_Brush_1, new PointF(0, 60));         //绘制投影
    g.ResetTransform();                                                      //变换矩阵重置为单位矩阵
    g.DrawString(Var_Str, Var_Font, Var_Brush_2, new PointF(0, 60));         //绘制文字
}
```

秘笈心法

心法领悟 528：调用控件的 CreateGraphics 方法创建 Graphics 对象。

在程序中绘图时，可以调用控件的 CreateGraphics 方法来获取对 Graphics 对象的引用，该对象表示控件的绘图画面。例如，本实例中创建 Graphics 对象的代码如下：

```
Graphics g = panel1.CreateGraphics();                                        //创建 Graphics 对象
```

实例 529　印版效果的文字

光盘位置：光盘\MR\22\529　　中级　　趣味指数：★★★★☆

实例说明

印版效果文字实际上就是让文字看上去具有一定的厚度。运行本实例，单击“效果”按钮，绘制指定厚度的文字，使其具有印版效果。实例运行效果如图 22.36 所示。

图 22.36　印版效果的文字

关键技术

本实例主要是使用 Graphics 对象的 DrawString 方法在指定的位置绘制文字，直到其具有一定的厚度，下面对其进行详细介绍。

DrawString 方法主要是通过 Brush 和 Font 对象在指定位置绘制文本字符串。语法格式如下：

```
public void DrawString (string s, Font font, Brush brush, PointF point)
```

DrawString 方法中的参数说明如表 22.11 所示。

表 22.11　DrawString 方法中的参数说明

参　数	描　述
s	要绘制的字符串
font	Font 对象，指定字符串的文本格式
brush	Brush 对象，指定绘制文本的颜色和纹理
point	PointF 结构，指定绘制文本的左上角的坐标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BlockCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制印版效果的文字；添加一个 Panel 控件，用来显示绘制的印版效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                          //创建控件的 Graphics 对象
    g.Clear(Color.White);                                          //以指定的颜色清除控件背景
    Brush Var_Brush_Back = Brushes.Black;                          //设置前景色
    Font Var_Font = new Font("宋体", 40);                          //设置字体样式
    string Var_Str = "印版效果的文字";                             //设置字符串
    SizeF Var_Size = g.MeasureString(Var_Str, Var_Font);           //获取字符串的大小
    int Var_X = (panel1.Width - Convert.ToInt32(Var_Size.Width)) / 2;   //设置平移的 X 坐标
    int Var_Y = (panel1.Height - Convert.ToInt32(Var_Size.Height)) / 2; //设置平移的 Y 坐标
    //实现印版文字
    for (int i = 0; i < 10; i++)
        g.DrawString(Var_Str, Var_Font, Var_Brush_Back, Var_X - i, Var_Y + i);
    g.DrawString(Var_Str, Var_Font, Var_Brush_Back, Var_X, Var_Y);  //绘制文字
}
```

秘笈心法

心法领悟 529：使用 Convert.ToInt32 方法转换 int 类型。

Convert 类用来将一个基本数据类型转换为另一个基本数据类型，其 ToInt32 方法用来将指定的值转换为 32 位有符号整数。

实例 530　阴影效果的文字　　中级

光盘位置：光盘\MR\22\530　　趣味指数：★★★★☆

实例说明

阴影效果文字就是在文字的后面显示其文字的阴影效果。运行本实例，单击“效果”按钮，绘制指定的文字，并在文字的后面绘制其阴影效果。实例运行效果如图 22.37 所示。

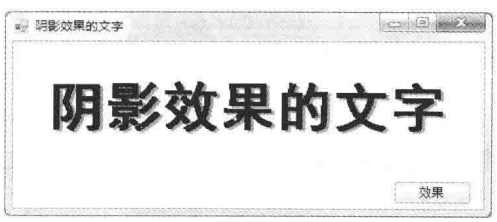


图 22.37　阴影效果的文字

关键技术

本实例主要使用 Graphics 对象的 DrawString 方法在指定的位置绘制阴影效果的文字。

说明：关于 Graphics 对象的 DrawString 方法的详细讲解请参见实例 529 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShadowCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制阴影效果的文字；添加一个 Panel 控件，用来显示绘制的阴影效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                                    //创建控件的 Graphics 对象
    g.Clear(Color.White);                                                    //以指定的颜色清除控件背景
    Brush Var_Brush_Back = Brushes.Gray;                                     //设置前景色
    Brush Var_Brush_Fore = Brushes.Black;                                    //设置背景色
    Font Var_Font = new Font("黑体", 40,FontStyle.Bold);                     //设置字体样式
    string Var_Str = "阴影效果的文字";                                       //设置字符串
    SizeF Var_Size = g.MeasureString(Var_Str, Var_Font);                     //获取字符串的大小
    int Var_X = (panel1.Width - Convert.ToInt32(Var_Size.Width)) / 2;        //设置平移的 X 坐标
    int Var_Y = (panel1.Height - Convert.ToInt32(Var_Size.Height)) / 2;      //设置平移的 Y 坐标
    g.DrawString(Var_Str, Var_Font, Var_Brush_Back, Var_X + 3, Var_Y + 2);   //绘制文字的阴影
    g.DrawString(Var_Str, Var_Font, Var_Brush_Fore, Var_X, Var_Y);           //绘制文字
}
```

秘笈心法

心法领悟 530：如何清空控件的背景色？

清空控件背景色时，可以使用 Graphics 类的 Clear 方法来实现。例如，本实例中清空 Panel 控件的背景色的实现代码如下：

```
Graphics g = panel1.CreateGraphics();                                        //创建控件的 Graphics 对象
g.Clear(Color.White);                                                        //以指定的颜色清除控件背景
```

实例 531　倾斜效果的文字

光盘位置：光盘\MR\22\531　　中级　　趣味指数：★★★★☆

实例说明

倾斜效果的文字就是将文字设置成具有一定的倾斜效果。运行本实例，单击“效果”按钮，绘制指定倾斜度的文字。实例运行效果如图 22.38 所示。

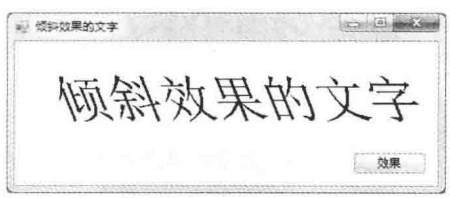


图 22.38　倾斜效果的文字

关键技术

本实例实现时，首先使用 Matrix 对象的 Shear 方法设置文字的倾斜度，然后使用 Graphics 对象的 DrawString 方法绘制倾斜文字。

说明：（1）关于 Matrix 对象的 Shear 方法的详细讲解请参见实例 528 中的关键技术。

（2）关于 Graphics 对象的 DrawString 方法的详细讲解请参见实例 529 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InclineCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制倾斜效果的文字；添加一个 Panel 控件，用来显示绘制的倾斜效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                                    //创建控件的 Graphics 对象
    g.Clear(Color.White);                                                    //以指定的颜色清除控件背景
    Brush Var_Brush_Back = Brushes.Black;                                    //设置前景色
    Brush Var_Brush_Fore = Brushes.Aquamarine;                               //设置背景色
    Font Var_Font = new Font("宋体", 40);                                    //设置字体样式
    string Var_Str = "倾斜效果的文字";                                       //设置字符串
    SizeF Var_Size = g.MeasureString(Var_Str, Var_Font);                     //获取字符串的大小
    int Var_X = (panel1.Width - Convert.ToInt32(Var_Size.Width)) / 2;        //设置平移的 X 坐标
    int Var_Y = (panel1.Height - Convert.ToInt32(Var_Size.Height)) / 2;      //设置平移的 Y 坐标
    g.TranslateTransform(Var_X, Var_Y);                                      //更改坐标系原点
    Matrix Var_Trans = g.Transform;                                          //获取几何世界的变换副本
    Var_Trans.Shear(0.40F, 0.00F);                                           //通过预先计算切变变换
    g.Transform = Var_Trans;                                                 //文字的左倾斜
    g.DrawString(Var_Str, Var_Font, Var_Brush_Back, 5, 5);                   //绘制文字
}
```

秘笈心法

心法领悟 531：使用 Graphics 对象的 Transform 属性实现全局变换。

全局变换是应用于由给定 Graphics 对象绘制的每个项目的变换。创建全局变换时，可以通过操作 Graphics 对象的 Transform 属性来实现，Transform 属性是 Matrix 类的对象。

实例 532　渐变效果的文字

光盘位置：光盘\MR\22\532　　中级　　趣味指数：★★★★☆

实例说明

渐变效果的文字就是在绘制文字时，使文字的前景色具有渐变的效果。运行本实例，单击“效果”按钮，绘制具有渐变颜色的文字。实例运行效果如图 22.39 所示。

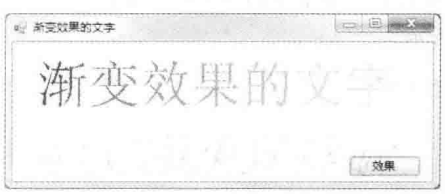


图 22.39　渐变效果的文字

关键技术

本实例主要使用 LinearGradientBrush 类来设置文字的渐变效果，下面对其进行详细介绍。

LinearGradientBrush 类使用线性渐变封装 Brush（画刷），也就是封装双色渐变和自定义多色渐变。所有渐变都是由矩形的宽度或两个点指定的直线绘制的。默认情况下，双色渐变是沿指定直线从起始色到结束色的均

匀水平线性混合。

技巧：使用 Blend 类、SetSigmaBellShape 方法或 SetBlendTriangularShape 方法可以自定义混合图案。

下面是 LinearGradientBrush 的构造函数，它主要是根据一个矩形、起始颜色和结束颜色以及方向模式，创建 LinearGradientBrush 类的新实例。语法格式如下：

```
public LinearGradientBrush (RectangleF rect, Color color1, Color color2, LinearGradientMode linearGradientMode)
```

LinearGradientBrush 构造函数语法中的参数说明如表 22.12 所示。

表 22.12　LinearGradientBrush 构造函数语法中的参数说明

参　数	描　述
rect	一个 RectangleF 结构，它指定线性渐变的界限
color1	表示渐变起始色的 Color 结构
color2	表示渐变结束色的 Color 结构
linearGradientMode	LinearGradientMode 枚举值之一，它指定渐变方向，渐变方向决定渐变的起点和终点。LinearGradientMode 枚举值及说明如表 22.13 所示

表 22.13　LinearGradientMode 枚举值及说明

枚 举 值	描　述
BackwardDiagonal	指定从右上到左下的渐变
ForwardDiagonal	指定从左上到右下的渐变
Horizontal	指定从左到右的渐变
Vertical	指定从上到下的渐变

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ShadeCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制渐变效果的文字；添加一个 Panel 控件，用来显示绘制的渐变效果文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                                    //创建控件的 Graphics 对象
    g.Clear(Color.White);                                                    //以指定的颜色清除控件背景
    Color Var_Color_Up = Color.Red;                                          //设置前景色
    Color Var_Color_Down = Color.Yellow;                                     //设置背景色
    Font Var_Font = new Font("宋体", 40);                                    //设置字体样式
    string Var_Str = "渐变效果的文字";                                       //设置字符串
    SizeF Var_Size = g.MeasureString(Var_Str, Var_Font);                     //获取字符串的大小
     PointF Var_Point = new PointF(5, 5);                                    //设置文字的显示位置
     //根据文字的大小及位置，创建 RectangleF 对象
    RectangleF Var_Rect = new RectangleF(Var_Point, Var_Size);
    LinearGradientBrush Var_LinearBrush = new LinearGradientBrush(Var_Rect, Var_Color_Up, Var_Color_Down,
LinearGradientMode.Horizontal);                                              //设置从左到右的线性渐变效果
    g.DrawString(Var_Str, Var_Font, Var_LinearBrush, Var_Point);             //绘制文字
}
```

秘笈心法

心法领悟 532：使用 LinerGradientBrush 类绘制渐变色彩。

LinerGradientBrush 类提供一种渐变色彩的特效，填满图形的内部区域，它位于 System.Drawing.Drawing2D 命名空间下。本实例中使用 LinerGradientBrush 类绘制渐变区域的代码如下：

```
    LinearGradientBrush Var_LinearBrush = new LinearGradientBrush(Var_Rect, Var_Color_Up, Var_Color_Down,
    LinearGradientMode.Horizontal);                                          //设置从左到右的线性渐变效果
```

实例 533 缩放效果的文字

光盘位置：光盘\MR\22\533　　中级　　趣味指数：★★★★☆

实例说明

缩放效果的文字实际上就是按一定的比例，对文字进行缩放。运行本实例，在“缩放比例”文本框中输入要缩放的比例，然后单击“效果”按钮，绘制指定的缩放文字。实例运行效果如图 22.40 所示。

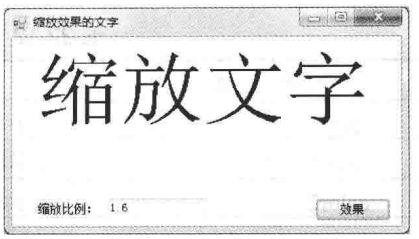


图 22.40　缩放效果的文字

关键技术

本实例实现时，首先使用 GraphicsPath 类的构造函数设置文字的绘制路径，然后使用 Graphics 对象的 FillPath 方法绘制缩放文字。下面对本实例中用到的关键技术进行详细讲解。

（1）GraphicsPath 类

该类表示一系列相互连接的直线和曲线，其构造函数的语法格式如下：

```
public GraphicsPath (Point[] pts,byte[] types)
```

参数说明

❶ pts：Point 结构的数组，它定义组成 GraphicsPath 的点的坐标。

❷ types：PathPointType 枚举元素的数组，它指定 pts 数组中各相应点的类型。

（2）FillPath 方法

该方法用于填充 GraphicsPath 对象的内部。语法格式如下：

```
public void FillPath (Brush brush,GraphicsPath path)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ path：GraphicsPath 图形路径，它表示要填充的路径。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ZoomCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入文字的缩放比例；添加一个 Button 控件，用来绘制缩放的文字；添加一个 Panel 控件，用来显示绘制的缩放文字。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = panel1.CreateGraphics();                                    //创建控件的 Graphics 对象
    g.Clear(Color.White);                                                    //以指定的颜色清除控件背景
    Brush Var_Back = Brushes.Black;                                          //设置画刷
    FontFamily Var_FontFamily = new FontFamily("宋体");                      //设置字体样式
    string Var_Str = "缩放文字";                                             //设置字符串
    GraphicsPath Var_Path = new GraphicsPath();                              //创建 GraphicsPath 对象
    //在路径中添加文本
    Var_Path.AddString(Var_Str, Var_FontFamily,(int)FontStyle.Regular,50,new Point(0,0),new StringFormat());
    PointF[] Var_PointS = Var_Path.PathPoints;                               //获取路径中的点
    Byte[] Car_Types = Var_Path.PathTypes;                                   //获取相应点的类型
    //设置矩阵
    Matrix Var_Matrix = new Matrix(Convert.ToSingle(textBox1.Text), 0.0F, 0.0F, Convert.ToSingle(textBox1.Text), 0.0F, 0.0F);
    Var_Matrix.TransformPoints(Var_PointS);                                  //设置几何变换
    GraphicsPath Var_New_Path = new GraphicsPath(Var_PointS, Car_Types);     //对 GraphicsPath 类进行初始化
    g.FillPath(Var_Back, Var_New_Path);                                      //绘制缩放的文字
}
```

秘笈心法

心法领悟 533：GraphicsPath 类的使用。

GraphicsPath 类表示一系列相互连接的直线和曲线，它主要用来将基本构造块收集到一个单元中作为一个完整的路径，它位于 System.Drawing.Drawing2D 命名空间中。

实例 534	辉光效果的文字	中级
	光盘位置：光盘\MR\22\534	趣味指数：★★★★☆

实例说明

辉光效果的文字就是在文字的外边框加上发光的效果。运行本实例，单击“效果”按钮，绘制指定的文字，然后在文字的外边框加上辉光效果。实例运行效果如图 22.41 所示。

图 22.41　辉光效果的文字

关键技术

本实例实现时，首先使用 Graphics 对象的 SmoothingMode、InterpolationMode 和 TextRenderingHint 属性对绘制文字的质量进行设置，然后使用其 DrawImageUnscaled 方法绘制辉光效果的文字。下面对本实例中用到的关键技术进行详细介绍。

（1）SmoothingMode 属性

该属性用于获取或设置 Graphics 对象的呈现质量。语法格式如下：

```
public SmoothingMode SmoothingMode { get; set; }
```

参数说明

属性值：SmoothingMode 枚举值之一，指定是否将平滑处理（消除锯齿）应用于直线、曲线和已填充区域的边缘。SmoothingMode 枚举值及说明如表 22.14 所示。

表 22.14　SmoothingMode 枚举值及说明

枚　举　值	描　　述
AntiAlias	指定消除锯齿的呈现
Default	指定不消除锯齿
HighQuality	指定高质量、低速度呈现
HighSpeed	指定高速度、低质量呈现
Invalid	指定一个无效模式
None	指定不消除锯齿

（2）InterpolationMode 属性

该属性用于获取或设置与 Graphics 对象关联的插补模式。语法格式如下：

```
public InterpolationMode InterpolationMode { get; set; }
```

参数说明

属性值：InterpolationMode 枚举值之一，指定在缩放或旋转图像时使用的算法。InterpolationMode 枚举值及说明如表 22.15 所示。

表 22.15　InterpolationMode 枚举值及说明

枚 举 值	描　述
Bicubic	指定双三次插值法，不进行预筛选。将图像收缩为原始大小的 25%以下时，此模式不适用
Bilinear	指定双线性插值法，不进行预筛选。将图像收缩为原始大小的 50%以下时，此模式不适用
Default	指定默认模式
High	指定高质量插值法
HighQualityBicubic	指定高质量的双三次插值法。执行预筛选以确保高质量的收缩。此模式可产生质量最高的转换图像
HighQualityBilinear	指定高质量的双线性插值法。执行预筛选以确保高质量的收缩
Invalid	等效于 QualityMode 枚举的 Invalid 元素
Low	指定低质量插值法
NearestNeighbor	指定最临近插值法

（3）TextRenderingHint 属性

该属性用于获取或设置与 Graphics 对象关联的文本的呈现模式。语法格式如下：

```
public TextRenderingHint TextRenderingHint { get; set; }
```

参数说明

属性值：TextRenderingHint 枚举值之一，指定文本呈现的质量。TextRenderingHint 枚举值及说明如表 22.16 所示。

表 22.16　TextRenderingHint 枚举值及说明

枚 举 值	描　述
AntiAlias	在无提示的情况下使用每个字符的消除锯齿效果标志符号位图来绘制字符
AntiAliasGridFit	在有提示的情况下使用每个字符的消除锯齿效果标志符号位图来绘制字符
ClearTypeGridFit	在有提示的情况下使用每个字符的标志符号 ClearType 位图来绘制字符
SingleBitPerPixel	使用每个字符的标志符号位图来绘制字符。不使用提示
SingleBitPerPixelGridFit	使用每个字符的标志符号位图来绘制字符。提示用于改善字符在主干和弯曲部分的外观
SystemDefault	在有系统默认呈现提示的情况下使用每个字符的标志符号位图来绘制字符

（4）DrawImageUnscaled 方法

该方法根据坐标指定的位置，使用图像的原始物理大小绘制指定的图像。语法格式如下：

```
public void DrawImageUnscaled (Image image,Point point)
```

参数说明

❶ image：要绘制的 Image 图像。

❷ point：Point 结构，它指定所绘制图像的左上角。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RadializedCharacter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来绘制辉光效果的文字；添加一个 Panel 控件，用来显示绘制的辉光效果文字。

（3）程序的主要代码如下：

```
public static Image ImageLightEffect(string Str, Font F, Color ColorFore, Color ColorBack, int BlurConsideration)
{
    Bitmap Var_Bitmap = null;                                          //创建 Bitmap 对象
    using (Graphics g = Graphics.FromHwnd(IntPtr.Zero))                //创建 Graphics 对象
    {
        SizeF Var_Size = g.MeasureString(Str, F);                      //对字符串进行测量
        //通过文字的大小创建 Bitmap 对象
        using (Bitmap Var_bmp = new Bitmap((int)Var_Size.Width, (int)Var_Size.Height))
        using (Graphics Var_G_Bmp = Graphics.FromImage(Var_bmp))       //创建 Bitmap 对象
```

```
        //根据 RGB 的值定义画刷
        using (SolidBrush Var_BrushBack = new SolidBrush(Color.FromArgb(16, ColorBack.R, ColorBack.G, ColorBack.B)))
        using (SolidBrush Var_BrushFore = new SolidBrush(ColorFore))                    //定义画刷
        {
            Var_G_Bmp.SmoothingMode = SmoothingMode.HighQuality;                        //设置为高质量
            Var_G_Bmp.InterpolationMode = InterpolationMode.HighQualityBilinear;        //设置为高质量的收缩
            Var_G_Bmp.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;           //消除锯齿
            Var_G_Bmp.DrawString(Str, F, Var_BrushBack, 0, 0);                          //绘制文字
            //根据辉光文字的大小创建 Bitmap 对象
            Var_Bitmap = new Bitmap(Var_bmp.Width + BlurConsideration, Var_bmp.Height + BlurConsideration);
            using (Graphics Var_G_Bitmap = Graphics.FromImage(Var_Bitmap))              //创建 Graphics 对象
            {
                Var_G_Bitmap.SmoothingMode = SmoothingMode.HighQuality;                 //设置为高质量
                Var_G_Bitmap.InterpolationMode = InterpolationMode.HighQualityBilinear; //设置为高质量的收缩
                Var_G_Bitmap.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;    //消除锯齿
                //遍历辉光文字的各像素点
                for (int x = 0; x <= BlurConsideration; x++)
                {
                    for (int y = 0; y <= BlurConsideration; y++)
                    {
                        Var_G_Bitmap.DrawImageUnscaled(Var_bmp, x, y);                  //绘制辉光文字的点
                    }
                }
                //绘制文字
                Var_G_Bitmap.DrawString(Str, F, Var_BrushFore, BlurConsideration / 2, BlurConsideration / 2);
            }
        }
    }
    return Var_Bitmap;                                                                  //返回辉光效果的文字
}
```

秘笈心法

心法领悟 534：SmoothingMode 枚举的使用。

SmoothingMode 枚举主要用于指定是否将平滑处理（消除锯齿）应用于直线、曲线和已填充区域的边缘，它一般用来消除图像的锯齿。

22.6　图 像 识 别

实例 535　查看图片的像素　　初级

光盘位置：光盘\MR\22\535　　趣味指数：★★★★

实例说明

图片的存储方式一般以像素为单位，像素越大，表示图片的面积越大，而且像素和分辨率紧密相关，也就是说像素越大，图片的清晰度就越高。本实例实现了查看图片像素大小的功能，实例运行效果如图 22.42 所示。

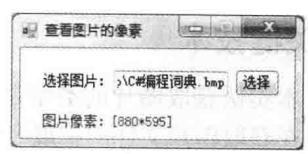


图 22.42　查看图片的像素

关键技术

本实例主要通过 Image 对象的 Width 和 Height 属性获得打开图片的宽度和高度，进而实现查看图片的像素功能，下面分别对它们进行详细讲解。

（1）Width 属性

该属性用来获取 Image 图像的宽度（以像素为单位）。语法格式如下：

```
[BrowsableAttribute(false)]
public int Width { get; }
```

参数说明

属性值：Image 图像的宽度（以像素为单位）。

（2）Height 属性

该属性用来获取 Image 图像的高度（以像素为单位）。语法格式如下：

```
[BrowsableAttribute(false)]
public int Height { get; }
```

参数说明

属性值：Image 图像的高度（以像素为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImagePels。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加一个 Button 控件，用来执行打开图片操作；添加一个 TextBox 控件，用来显示选择的图片文件路径；添加一个 Label 控件，用来显示图片的像素。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "*.jpg,*.jpeg,*.bmp,*.gif,*.ico,*.png,*.tif,*.wmf|*.jpg;*.jpeg;*.bmp;*.gif;*.ico;*.png;*.tif;*.wmf";
    openFileDialog1.ShowDialog();                                          //“打开文件”对话框
    textBox1.Text = openFileDialog1.FileName;                              //显示选择图片的路径
    //根据文件的路径创建 Image 对象
    Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);
    label2.Text = "图片像素：[" + myImage.Width + "*" + myImage.Height + "]";   //获取图片的大小
}
```

秘笈心法

心法领悟 535：Image 类的使用。

Image 类为源自 Bitmap 和 Metafile 的类提供功能的抽象基类，它位于 System.Drawing 命名空间下。

实例 536　获取图片类型

光盘位置：光盘\MR\22\536　　初级　　趣味指数：★★★★

实例说明

实际程序开发过程中，有时需要用指定类型的图片，那么如何在程序中获取图片的类型呢？本实例实现了一种简单的获取图片类型的方法，实例运行效果如图 22.43 所示。

图 22.43　获取图片类型

关键技术

本实例获取图片的类型是通过获取选定图片文件的文件扩展名来实现的，在获取图片文件的扩展名时，需要使用 Substring 方法，下面对其进行详细讲解。

Substring 方法用来从指定的字符位置开始截取指定长度的字符串。语法格式如下：

```
public string Substring ( int startIndex, int length)
```

参数说明

❶ startIndex：子字符串的起始位置的索引。

❷ length：要截取的字符数。

❸ 返回值：一个 String 文本字符串，表示截取的字符串。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ImageStyle。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示“打开文件”对话框；添加一个 Button 控件，用来执行打开图片操作；添加一个 PictureBox 控件，用来显示选择的图片；添加一个 TextBox 控件，用来显示图片的类型。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "*.jpg,*.jpeg,*.bmp,*.gif,*.ico,*.png,*.tif,*.wmf|*.jpg;*.jpeg;*.bmp;*.gif;*.ico;*.png;*.tif;*.wmf";
    openFileDialog1.ShowDialog();                                                    //“打开文件”对话框
    //根据文件的路径创建 Image 对象
    Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);
    pictureBox1.Image = myImage;                                                     //显示打开的图片
    textBox1.Text = openFileDialog1.FileName.Substring(openFileDialog1.FileName.LastIndexOf(".") + 1,
        openFileDialog1.FileName.Length - openFileDialog1.FileName.LastIndexOf(".") - 1);   //获取当前图片的扩展名
}
```

秘笈心法

心法领悟 536：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加载图像，或者使用 LoadAsync 方法异步加载图像。

实例 537　获取指定点的 RGB 值

光盘位置：光盘\MR\22\537　　初级　　趣味指数：★★★★☆

实例说明

在一些图片处理软件中，可以根据选择的像素点，获得该点的 R、G、B 值。本实例实现了获得图片中任意点的 R、G、B 值的功能，实例运行效果如图 22.44 所示。

图 22.44　获取指定点的 RGB 值

关键技术

本实例获取图像中指定点的 R、G、B 值时，主要是通过 Color 对象的 R、G、B 属性实现的，其中，R 属性用来获取 Color 结构的红色分量值，G 属性用来获取 Color 结构的绿色分量值，B 属性用来获取 Color 结构的蓝色分量值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RGBValue。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显

示“打开文件”对话框；添加一个 Button 控件，用来执行打开图片操作；添加 3 个 TextBox 控件，分别用来显示指定像素点的 R、G、B 值。

（3）程序的主要代码如下：

在 button1 控件的 Click 事件中，打开“打开文件”对话框，选择指定类型的图片，并显示在 pictureBox1 控件上。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "*.jpg,*.jpeg,*.bmp,*.gif,*.ico,*.png,*.tif,*.wmf|*.jpg;*.jpeg;*.bmp;*.gif;*.ico;*.png;*.tif;*.wmf";
    openFileDialog1.ShowDialog( );                                          //“打开文件”对话框
    //根据文件的路径创建 Image 对象
    Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);
    pictureBox1.Image = myImage;                                            //显示打开的图片
}
```

在 pictureBox1 控件上单击鼠标右键并释放鼠标时，获取鼠标所在像素的颜色值。代码如下：

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    Bitmap bmp = (Bitmap)pictureBox1.Image;                                 //创建 Bitmap 对象
    try
    {
        Color pointColor = bmp.GetPixel(e.X, e.Y);                          //获取当前像素的颜色值
        //分别通过调用 Color 对象的 R、G、B 属性获得指定点的 R、G、B 值
        textBox1.Text = pointColor.R.ToString( );
        textBox2.Text = pointColor.G.ToString( );
        textBox3.Text = pointColor.B.ToString( );
    }
    catch { }
}
```

秘笈心法

心法领悟 537：使用 OpenFileDialog 控件作为“打开”对话框。

打开对话框使用 OpenFileDialog 控件来表示，它表示一个通用对话框，用户可以使用此对话框来指定一个或多个要打开的文件的文件名。开发人员可以使用 OpenFileDialog 控件的 ShowDialog 方法显示“打开”对话框。

实例 538　设置图像中指定位置的像素值

光盘位置：光盘\MR\22\538　　初级　　趣味指数：★★★★☆

实例说明

本实例是用指定的颜色替换用鼠标在图片上拖放矩形内的像素颜色。运行程序，在图片的上面用鼠标拖放一个矩形，然后设置要替换的颜色，单击“设置”按钮，将矩形框中的颜色进行替换。实例运行效果如图 22.45 所示。

图 22.45　设置图像中指定位置的像素值

关键技术

本实例中通过使用 Bitmap 对象的 SetPixel 方法以描点的方式更改图片上指定区域的像素颜色。

说明：关于 Bitmap 对象的 SetPixel 方法的详细讲解请参见实例 510 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SetupPelsImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 8 个 Label 控件，用于显示矩形框的位置及大小；添加一个 Panel 控件，用于显示替换的颜色；添加一个 PictureBox 控件，用于显示图片；添加一个 ColorDialog 组件，用于选择指定的颜色；添加两个 Button 控件，用于弹出颜色对话框，以及对指定区域的颜色进行替换。

（3）程序的主要代码如下.

在 pictureBox1 控件的 MouseDown 中获取鼠标按下时的坐标。代码如下：

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    Var_MouseX = e.X;                                    //记录鼠标按下时的 X 坐标值
    Var_MouseY = e.Y;                                    //记录鼠标按下时的 Y 坐标值
}
```

在 pictureBox1 控件的 MouseUp 中获取矩形框的高度和宽度。代码如下：

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    Var_MouseW = e.X - Var_MouseX;                       //获取鼠标拖动矩形框的宽度
    Var_MouseH = e.Y - Var_MouseY;                       //获取鼠标拖动矩形框的高度
    if (Var_MouseW == 0)                                 //矩形框的宽度为 0
        Var_MouseW = 1;                                  //设置宽度为 1
    if (Var_MouseH == 0)                                 //矩形框的高度为 0
        Var_MouseH = 1;                                  //设置高度为 1
    label_X.Text = Var_MouseX.ToString();                //显示 X 的坐标值
    label_Y.Text = Var_MouseY.ToString();                //显示 Y 的坐标值
    label_W.Text = Var_MouseW.ToString();                //显示宽度
    label_H.Text = Var_MouseH.ToString();                //显示高度
}
```

自定义方法 SetPels 用于在图片的指定区域内，用指定的颜色替换其像素颜色。代码如下：

```
public Image SetPels(PictureBox Pict, Panel panel, Color color, int x, int y, int w, int h)
{
    Bitmap Var_Bmp = new Bitmap(w, h);                   //根据矩形框的大小创建 Bitmap 对象
    Bitmap Var_SaveBmp = (Bitmap)Pict.Image;             //创建 Bitmap 对象
    //遍历矩形框内的各像素点
    for (int i = x; i < x + w; i++)
        for (int j = y; j < y + h; j++)
        {
            Var_SaveBmp.SetPixel(i, j, color);           //设置当前像素点的颜色
        }
    return Var_SaveBmp;
}
```

“设置”按钮的 Click 事件用于调用自定义方法 SetPels，实现设置图像中指定位置的像素值。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    pictureBox1.Image = SetPels(pictureBox1, panel1, panel1.BackColor, Var_MouseX, Var_MouseY, Var_MouseW,
Var_MouseH);                                             //调用自定义方法修改图片的颜色
}
```

秘笈心法

心法领悟 538：使用 Bitmap 类的 SetPixel 方法实现图像的特殊效果。

Bitmap 类的 SetPixel 方法主要用来设置 Bitmap 图像中指定像素的颜色，通过使用该方法为图像重新着色，可以实现一些常见的图像特殊效果，如百叶窗效果、交错效果、光晕效果等。

实例 539 在图像文件中实现自定义标记

光盘位置：光盘\MR\22\539　　初级　　趣味指数：★★★★☆

实例说明

在图像文件中实现自定义标记实际上就是在图片的指定位置绘制文字。运行本实例，在“标记文字”文本框中设置要显示的标记文字，在“标记位置”下拉列表框中设置文字的显示位置，用“字体”按钮设置文字的字体样式，然后单击“设置标记”按钮，在图片的指定位置绘制文字。实例运行效果如图 22.46 所示。

图 22.46　在图像文件中实现自定义标记

关键技术

本实例实现时，首先使用 Graphics 类的 MeasureString 方法来获取指定文本格式的文字大小，然后使用 Color 类的 FromArgb 方法将设置后的 R、G、B 的值组合成 Color 值来改变字体的颜色。

说明：（1）关于 Graphics 类的 MeasureString 方法的详细讲解请参见实例 527 中的关键技术。

（2）关于 Color 类的 FromArgb 方法的详细讲解请参见实例 517 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BadgeImage。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用于输入要显示的字符串；添加一个 ComboBox 控件，用于设置文字的显示位置；添加两个 Button 控件，分别用于设置字体的文本样式，以及实现文字标识的功能。

（3）程序的主要代码如下：

```
public Image SetBadge(PictureBox Pict, String Str, Font font, int place)
{
    Image Var_Image = Pict.Image;                                          //根据图片创建 Image 对象
    int Var_FontSize = (int)font.Size;                                     //获取字体大小
    bool Var_isSetFont = false;                                            //判断当前文字是否超出图片的大小
    int Var_W = Var_Image.Width;                                           //获取图片的宽度
    int Var_H = Var_Image.Height;                                          //获取图片的高度
    int Var_StrX = 0;                                                      //记录文字的 X 位置
    int Var_StrY = 0;                                                      //记录文字的 Y 位置
    Bitmap Var_bmp = new Bitmap(Var_W, Var_H);                             //创建 Image 对象
    Bitmap Var_SaveBmp = new Bitmap(Var_Image);                            //创建 Image 对象
    Graphics g = Graphics.FromImage(Var_bmp);                              //用指定的 Bitmap 创建 Graphics 对象
    Graphics tem_Graphics = Graphics.FromImage(Var_Image);                 //用指定的 Bitmap 创建 Graphics 对象
    SizeF Var_Size = new SizeF(Var_W, Var_H);                              //创建 SizeF 对象
    Font tem_Font = font;                                                  //获取文字的设置文本
    g.Clear(Color.White);                                                  //清空图片
    while (Var_isSetFont == false)                                         //如果文字超出图片的大小
    {
        //设置文字的文本
        tem_Font = new Font(font.Name, Var_FontSize, font.Bold ? FontStyle.Bold : FontStyle.Regular);
        Var_Size = g.MeasureString(Str, tem_Font);                         //对文字进行测量
        if (Var_Size.Width < Var_bmp.Width - 10)                           //如果文字的宽度没有超出图片
        {
            if (Var_Size.Height < Var_bmp.Height - 10)                     //如果文字的高度没有超出图片
                Var_isSetFont = true;                                      //不减小文字的大小
```

```
        }
        else
            Var_FontSize = Var_FontSize - 1;                    //文字的字体大小减 1
    }
    switch (place)                                              //选择文字的显示位置
    {
        case 1:                                                 //右下角
            {
                Var_StrX = (int)(Var_bmp.Width - Var_Size.Width-3);     //设置文字的 X 坐标值
                Var_StrY = (int)(Var_bmp.Height - Var_Size.Height);     //设置文字的 Y 坐标值
                break;
            }
        case 2:                                                 //右上角
            {
                Var_StrX = (int)(Var_bmp.Width - Var_Size.Width - 3);
                Var_StrY = 1;
                break;
            }
        case 3:                                                 //左下角
            {
                Var_StrX = 1;
                Var_StrY = (int)(Var_bmp.Height - Var_Size.Height);
                break;
            }
        case 4:                                                 //左上角
            {
                Var_StrX = 1;
                Var_StrY = 1;
                break;
            }
        case 5:                                                 //顶居中
            {
                Var_StrX = (int)(Var_bmp.Width - Var_Size.Width-2)/2;
                Var_StrY = 1;
                break;
            }
        case 6:                                                 //底居中
            {
                Var_StrX = (int)(Var_bmp.Width - Var_Size.Width - 2) / 2;
                Var_StrY = (int)(Var_bmp.Height - Var_Size.Height);
                break;
            }
    }
    g.DrawString(Str, tem_Font, new SolidBrush(Color.Black), Var_StrX, Var_StrY);   //绘制前景色为黑色的文字
    int tem_Become = 40;                                        //设置文字的变色深度
    //遍历图片的所有像素
    for (int x = 1; x < Var_bmp.Width; x++)
    {
        for (int y = 1; y < Var_bmp.Height; y++)
        {
            int tem_a, tem_r, tem_g, tem_b, tem_r1, tem_g1, tem_b1;     //定义变量
            if (Var_bmp.GetPixel(x, y).ToArgb() == Color.Black.ToArgb())   //如果当前像素的颜色为黑色
            {
                tem_a = Var_SaveBmp.GetPixel(x, y).A;           //获取当前像素的 alpha 分量值
                tem_r = Var_SaveBmp.GetPixel(x, y).R;           //获取当前像素的 R 色值
                tem_g = Var_SaveBmp.GetPixel(x, y).G;           //获取当前像素的 G 色值
                tem_b = Var_SaveBmp.GetPixel(x, y).B;           //获取当前像素的 B 色值
                tem_r1 = tem_r;                                 //临时存储 R 色值
                tem_g1 = tem_g;                                 //临时存储 G 色值
                tem_b1 = tem_b;                                 //临时存储 B 色值
                //根据加深后的图片背景显示文字
                if (tem_b + tem_Become < 255)                   //如果 B 色值加上当前深度小于 255
                    tem_b = tem_b + 255;                        //B 色值加上深度值
                if (tem_g + tem_Become < 255)
                    tem_g = tem_g + 255;
                if (tem_r + tem_Become < 255)
                    tem_r = tem_r + 255;
                if (tem_r1 - tem_Become > 0)                    //如果 B 色值加上当前深度大于 0
                    tem_r1 = tem_r1 - tem_Become;               //B 色值减去深度值
```

```
                if (tem_g1 - tem_Become > 0)
                    tem_g1 = tem_g1 - tem_Become;
                if (tem_b1 - tem_Become > 0)
                    tem_b1 = tem_b1 - tem_Become;
                tem_Graphics.DrawEllipse(new Pen(new SolidBrush(Color.Black)), x, y + 1, 3, 3);//绘制文字的阴影
                //以加深后的图片背景显示文字
                tem_Graphics.DrawEllipse(new Pen(new SolidBrush(Color.FromArgb(tem_a, tem_r1, tem_g1, tem_b1))), x, y, 1, 1);
            }
        }
    }
    return Var_Image;
}
```

秘笈心法

心法领悟 539：使用 DrawString 方法实现水印文字。

对于一些有可能侵权的图片，或者要作为宣传的图片，可以通过使用 DrawString 方法将用到的文字绘制到图片上，从而得到所谓的“水印文字”。

22.7 常用的图像工具

实例 540	屏幕抓图	高级
	光盘位置：光盘\MR\22\540	趣味指数：★★★★★

实例说明

抓图工具在实际生活中被广泛应用，通过抓图工具可以将当前屏幕中显示的内容抓取成图片保存起来，以便提取其中的内容。系统自带的抓屏工具局限性很大，如无法获取鼠标等。本实例可以设置是否抓取鼠标，并且还提供了快捷键，方便使用者抓取屏幕。实例运行效果如图 22.47 所示。

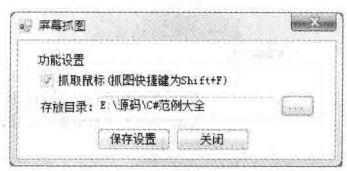


图 22.47 屏幕抓图

关键技术

本实例实现时主要用到了系统 API 函数 GetSystemMetrics 和 CopyIcon，以及 Graphics 类的 CopyFromScreen 方法。下面对本实例中用到的关键技术进行详细讲解。

（1）GetSystemMetrics 函数

该函数用来返回与 Windows 环境有关的信息。语法格式如下：

```
[DllImport("user32.dll", EntryPoint = "GetSystemMetrics")]
private static extern int GetSystemMetrics(int mVal);
```

参数说明

mVal：指定欲获取的信息。

（2）CopyIcon 函数

该函数用来制作指定图标或鼠标指针的一个副本。语法格式如下：

```
[DllImport("user32.dll", EntryPoint = "CopyIcon")]
private static extern IntPtr CopyIcon(IntPtr hIcon);
```

参数说明

hIcon：欲复制的图标或指针的句柄。

（3）CopyFromScreen 方法

该方法用来执行颜色数据从屏幕到 Graphics 的绘图图面的位块传输。语法格式如下：

```
public void CopyFromScreen(int sourceX,int sourceY,int destinationX,int destinationY,Size blockRegionSize)
```

CopyFromScreen 方法中的参数说明如表 22.17 所示。

表 22.17　CopyFromScreen 方法中的参数说明

参　数	描　述
sourceX	位于源矩形左上角的点的 x 坐标
sourceY	位于源矩形左上角的点的 y 坐标
destinationX	位于目标矩形左上角的点的 x 坐标
destinationY	位于目标矩形左上角的点的 y 坐标
blockRegionSize	要传输的区域大小

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetScreen。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 CheckBox 控件，用来设置是否抓取鼠标；添加 3 个 Button 控件，分别用来执行选择保存路径、保存用户设置和退出程序功能；添加一个 FolderBrowserDialog 控件，用来显示“浏览文件夹”对话框；添加一个 TextBox 控件，用来显示保存路径；添加一个 Timer 组件，用来控制快捷键的设置；添加一个 ContextMenuStrip 控件，用来作为程序的快捷菜单。

（3）程序的主要代码如下。

当抓取屏幕时，可以设置是否抓取鼠标，如果抓取没有鼠标的屏幕图片，则调用 CaptureNoCursor 方法，由于不需要抓取鼠标，所以此方法中直接使用 CopyFromScreen 方法捕获屏幕。代码如下：

```
private Bitmap CaptureNoCursor()                                    //抓取没有鼠标的桌面
{
    //根据屏幕大小创建一个 Bitmap 对象
    Bitmap _Source = new Bitmap(GetSystemMetrics(0), GetSystemMetrics(1));
    using (Graphics g = Graphics.FromImage(_Source))                //创建 Graphics 对象
    {
        g.CopyFromScreen(0, 0, 0, 0, _Source.Size);                 // CopyFromScreen 方法捕获桌面图像
        g.Dispose();                                                //释放
    }
    return _Source;
}
```

如果想抓取带鼠标的屏幕图片，则调用 CaptureDesktop 方法。在此方法中首先还是调用 CopyFromScreen 方法捕获屏幕图片，然后通过 CaptureCursor 方法获取鼠标的样式和坐标，最后将鼠标绘制到捕获的屏幕图片上，从而实现抓取带鼠标的屏幕图片。代码如下：

```
private Bitmap CaptureDesktop()                                     //抓取带鼠标的桌面
{
    try
    {
        int _CX = 0, _CY = 0;
        //根据屏幕大小创建一个 Bitmap 对象
        Bitmap _Source = new Bitmap(GetSystemMetrics(0), GetSystemMetrics(1));
        using (Graphics g = Graphics.FromImage(_Source))            //创建 Graphics 对象
        {
            g.CopyFromScreen(0, 0, 0, 0, _Source.Size);             // CopyFromScreen 方法捕获桌面图像
            g.DrawImage(CaptureCursor(ref _CX, ref _CY), _CX, _CY); //将鼠标样式绘制到捕获的桌面图像上
            g.Dispose();                                            //释放
        }
        _X = (800 - _Source.Width) / 2;
        _Y = (600 - _Source.Height) / 2;
        return _Source;
    }
    catch
    {
        return null;
    }
}
```

CaptureCursor 方法用于获取鼠标的样式和坐标，当抓取带鼠标的屏幕图片时，会用到此方法，其返回值是鼠标的 Bitmap 对象。代码如下：

```
private Bitmap CaptureCursor(ref int _CX, ref int _CY)
{
    IntPtr _Icon;                                          //鼠标句柄
    CURSORINFO _CursorInfo = new CURSORINFO();             //创建 CURSORINFO 对象
    ICONINFO _IconInfo;
    _CursorInfo.cbSize = Marshal.SizeOf(_CursorInfo);
    if (GetCursorInfo(ref _CursorInfo))                    //获取鼠标信息
    {
        if (_CursorInfo.flags == 0x00000001)
        {
            _Icon = CopyIcon(_CursorInfo.hCursor);         //获取鼠标句柄
            if (GetIconInfo(_Icon, out _IconInfo))         //取得与图标有关的信息
            {
                _CX = _CursorInfo.ptScreenPos.X - _IconInfo.xHotspot;   //获取鼠标 x 坐标
                _CY = _CursorInfo.ptScreenPos.Y - _IconInfo.yHotspot;   //获取鼠标 y 坐标
                return Icon.FromHandle(_Icon).ToBitmap();               //返回鼠标图像
            }
        }
    }
    return null;
}
```

秘笈心法

心法领悟 540：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 541　抓取网站整页面

光盘位置：光盘\MR\22\541　　高级　　趣味指数：★★★★★

实例说明

当浏览网站时，如果想保存网页，通常是选择浏览器中的“文件”/“另存为”命令将网页保存到本地磁盘中。但有时网页不允许用这样的方法保存，所以就需要将整个网页内容抓取成图片保存起来，从而实现保存网站内容的功能。本实例实现了抓取网站整页面的功能，实例运行效果如图 22.48 所示。

图 22.48　抓取网站整页面

关键技术

本实例在实现抓取网站整页面功能时，首先需要获取网页工作区的范围；然后判断当前网页是否存在滚动条，如果存在滚动条，说明网页的内容已经超出当前页面，这时需要将当前页面以外的内容抓取出来，并与当前页面的图片合并，从而实现动态抓取网站整页面的功能。下面对本实例中用到的关键技术进行详细讲解。

（1）滚动条控制

本实例通过使用 WebBrowser 控件对网页进行显示与控制。WebBrowser 控件是网页浏览器控件，使用该控件可以实现与 IE 浏览器同样的功能，在抓取网站整页面时是通过控制滚动条位置实现滚动抓图的，这里主要用到了其 Body 对象。Body 对象的属性可以用来控制页面中整个主体内容的外观，如设置主体内容的背景颜色、设置左右边距等。Body 对象常用的属性及说明如表 22.18 所示。

表 22.18　Body 对象常用的属性及说明

属　性	说　明
clientWidth	网页可见区域的宽度
clientHeight	网页可见区域的高度
scrollWidth	网页正文的宽度
scrollHeight	网页正文的高度
scrollLeft	获取或设置横向滚动条的坐标值
scrollTop	获取或设置纵向滚动条的坐标值

（2）场景拼凑

网页抓图的功能是通过场景的拼凑实现的，其设计思路如下：

- 抓取网页可见区域场景。
- 当已抓取的场景的宽度小于网页宽度时，移动网页横向滚动条（如移动 199 个像素），获取可见区域的右侧 199 个像素的场景，并与原场景拼凑。
- 将横向滚动最后剩余的场景捕获，并与已获取的场景拼凑。

纵向场景捕获与横向同理。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 WebSnap。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为窗体的工具栏；添加一个 SaveFileDialog 控件，用来选择保存位置；添加一个 WebBrowser 控件，用来显示加载的网页；添加一个 StatusStrip 控件，用来作为窗体的状态栏。

（3）程序的主要代码如下：

```
private void Snapweb()
{
    this.TopMost = true;                                                    //窗体处于所有窗体的最前端
    SelectObject(DestDC, Bhandle);                                          //选择场景对象
    int linewidth=0;                                                        //宽
    int lineheight=0;                                                       //高
    int i=0;                                                                //纵向计数器
    int j=0;                                                                //横向计数器
    winpoint = webBrowser1.Handle;                                          // webBrowser1 句柄
    GetWindowRect(winpoint,ref Rectangles);                                 //获得整个窗口的范围矩形
    while (lineheight < webBrowser1.Document.Body.ScrollRectangle.Height - 199)
    {
        if (webBrowser1.Document.Body.ScrollTop == 0)
        {
            inputHide();                                                    //屏蔽输入法
            //获取可见区域场景
            BitBlt(DestDC, 0, 0, webBrowser1.Document.Body.ClientRectangle.Width, webBrowser1.Document.Body.ClientRectangle.Height, SourceDC,
Rectangles.Left, Rectangles.Top, 13369376);
```

的颜色值，从而实现对屏幕颜色的获取。下面对本实例中用到的关键技术进行详细讲解。

（1）CreateDC 函数

该函数是指为专门设备创建设备场景。语法格式如下：

```
[DllImport("gdi32.dll")]
static public extern IntPtr CreateDC(string driverName,string deviceName,string output,IntPtr lpinitData);
```

CreateDC 函数中的参数说明如表 22.19 所示。

表 22.19　CreateDC 函数中的参数说明

参　数	描　述
driverName	如果此参数为 DISPLAY，代表获取整个屏幕的设备场景；如果为 WINSPOOL，则是访问打印驱动
deviceName	所用专门设备的名称
output	用 null 值给该参数
lpinitData	DEVMODE，这个结构保存初始值

（2）GetPixel 函数

该函数是指在指定的设备场景中取得一个像素的 RGB 值。语法格式如下：

```
[DllImport("gdi32.dll")]
static public extern uint GetPixel(IntPtr hDC,int XPos,int YPos);
```

参数说明

❶ hDC：一个设备场景的句柄。

❷ XPos：逻辑坐标中要检查的点的 X 坐标。

❸ YPos：逻辑坐标中要检查的点的 Y 坐标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GetColor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 CheckBox 控件，用来设置窗体是否显示在最顶层；添加一个 Panel 控件，用来显示屏幕的颜色；添加 3 个 TextBox 控件，分别用来显示鼠标位置。指定点的 R、G、B 值和网页颜色值；添加一个 Button 控件，用来执行退出窗体操作。

（3）程序的主要代码如下。

Frm_Main 窗体中，自定义一个 GetColor 方法用于获取指定坐标处的颜色值，其返回值是 Color 类型。该方法中分别调用了 GetRValue、GetGValue 和 GetBValue 方法获取指定坐标处的 R、G、B 值。代码如下：

```
public Color GetColor(Point screenPoint)
{
    IntPtr displayDC=CreateDC("DISPLAY", null,    null, IntPtr.Zero);          //创建一个屏幕句柄
    uint colorref=GetPixel(displayDC, screenPoint.X, screenPoint.Y);           //取得鼠标所在位置的像素的 RGB 值
    DeleteDC(displayDC);                                                       //删除场景
    byte Red = GetRValue(colorref);                                            //获取 R 值
    byte Green = GetGValue(colorref);                                          //获取 G 值
    byte Blue = GetBValue(colorref);                                           //获取 B 值
    return Color.FromArgb(Red, Green, Blue);                                   //返回 RGB 值对应的 Color 值
}
```

为了能即时获取鼠标所在坐标处的颜色值，向窗体中添加一个 Timer 组件，在该组件的 Tick 事件中调用 GetColor 方法获取指定坐标处的颜色值。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    //显示鼠标坐标
    txtPoint.Text=Control.MousePosition.X.ToString()+","+Control.MousePosition.Y.ToString();
    Point pt = new Point(Control.MousePosition.X, Control.MousePosition.Y);    //创建 Point 对象
    Color cl = GetColor(pt);                                                   //获取鼠标所在位置的 Color 值
    panel1.BackColor = cl;                                                     //显示该颜色
    txtRGB.Text = cl.R + "," + cl.G + "," + cl.B;                              //显示 RGB 值
    txtColor.Text = ColorTranslator.ToHtml(cl).ToString();                     //显示对应的网页颜色值
    RegisterHotKey(Handle, 81, KeyModifiers.Ctrl, Keys.F);                     //注册热键
}
```

```
        if(linewidth < webBrowser1.Document.Body.ScrollRectangle.Width)                    //当前宽度小于网页宽度时
        {
            //获取整个宽度场景
            while(linewidth<webBrowser1.Document.Body.ScrollRectangle.Width-199)
            {
                inputHide();
                webBrowser1.Document.Body.ScrollLeft+=199;
                BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, webBrowser1.Document.Body.ClientRectangle.
Height + 199 * i, webBrowser1.Document.Body.ClientRectangle.Width, (webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), SourceDC,
Rectangles.Left + webBrowser1.Document.Body.ClientRectangle.Width - 199, Rectangles.Top + webBrowser1.Document.Body.ClientRectangle.Height
- (webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), 13369376);
                linewidth = linewidth + 199;                                               //累计当前高度
                j = j + 1;                                                                 //纵向计数器累计
            }
            if(linewidth >=webBrowser1.Document.Body.ScrollRectangle.Width-199)
            {
                inputHide();
                webBrowser1.Document.Body.ScrollLeft+=199;                                 //调整纵向滚动条位置
                BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, webBrowser1.Document.Body.ClientRectangle.Height +
199 * i, webBrowser1.Document.Body.ScrollRectangle.Width - linewidth, (webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), SourceDC,
Rectangles.Left + webBrowser1.Document.Body.ClientRectangle.Width - (webBrowser1.Document.Body.ScrollRectangle.Width - linewidth), Rectangles.
Top + webBrowser1.Document.Body.ClientRectangle.Height - (webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), 13369376);
                webBrowser1.Document.Body.ScrollLeft=0;
            }
        }
        i=0;
        OpenClipboard(this.Handle);
        EmptyClipboard();
        SetClipboardData(2,Bhandle);
        CloseClipboard();
        Image ig = (Image)Clipboard.GetImage();
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string path = saveFileDialog1.FileName;
            ig.Save(path);
            MessageBox.Show("保存成功","提示",MessageBoxButtons.OK,MessageBoxIcon.Exclamation);
            webBrowser1.Document.Body.ScrollTop = 0;
            ig.Dispose();
        }
    }
}
```

秘笈心法

心法领悟 541：使用 WebBrowser 控件轻松地在窗体中浏览网页信息。

WebBrowser 控件可以在 Windows 窗体客户端应用程序中显示网页，使用 WebBrowser 控件显示网页信息非常方便，只需执行 Navigate 方法，并在方法参数中加入网址信息字符串即可。

实例 542　屏幕颜色拾取器

光盘位置：光盘\MR\22\542　　高级　　趣味指数：★★★★★

实例说明

在处理图片或者制作网页过程中，有时需要获取某个点的颜色值，传统的方法是将屏幕抓取下来保存成图片，然后使用 Photoshop 等软件打开此图片获取颜色值，显然这样做过于繁琐。因此，本实例开发出屏幕颜色拾取器用于获取鼠标所在位置的颜色值。实例运行效果如图 22.49 所示。

图 22.49　屏幕颜色拾取器

关键技术

本实例主要通过调用 API 函数 CreateDC 和 GetPixel 获取指定坐标处

```
            linewidth = webBrowser1.Document.Body.ClientRectangle.Width+5;                    //赋值宽
            if (linewidth < webBrowser1.Document.Body.ScrollRectangle.Width)                  //当前宽小于网页宽度时
            {
                //获取整个宽度场景
                while (linewidth < webBrowser1.Document.Body.ScrollRectangle.Width - 199)
                {
                    inputHide();
                    webBrowser1.Document.Body.ScrollLeft += 199;
                    BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, 0, 199, webBrowser1.Document.Body.ClientRectangle.
Height, SourceDC, Rectangles.Left + webBrowser1.Document.Body.ClientRectangle.Width - 199, Rectangles.Top, 13369376);
                    linewidth = linewidth + 199;
                    j = j + 1;
                }
                if (linewidth >= webBrowser1.Document.Body.ScrollRectangle.Width - 199)
                {
                    inputHide();
                    webBrowser1.Document.Body.ScrollLeft += 199;
                    BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, 0, webBrowser1.Document.Body.ScrollRectangle.
Width - linewidth, webBrowser1.Document.Body.ClientRectangle.Height, SourceDC, Rectangles.Left + webBrowser1.Document.Body.ClientRectangle.
Width - (webBrowser1.Document.Body.ScrollRectangle.Width - linewidth), Rectangles.Top, 13369376);
                    webBrowser1.Document.Body.ScrollLeft= 0;
                    j = 0;
                }
            }
            lineheight = lineheight + webBrowser1.Document.Body.ClientRectangle.Height;       //累计当前高度
        }
        else
        {
            inputHide();
            BitBlt(DestDC, 0, webBrowser1.Document.Body.ClientRectangle.Height + 199 * i, webBrowser1.Document.Body.ClientRectangle.Width,
199, SourceDC, Rectangles.Left, Rectangles.Top + webBrowser1.Document.Body.ClientRectangle.Height - 199, 13369376);
                                                                                              //获取可见区域场景
            linewidth = webBrowser1.Document.Body.ClientRectangle.Width;                      //当前宽度
            if (linewidth < webBrowser1.Document.Body.ScrollRectangle.Width)
            {
                while (linewidth < webBrowser1.Document.Body.ScrollRectangle.Width-199)
                {
                    inputHide();
                    webBrowser1.Document.Body.ScrollLeft += 199;
                    BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, webBrowser1.Document.Body.ClientRectangle.
Height + 199 * i, webBrowser1.Document.Body.ClientRectangle.Width, 199, SourceDC, Rectangles.Left + webBrowser1.Document.Body.ClientRectangle.
Width - 199, Rectangles.Top + webBrowser1.Document.Body.ClientRectangle.Height - 199, 13369376);
                    linewidth = linewidth + 199;
                    j = j + 1;
                }
                if(linewidth >=webBrowser1.Document.Body.ScrollRectangle.Width-199)
                {
                    inputHide();
                    webBrowser1.Document.Body.ScrollLeft+=199;
                    BitBlt(DestDC, webBrowser1.Document.Body.ClientRectangle.Width + 199 * j, webBrowser1.Document.Body.ClientRectangle.
Height + 199 * i, webBrowser1.Document.Body.ScrollRectangle.Width - linewidth, 199, SourceDC, Rectangles.Left + webBrowser1.Document.Body.
ClientRectangle.Width - (webBrowser1.Document.Body.ScrollRectangle.Width - linewidth), Rectangles.Top + webBrowser1.Document.Body.ClientRectangle.
Height - 199, 13369376);
                    webBrowser1.Document.Body.ScrollLeft=0;
                    j = 0;
                }
            }
            i = i + 1;                                                                        //纵向计数器累计
            lineheight = lineheight + 199;
        }
        webBrowser1.Document.Body.ScrollTop+=199;                                             //调整纵向滚动条位置
    }
    if (lineheight >= webBrowser1.Document.Body.ScrollRectangle.Height - 199)
    {
        inputHide();
        BitBlt(DestDC, 0, webBrowser1.Document.Body.ClientRectangle.Height + 199 * i, webBrowser1.Document.Body.ClientRectangle.Width,
(webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), SourceDC, Rectangles.Left, Rectangles.Top + webBrowser1.Document.Body.
ClientRectangle.Height - (webBrowser1.Document.Body.ScrollRectangle.Height - lineheight), 13369376);
        linewidth = webBrowser1.Document.Body.ClientRectangle.Width;                          //当前宽度
```

秘笈心法

心法领悟 542：使用 Color 结构获取颜色。

Color 结构表示一种 ARGB 颜色（alpha、红色、绿色、蓝色），现在已经命名的颜色都可以使用 Color 结构的属性来表示，例如，蓝色可以用 Color.Blue 来表示。

实例 543	不失真压缩图片	高级
	光盘位置：光盘\MR\22\543	趣味指数：★★★★★

实例说明

随着数码相机的普及，很多相片都是用数码相机拍摄出来的，清晰度要比用胶卷的相机好很多，但是数码相机拍摄出来的相片体积都很大，不便在网络上进行传输，所以要对这些图片进行处理，如果用一般的工具改变图片体积大小，随之而来清晰度也会下降，因此，本实例使用 C#开发了不失真压缩图片工具，该工具能够对图片进行高效的批量压缩，并且保证图片的清晰度为最佳状态。实例运行效果如图 22.50 所示。

图 22.50　不失真压缩图片

关键技术

本实例在实现时，首先需要设置缩放的比例，然后设置画布的描绘质量，使重绘后的图片不失真，最后在设置保存图片时，指定图片的压缩质量，这样，才能保证在不失真的情况下压缩图片的体积。实现过程中主要用到 Graphics 类的 CompositingQuality、SmoothingMode 和 InterpolationMode 属性。下面对本实例中用到的关键技术进行详细讲解。

（1）CompositingQuality 属性

该属性用来获取或设置绘制到此 Graphics 的合成图像的呈现质量。语法格式如下：

```
public CompositingQuality CompositingQuality { get; set; }
```

参数说明

属性值：CompositingQuality 枚举值之一，用于指定在复合期间使用的质量等级。CompositingQuality 枚举值及说明如表 22.20 所示。

表 22.20　CompositingQuality 枚举值及说明

枚 举 值	描　述
Invalid	无效质量
Default	默认质量
HighSpeed	高速度、低质量
HighQuality	高质量、低速度复合
GammaCorrected	使用灰度校正
AssumeLinear	假定线性值

（2）SmoothingMode 属性

该属性用来获取或设置此 Graphics 的呈现质量。语法格式如下：

```
public SmoothingMode SmoothingMode { get; set; }
```

参数说明

属性值：SmoothingMode 枚举值之一，用于指定是否将平滑处理（消除锯齿）应用于直线、曲线和已填充区域的边缘。SmoothingMode 枚举值及说明如表 22.21 所示。

表 22.21　SmoothingMode 枚举值及说明

枚 举 值	描　述
Invalid	指定一个无效模式
Default	指定不消除锯齿
HighSpeed	指定高速度、低质量呈现
HighQuality	指定高质量、低速度呈现
None	指定不消除锯齿
AntiAlias	指定消除锯齿的呈现

（3）InterpolationMode 属性

该属性用来获取或设置与此 Graphics 关联的插补模式。语法格式如下：

```
public InterpolationMode InterpolationMode { get; set; }
```

参数说明

属性值：InterpolationMode 枚举值之一，用于指定在缩放或旋转图像时使用的算法。InterpolationMode 枚举值及说明如表 22.22 所示。

表 22.22　InterpolationMode 枚举值及说明

枚 举 值	描　述
Invalid	等效 QualityMode 枚举的 Invalid 元素
Default	指定默认模式
Low	指定低质量插值法
High	指定高质量插值法
Bilinear	指定双线性插值法，不进行预筛选。将图像收缩为原始大小的 50% 以下时，此模式不适用
Bicubic	指定双三次插值法，不进行预筛选。将图像收缩为原始大小的 25% 以下时，此模式不适用
NearestNeighbor	指定最临近插值法
HighQualityBilinear	指定高质量的双线性插值法，执行预筛选以确保高质量的收缩
HighQualityBicubic	指定高质量的双三次插值法，执行预筛选以确保高质量的收缩，此模式可产生质量最高的转换图像

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CompressImg。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 PictureBox 控件，分别用来选择原始图片路径和处理后图片的保存路径；添加两个 FolderBrowserDialog 控件，用来显示“浏览文件夹”对话框；添加两个 TextBox 控件，分别用来显示原始图片路径和处理后图片的保存路径；添加一个 RadioButton 控件，用来设置是否按百分比压缩图片；添加一个 NumericUpDown 控件，用来选择图片的压缩比例；添加两个 Label 控件，分别用来显示原图片的数目和已成功处理的图片数目；添加一个 ProgressBar 控件，用来显示压缩进度；添加两个 Button 控件，分别用来执行压缩图片和退出程序操作。

（3）程序的主要代码如下：

```
public static bool GetPicThumbnail(string sFile, string dFile, int dHeight, int dWidth)
{
    Image iSource = Image.FromFile(sFile);                          //创建 Image 实例
    ImageFormat tFormat = iSource.RawFormat;                        //设置保存格式
    int sW = 0, sH = 0;                                             //记录宽度和高度
     Size tem_size = new Size(iSource.Width,iSource.Height);        //创建 Size 对象
    if (tem_size.Height> dHeight || tem_size.Width> dWidth)         //判断原图大小是否大于指定大小
    {
        if ((tem_size.Width * dHeight) > (tem_size.Height * dWidth))
        {
            sW = dWidth;
```

```
                sH = (dWidth * tem_size.Height) / tem_size.Width;
            }
            else
            {
                sH = dHeight;
                sW = (tem_size.Width * dHeight) / tem_size.Height;
            }
        }
        else                                                    //如果原图大小小于指定的大小
        {
            sW = tem_size.Width;                                //原图宽度等于指定宽度
            sH = tem_size.Height;                               //原图高度等于指定高度
        }
        Bitmap oB = new Bitmap(dWidth, dHeight);                //创建 Bitmap 对象
        Graphics g = Graphics.FromImage(oB);                    //创建 Graphics 对象
        g.Clear(Color.WhiteSmoke);                              //设置画布背景颜色
        g.CompositingQuality = CompositingQuality.HighQuality;  // Graphics 的合成图像的呈现质量
        g.SmoothingMode = SmoothingMode.HighQuality;            // Graphics 的呈现质量
        g.InterpolationMode = InterpolationMode.HighQualityBicubic;  // Graphics 关联的插补模式
        g.DrawImage(iSource, new Rectangle((dWidth - sW) / 2, (dHeight - sH) / 2, sW, sH), 0, 0,
iSource.Width, iSource.Height, GraphicsUnit.Pixel);             //开始重新绘制图像
        g.Dispose();
        //以下代码为保存图片时，设置压缩质量
        EncoderParameters eP = new EncoderParameters();
        long[] qy = new long[1];
        qy[0] = 100;
        EncoderParameter eParam = new EncoderParameter(System.Drawing.Imaging.Encoder.Quality, qy);
        eP.Param[0] = eParam;
        try
        {
            //获得包含有关内置图像编码解码器的信息的 ImageCodecInfo 对象
            ImageCodecInfo[] arrayICI = ImageCodecInfo.GetImageEncoders();
            ImageCodecInfo jpegICIinfo = null;
              for (int x = 0; x < arrayICI.Length; x++)
              {
                  if (arrayICI[x].FormatDescription.Equals("JPEG"))
                  {
                      jpegICIinfo = arrayICI[x];                //设置 JPEG 编码
                      break;
                  }
              }
              if (jpegICIinfo != null)
              {
                  oB.Save(dFile, jpegICIinfo, eP);
              }
              else
              {
                  oB.Save(dFile, tFormat);
              }
              return true;
        }
        catch
        {
            return false;
        }
        finally
        {
            iSource.Dispose();
            oB.Dispose();
        }
    }
```

秘笈心法

心法领悟 543：SmoothingMode 枚举的使用。

SmoothingMode 枚举主要用于指定是否将平滑处理（消除锯齿）应用于直线、曲线和已填充区域的边缘，它一般用来消除图像的锯齿。

22.8 图 像 应 用

实例 544	随机更换壁纸程序 光盘位置：光盘\MR\22\544	中级 趣味指数：★★★★☆

实例说明

Windows 系统的壁纸可以被随意地更换，漂亮的壁纸能使操作者心情舒畅。本实例实现了随机自动更换壁纸功能，运行本实例后，每次重新启动计算机都会自动更换 Windows 系统的桌面壁纸。实例运行效果如图 22.51 所示。

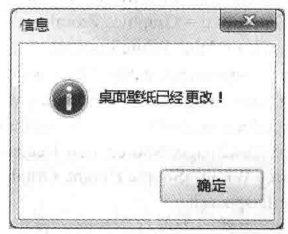


图 22.51 随机更换壁纸程序

关键技术

本实例实现更换桌面壁纸时，主要是通过修改注册表来实现的。更换桌面壁纸可以通过修改注册表中“HKEY_CURRENT_USER\Control Panel\Desktop”子键下的 WallPaper 和 TitleWallPaper 键值来实现。在实现过程中主要用到 RegistryKey 类的 SetValue 方法，下面对其进行详细讲解。

SetValue 方法用来设置指定的名称/值对。语法格式如下：

```
public void SetValue ( string name, Object value)
```

参数说明

❶ name：要存储的值的名称。

❷ value：要存储的数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Wallpaper。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    string strPath = Application.StartupPath.Substring(0, Application.StartupPath.Substring(0, Application.
StartupPath.LastIndexOf("\\")).LastIndexOf("\\")) + @"\Image\";                    //获取图片的所在路径
    DirectoryInfo DInfo = new DirectoryInfo(strPath);                              //创建 DirectoryInfo 对象
    FileInfo[] FInfo = DInfo.GetFiles();                                           //获取当前文件夹下的所有文件
    Random rand = new Random();                                                    //创建 Random 对象
    int i = rand.Next(FInfo.Length);                                               //获取随机数
    RegistryKey myRKey = Registry.CurrentUser;                                     //获取注册表中的基表
    myRKey = myRKey.OpenSubKey("Control Panel\\Desktop", true);                    //检索指定的子项
    //通过调用 RegistryKey 对象的 SetValue 方法随机设置桌面壁纸
    myRKey.SetValue("WallPaper", strPath + FInfo[i].Name);
    myRKey.SetValue("TitleWallPaper", "2");
    myRKey.Close();
    MessageBox.Show("桌面壁纸已经更改！", "信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

秘笈心法

心法领悟 544：注册表的使用。

注册表的层次结构非常类似于文件系统，它记录了用户账号、服务器硬件以及应用程序的设置信息等。同 INI 文件相比，注册表可以控制的数据更多，而且不仅限于处理字符串类型的数据。注册表也包含了一些系统配

置的信息，这些信息根据操作系统的不同而不同。用户选择“开始”/“运行”命令，在弹出对话框的“打开”文本框中输入“regedit”后，单击“确定”按钮即可打开注册表编辑器。

实例 545	制作屏幕保护程序 光盘位置：光盘\MR\22\545	中级 趣味指数：★★★★☆

实例说明

屏幕保护是当用户长时间不对计算机做任何操作时，系统为了保护屏幕而启动的一种保护程序。屏幕保护程序启动后会占据整个屏幕，并且具有一些缓慢变动的图像效果。本实例使用 C#语言设计了一个屏幕保护程序，实例运行效果如图 22.52 所示。

图 22.52　制作屏幕保护程序

关键技术

屏幕保护程序其实是一个无边框样式的 Windows 窗体应用程序，本实例在制作屏幕保护程序时，新建了一个窗体，并将该窗体的大小设置为屏幕大小相同，然后设置其 ControlBox、MaximizeBox、MinimizeBox 和 ShowInTaskbar 的属性均为 false，FormBorderStyle 属性设置为 None，这些属性的详细说明如表 22.23 所示。

表 22.23　本实例中用到的 Form 类的属性及说明

属　性	描　述
ControlBox	获取或设置一个值，该值指示在该窗体的标题栏中是否显示控件框
MaximizeBox	获取或设置一个值，该值指示是否在窗体的标题栏中显示“最大化”按钮
MinimizeBox	获取或设置一个值，该值指示是否在窗体的标题栏中显示“最小化”按钮
ShowInTaskbar	获取或设置一个值，该值指示是否在 Windows 任务栏中显示窗体
FormBorderStyle	获取或设置窗体的边框样式

提示：运行本程序时，需要把 ScreenProtect.exe 改为 ScreenProtect.scr，然后复制到 Windows 系统目录（如 C:\WINDOWS\system32 目录）中，这样即可运行该屏幕保护程序。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ScreenProtect。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并将该窗体的 FormBoderStyle 属性设置为 None，将

窗体的大小调整为与屏幕大小相同，在 Frm_Main 窗体中添加一个 Label 控件，用来加载屏幕保护程序运行时，在窗体中移动的文字；添加一个 PictureBox 控件，用来加载屏幕保护程序运行时，在窗体上浮动的图片；添加一个 Timer 组件，用来适时启动屏幕保护程序。

（3）程序的主要代码如下：

```
private int iSpeed = 2;
private string str = "吉林省明日科技有限公司";
private System.Drawing.Font TextStringFont = new System.Drawing.Font("宋体", 24, System.Drawing.FontStyle.Bold);
private Color TextStringcolor = System.Drawing.Color.Red;
private int iDistance;
private int ixStart = 0;
private int iyStart = 0;
private int speed;
private int x1, y1;
int width1, height1;
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Visible = true;                                          //当前控件可见
    label1.Text = str;                                              //显示文本
    label1.Height = label1.Font.Height;                             //设置文本的高度
    label1.Width = label1.Text.Length * (int)label1.Font.Size * 2;  //设置文本的宽度
    PlayScreenSaver( );                                             //开始屏幕保存
}
private void Form1_KeyDown(object sender, KeyEventArgs e)           //当按下键盘时，停止屏幕保护
{
    StopScreenSaver( );
}
private void Form1_MouseDown(object sender, MouseEventArgs e)       //当按下鼠标时，停止屏幕保护
{
    StopScreenSaver( );
}
private void Form1_Load(object sender, EventArgs e)
{
    speed = 0;
    width1 = this.Width;                                            //记录窗体的宽度
    height1 = this.Height;                                          //记录窗体的高度
}
private void Form1_MouseMove(object sender, MouseEventArgs e)       //当移动鼠标时，如果位置发生改变，停止屏幕保护
{
    if (ixStart == 0 && iyStart == 0)                               //如果移动了鼠标
    {
        ixStart = e.X;                                              //获取当前鼠标的位置
        iyStart = e.Y;
        return;
    }
    else if (e.X != ixStart || e.Y != iyStart)                      //如果鼠标的位置改变
        StopScreenSaver( );                                         //停止屏幕保护
}

private void PlayScreenSaver( )                                     //开始屏幕保护
{
    //下面设置文本显示框的位置坐标
    label1.Location = new System.Drawing.Point(width1 - iDistance, label1.Location.Y);
    label1.Visible = true;                                          //设置为可见
    iDistance += iSpeed;
    if (label1.Location.X <= -(label1.Width))                       //如果超出左边界
    {
        iDistance = 0;
        //循环移动文字
        if (label1.Location.Y == 0)
            label1.Location = new System.Drawing.Point(label1.Location.X, height1 / 2);
        else if (label1.Location.Y == height1 / 2)
            label1.Location = new System.Drawing.Point(label1.Location.X, height1 - label1.Height);
        else
            label1.Location = new System.Drawing.Point(label1.Location.X, 0);
    }
```

```
    //下面是计算图片移动坐标
    speed++;
    if (speed <= 2 * height1)
    {
        x1 = System.Math.Abs(width1 - speed);
        y1 = System.Math.Abs(height1 - speed);
    }
    else if (speed > 2 * height1 && speed <= 2 * width1)
    {
        x1 = System.Math.Abs(width1 - speed);
        y1 = System.Math.Abs(height1 - (speed - speed / height1 * height1));
    }
    else if (speed > 2 * width1 && speed <= 3 * height1)
    {
        x1 = System.Math.Abs(width1 - (speed - speed / width1 * width1));
        y1 = System.Math.Abs(height1 - (speed - speed / height1 * height1));
    }
    else if (speed > 3 * height1 && speed < 4 * height1)
    {
        x1 = System.Math.Abs(width1 - (speed - speed / width1 * width1));
        y1 = System.Math.Abs(speed - speed / height1 * height1);
    }
    else if (speed >= 4 * height1 && speed < 5 * height1)
    {
        x1 = System.Math.Abs(speed - speed / width1 * width1);
        y1 = System.Math.Abs(height1 - (speed - speed / height1 * height1));
    }
    else if (speed >= 5 * height1 && speed < 4 * width1)
    {
        x1 = System.Math.Abs(speed - speed / width1 * width1);
        y1 = System.Math.Abs(speed - speed / height1 * height1);
    }
    else if (speed >= 4 * width1 && speed < 6 * height1)
    {
        x1 = System.Math.Abs(width1 - (speed - speed / width1 * width1));
        y1 = System.Math.Abs(speed - speed / height1 * height1);
    }
    else if (speed >= 6 * height1 && speed < 5 * width1)
    {
        x1 = System.Math.Abs(width1 - (speed - speed / width1 * width1));
        y1 = System.Math.Abs(height1 - (speed - speed / height1 * height1));
    }
    else if (speed >= 5 * width1 && speed < 7 * height1)
    {
        x1 = System.Math.Abs(speed - speed / width1 * width1);
        y1 = System.Math.Abs(height1 - (speed - speed / height1 * height1));
    }
    else if (speed >= 7 * height1 && speed < 6 * width1)
    {
        x1 = System.Math.Abs(speed - speed / width1 * width1);
        y1 = System.Math.Abs(speed - speed / height1 * height1);
    }
    if (speed == 6 * width1)
        speed = 0;
    pictureBox1.Location = new System.Drawing.Point(x1, y1);          //设置图片的位置
}
private void StopScreenSaver( )                                       //停止屏幕保护
{
    System.Windows.Forms.Cursor.Show( );
    timer1.Enabled = false;                                           //停止计时器
    Application.Exit( );                                              //关闭工程
}
```

秘笈心法

心法领悟 545：如何隐藏窗体的标题栏？

开发人员可以通过设置窗体的 FormBorderStyle 属性为 None，实现隐藏窗体标题栏功能。

运行效果如图 22.54 所示。

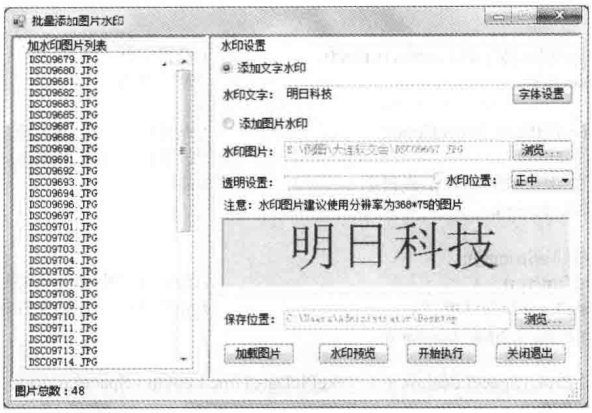


图 22.54　批量添加图片水印

关键技术

本实例在实现为图片添加水印功能时，主要用到了 Graphics 类中的 DrawString 和 DrawImage 方法。

说明：（1）关于 DrawString 方法的详细讲解请参见实例 529 中的关键技术。

（2）关于 DrawImage 方法的详细讲解请参见实例 500 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IMGwatermark。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来显示图片列表；添加两个 OpenFileDialog 控件，分别用来选择所有图片和水印图片；添加一个 ColorDialog 控件，用来选择字体颜色；添加一个 FontDialog 控件，用来设置字体；添加一个 FolderBrowserDialog，用来选择保存位置；添加一个 PictureBox 控件，用来显示添加水印的效果；添加一个 TrackBar 控件，用来设置水印图片透明度；添加一个 ComboBox 控件，用来选择添加水印的位置；添加 7 个 Button 控件，分别用来执行选择字体、选择水印图片、选择保存位置、加载图片、预览水印效果、添加水印效果和退出程序的功能。

（3）程序的主要代码如下：

```
private void AddFontWatermark(string txt,string Iname,int i)
{
    b = new SolidBrush(fontColor);                                    //创建 SolidBrush 对象
    bt = new Bitmap(368,75);                                          //创建 Bitmap 对象，设置水印图片大小
    BigBt = new Bitmap(Image.FromFile(ImgDirectoryPath + "\\" +Iname));  //通过图片路径创建 Bitmap 对象
    Graphics g = Graphics.FromImage(bt);                              //创建 Graphics 对象
    Graphics g1 = Graphics.FromImage(BigBt);                          //创建 Graphics 对象
    g.Clear(Color.Gainsboro);                                         //设置画布背景颜色
    pbImgPreview.Image = bt;                                          //设置 Image 属性
    if (FontF == null)                                                //判断是否设置字体
    {
        f = new Font(txt,fontSize);                                   //创建字体对象
        SizeF XMaxSize = g.MeasureString(txt, f);                     //获取字体的大小
        Fwidth = (int)XMaxSize.Width;                                 //字体宽度
        Fheight = (int)XMaxSize.Height;                               //字体高度
        g.DrawString(txt,f, b, (int)(368 - Fwidth) / 2, (int)(75 - Fheight) / 2);  //绘制文字水印
        if (cbbPosition.SelectedIndex==0)                             //正中
        {
            g1.DrawString(txt, f, b, (int)(BigBt.Width - Fwidth) / 2, (int)(BigBt.Height - Fheight) / 2);
        }
        if (cbbPosition.SelectedIndex == 1)                           //左上
        {
            g1.DrawString(txt, f, b, 30,30);
```

（3）程序的主要代码如下：

```
private void AddDate()
{
    Font normalContentFont = new Font("宋体", 36,FontStyle.Bold);          //设置字体
    Color normalContentColor = Color.Red;                                  //设置颜色
    int kk = 1;
    toolStripProgressBar1.Maximum = listBox1.Items.Count;                  //设置进度条的最大值
    toolStripProgressBar1.Minimum = 1;                                     //设置进度条的最小值
    toolStripStatusLabel1.Text = "开始添加数码相片拍摄日期";                 //显示状态
    for (int i = 0; i < listBox1.Items.Count; i++)
    {
        pi = GetExif(listBox1.Items[i].ToString());
        TakePicDateTime = GetDateTime(pi);                                 //获取元数据中的拍照日期时间，以字符串形式保存
        SpaceLocation = TakePicDateTime.IndexOf(" ");                      //分析字符串分别保存拍照日期和时间的标准格式
        pdt = TakePicDateTime.Substring(0, SpaceLocation);
        pdt = pdt.Replace(":", "-");
        ptm = TakePicDateTime.Substring(SpaceLocation + 1, TakePicDateTime.Length - SpaceLocation - 2);
        TakePicDateTime = pdt + " " + ptm;
        Pic = new Bitmap(listBox1.Items[i].ToString());                    //由列表中的文件创建内存位图对象
        g = Graphics.FromImage(Pic);                                       //由位图对象创建 Graphics 对象
        //绘制数码照片的日期/时间
        g.DrawString(TakePicDateTime, normalContentFont, new SolidBrush(normalContentColor),
        Pic.Width - 700, Pic.Height - 200);
        if (txtSavePath.Text.Length == 3)                                  //将添加日期/时间戳后的图像进行保存
        {
            Pic.Save(txtSavePath.Text + Path.GetFileName(listBox1.Items[i].ToString()));
        }
        else
        {
            Pic.Save(txtSavePath.Text +"\\"+ Path.GetFileName(listBox1.Items[i].ToString()));
        }
        Pic.Dispose();                                                     //释放内存位图对象
        toolStripProgressBar1.Value = kk;
        if (kk == listBox1.Items.Count)
        {
            toolStripStatusLabel1.Text = "全部数码相片拍摄日期添加成功";
            toolStripProgressBar1.Visible = false;
            flag = null;
            listBox1.Items.Clear();
        }
        kk++;
    }
}
```

秘笈心法

心法领悟 546：使用线程代替 Timer 组件。

Timer 组件可以按用户定义的时间间隔来引发 Tick 事件，该事件一般为周期性的，每隔若干秒或若干毫秒执行一次，Timer 组件工作在窗体线程中，如果该组件中执行了较为耗时的操作，会增加窗体线程的负担，导致窗体中其他操作不能及时得到 CPU 资源，出现窗体长时间或短时间无响应的情况。在适当的情况下可以使用线程来代替 Timer 组件，这样会减少窗体线程的负担。

实例 547　批量添加图片水印

光盘位置：光盘\MR\22\547　　高级　　趣味指数：★★★★★

实例说明

现在网络已经非常普及，很多人选择网上购物。在网上开店的店主们会精心地处理自己的商品图片，并且还会在图片上加上水印，突出自己的店铺风格，实现广告效应。如果多个图片想添加同一个水印，那么会变得非常的复杂，所以本实例实现批量添加水印的功能，通过本实例可以为多个图片添加文字或者图片水印。实例

实例 546	为数码照片添加日期	中级
	光盘位置：光盘\MR\22\546	趣味指数：★★★★★

实例说明

随着数码相机逐渐成为主流，每人具备一台数码相机已经不是什么新鲜事了。与以胶卷为感光介质的普通相机相比，数码相机可以将所照图像即刻转换成计算机可识别的图像文件格式，以便浏览、共享和打印。数码相机拍摄的相片通常都会显示拍摄日期，但是，如果没有开启数码相机的这个功能，可能导致数码照片上没有拍摄日期，为了解决这个问题，开发出本实例批量为数码照片添加拍摄日期。实例运行效果如图 22.53 所示。

图 22.53　为数码照片添加日期

关键技术

本实例在实现使背景图片居中显示时，首先需要通过窗体的 BackgroundImage 属性设置图片，然后将窗体的 BackgroundImageLayout 属性设置为 ImageLayout.Tile 枚举值。

目前大部分数码相机都将拍摄的图像保存成 JPG 格式，而像拍摄日期这样的信息统称为 EXIF 信息。EXIF 信息是由数码相机在拍摄过程中采集的一系列信息，然后把信息放置在 JPG 文件或者 TIFF 文件的头部，也就是说 EXIF 信息是镶嵌在图像文件格式内的一组拍摄参数，主要包括摄影时的光圈、快门、ISO、拍摄日期等各种与当时摄影条件相关的信息。

EXIF 信息以键值对的方式保存在数码 JPG 图像文件的头部，在.NET 平台中，所有图像文件头部信息统称为元数据，可以使用 GDI+读取现有的元数据，也可以将新的元数据写入图像文件中。GDI+将单独的元数据段存储在 PropertyItem 对象中，读取 Image 对象的 PropertyItems 属性以便从某个文件中检索所有的元数据。PropertyItems 属性返回一个 PropertyItem 对象的数组，PropertyItem 对象具有以下 4 个属性。

- Id：用于标识元数据项的标记。
- Value：为数组值，这些值的格式由 Type 属性确定。
- Len：指向的值的数组长度（以字节表示）。
- Type：指向数组中值的数据类型。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 IMGAddDate。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来显示需要添加拍摄日期的图片列表；添加一个 OpenFileDialog 控件，用来选择要添加拍摄日期的图片；添加一个 FolderBrowserDialog 控件，用来选择保存路径；添加一个 TextBox 控件，用来显示保存路径；添加两个 Button 控件，分别用来执行添加拍摄日期和退出程序操作。

```
            }
            if (cbbPosition.SelectedIndex ==2)                                    //左下
            {
                g1.DrawString(txt, f, b, 30, (int)(BigBt.Height - Fheight)-30);
            }
            if (cbbPosition.SelectedIndex == 3)                                   //右上
            {
                g1.DrawString(txt, f, b, (int)(BigBt.Width - Fwidth), 30);
            }
            if (cbbPosition.SelectedIndex == 4)                                   //右下
            {
                g1.DrawString(txt, f, b, (int)(BigBt.Width - Fwidth), (int)(BigBt.Height - Fheight)-30);
            }
        }
        else                                                                      //如果设置了字体
        {
            f = new Font(FontF, fontSize, fontStyle);                             //创建字体对象
            SizeF XMaxSize = g.MeasureString(txt, f);                             //获取字体的大小
            Fwidth = (int)XMaxSize.Width;                                         //字体宽度
            Fheight = (int)XMaxSize.Height;                                       //字体高度
            //绘制文字水印
            g.DrawString(txt, new Font(FontF, fontSize, fontStyle), b, (int)(368 - Fwidth) / 2, (int)(75 - Fheight) / 2);
            if (cbbPosition.SelectedIndex == 0)                                   //正中
            {
                g1.DrawString(txt, new Font(FontF, fontSize, fontStyle), b, (int)(BigBt.Width - Fwidth) / 2,
                (int)(BigBt.Height - Fheight) / 2);
             }
             if (cbbPosition.SelectedIndex == 1)                                  //左上
             {
                g1.DrawString(txt, new Font(FontF, fontSize, fontStyle), b, 30, 30);
             }
             if (cbbPosition.SelectedIndex == 2)                                  //左下
             {
                g1.DrawString(txt, new Font(FontF, fontSize, fontStyle), b, 30, (int)(BigBt.Height - Fheight)-30);
             }
             if (cbbPosition.SelectedIndex == 3)                                  //右上
             {
                g1.DrawString(txt, new Font(FontF, fontSize, fontStyle), b, (int)(BigBt.Width - Fwidth), Fheight);
             }
             if (cbbPosition.SelectedIndex == 4)                                  //右下
             {
                g1.DrawString(txt, new Font(FontF, fontSize, fontStyle), b,
                  (int)(BigBt.Width - Fwidth), (int)(BigBt.Height - Fheight)-30);
        }
    }
    if (i == 1)                                                                   //i 为 1 时，保存绘制了文字水印的图片
    {
        string ipath;                                                             //加水印后的图片路径
        if (NewFolderPath.Length == 3)                                            //判断是不是系统磁盘
            ipath = NewFolderPath.Remove(NewFolderPath.LastIndexOf(":") + 1);     //如果是系统磁盘则删除“：”后边的字符
        else                                                                      //否则
            ipath = NewFolderPath;                                                //获取路径
        //获取图片类型
        string imgstype = Iname.Substring(Iname.LastIndexOf(".") + 1, Iname.Length - 1 - Iname.LastIndexOf("."));
        if (imgstype.ToLower() == "jpeg" || imgstype.ToLower() == "jpg")          //如果原图是 jpeg 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Jpeg);                  //加水印后依然保存成 jpeg
        }
        if (imgstype.ToLower() == "png")                                          //如果原图是 png 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Png);                   //加水印后依然保存成 png 格式
        }
        if (imgstype.ToLower() == "bmp")                                          //如果原图是 bmp 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Bmp);                   //加水印后依然保存成 bmp
        }
        if (imgstype.ToLower() == "gif")                                          //如果原图是 gif 格式
```

```
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Gif);                    //加水印后依然保存成 gif
        }
        g1.Dispose();
        BigBt.Dispose();
    }
    if (i == 2)                                                                    //加完图片水印后预览
    {
        if (cbbPosition.SelectedIndex == 0)                                        //正中
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width) / 2, (int)(BigBt.Height - effect.Height) / 2);
        }
        if (cbbPosition.SelectedIndex == 1)                                        //左上
        {
            g1.DrawImage(effect, 30, 30);
        }
        if (cbbPosition.SelectedIndex == 2)                                        //左下
        {
            g1.DrawImage(effect, 30, (int)(BigBt.Height - effect.Height) - 30);
        }
        if (cbbPosition.SelectedIndex == 3)                                        //右上
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width)-30, 30);
        }
        if (cbbPosition.SelectedIndex == 4)                                        //右下
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width)-30, (int)(BigBt.Height - effect.Height) - 30);
        }
    }
    if (i == 3)                                                                    //加完图片水印后保存
    {
        if (cbbPosition.SelectedIndex == 0)                                        //正中
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width) / 2, (int)(BigBt.Height - effect.Height) / 2);
        }
        if (cbbPosition.SelectedIndex == 1)                                        //左上
        {
            g1.DrawImage(effect, 30, 30);
        }
        if (cbbPosition.SelectedIndex == 2)                                        //左下
        {
            g1.DrawImage(effect, 30, (int)(BigBt.Height - effect.Height) - 30);
        }
        if (cbbPosition.SelectedIndex == 3)                                        //右上
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width), 30);
        }
        if (cbbPosition.SelectedIndex == 4)                                        //右下
        {
            g1.DrawImage(effect, (int)(BigBt.Width - effect.Width), (int)(BigBt.Height - effect.Height) - 30);
        }
        string ipath;
        if (NewFolderPath.Length == 3)
            ipath = NewFolderPath.Remove(NewFolderPath.LastIndexOf(":") + 1);
        else
            ipath = NewFolderPath;
        string imgstype = Iname.Substring(Iname.LastIndexOf(".") + 1, Iname.Length - 1 - Iname.LastIndexOf("."));
        if (imgstype.ToLower() == "jpeg" || imgstype.ToLower() == "jpg")           //如果原图是 jpeg 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Jpeg);                   //加水印后依然保存成 jpeg 格式
        }
        if (imgstype.ToLower() == "png")                                           //如果原图是 png 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Png);                    //加水印后依然保存成 png 格式
        }
        if (imgstype.ToLower() == "bmp")                                           //如果原图是 bmp 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Bmp);                    //加水印后依然保存成 bmp 格式
```

```
        }
        if (imgstype.ToLower() == "gif")                    //如果原图是 gif 格式
        {
            BigBt.Save(ipath + "\\_" + Iname, ImageFormat.Gif);    //加水印后依然保存成 gif 格式
        }
    }
}
```

秘笈心法

心法领悟 547：如何保存图片为指定格式？

在保存图片为指定格式时，可以使用 Bitmap 类的 Save 方法来实现，该方法有多种重载形式，其最常用的一种用来将 Image 图片以指定格式保存到指定文件。语法格式如下：

```
public void Save(string filename,ImageFormat format)
```

参数说明

❶ filename：字符串，包含要将 Image 图片保存到的文件的名称。

❷ format：Image 图片的保存格式，通过 ImageFormat 类的静态属性设置。

实例 548　仿 QQ 截图

光盘位置：光盘\MR\22\548　　高级　　趣味指数：★★★★★

实例说明

在工作中，有时需要抓取图片。如果通过 Windows 系统自带的抓图工具，只能抓取当前窗体的图片，而不能抓取局部图片。QQ 提供了抓图工具，可以通过鼠标选取要抓取的范围，使用起来非常方便，本实例实现了与 QQ 截图类似的功能。实例运行效果如图 22.55 所示。

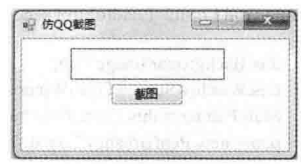


图 22.55　仿 QQ 截图

关键技术

本实例实现仿 QQ 截图功能，在绘制鼠标拖出的区域时用到了 Graphics 类的 DrawRectangle 方法；双击鼠标将选择的区域保存在系统剪贴板中，这一功能是通过 Clipboard 类的 SetImage 方法实现的。下面对本实例中用到的关键技术进行详细讲解。

（1）Graphics 类的 DrawRectangle 方法

Graphics 类的 DrawRectangle 方法用来绘制由坐标对、宽度和高度指定的矩形。语法格式如下：

```
public void DrawRectangle(Pen pen,int x,int y,int width,int height)
```

DrawRectangle 方法中的参数说明如表 22.24 所示。

表 22.24　DrawRectangle 方法中的参数说明

参　数	描　述
pen	Pen 对象，确定矩形的颜色、宽度和样式
x	要绘制矩形的左上角的 x 坐标
y	要绘制矩形的左上角的 y 坐标
width	要绘制矩形的宽度
height	要绘制矩形的高度

（2）Clipboard 类的 SetImage 方法

Clipboard 类的 SetImage 方法用来将 Bitmap 数据存储在剪贴板上。语法格式如下：

```
public static void SetImage(BitmapSource image)
```

参数说明

image：一个 BitmapSource 对象，其中包含要存储在剪贴板上的图像数据。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ScreenCutter。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件，用来实现仿 QQ 截图功能。

（3）在项目中添加一个新的 Windows 窗体，并将其命名为 Frm_Browser.cs，用来实现截图功能。

（4）程序的主要代码如下。

Frm_Main 窗体的后台代码中，首先获取整个屏幕的抓图，然后将其传递给第二个窗体，以便截取指定区域，代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //创建 Image 对象，用于存储整个屏幕的抓图
    Image img = new Bitmap(Screen.AllScreens[0].Bounds.Width, Screen.AllScreens[0].Bounds.Height);
    Graphics g = Graphics.FromImage(img);                                    //创建 Graphics 对象
    g.CopyFromScreen(new Point(0, 0), new Point(0, 0), Screen.AllScreens[0].Bounds.Size);  // CopyFromScreen 方法捕获整个屏幕图像
    IntPtr dc = g.GetHdc();                                                  //获取句柄
    g.ReleaseHdc(dc);                                                        //释放
    Frm_Browser frm2 = new Frm_Browser ();                                   //创建 Frm_Browser 对象
    frm2.ig = img;                                                           //将 img 传递给 ig
    frm2.Show();;                                                            //打开 Frm_Browser 窗体
}
```

Frm_Browser 窗体的后台代码中，窗体的 Load 事件首先将 Frm_Main 传递过来的屏幕截图设为窗体背景，然后最大化窗体，这样就可以在 Frm_Browser 窗体上使用鼠标选择截图的区域，以便保存。代码如下：

```
private void Form2_Load(object sender, EventArgs e)
{
    this.BackgroundImage = ig;                                               //将捕获的屏幕图像作为窗体的背景
    this.WindowState = FormWindowState.Maximized;                            //最大化窗体
    MainPainter = this.CreateGraphics();                                     //创建 Graphics 对象
    pen = new Pen(Brushes.Tomato);                                           //创建 Pen 对象
    isDowned = false;                                                        //判断鼠标是否按下
    baseImage = this.BackgroundImage;                                        //获取捕获的屏幕图像
    Rect = new Rectangle();                                                  //创建 Rectangle 对象
    RectReady = false;                                                       //是否出现边框
    change = false;
    Rectpoints = new Rectangle[8];
    for (int i = 0; i < Rectpoints.Length; i++)
    {
        Rectpoints[i].Size = new Size(4, 4);
    }
}
```

当鼠标移动时，会触发窗体的 MouseMove 事件，在此事件中，判断鼠标左键是否按下，如果按下，则开始获取鼠标拖动后绘制的选区。代码如下：

```
private void Form2_MouseMove(object sender, MouseEventArgs e)
{
    if (RectReady == true)
    {
        if (Rect.Contains(e.X, e.Y))
        {
            if (isDowned == true && change == false)
            {
                //和上一次的位置比较获取偏移量
                Rect.X = Rect.X + e.X - tmpx;
                Rect.Y = Rect.Y + e.Y - tmpy;
                //记录现在的位置
                tmpx = e.X;
                tmpy = e.Y;
                MoveRect((Image)baseImage.Clone(), Rect);                    //移动选择区域
            }
        }
    }
```

```
        else
        {
            if (isDowned == true)                                    //如果按下鼠标左键
            {
                Image New = DrawScreen((Image)baseImage.Clone(), e.X, e.Y);    //开始绘制选区
                MainPainter.DrawImage(New, 0, 0);
                New.Dispose();
            }
        }
}
```

当双击鼠标时，会触发窗体的 DoubleClick 事件，在此事件中，判断是否双击了鼠标左键，如果双击鼠标左键则获取鼠标选择的区域，并存入系统剪贴板中。代码如下：

```
private void Form2_DoubleClick(object sender, EventArgs e)
{
    if (((MouseEventArgs)e).Button == MouseButtons.Left && Rect.Contains(((MouseEventArgs)e).X, ((MouseEventArgs)e).Y))
    {
        Image memory = new Bitmap(Rect.Width-1, Rect.Height-1);          //创建 Image 对象，用于记录选定的区域
        Graphics g = Graphics.FromImage(memory);                         //创建 Graphics 对象
        g.CopyFromScreen(Rect.X+1, Rect.Y+1,0, 0, Rect.Size);            // Graphics 方法捕获选定的区域
        Clipboard.SetImage(memory);                                      //复制到剪贴板
        this.Close();                                                    //关闭当前窗口
    }
}
```

秘笈心法

心法领悟 548：仿 QQ 截图的实现原理。

在实现仿 QQ 截图功能时，首先需要抓取整个屏幕图片，然后在图片上按住鼠标左键，拖动鼠标选择区域，当双击鼠标左键时，将选择的区域保存下来。

实例 549　模拟石英钟

光盘位置：光盘\MR\22\549　　中级　　趣味指数：★★★★★

实例说明

实际生活中大家对石英钟并不陌生，它能够帮助人们计量时间。随着计算机的普及，电子石英钟也随之诞生。电子石英钟能够模仿真实石英钟的样式，以系统时间为基础，显示当前时间。本实例使用 C#制作了一个模拟电子石英钟，实例运行效果如图 22.56 所示。

图 22.56　模拟石英钟

关键技术

本实例实现时的主要难点是如何计算和绘制电子时钟的指针。在计算指针长度时使用了 Math 类的 Min 和 Cos 方法，绘制电子时钟指针时主要用到了 Graphics 类的 DrawLine 方法。下面对本实例中用到的关键技术进行详细讲解。

（1）Math 类的 Min 方法

Math 类为三角函数、对数函数和其他通用数学函数提供常数和静态方法，其 Min 方法用来返回两个 32 位有符号整数中较小的一个，该方法的语法格式如下：

```
public static int Min(int val1, int val2)
```

参数说明

❶ val1：要比较的两个 32 位有符号的整数中的第一个。

❷ val2：要比较的两个 32 位有符号的整数中的第二个。

❸ 返回值：val1 或 val2 参数中较小的一个。

（2）Math 类的 Cos 方法

该方法返回指定角度的余弦值。语法格式如下：

```
public static double Cos(double d)
```

参数说明

❶ d：以弧度计量的角度。

❷ 返回值：d 的余弦值。

（3）Graphics 类的 DrawLine 方法

Graphics 类的 DrawLine 方法主要用来绘制一条连接由坐标对指定的两个点的线条。语法格式如下：

```
public void DrawLine (Pen pen,Point pt1,Point pt2)
```

参数说明

❶ pen：Pen 对象，它确定线条的颜色、宽度和样式。

❷ pt1：Point 结构，它表示要连接的第一个点。

❸ pt2：Point 结构，它表示要连接的第二个点。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 QuartzClock。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来设置时钟的背景图片；添加一个 Timer 计时器组件，用来控制时分秒针的走动；添加两个 ContextMenuStrip 控件，分别用来作为时钟的快捷菜单和时钟托盘图标的快捷菜单；添加一个 NotifyIcon 控件，用来显示时钟托盘图标。

（3）程序的主要代码如下：

```
private void clock(Graphics g)
{
    const double pi = Math.PI;                                          //声明数学公式中的 pi
    Point center = new Point(0, 0);                                     //创建 Point 实例
    DateTime time = DateTime.Now;                                       //获取当前时间
    //读取时间
    double sec = 2.0 * pi * time.Second / 60.0;
    double min = 2.0 * pi * (time.Minute + time.Second / 60.0) / 60.0;
    double hour= 2.0 * pi * (time.Hour + time.Minute / 60.0) / 12.0;
    //各指针单位换算
    int r = Math.Min(pictureBox1.Image.Width, pictureBox1.Image.Height) / 2;
    int secHandLength = (int)(0.55 * r);
    int minHandLength = (int)(0.55 * r);
    int hourHandLength = (int)(0.45 * r);
    //刷新指针
    Pen SPen = new Pen(Color.White, 1);
    g.DrawLine(SPen, center, new Point((int)(secHandLength * Math.Sin(sec)),
      (int)(-secHandLength * Math.Cos(sec))));                          //绘制秒针
    Pen MPen = new Pen(Color.White, 2);
    g.DrawLine(MPen, center, new Point((int)(minHandLength * Math.Sin(min)),
      (int)(-minHandLength * Math.Cos(min))));                          //绘制分针
    Pen HPen = new Pen(Color.White, 2);
    g.DrawLine(HPen, center, new Point((int)(hourHandLength * Math.Sin(hour)),
      (int)(-hourHandLength * Math.Cos(hour))));                        //绘制时针
}
```

秘笈心法

心法领悟 549：如何获取当前系统时间？

获取当前系统时间时，可以使用 DateTime.Now 属性来实现，该属性用来获取一个 DateTime 对象，该对象设置为此计算机上的当前日期和时间，表示为本地时间。使用 DateTime.Now 属性获取当前系统时间之后，可以使用 ToLongDateString 方法或者 ToShortDateString 方法将其转换为长日期或短日期格式；另外，也可以使用 ToLongTimeString 方法或者 ToShortTimeString 方法将其转换为长时间或短时间格式。

实例 550　制作画桃花小游戏

光盘位置：光盘\MR\22\550

中级

趣味指数：★★★★★

实例说明

本实例通过动态创建 PictureBox 控件实例实现了绘制桃花的效果。运行本实例，在窗体的左侧区域单击“花骨朵”“花蕾”“开花”中的任意图片控件，然后在窗体的右侧区域单击鼠标，则会在鼠标的单击位置出现对应的图形。实例运行效果如图 22.57 所示。

图 22.57　制作画桃花小游戏

关键技术

本实例在绘制桃花的过程中，主要用到 Point 类的构造方法、Size 类的构造方法和 ControlCollection 类的 Add 方法，下面分别对这 3 个方法进行详细讲解。

（1）Point 类的构造方法

Point 类的构造方法用来创建点结构的实例，其重载方式有多种，本实例中用到的语法格式如下：

```
public Point(int x, int y);
```

参数说明

❶ x：点的水平位置。

❷ y：点的垂直位置。

（2）Size 类的构造方法

Size 类的构造方法用来创建 Size 实例，其重载方式有多种，本实例中用到的语法格式如下：

```
public Size(int width, int height);
```

参数说明

❶ width：Size 实例的宽度分量。

❷ height：Size 实例的高度分量。

（3）ControlCollection 类的 Add 方法

该方法用于将指定的控件添加到控件集合中。语法格式如下：

```
public virtual void Add(Control value);
```

参数说明

value：要添加到控件集合中的控件实例。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawPeachBlossom。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 4 个 PictureBox 控件，分别用来显示桃树、花骨朵、花蕾和开花效果的图片。

（3）程序的主要代码如下：

```
private void pictureBox1_MouseClick(object sender, MouseEventArgs e)
{
    Point myPT = new Point(e.X, e.Y);                                          //获取鼠标单击位置
    PictureBox pbox = new PictureBox();                                        //创建 PictureBox 对象
    pbox.Location = myPT;                                                      //指定 PictureBox 控件的位置
     pbox.BackColor = Color.Transparent;                                       //设置 PictureBox 控件的背景色
    pbox.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage;      //设置 PictureBox 控件的图片显示方式
    switch (flag)                                                              //判断标记
    {
        case 0:
            pbox.Size = new System.Drawing.Size(20, 18);                       //设置 PictureBox 控件大小
            pbox.Image = Properties.Resources._2;                              //设置 PictureBox 控件要显示的图像
            break;
        case 1:
            pbox.Size = new System.Drawing.Size(30, 31);                       //设置 PictureBox 控件大小
            pbox.Image = Properties.Resources._3;                              //设置 PictureBox 控件要显示的图像
            break;
        case 2:
            pbox.Size = new System.Drawing.Size(34, 30);                       //设置 PictureBox 控件大小
            pbox.Image = Properties.Resources._1;                              //设置 PictureBox 控件要显示的图像
            break;
    }
    if (e.Button == MouseButtons.Left)                                         //判断是否单击了鼠标左键
    {
        pictureBox1.Controls.Add(pbox);                                        //把图片控件添加到桃树上
    }
}
```

秘笈心法

心法领悟 550：逻辑标记在程序设计中的重要性。

本实例中定义了一个整型字段 flag，这个字段十分重要，是本实例得以顺利实现的重要逻辑标记。当鼠标单击窗体左侧的桃花图案时，程序将为字段 flag 赋一个整型值，当鼠标在窗体右侧单击时，程序将根据 flag 标记为创建的图片控件设置不同的 Image 属性值。从本实例使用 flag 标记中，总结出这样的开发经验，当一个逻辑的执行有多种情况时，可以首先定义一个逻辑标记，用来保存当前的执行状态，然后程序通过判断该逻辑标记的值来执行对应的程序代码。

实例 551　打造自己的开心农场

光盘位置：光盘\MR\22\551　　高级　　趣味指数：★★★★★

实例说明

开心农场、QQ 农场可谓近几年来最为广泛流行的虚拟现实的小游戏，本实例即实现了类似 QQ 农场的小游戏。运行本实例，单击窗体中的“播种”“生长”“开花”等按钮，在窗体中将显示出农作物在不同时期的图像，如果单击按钮的顺序不符合农作物的实际生长过程，则程序会给予信息提示。实例运行效果如图 22.58 所示。

图 22.58　打造自己的开心农场

关键技术

本实例实现时主要用到了 Rectangle 类的 Contains 方法、IEnumerable<T>泛型接口和 Enumerable 类的 Cast 方法，下面分别对它们进行详细讲解。

（1）判断鼠标是否在窗体工作区之内

在本实例中，在单击“播种”按钮之后，种子图片将跟随鼠标移动，但要求鼠标必须在窗体的工作区之内，判断鼠标是否在窗体的工作区之内，可以使用 Rectangle 类的 Contains 方法，该方法的重载方式有多种，本实例用到的语法格式如下：

```
public bool Contains(Point pt);
```

参数说明

pt：要测试的点实例。

（2）使用 IEnumerable<T>泛型接口枚举集合元素

本实例使用 Linq 查询技术获取在窗体中创建的所有 CPictureBox 控件的集合，并将这些控件集合传递给 IEnumerable<Control>接口的引用。IEnumerable<Control>接口是 IEnumerable<T>泛型接口的一种形式，IEnumerable<T>泛型接口是所有 C# 2.0 常规集合类和数组都实现的一个接口，这个接口可以用来枚举集合或数组中的元素。

本实例使用 IEnumerable<Control >接口枚举所有 CPictureBox 控件。代码如下：

```
IEnumerable<Control> cons = this.GetCPictureBoxes();                    //获取所有 CPictureBox 控件的集合
foreach (CPictureBox cpbx in cons)                                       //遍历 CPictureBox 控件集合
{
    cpbx.Image = ChuffedFarm.Properties.Resources.fruit;                 //设置 CPictureBox 控件的 Image 属性
}
```

（3）使用 Enumerable 类的 Cast 扩展方法实现类型转换

本实例通过窗体的 Controls 属性获取了窗体中的所有控件的集合，该属性的类型为 Control.ControlCollection，这种类型的集合无法直接使用 Linq 技术进行查询，需要转换类型，所以本实例使用 Enumerable 类的 Cast 扩展方法将 Control.ControlCollection 类型的集合转换为 IEnumerable<Control>类型。Cast 方法的语法格式如下：

```
public static IEnumerable<TResult> Cast<TResult>(this IEnumerable source);
```

参数说明

❶ TResult：集合中元素要转换的数据类型。

❷ source：包含要转换类型的元素的集合。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ChuffedFarm。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 5 个 PictureBox 控件，分别用来实现播种、生长、开花、结果和收获功能；添加一个 Label 控件，用来显示仓库中果实的信息；添加一个 ToolTip 控件，用来作为图片控件的信息提示助手。

（3）程序的主要代码如下。

Frm_Main 窗体中，单击“播种”图片控件，将触发该图片控件的 Click 事件。在该事件中，程序首先创建

一个 CPictureBox 图片控件，用来显示种子图片，然后设置该图片控件的相关属性。“播种”图片控件的 Click 事件代码如下：

```
private void pbxInseminate_Click(object sender, EventArgs e)
{
    if (PlantState != PlantState.Nothing && PlantState != PlantState.Harvest&&PlantState!=
PlantState.Inseminate)                                          //若农场中存在未收获的农作物
    {
        MessageBox.Show("还未收获，无法播种!");
        return;
    }
    this.PlantState = PlantState.Inseminate;                    //设置农作物处于播种状态
    this.cpbxSeed = new CPictureBox();                          //创建 CPictureBox 图片控件
    this.cpbxSeed.BackColor = System.Drawing.Color.Transparent; //设置图片控件的背景颜色为透明色
    //设置图片控件背景图像的布局，这里设置图像将沿图片控件的矩形工作区拉伸
    this.cpbxSeed.BackgroundImageLayout = System.Windows.Forms.ImageLayout.Stretch;
    this.cpbxSeed.Image = ChuffedFarm.Properties.Resources.seed2;   //设置 Image 属性，让图片控件显示种子图片
    this.cpbxSeed.Size = new System.Drawing.Size(ChuffedFarm.Properties.Resources.seed2.Width,
ChuffedFarm.Properties.Resources.seed2.Height);                 //设置图片控件大小
    //设置图片控件位置
    this.cpbxSeed.Location = new
System.Drawing.Point(this.pbxInseminate.Location.X-50,this.pbxInseminate.Location.Y - 80);
    this.cpbxSeed.TabStop = true;                               //设置图片控件接收 Tab 键的焦点
    this.cpbxSeed.IsInseminate = false;                         //表示种子还未种下
    //把图片控件的 Click 事件绑定到 cpbxSeed_Click 方法
    this.cpbxSeed.Click += new System.EventHandler(this.cpbxSeed_Click);
    this.Controls.Add(this.cpbxSeed);                           //把图片控件添加到窗体中
    tipSeed.SetToolTip(this.cpbxSeed, "这是种子");               //给图片控件设置提示助手
}
```

单击“播种”图片控件之后，在窗体中移动鼠标，会发现生成的图片控件随着鼠标移动。鼠标在窗体中移动时，会触发窗体的 MouseMove 事件，在该事件中设置了图片控件的 Location 属性值为鼠标当前的位置。窗体的 MouseMove 事件代码如下：

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    if (this.cpbxSeed != null)                                  //若已生成 CPictureBox 控件实例
    {
        if (this.Bounds.Contains(Cursor.Position))              //若光标在窗体内
        {
            if (this.cpbxSeed.IsInseminate == false)            //若种子还未种下
            {
                if (this.PlantState == PlantState.Inseminate)   //若农作物处于种植状态
                {

                    this.cpbxSeed.Location = new Point(e.X-20 , e.Y-40 );   //设置图片控件的 Location 属性值为鼠标当前的位置
                }
            }
        }
    }
}
```

在种子种下之后，可按照农作物的生长顺序，依次单击各个图片按钮，农场中的农作物会呈现不同的外观状态，当农作物结果之后，单击“收获”图片控件，程序将清空农场中的所有农作物，并记录下收获的果实数量。单击“收获”图片控件，会触发该控件的 Click 事件。代码如下：

```
private void pbxHarvest_Click(object sender, EventArgs e)
{
    if (PlantState == PlantState.Nothing)                       //若还未播种
    {
        MessageBox.Show("还未播种，能收获吗？","信息提示！");
        return;
    }
    if (PlantState == PlantState.Inseminate)                    //若刚播完种子
    {
        MessageBox.Show("刚刚播完种，还未生长，能收获吗？","信息提示！");
        return;
    }
```

```
    if (PlantState == PlantState.Vegetate)                          //若农作物还处在生长期
    {
        MessageBox.Show("还处在生长期，并未开花，能收获吗？", "信息提示！");
        return;
    }
    if (PlantState == PlantState.BlossomOut)                        //若农作物还处在开花期
    {
        MessageBox.Show("正在花期，并未结果，能收获吗？", "信息提示！");
        return;
    }
    if (PlantState == PlantState.Harvest)                           //若农作物已被收获完毕
    {
        MessageBox.Show("都已经收过了？", "信息提示！");
        return;
    }
    IEnumerable<Control> cons = this.GetCPictureBoxes();            //获取所有的 CPictureBox 控件实例
    foreach (CPictureBox cpbx in cons)
    {
        intAmount++;                                                //记录收获果实的数量
        cpbx.Dispose();                                             //销毁当前的 CPictureBox 控件实例
    }
    if (cons.Count<Control>() > 0)                                  //若图片控件集合中还有元素
    {
        pbxHarvest_Click(sender, e);                                //递归调用 pbxHarvest_Click 方法
    }
    this.PlantState = PlantState.Harvest;                           //设置农作物处于收获完毕状态
    lbAmount.Text = "你的仓库里有" + intAmount.ToString()+"个果实！";
}
```

秘笈心法

心法领悟 551：轻松实现动画按钮。

当鼠标移入或移出图片按钮时，可以使图片按钮显示动态效果，这种动态效果增强了软件的操作性。当鼠标移入图片控件区域时，会触发图片控件的 MouseEnter 事件；当鼠标移出图片控件区域时，会触发图片控件的 MouseLeave 事件，在这两个事件中为图片控件设置不同的 Image 属性值，即可实现图片按钮的动态效果。

第23章

常用图表应用

- 简单图表的绘制
- 柱形图表
- 折线图表
- 饼型图表

23.1　简单图表的绘制

实例 552	绘制柱形图 光盘位置：光盘\MR\23\552	初级 趣味指数：★★★★

实例说明

本实例通过绘制简单的柱形图让读者初步了解绘制柱形图的技术，实例运行效果如图 23.1 所示。

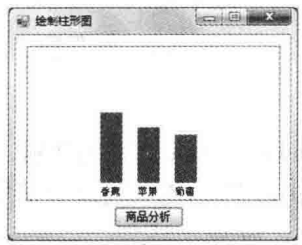


图 23.1　绘制柱形图

关键技术

本实例在绘制柱形图时，主要通过调用 Graphics 类中的 FillRectangle 方法实现，下面对其进行详细介绍。

Graphics 类中的 FillRectangle 方法用于填充由一对坐标、一个宽度和一个高度指定的矩形的内部。语法格式如下：

```
public void FillRectangle(Brush brush,int x,int y,int width,int height)
```

FillRectangle 方法中的参数说明如表 23.1 所示。

表 23.1　FillRectangle 方法中的参数说明

参　数	说　明
brush	确定填充特性的 Brush
x	要填充的矩形的左上角的 x 坐标
y	要填充的矩形的左上角的 y 坐标
width	要填充的矩形的宽度
height	要填充的矩形的高度

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawColumn。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用于显示绘制的柱形图。

（3）程序的主要代码如下：

```
private void ShowPic()
{
    Conn();                                                                    //打开数据库连接
    using (cmd = new SqlCommand("SELECT TOP 3 * FROM tb_Rectangle order by t_Num desc", con))
    {
        SqlDataReader dr = cmd.ExecuteReader();                                //创建 SqlDataReader 对象
        Bitmap bitM = new Bitmap(this.panel1.Width, this.panel1.Height);       //创建画布
        Graphics g = Graphics.FromImage(bitM);                                 //创建 Graphics 对象
        g.Clear(Color.White);                                                  //设置画布背景
        for (int j = 0; j < 4; j++)                                            //开始读取数据库中的数据并绘图
```

```
        {
            if (dr.Read())                                                    //读取记录集
            {
                int x, y, w, h;                                               //声明变量存储坐标和大小
                g.DrawString(dr[0].ToString(), new Font("宋体", 8, FontStyle.Regular), new SolidBrush(Color.Black), 76 + 40 * j, this.panel1.Height - 16);
                                                                              //绘制文字
                x = 78 + 40 * j;                                              //x 坐标
                //y 坐标
                y = this.panel1.Height - 20 - Convert.ToInt32((Convert.ToDouble(Convert.ToDouble(dr[1].ToString()) * 20 / 100)));
                 w = 24;                                                      //宽
                 h = Convert.ToInt32(Convert.ToDouble(dr[1].ToString()) * 20 / 100);   //高
                 g.FillRectangle(new SolidBrush(Color.FromArgb(56, 129, 78)), x, y, w, h); //开始绘制柱形图
            }
        }
        this.panel1.BackgroundImage = bitM;                                   //显示绘制的柱形图
    }
}
```

秘笈心法

心法领悟 552：FillRectangle 方法和 DrawRectangle 方法的区别。

FillRectangle 方法用来填充矩形的内部，而 DrawRectangle 方法用来绘制矩形。

实例 553	绘制面形图	初级
	光盘位置：光盘\MR\23\553	趣味指数：★★★★

实例说明

面形图在图表控件中是十分重要的，通过面形图可以直观地显示出各数据之间的变化趋势。本实例绘制一个单列显示数据的面形图，实例运行效果如图 23.2 所示。

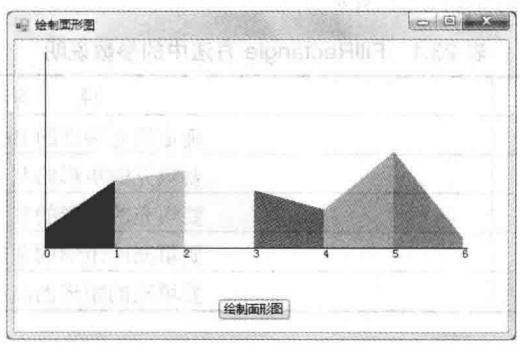


图 23.2　绘制面形图

关键技术

本实例在绘制面形图时，主要用到了 Graphics 类中的 DrawLine 和 FillPolygon 方法，下面分别对它们进行详细讲解。

（1）DrawLine 方法

该方法用于绘制一条连接两个 Point 结构的线。语法格式如下：

```
public void DrawLine(Pen pen,Point pt1,Point pt2)
```

参数说明

❶ pen：Pen 对象，确定线条的颜色、宽度和样式。

❷ pt1：Point 结构，表示要连接的第一个点。

❸ pt2：Point 结构，表示要连接的第二个点。

（2）FillPolygon 方法

该方法用于填充 Point 结构指定的点数组所定义的多边形的内部。语法格式如下：

```
public void FillPolygon(Brush brush,Point[ ] point)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ point：Point 结构数组，这些结构表示要填充的多边形的顶点。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DrawPlane。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用来显示绘制的面形图。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Bitmap bt = new Bitmap(panel1.Width, panel1.Height);                                    //创建一个 Bitmap 对象
    int flag = (panel1.Width-4 )/ 6;                                                        //X 轴的增值
    g = Graphics.FromImage(bt);                                                             //创建 Graphics 对象
    Pen p = new Pen(Color.Black, 1);                                                        //设置 Pen 对象
    g.DrawLine(p, new Point(0, 0), new Point(0, panel1.Height-20));                         //绘制 Y 轴
    g.DrawLine(p, new Point(0, panel1.Height - 20), new Point(panel1.Width - 4, panel1.Height - 20));   //绘制 X 轴
    //声明一个用于绘制颜色的数组
    Color[] cl = new Color[] { Color.Red, Color.Blue, Color.YellowGreen, Color.Yellow, Color.RoyalBlue, Color.Violet, Color .Tomato};
    int[] points = { 20,70,80,60,40,100,10};                                                //声明一个计算走势峰值的数组
    Point pt1 = new Point(0, panel1.Height - 20 - points[0]);                               //记录绘制四边形的第一个点
    Point pt2 = new Point(0, panel1.Height - 20);                                           //记录绘制四边形的第二个点
    for (int i = 0; i <= 6; i++)                                                            //通过 for 循环绘制月份和面形图
    {
        PointF p1 = new PointF(flag * i, panel1.Height - 20);                               //计算每个月份数字的坐标
        //绘制显示月份的数字
        g.DrawString(i.ToString(), new Font("宋体", 9), new SolidBrush(Color.Black), new PointF(p1.X - 2, p1.Y));
        Point pt3 = new Point(flag * i, panel1.Height - 20);                                //记录绘制四边形的第 3 个点
        Point pt4 = new Point(flag * i, panel1.Height - 20 - points[i]);                    //记录绘制四边形的第 4 个点
        Point[] pt={pt1,pt2,pt3,pt4};                                                       //声明一个 Point 数组
        g.FillPolygon(new SolidBrush(cl[i]), pt);                                           //填充四边形的颜色
        //当继续绘制下一个四边形时，前一个四边形的最后两个点作为下一个四边形的起始点
        pt1 = pt4;
        pt2 = pt3;
    }
    panel1.BackgroundImage = bt;                                                            //显示绘制的面形图
}
```

秘笈心法

心法领悟 553：使用 DrawLine 方法绘制坐标轴。

一般在绘制坐标轴时，都需要用到 DrawLine 方法。

23.2　柱 形 图 表

实例 554	利用柱形图分析汇总数据	中级
	光盘位置：光盘\MR\23\554	趣味指数：★★★★☆

实例说明

在开发数据库系统管理软件时，通常使用图表分析数据库中的汇总数据。本实例利用柱形图对水果市场进

行抽样调查，评选出最受欢迎的 3 种水果。实例运行效果如图 23.3 所示。

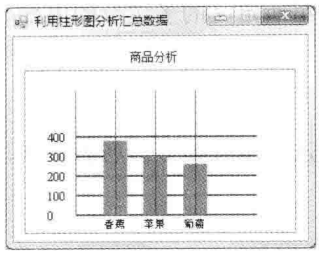


图 23.3　利用柱形图分析汇总数据

关键技术

本实例主要通过调用 Graphics 类的 FillRectangle 方法来绘制表示汇总数据的柱形图。

说明：关于 Graphics 类的 FillRectangle 方法的详细讲解请参见实例 552 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GatherDataAnalyse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用于显示绘图结果。

（3）程序的主要代码如下：

```
private void ShowPic()
{
    Conn();                                                                    //打开数据库连接
    //创建 SqlCommand 对象
    using (cmd = new SqlCommand("SELECT TOP 3 * FROM tb_Rectangle order by t_Num desc", con));
    {
        SqlDataReader dr = cmd.ExecuteReader();                                //创建 SqlDataReader 对象
        Bitmap bitM = new Bitmap(this.panel1.Width, this.panel1.Height);       //创建一个新画布
        Graphics g = Graphics.FromImage(bitM);                                 //创建 Graphics 对象
        g.Clear(Color.White);                                                  //设置画布背景
        for (int i = 0; i < 5; i++)
        {
            g.DrawLine(new Pen(new SolidBrush(Color.Red), 2.0f), 50, this.panel1.Height - 20 - i * 20, this.panel1.Width - 40, this.panel1.Height -
20 - i * 20);                                                                  //绘制水平直线
            g.DrawString(Convert.ToString(i * 100), new Font("Times New Roman", 10, FontStyle.Regular), new SolidBrush(Color.Black), 20,
this.panel1.Height - 27 - i * 20);                                             //绘制文字
        }
        for (int j = 0; j < 4; j++)
        {
            g.DrawLine(new Pen(new SolidBrush(Color.Red), 1.0f), 50+ 40 * j, this.panel1.Height - 20, 50 + 40 * j, 20);
                                                                               //绘制垂直直线
            if (dr.Read())
            {
                int x,y,w,h;                                                   //声明变量存储坐标和宽高
                g.DrawString(dr[0].ToString(), new Font("宋体", 8, FontStyle.Regular), new SolidBrush(Color.Black), 76 + 40 * j,
this.panel1.Height-16);                                                        //绘制说明文字
                x = 78+ 40 * j;                                                //X 轴
                //Y 轴
                y = this.panel1.Height - 20 - Convert.ToInt32((Convert.ToDouble(Convert.ToDouble(dr[1].ToString()) * 20 / 100)));
                w = 24;                                                        //宽
                h = Convert.ToInt32(Convert.ToDouble(dr[1].ToString()) * 20 / 100);   //高
                g.FillRectangle(new SolidBrush(Color.FromArgb(56,129,78)), x,y,w,h);  //绘制柱形图
            }
        }
        this.panel1.BackgroundImage = bitM;                                    //将画布设为 panel1 控件的背景图
    }
}
```

秘笈心法

心法领悟 554：有效回收对象资源。

在程序设计过程中，可以手动回收文件资源，但是使用 using 关键字则更为简洁友好。当 using 关键字下面的代码执行完毕后，系统会自动将 using 小括号中创建的对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 555	利用柱形图表分析商品走势 光盘位置：光盘\MR\23\555	中级 趣味指数：★★★★☆

实例说明

在实现一个具有分析功能的软件时，经常使用图表显示分析结果，图表可以使用户更直观地了解所关注的信息。本实例通过图表技术来动态地分析某商品每年的走势情况。实例运行效果如图 23.4 所示。

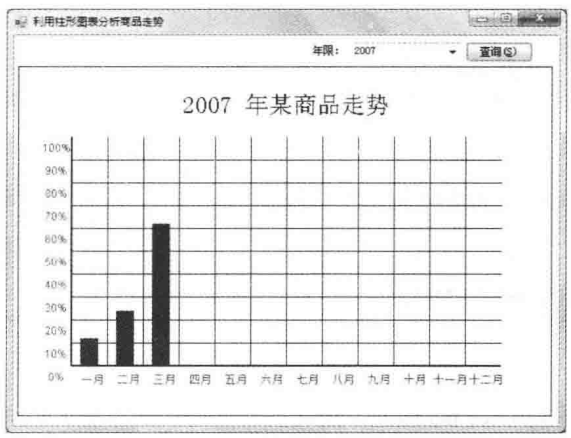


图 23.4　利用柱形图表分析商品走势

关键技术

本实例主要通过调用 Graphics 类的 FillRectangle 方法来绘制分析商品走势的柱形图。

说明：关于对 Graphics 类的 FillRectangle 方法的详细讲解请参见实例 552 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AnalyseGoodsTrend。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件、一个 ComboBox 控件和一个 Button 控件，分别用于显示绘图结果、统计年份和绘制图形。

（3）程序的主要代码如下：

```
private void CreateImage(int Year)
{
    int height = 400, width = 600;                                          //设置画布的高和宽
    System.Drawing.Bitmap image = new System.Drawing.Bitmap(width, height);  //创建一个新画布
    Graphics g = Graphics.FromImage(image);                                 //创建 Graphics 对象
    try
    {
        g.Clear(Color.White);                                               //设置画布背景色
        Font font = new System.Drawing.Font("Arial", 9, FontStyle.Regular);  //设置字体
        Font font1 = new System.Drawing.Font("宋体", 20, FontStyle.Regular);  //设置字体
        System.Drawing.Drawing2D.LinearGradientBrush brush = new System.Drawing.Drawing2D.LinearGradientBrush(new Rectangle(0, 0, image.
```

```
Width, image.Height), Color.Blue, Color.Blue, 1.2f, true);                                    //创建 LinearGradientBrush 对象
        g.FillRectangle(Brushes.WhiteSmoke, 0, 0, width, height);                             //绘制柱形图
        Brush brush1 = new SolidBrush(Color.Blue);                                            //创建 Brush 对象
        g.DrawString("" + Year + " 年某商品走势", font1, brush1, new PointF(180, 30));          //绘制说明文字
        g.DrawRectangle(new Pen(Color.Blue), 0, 0, image.Width - 4, image.Height - 4);        //画图片的边框线
        Pen mypen = new Pen(brush, 1);                                                        //创建 Pen 对象
        //绘制横向线条
        int x = 100;
        for (int i = 0; i < 11; i++)
        {
            g.DrawLine(mypen, x, 80, x, 340);                                                 //绘制线条
            x = x + 40;
        }
        Pen mypen1 = new Pen(Color.Blue, 2);                                                  //创建 Pen 对象
        g.DrawLine(mypen1, x - 480, 80, x - 480, 340);
        //绘制纵向线条
        int y = 106;
        for (int i = 0; i < 9; i++)
        {
            g.DrawLine(mypen, 60, y, 540, y);                                                 //绘制线条
            y = y + 26;
        }
        g.DrawLine(mypen1, 60, y, 540, y);
        //X 轴
        String[] n = {"一月", "二月", "三月", "四月", "五月", "六月", "七月",
                      "八月",  九月", "十月", "十一月", "十二月"};                              //绘制 X 轴的月份
        x = 62;
        for (int i = 0; i < 12; i++)
        {
            g.DrawString(n[i].ToString(), font, Brushes.Red, x, 348);                         //设置文字内容及输出位置
            x = x + 40;
        }
        //Y 轴
        String[] m = {"100%", " 90%", " 80%", " 70%", " 60%", " 50%", " 40%", " 30%",
            " 20%", " 10%", "   0%"};                                                         //绘制 Y 轴的商品上升百分比
        y = 85;
        for (int i = 0; i < 11; i++)
        {
            g.DrawString(m[i].ToString(), font, Brushes.Red, 25, y);                          //设置文字内容及输出位置
            y = y + 26;
        }
        int[] Count = new int[12];
        string cmdtxt2 = "SELECT * FROM tb_Stat WHERE ShowYear=" + Year + "";
        SqlConnection Con = new SqlConnection("server=mrwxk\\wangxiaoke;uid=sa;pwd=;database=db_TomeOne");  //建立数据库连接
        Con.Open();                                                                           //打开连接
        SqlCommand Com = new SqlCommand(cmdtxt2, Con);                                        //创建 SqlCommand 对象
        SqlDataAdapter da = new SqlDataAdapter();                                             //创建 SqlDataAdapter 对象
        da.SelectCommand = Com;
        DataSet ds = new DataSet();                                                           //创建 DataSet 对象
        da.Fill(ds);                                                                          //Fill 方法填充 DataSet
        int j = 0;
        int number = SumYear(Year);
        for (j = 0; j < 12; j++)
        {
            Count[j] = Convert.ToInt32(ds.Tables[0].Rows[0][j + 1].ToString()) * 100 / number;
        }
        //显示柱状效果
        x = 70;
        for (int i = 0; i < 12; i++)
        {
            SolidBrush mybrush = new SolidBrush(Color.Red);
            g.FillRectangle(mybrush, x, 340 - Count[i] * 26 / 10, 20, Count[i] * 26 / 10);
            x = x + 40;
        }
        this.panel1.BackgroundImage = image;
    }
    catch (Exception ey)
    {
```

```
        MessageBox.Show(ey.Message);
    }
}
```

秘笈心法

心法领悟 555：SolidBrush 类的使用。

SolidBrush 类主要用来定义单色画笔，画笔用于填充图形形状，如矩形、椭圆、扇形、多边形和封闭路径等。

实例 556　对排序数据进行分析　　中级

光盘位置：光盘\MR\23\556　　趣味指数：★★★★★

实例说明

使用图表显示统计信息时，一般数据量很大并且会使图表看起来很杂乱。如果将图表按一定条件进行排序再显示给用户，这样可以解决数据量大、图表多、用户无法很快得到想要的信息等这些棘手的问题。本实例将通过柱形图对排序数据进行分析，实例运行效果如图 23.5 和图 23.6 所示。

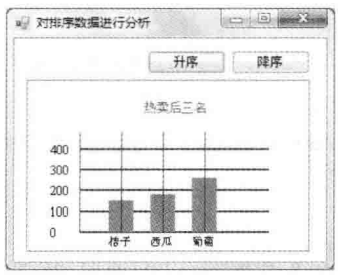


图 23.5　升序排序

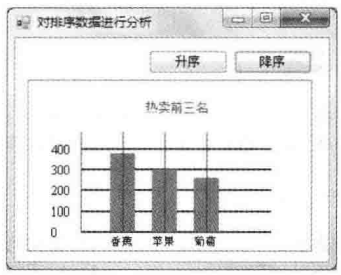


图 23.6　降序排序

关键技术

本实例实现的重点是如何通过 SQL 语句对数据进行排序，而在排完序之后，即可调用 Graphics 类的 FillRectangle 方法绘制柱形图。下面对本实例中用到的关键技术进行详细介绍。

SQL 语句的语法在索引中的各列上支持关键字 ASC（升序）和 DESC（降序），默认为升序。如何使用关键字 ASC（升序）和 DESC（降序），请看下面的例子，将 tb_Rectangle 表按销售量（t_Num 字段）查询，并显示出销售量最低的 3 种商品。代码如下：

```
SELECT TOP 3 * FROM tb_Rectangle order by t_Num desc
```

说明：关于对 Graphics 类的 FillRectangle 方法的详细讲解请参见实例 552 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AnalyseSortData。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件和两个 Button 控件，分别用来显示绘图结果和执行升序、降序绘图操作。

（3）程序的主要代码如下：

```
private void ShowPic(string str)
{
    Conn();                                                          //打开数据库连接
    //生成 SqlCommand 对象
    using (cmd = new SqlCommand("SELECT TOP 3 * FROM tb_Rectangle order by t_Num " + str, con))
    {
        SqlDataReader dr = cmd.ExecuteReader();                      //创建 SqlDataReader 对象
        Bitmap bitM = new Bitmap(this.panel1.Width, this.panel1.Height);  //新建一张画布
        Graphics g = Graphics.FromImage(bitM);                       //创建 Graphics 对象用于绘图
```

```
            g.Clear(Color.White);                                                    //设置画布背景
            for (int i = 0; i < 5; i++)
            {
                g.DrawLine(new Pen(new SolidBrush(Color.Red), 2.0f), 50, this.panel1.Height - 20 - i * 20, this.panel1.Width - 40, this.panel1.Height -
20 - i * 20);                                                                        //绘制水平线条
                g.DrawString(Convert.ToString(i * 100), new Font("Times New Roman", 10, FontStyle.Regular), new SolidBrush(Color.Black), 20,
this.panel1.Height - 27 - i * 20);                                                   //绘制商品上升数量
            }
            for (int j = 0; j < 4; j++)
            {
                //绘制垂直线条
                g.DrawLine(new Pen(new SolidBrush(Color.Red), 1.0f), 50 + 40 * j, this.panel1.Height - 20, 50 + 40 * j, 50);
                if (dr.Read())
                {
                    int x, y, w, h;                                                  //声明变量存储坐标与宽高
                    g.DrawString(dr[0].ToString(), new Font("宋体", 8, FontStyle.Regular), new SolidBrush(Color.Black), 76 + 40 * j, this.panel1.
Height - 16);                                                                        //绘制商品名称
                    x = 78 + 40 * j;                                                 //X 坐标
                    //Y 坐标
                    y = this.panel1.Height - 20 - Convert.ToInt32((Convert.ToDouble(Convert.ToDouble(dr[1].ToString()) * 20 / 100)));
                    w = 24;                                                          //宽度
                    h = Convert.ToInt32(Convert.ToDouble(dr[1].ToString()) * 20 / 100);  //高度
                    g.FillRectangle(new SolidBrush(Color.FromArgb(56, 129, 78)), x, y, w, h);  //开始绘制柱形图
                }
            }
            if (str == "desc")                                                       //如果按降序排列
            {
                //绘制提示文字，提示商品是按降序排列的
                g.DrawString("热卖前三名", new Font("宋体", 9), new SolidBrush(Color.Red), this.panel1.Width / 2 - 26, 20);
            }
            else if (str == "asc")                                                   //如果按升序排列
            {
                //绘制提示文字，提示商品是按升序排列的
                g.DrawString("热卖后三名", new Font("宋体", 9), new SolidBrush(Color.Red), this.panel1.Width / 2 - 26, 20);
            }
            this.panel1.BackgroundImage = bitM;                                      //显示绘制的图形
        }
    }
```

秘笈心法

心法领悟 556：使用 DrawString 方法绘制文本。

DrawString 方法用来在指定位置并且用指定的 Brush 和 Font 对象绘制指定的文本字符串。

实例 557　利用控件实现柱形图分析　中级

光盘位置：光盘\MR\23\557　　趣味指数：★★★★☆

实例说明

读者已经了解了通过 GDI+来绘制图表的方法，那么不用 GDI+能不能实现图形显示的效果呢？答案是肯定的。本实例通过控件来完成柱形图统计功能，实例运行效果如图 23.7 所示。

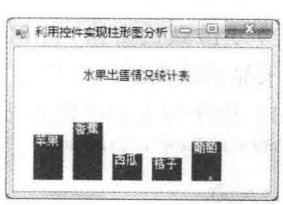


图 23.7　利用控件实现柱形图分析

关键技术

实现动态分析数据信息时需要通过 SQL 语句查询要分析的数据信息，并且将查询出的字段赋给相应的变量，然后通过变量来控制控件的显示。由于分析结果为动态值，所以定义一个动态控件数组来存放信息。最后通过 Controls 类的 Add 方法将控件添加到窗体上。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ColumnByControl。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    //创建一个 SqlConnection 对象
    using (SqlConnection con = new SqlConnection("server=mrwxk\\wangxiaoke;uid=sa;pwd=;database=db_TomeOne"))
    {
        int XValse = 20;
        DataSet ds = new DataSet();                                          //创建 DataSet 对象
        //创建 SqlCommand 对象
        SqlCommand cmd = new SqlCommand("select * from tb_Rectangle select Sum(t_Num) from tb_Rectangle", con);
        SqlDataAdapter da = new SqlDataAdapter();                            //创建 SqlDataAdapter 对象
        da.SelectCommand = cmd;
        da.Fill(ds);                                                         //Fill 方法填充 DataSet
        Panel[] p = new Panel[ds.Tables[0].Rows.Count];                      //创建一个 Panel 数组
        int Values = Convert.ToInt32(ds.Tables[1].Rows[0][0].ToString());    //商品总数
        for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
        {
            ds.Tables[0].Rows[i][0].ToString();
            float f = Convert.ToInt32(ds.Tables[0].Rows[i][1].ToString());   //获取每个商品的数量
            Size s = new Size();                                             //创建 Size 对象
            s.Width = 30;                                                    //设置柱形图宽度
            s.Height = Convert.ToInt32(f / Values * 200);                    //计算柱形图高度
            Point pint = new Point();                                        //创建 Point 对象
            pint.X = XValse;                                                 //X 坐标
            pint.Y = this.Height - 50 - s.Height;                            //Y 坐标
            p[i] = new Panel();                                              //创建一个 Panel 对象
            p[i].Location = pint;                                            //设置位置
            p[i].BackColor = Color.Red;                                      //设置背景颜色
            p[i].Size = s;                                                   //设置大小
            XValse += 40;                                                    //设置 XValse 变量的值
            Label lbl = new Label();                                         //创建 Label 对象
            lbl.Text = ds.Tables[0].Rows[i][0].ToString();                   //设置 Label 显示的文本
            lbl.Font = new Font("宋体", 9, FontStyle.Regular);               //设置 Label 的 Font 属性
            lbl.ForeColor = Color.White;                                     //设置 Label 的 ForeColor 属性
            p[i].Controls.Add(lbl);                                          //添加控件
            this.Controls.Add(p[i]);                                         //将控件数组添加到当前容器中
        }
    }
}
```

秘笈心法

心法领悟 557：有效回收对象资源。

程序设计过程中，可以手动回收文件资源，但是使用 using 关键字则更为简洁友好。当 using 关键字下面的代码执行完毕后，系统会自动将 using 小括号中创建的对象隐式转换为 IDisposable 接口类型，并调用接口方法 Dispose 释放对象资源。

实例 558　在柱形图的指定位置显示说明文字

高级

光盘位置：光盘\MR\23\558

趣味指数：★★★★★

实例说明

在实际生活中使用柱形图统计分析一些数据时，通常都会在柱形图上显示说明文字，以便能够让用户更清楚地了解统计分析的数据。本实例使用 C#实现了在柱形图上绘制说明文字的功能，实例运行效果如图 23.8 所示。

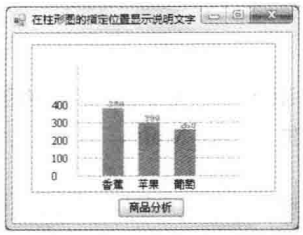


图 23.8　在柱形图的指定位置显示说明文字

关键技术

本实例在绘制柱形图及柱形图上的文字时，主要用到了 Graphics 类的 FillRectangle 和 DrawString 方法，下面分别对它们进行详细讲解。

说明： 关于对 Graphics 类的 FillRectangle 方法的详细讲解请参见实例 552 中的关键技术。

Graphics 类的 DrawString 方法主要用于在指定位置用指定的 Brush 和 Font 对象绘制文本字符串。语法格式如下：

```
public void DrawString(string s,Font font,Brush brush,PointF point)
```

DrawString 方法中的参数说明如表 23.2 所示。

表 23.2　DrawString 方法中的参数说明

参　数	说　明
s	要绘制的字符串
font	定义字符串的文本格式
brush	确定所绘制文本的颜色和纹理
point	指定所绘制文本的左上角

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TextInColumn。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用于显示绘制的标有说明文字的柱形图。

（3）程序的主要代码如下：

```
private void ShowPic()
{
    Conn();                                                                    //打开数据库
    using (cmd = new SqlCommand("SELECT TOP 3 * FROM tb_Rectangle order by t_Num desc", con))
    {
        SqlDataReader dr = cmd.ExecuteReader();                                //创建 SqlDataReader 对象
        Bitmap bitM = new Bitmap(this.panel1.Width, this.panel1.Height);       //创建画布
        Graphics g = Graphics.FromImage(bitM);                                 //创建 Graphics 对象
        Pen p = new Pen(new SolidBrush(Color.SlateGray), 1.0f);                //创建 Pen 对象
        p.DashStyle= System.Drawing.Drawing2D.DashStyle.Dash;                  //设置虚线
        g.Clear(Color.White);                                                  //设置画布颜色
        for (int i = 0; i < 5; i++)
        {
            //绘制水平线条
            g.DrawLine(p, 50, this.panel1.Height - 20 - i * 20, this.panel1.Width - 40, this.panel1.Height - 20 - i * 20);
            g.DrawString(Convert.ToString(i * 100), new Font("Times New Roman", 10, FontStyle.Regular), new SolidBrush(Color.Black), 20,
this.panel1.Height - 27 - i * 20);                                             //绘制商品的增长值
        }
        for (int j = 0; j < 4; j++)
        {
            g.DrawLine(p, 50 , this.panel1.Height - 20, 50, 20);               //绘制垂直线条
            if (dr.Read())
            {
```

```
                int x, y, w, h;                                          //声明变量存储坐标和大小
                g.DrawString(dr[0].ToString(), new Font("宋体", 9, FontStyle.Regular), new SolidBrush(Color.Black), 76 + 40 * j, this.panel1.
Height - 16);                                                            //绘制商品名称
                x = 78 + 40 * j;                                         //X 坐标
                //Y 坐标
                y = this.panel1.Height - 20 - Convert.ToInt32((Convert.ToDouble(Convert.ToDouble(dr[1].ToString()) * 20 / 100)));
                w = 24;                                                  //宽度
                h = Convert.ToInt32(Convert.ToDouble(dr[1].ToString()) * 20 / 100);   //高度
                g.FillRectangle(new SolidBrush(Color.SlateGray), x, y, w, h);         //绘制柱形图
                //在柱形图指定的位置绘制文字
                g.DrawString((h*100/20).ToString(), new Font("宋体", 8,FontStyle.Bold), new SolidBrush(Color.Tomato), new Point(x+4,y-10));
            }
        }
        this.panel1.BackgroundImage = bitM;                              //显示绘制的图形
    }
}
```

秘笈心法

心法领悟 558：FillRectangle 方法的使用。

FillRectangle 方法为可重载方法，它有 4 种重载形式，分别用来填充由 Rectangle 结构、RectangleF 结构以及一对坐标、一个宽度、一个高度指定的矩形的内部。

23.3　折 线 图 表

实例 559	利用折线图分析产品销售走势	中级
	光盘位置：光盘\MR\23\559	趣味指数：★★★★★

实例说明

在商品销售企业中商品的销售额是决策者最关心的，及时、清晰、准确地分析产品销售走势是每个商业企业所必需的。本实例通过对某地区的白金市场近半年白金价格的走势进行统计分析，实例运行效果如图 23.9 所示。

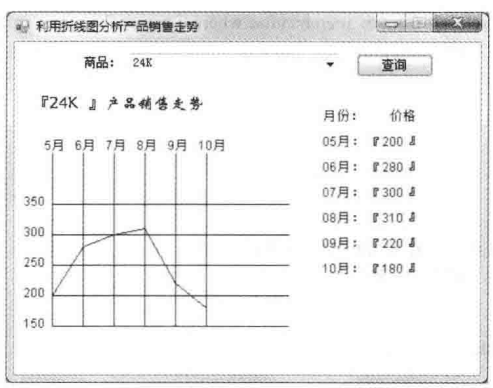


图 23.9　利用折线图分析产品销售走势

关键技术

本实例实现时主要用到了 Graphics 类中的 DrawLines、DrawLine 和 DrawString 方法，下面对本实例中用到的关键技术进行详细讲解。

DrawLines 方法用于绘制一系列连接一组 Point 结构的线段。语法格式如下：

```
public void DrawLines(Pen pen,Point[] points)
```

参数说明

❶ pen：Pen 对象，确定线段的颜色、宽度和样式。

❷ points：Point 结构数组，这些结构表示要连接的点。

说明：（1）关于 DrawLine 的详细讲解请参见实例 553 中的关键技术。

（2）关于 DrawString 的详细讲解请参见实例 558 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 GoodsSallTrend。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，用来显示白金型号；添加一个 Button 控件，用来执行绘制折线图操作；添加一个 Panel 控件，用来显示描绘图；添加两个 GroupBox 控件，分别用来显示分析图和详细信息列表图。

（3）程序的主要代码如下。

自定义 DrowPic 方法，用来绘制折线图并显示分析产品销售走势。代码如下：

```
private void DrowPic(string str)
{
    int MaxValue, MinValue;                                                    //声明变量记录最大值和最小值
    using (cmd = new SqlCommand("select Max(t_price) from tb_merchandise where t_name='" + str + "'", con))
    {
        con.Open();                                                            //打开数据库连接
        MaxValue = Convert.ToInt16(cmd.ExecuteScalar());                       //获取最大值
        con.Close();                                                           //关闭数据库连接
    }
    using (cmd = new SqlCommand("select Min(t_price) from tb_merchandise where t_name='" + str + "'", con))
    {
        con.Open();                                                            //打开数据库连接
        MinValue = Convert.ToInt16(cmd.ExecuteScalar());                       //获取最小值
        con.Close();                                                           //关闭数据库连接
    }
    Graphics g = this.groupBox1.CreateGraphics();                              //创建 Graphics 对象
    g.Clear(Color.SeaShell);                                                   //设置背景
    Brush b = new SolidBrush(Color.Blue);                                      //创建 Brush 对象
    Font f = new Font("Arial", 9, FontStyle.Regular);                          //创建 Font 对象
    Pen p=new Pen(b);                                                          //创建 Pen 对象
    using (sqlAda = new SqlDataAdapter("select * from tb_merchandise where t_name='" + str + "' order by t_date", con))
    {
        ds = new DataSet();                                                    //创建 DataSet 对象
        sqlAda.Fill(ds,"t_date");                                              //Fill 方法填充对象
        int M = MaxValue / 50+1;                                               //最大值
        int N = MinValue / 50;                                                 //最小值
        int T = N;
    for(int i=0;i<=M-N;i++)
    {
        g.DrawString(Convert.ToString(T * 50), f, b,0, 190 - 30 * i);
        g.DrawLine(p, 30, 200 - 30 * i, 260, 200 -30 * i);
        T++;
    }
    int Num = ds.Tables[0].Rows.Count;
    int[] Values = new int[Num];
    for (int C = 0; C <Num; C++)
    {
        Values[C] =Convert.ToInt32(ds.Tables[0].Rows[C][3].ToString());
        g.DrawString(Convert.ToDateTime(ds.Tables[0].Rows[C][2].ToString()).Month + "月", f, b, 30 * (C + 1)-10, 15);
        g.DrawLine(p, 30*(C+1), 200, 30*(C+1), 30);
    }
    Point[] P = new Point[Num];
    for (int i = 0; i < Num; i++)
    {
        P[i].X=30*(i+1);
        P[i].Y = 290 - Convert.ToInt32(Values[i] / 50f* 30);
```

```
        }
        g.DrawLines(p, P);
    }
}
```

自定义 DrowInfo 方法，用来绘制提示信息并显示各种商品的出售情况。代码如下：

```
private void DrowInfo(string str)
{
    Graphics g = this.groupBox2.CreateGraphics();                                          //创建 Graphics 对象
    g.Clear(Color.SeaShell);                                                               //设置背景颜色
    Brush b=new SolidBrush(Color.Blue);                                                    //创建 Brush 对象
    Font f=new Font("Arial", 9, FontStyle.Regular);                                        //设置 Font 对象
    using (sqlAda = new SqlDataAdapter("select * from tb_merchandise where t_name='" + str + "' order by t_date", con))
    {
        DataSet ds = new DataSet();                                                        //创建 DataSet 对象
        sqlAda.Fill(ds, "tb_09");                                                          //Fill 方法填充对象
        g.DrawString( "月份：          " + "价格", f, b, 10.0f,   25.0f);                  //绘制标题
        for (int i = 0; i < ds.Tables[0].Rows.Count; i++)                                  //绘制月份及相应的产品价格
        {
            int month=Convert.ToDateTime(ds.Tables[0].Rows[i][2].ToString()).Month;        //获取月份
            if (month >= 10)
            {
                //绘制月份及价格
                g.DrawString(month + "月：  " + "『" + ds.Tables[0].Rows[i][3].ToString() + "』", f, b, 10.0f, (i + 2) * 25.0f);
            }
            else
            {
                g.DrawString("0"+month + "月：  " + "『" + ds.Tables[0].Rows[i][3].ToString() + "』", f, b, 10.0f, (i + 2) * 25.0f);
            }
        }
    }
}
```

秘笈心法

心法领悟 559：如何清空控件的背景色？

清空控件背景色时，可以使用 Graphics 类的 Clear 方法来实现。例如，本实例中清空 GroupBox 控件的背景色的实现代码如下：

```
Graphics g = this.groupBox2.CreateGraphics();                                          //创建 Graphics 对象
g.Clear(Color.SeaShell);                                                               //设置背景颜色
```

实例 560　利用折线图分析彩票中奖情况

光盘位置：光盘\MR\23\560　　中级　　趣味指数：★★★★☆

实例说明

彩票销售点经常会挂着折线图，将近期本销售点中奖金额信息清晰地表示出来。本实例将实现利用折线图分析彩票中奖情况的功能，运行本程序，选择要分析的起始日期和终止日期，单击“分析”按钮，即可将本时间段的彩票中奖情况使用折线图描绘出来。实例运行效果如图 23.10 所示。

图 23.10　利用折线图分析彩票中奖情况

关键技术

本实例首先通过 SQL 语句从数据库中检索出相关数据，在绘制折线图时，将对应的数据进行计算，并以适当的比例绘制计算结果，实现过程中主要使用 Graphics 类的 DrawLines 方法来绘制折线。

说明：关于 DrawLines 方法的详细讲解请参见实例 559 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AnalyseLottery。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 DateTimePicker 控件，分别用来选择起始日期和终止日期；添加一个 Button 控件，用来查询指定时间段的彩票中奖情况，并绘制折线图；添加两个 Panel 控件，分别用来显示绘制的折线图和彩票中奖详细列表。

（3）程序的主要代码如下：

```
private void DrowInfo(string SQL)
{
    try
    {
        System.Drawing.Bitmap bmp = new Bitmap(this.panel1.Width, this.panel1.Height);    //定义画布
        Graphics g = Graphics.FromImage(bmp);                                              //创建 Graphics 对象
        g.Clear(Color.White);                                                              //设置画布背景颜色
        Brush bru = new SolidBrush(Color.Blue);                                            //创建 Brush 对象
        Pen p = new Pen(bru);                                                              //定义画笔
        Font font = new Font("Arial", 9, FontStyle.Bold);                                  //定义字体
        Conn();                                                                            //连接数据库
        cmd = new SqlCommand(SQL, con);                                                    //创建 SqlCommand 对象
        SqlDataReader dr = cmd.ExecuteReader();                                            //创建 SqlDataReader 对象
        int i = 0;
        Pen pLine = new Pen(Color.Orange, 4.0f);                                           //定义画笔
        string str = null;
        float f = 0.0f;
        while (dr.Read())                                                                  //开始读取数据库中的数据
        {
            i++;
            g.DrawString(dr[0].ToString().Substring(0, 7) + "月---", font, bru, 10, 15.0f * i);  //绘制月份
            g.DrawString(dr[1].ToString(), font, bru, this.panel1.Width - 50, 15.0f * i);        //绘制每个月份的中奖情况
            str += dr[1].ToString() + "#";
            f += Convert.ToSingle(dr[1].ToString());
        }
        dr.Close();                                                                        //关闭 SqlDataReader 对象
        this.panel1.BackgroundImage = bmp;                                                 //显示绘制的图像
        Bitmap bmpP = new Bitmap(this.panel3.Width, this.panel3.Height);                   //定义画布
        Graphics gP = Graphics.FromImage(bmpP);                                            //创建 Graphics 对象
        gP.Clear(Color.White);                                                             //设置背景颜色
        Brush bruImg = new SolidBrush(Color.Orange);                                       //定义笔刷
        Pen Pg = new Pen(bruImg, 1.0f);                                                    //定义画笔
        string[] strCount = str.Split('#');
        int[] ICount = new int[strCount.Length];
        for (int l = 0; l < strCount.Length - 1; l++)
        {
            ICount[l] = Convert.ToInt32(strCount[l]);
        }
        Point[] P = new Point[ICount.Length - 1];                                          //用于存储直线的坐标
        for (int j = 0; j < ICount.Length - 1; j++)
        {
            P[j].X = 35 + 28 * j;                                                          //设置 X 坐标
            P[j].Y = this.panel3.Height - 20 - Convert.ToInt32(ICount[j] / f * (this.panel3.Height + 20));    //设置 Y 坐标
        }
        f = 0.0f;
        str = null;
        gP.DrawLines(new Pen(new SolidBrush(Color.Red)), P);                               //绘制走势图
        gP.DrawString("分析结果走势图", new Font("宋体", 16), bru, this.panel3.Width / 2 - 80, 10);    //绘制标题
        this.panel3.BackgroundImage = bmpP;                                                //显示绘制的图像
```

```
    }
    catch
    {
        MessageBox.Show("此范围内没有任何信息！！！");
        return;
    }
}
```

秘笈心法

心法领悟 560：return 语句的使用。

return 语句用于退出类的方法，并控制返回方法的调用者，如果方法有返回类型，return 语句必须返回该类型的值；如果方法没有返回类型，应使用没有表达式的 return 语句。

实例 561　多曲线数据分析

光盘位置：光盘\MR\23\561　　高级　　趣味指数：★★★★★

实例说明

在人事管理系统中经常会用到统计功能，如统计员工、工资、性别比例等。本实例使用多曲线对员工进行统计，分析公司内正式员工和试用员工的流动情况。实例运行效果如图 23.11 所示。

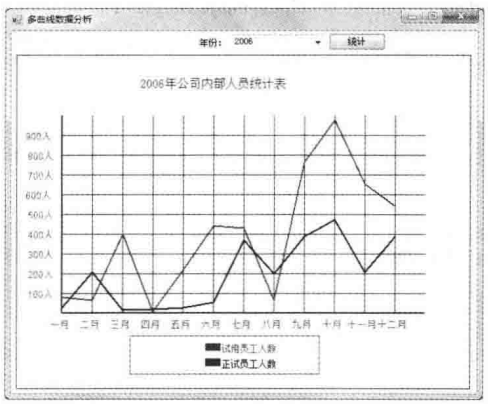


图 23.11　多曲线数据分析

关键技术

本实例实现时，首先需要通过 SQL 语句从数据库中检索出相关数据，然后将这些数据信息以适当的比例绘制成多曲线图，实现过程中主要使用 Graphics 类的 DrawLines 方法来绘制多曲线。

说明：关于 DrawLines 方法的详细讲解请参见实例 559 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiCurve。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，用来显示年度列表；添加一个 Panel 控件，用来显示绘制的多曲线图；添加一个 Button 控件，用来从数据库中查询信息，并根据查询的信息绘制多曲线图。

（3）程序的主要代码如下：

```
private void CreateImage(int ID)
{
    int height = 440, width = 600;                                    //设置画布宽和高
    System.Drawing.Bitmap image = new System.Drawing.Bitmap(width, height);    //创建画布
```

```
Graphics g = Graphics.FromImage(image);                                    //创建 Graphics 对象
try
{
    g.Clear(Color.White);                                                  //清空图片背景色
    Font font = new System.Drawing.Font("Arial", 9, FontStyle.Regular);    //设置字体
    Font font1 = new System.Drawing.Font("宋体", 12, FontStyle.Regular);   //设置字体
    Font font2 = new System.Drawing.Font("Arial", 8, FontStyle.Regular);   //设置字体
    System.Drawing.Drawing2D.LinearGradientBrush brush = new System.Drawing.Drawing2D.LinearGradientBrush(new Rectangle(0, 0,
image.Width, image.Height), Color.Blue, Color.Blue, 1.2f, true);           //创建 LinearGradientBrush 对象
    g.FillRectangle(Brushes.AliceBlue, 0, 0, width, height);               //绘制矩形框
    Brush brush1 = new SolidBrush(Color.Blue);                             //创建笔刷
    Brush brush2 = new SolidBrush(Color.SaddleBrown);                      //创建笔刷
    string str = "SELECT * FROM tb_curve WHERE Years=" + ID + "";          //声明 SQL 语句
    //建立数据库连接
    SqlConnection Con = new SqlConnection("server=mrwxk\\wangxiaoke;pwd=;uid=sa;database=db_TomeOne");
    Con.Open();                                                            //打开数据库连接
    SqlCommand Com = new SqlCommand(str, Con);                             //创建 SqlCommand 对象
    SqlDataReader dr = Com.ExecuteReader();                                //创建 SqlDataReader 对象
    dr.Read();                                                             //开始读取记录
    if (dr.HasRows)                                                        //如果有记录
    {
        //绘制标题
        g.DrawString("" + ID + "年公司内部人员统计表", font1, brush1, new PointF(160, 30));
    }
    dr.Close();                                                            //关闭 SqlDataReader 对象
    //画图片的边框线
    g.DrawRectangle(new Pen(Color.Blue), 0, 0, image.Width - 1, image.Height - 1);
    Pen mypen = new Pen(brush, 1);                                         //创建画笔
    Pen mypen2 = new Pen(Color.Red, 2);                                    //创建画笔
    int x = 60;
    for (int i = 0; i < 12; i++)
    {
        g.DrawLine(mypen, x, 80, x, 340);                                  //绘制纵向线条
        x = x + 40;
    }
    Pen mypen1 = new Pen(Color.Blue, 2);                                   //创建画笔
    g.DrawLine(mypen1, x - 480, 80, x - 480, 340);                         //绘制线条
    int y = 106;
    for (int i = 0; i < 9; i++)
    {
        g.DrawLine(mypen, 60, y, 540, y);                                  //绘制横向线条
        y = y + 26;
    }
    g.DrawLine(mypen1, 60, y, 540, y);
    //X 轴
    String[] n = {"一月", "二月", "三月", "四月", "五月", "六月", "七月",
        "八月", "九月", "十月", "十一月", "十二月"};                          //绘制月份
    x = 35;
    for (int i = 0; i < 12; i++)
    {
        g.DrawString(n[i].ToString(), font, Brushes.Red, x, 348);          //设置文字内容及输出位置
        x = x + 40;
    }
    //Y 轴
    String[] m = {"900 人", " 800 人", " 700 人", "600 人", " 565 人", " 400 人", " 300 人", " 200 人",
        " 100 人"};                                                        //绘制人数
    y = 100;
    for (int i = 0; i < 9; i++)
    {
        g.DrawString(m[i].ToString(), font, Brushes.Red, 10, y);           //设置文字内容及输出位置
        y = y + 26;
    }
    int[] Count1 = new int[12];
    int[] Count2 = new int[12];
    string[] NumChr = new string[12];
    string cmdtxt2 = "SELECT * FROM tb_curve WHERE Years=" + ID + "";      //声明 SQL 语句
    SqlCommand Com1 = new SqlCommand(cmdtxt2, Con);                        //创建 SqlCommand 对象
```

```
        SqlDataAdapter da = new SqlDataAdapter();                          //创建 SqlDataAdapter 对象
        da.SelectCommand = Com1;
        DataSet ds = new DataSet();                                        //创建 DataSet 对象
        da.Fill(ds);                                                       //Fill 方法填充 DataSet
        int j = 0;
        for (int i = 0; i < 12; i++)
        {
            NumChr[i] = ds.Tables[0].Rows[0][i + 1].ToString();
        }
        for (j = 0; j < 12; j++)
        {
            Count1[j] = Convert.ToInt32(NumChr[j].Split('|')[0].ToString()) * 26 / 100;
        }
        for (int k = 0; k < 12; k++)
        {
            Count2[k] = Convert.ToInt32(NumChr[k].Split('|')[1].ToString()) * 26 / 100;
        }
        //显示折线效果
        SolidBrush mybrush = new SolidBrush(Color.Red);                    //创建 SolidBrush 对象
        Point[] points1 = new Point[12];
        points1[0].X = 60; points1[0].Y = 340 - Count1[0];
        points1[1].X = 100; points1[1].Y = 340 - Count1[1];
        points1[2].X = 140; points1[2].Y = 340 - Count1[2];
        points1[3].X = 180; points1[3].Y = 340 - Count1[3];
        points1[4].X = 220; points1[4].Y = 340 - Count1[4];
        points1[5].X = 260; points1[5].Y = 340 - Count1[5];
        points1[6].X = 300; points1[6].Y = 340 - Count1[6];
        points1[7].X = 340; points1[7].Y = 340 - Count1[7];
        points1[8].X = 380; points1[8].Y = 340 - Count1[8];
        points1[9].X = 420; points1[9].Y = 340 - Count1[9];
        points1[10].X = 460; points1[10].Y = 340 - Count1[10];
        points1[11].X = 565; points1[11].Y = 340 - Count1[11];
        g.DrawLines(mypen2, points1);                                      //绘制折线
        Pen mypen3 = new Pen(Color.Black, 2);                              //创建画笔
        Point[] points2 = new Point[12];
        points2[0].X = 60; points2[0].Y = 340 - Count2[0];
        points2[1].X = 100; points2[1].Y = 340 - Count2[1];
        points2[2].X = 140; points2[2].Y = 340 - Count2[2];
        points2[3].X = 180; points2[3].Y = 340 - Count2[3];
        points2[4].X = 220; points2[4].Y = 340 - Count2[4];
        points2[5].X = 260; points2[5].Y = 340 - Count2[5];
        points2[6].X = 300; points2[6].Y = 340 - Count2[6];
        points2[7].X = 340; points2[7].Y = 340 - Count2[7];
        points2[8].X = 380; points2[8].Y = 340 - Count2[8];
        points2[9].X = 420; points2[9].Y = 340 - Count2[9];
        points2[10].X = 460; points2[10].Y = 340 - Count2[10];
        points2[11].X = 565; points2[11].Y = 340 - Count2[11];
        g.DrawLines(mypen3, points2);                                      //绘制折线
        //绘制标识
        g.DrawRectangle(new Pen(Brushes.Red), 150, 370, 250, 50);          //绘制范围框
        g.FillRectangle(Brushes.Red, 250, 380, 20, 10);                    //绘制小矩形
        g.DrawString("试用员工人数", font2, Brushes.Red, 270, 380);        //绘制试用员工人数
        g.FillRectangle(Brushes.Black, 250, 400, 20, 10);                  //绘制小矩形
        g.DrawString("正式员工人数", font2, Brushes.Black, 270, 400);      //绘制正式员工人数
        this.panel1.BackgroundImage = image;                               //显示绘制的图像
    }
    catch
    { }
}
```

秘笈心法

心法领悟 561：折线图的使用场合。

一般在对一些销售数据、趋势走势对比等数据分析时，使用折线图显示，可以更加直观，例如，开发人员熟悉的 TIBOE 编程语言排行榜就使用折线图显示。

实例 562 网站人气指数曲线分析

光盘位置：光盘\MR\23\562　　中级　　趣味指数：★★★★☆

实例说明

网站访问量可以决定网站的人气，那么对网站月访问量进行统计分析，可以让网站管理员准确地判断网民上网的需求周期，从而调整网站的内容，更好地为广大网民服务。本实例通过对数据库中的数据进行分析，将每月份网站访问量以折线形式反映给用户。实例运行效果如图 23.12 所示。

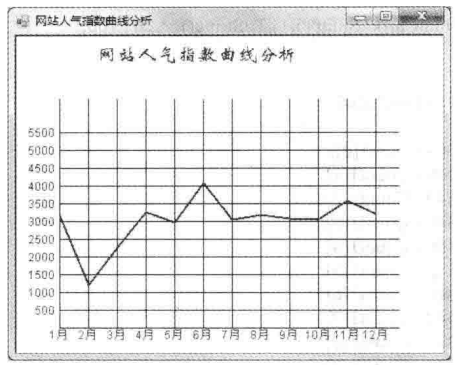


图 23.12　网站人气指数曲线分析

关键技术

本实例主要使用 Graphics 类的 DrawLines 方法来绘制网站人气指数曲线图。

说明：关于 DrawLines 方法的详细讲解请参见实例 559 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SiteVisterAnalyse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用于显示绘制的网站人气指数折线图。

（3）程序的主要代码如下：

```
private void drowPic()
{
    Graphics g = this.CreateGraphics();                                    //创建 Graphics 对象
    g.Clear(Color.WhiteSmoke);                                             //设置背景色
    Pen p = new Pen(Color.Blue);                                           //绘制画笔
    //设置用到的字体
    Font fontO = new System.Drawing.Font("Arial", 9, FontStyle.Regular);
    Font fontT = new System.Drawing.Font("华文新魏", 16, FontStyle.Regular);
    Point pointStart = new Point(0, 0);                                    //绘制边框与显示字体
    Size sizeWindows = new Size(this.Width - 8, this.Height - 34);         //创建 Size 对象
    Rectangle rect = new Rectangle(pointStart, sizeWindows);               //创建 Rectangle 对象
    g.DrawRectangle(p, rect);                                              //绘制矩形
    Brush brus = new SolidBrush(Color.Red);                                //创建笔刷
    g.DrawString("网站人气指数曲线分析", fontT, brus, this.Width / 2.00f - 150, 10.00f);
    //绘制网格线
    int x = this.Width / 10;
    int y = this.Height / 14;
    int z = this.Width / 10;
    int k = y * 12;
    //X
    for (int i = 0; i < 12; i++)
```

```
{
    g.DrawLine(p, x, y * 3 - 10, x, y * 12);                    //绘制水平线条
    x = x + (this.Width - 34) / 14;
}
//X 轴
String[] n = {" 1 月", " 2 月", " 3 月", " 4 月", " 5 月", " 6 月", " 7 月",
 " 8 月", " 9 月", "10 月", "11 月", "12 月"};                   //绘制月份
x = this.Width / 10 - 16;
for (int i = 0; i < 12; i++)
{
    g.DrawString(n[i].ToString(), fontO, Brushes.Red, x, y * 12);   //设置文字内容及输出位置
    x = x + (this.Width - 34) / 14;
}
//Y
for (int i = 0; i < 12; i++)
{
    g.DrawLine(p, z, k, x + 10, k);                             //绘制垂直线条
    k = k - (y * 12) / 16;
}
//Y 轴
int h = k;
String[] m = {"5565","5650","4565", "4000", "3565", "3000", "2565", "2000", "1565", "1000",
 "   565"};                                                     //绘制 Y 轴显示的文字
k = y * 12;
for (int i = 0; i < 11; i++)
{
    g.DrawString(m[10 - i].ToString(), fontO, Brushes.Red, z - 35, k - y);   //开始绘制文字
    k = k - (y * 12) / 16;
}
int[] Count = new int[12];
Pen mypen = new Pen(Color.Red, 2);                              //创建画笔
Point[] points = new Point[12];
x = this.Width / 10;
k = y * 12;
SqlConnection Con = new SqlConnection("Server=mrwxk\\wangxiaoke;DataBase=db_TomeOne;Uid=sa;Pwd=");
string cmdtxt2 = "SELECT * FROM tb_reticulation";               //声明 SQL 语句
SqlCommand Com1 = new SqlCommand(cmdtxt2, Con);                 //创建 SqlCommand 对象
SqlDataAdapter da = new SqlDataAdapter();                       //创建 SqlDataAdapter 对象
da.SelectCommand = Com1;
DataSet ds = new DataSet();                                     //创建 DataSet 对象
da.Fill(ds);                                                    //Fill 方法填充 DataSet 对象
int j = 0;
for (j = 0; j < 12; j++)
{
    //生成 Y 轴
    Count[j] = Convert.ToInt32(ds.Tables[0].Rows[0][j + 2].ToString()) * (y * 12) / 16 / 565;
}
//设置绘制曲线的坐标数组
points[0].X = x; points[0].Y = k - Count[0];
x = x + (this.Width - 34) / 14;
points[1].X = x; points[1].Y = k - Count[1];
x = x + (this.Width - 34) / 14;
points[2].X = x; points[2].Y = k - Count[2];
x = x + (this.Width - 34) / 14;
points[3].X = x; points[3].Y = k - Count[3];
x = x + (this.Width - 34) / 14;
points[4].X = x; points[4].Y = k - Count[4];
x = x + (this.Width - 34) / 14;
points[5].X = x; points[5].Y = k - Count[5];
x = x + (this.Width - 34) / 14;
points[6].X = x; points[6].Y = k - Count[6];
x = x + (this.Width - 34) / 14;
points[7].X = x; points[7].Y = k - Count[7];
x = x + (this.Width - 34) / 14;
points[8].X = x; points[8].Y = k - Count[8];
x = x + (this.Width - 34) / 14;
points[9].X = x; points[9].Y = k - Count[9];
x = x + (this.Width - 34) / 14;
```

```
    points[10].X = x; points[10].Y = k - Count[10];
    x = x + (this.Width - 34) / 14;
    points[11].X = x; points[11].Y = k - Count[11];
    g.DrawLines(mypen, points);                                    //绘制折线
}
```

秘笈心法

心法领悟 562：使用 DataSet 对象作为数据集。

DataSet 对象是 ADO.NET 的核心成员，它是支持 ADO.NET 断开式、分布式数据方案的核心对象，也是实现基于非连接的数据查询的核心组件。DataSet 对象是创建在内存中的集合对象，它可以包含任意数量的数据表以及所有表的约束、索引和关系等，它实质上相当于在内存中的一个小型关系数据库。

23.4 饼 型 图 表

实例 563	利用饼型图分析公司男女比率	中级
	光盘位置：光盘\MR\23\563	趣味指数：★★★★☆

实例说明

开发人力资源管理系统时，利用饼型图分析公司男女比例是最佳的选择。本实例对某公司的男女比例使用饼型图进行分析，实例运行效果如图 23.13 所示。

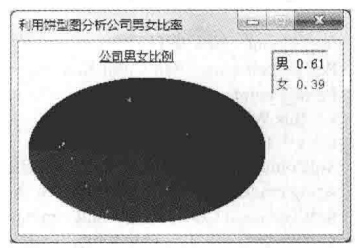


图 23.13 利用饼型图分析公司男女比率

关键技术

本实例实现时主要用到了 Graphics 类的 FillEllipse 和 FillPie 方法，下面分别对它们进行详细讲解。

（1）FillEllipse 方法

该方法用于填充 Rectangle 结构指定的边框所定义的椭圆的内部。语法格式如下：

```
public void FillEllipse(Brush brush,Rectangle rect)
```

参数说明

❶ brush：确定填充特性的 Brush。

❷ rect：Rectangle 结构，它表示定义椭圆的边框。

（2）FillPie 方法

该方法用于填充椭圆所定义的扇形区的内部，该椭圆由 Rectangle 结构和两条射线指定。语法格式如下：

```
public void FillPie(Brush brush,Rectangle rect,float startAngle,float sweepAngle)
```

FillPie 方法中的参数说明如表 23.3 所示。

表 23.3 FillPie 方法中的参数说明

参 数	说 明
brush	确定填充特性的 Brush
rect	Rectangle 结构，它表示定义该扇形区所属的椭圆的边框
startAngle	从 X 轴沿顺时针方向旋转到扇形区第一个边所测得的角度（以度为单位）
sweepAngle	从 startAngle 参数沿顺时针方向旋转到扇形区第二个边所测得的角度（以度为单位）

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CAnalyseSex。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Panel 控件，分别用来显示绘制的饼型图和说明信息。

（3）程序的主要代码如下：

```
private void ShowPic(string SexCode,float f)
{
    Graphics g = this.panel1.CreateGraphics();                    //通过 panel1 控件创建一个 Graphics 对象
    Pen p=new Pen(new SolidBrush(Color.Blue));                    //创建画笔
    Point p1=new Point(0,0);                                      //创建 Point 对象
    Size s=new Size(this.panel1.Width,this.panel1.Height);        //创建 Size 对象
    Rectangle trct = new Rectangle(p1, s);                        //创建 Rectangle 对象
    g.FillEllipse(new SolidBrush(Color.Red), trct);               //绘制椭圆
    g.FillPie(new SolidBrush(Color.Blue), trct, 180, f*360);      //绘制扇形
    Graphics ginfo = this.panel2.CreateGraphics();                //通过 panel2 控件创建一个 Graphics 对象
    Font font=new Font("宋体",10,FontStyle.Regular);              //设置字体
    //绘制性别
    ginfo.DrawString(SexCode +" "+f.ToString().Substring(0,4), font, new SolidBrush(Color.Blue), 0, 5);
    ginfo.DrawString("女" + " " + (1.0 - Convert.ToDouble(f.ToString().Substring(0, 4))).ToString().Substring(0, 4), font, new SolidBrush(Color.Red),
0, 25);                                                           //绘制比例
}
```

秘笈心法

心法领悟 563：饼形图的使用场合。

饼形图在一些人力资源管理系统或者行业数据分析系统中经常用到，例如，现在大家所熟悉的各个品牌手机的市场份额占比情况，就是使用饼形图显示的。

实例 564　利用饼型图分析产品市场占有率

光盘位置：光盘\MR\23\564　　中级　　趣味指数：★★★★☆

实例说明

开发商品销售管理系统的过程中，为了让用户更清晰地了解产品在市场上的占有率，使用饼图分析产品市场占有率是最佳的选择。本实例利用饼型图来分析某电子产品的市场占有率，实例运行效果如图 23.14 所示。

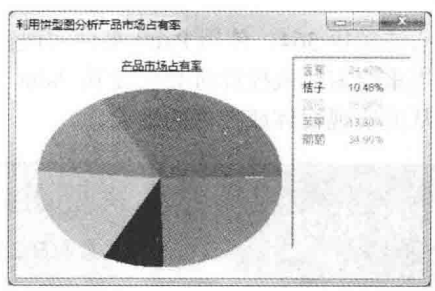


图 23.14　利用饼型图分析产品市场占有率

关键技术

本实例在实现时，首先通过 SQL 语句统计商品在市场的占有率，并将其字段名与数量存放于 Hashtable（哈希表）中，然后遍历“哈希表”计算出每种商品所占的比例，最后通过 Graphics 类的 FillPie 方法绘制饼型图。

说明： 关于 FillPie 方法的详细讲解请参见实例 563 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CAnalyseGoodsScale。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Panel 控件，分别用来显示绘制的饼型图和说明信息。

（3）程序的主要代码如下：

```
private void showPic(float f,Brush B)
{
    Graphics g = this.panel1.CreateGraphics();                    //通过 panel1 控件创建一个 Graphics 对象
```

```
    if (TimeNum == 0.0f)
    {
        g.FillPie(B, 0, 0, this.panel1.Width, this.panel1.Height, 0, f * 360);          //绘制扇形
    }
    else
    {
        g.FillPie(B, 0, 0, this.panel1.Width, this.panel1.Height, TimeNum, f * 360);
    }
    TimeNum += f * 360;
}
private void Form1_Paint(object sender, PaintEventArgs e)                               //在 Paint 事件中绘制窗体
{
    ht.Clear();
    Conn();                                                                             //连接数据库
    Random rnd = new Random();                                                          //生成随机数
    using (cmd = new SqlCommand("select t_Name,sum(t_Num) as Num   from tb_14 group by t_Name", con))
    {
        Graphics g2 = this.panel2.CreateGraphics();                                     //通过 panel2 控件创建一个 Graphics 对象
        SqlDataReader dr = cmd.ExecuteReader();                                         //创建 SqlDataReader 对象
        while (dr.Read())                                                               //读取数据
        {
            ht.Add(dr[0],Convert.ToInt32(dr[1]));                                       //将数据添加到 Hashtable 中
        }
        float[] flo = new float[ht.Count];
        int T = 0;
        foreach (DictionaryEntry de in ht)                                              //遍历 Hashtable
        {
            flo[T] = Convert.ToSingle((Convert.ToDouble(de.Value) / SumNum).ToString().Substring(0, 6));
            Brush Bru = new SolidBrush(Color.FromArgb(rnd.Next(255), rnd.Next(255), rnd.Next(255)));
            //绘制商品及百分比
            g2.DrawString(de.Key + "   " + flo[T] * 100 + "%", new Font("Arial", 8, FontStyle.Regular), Bru, 7, 5 + T * 18);
            showPic(flo[T], Bru);                                                       //调用 showPic 方法绘制饼型图
            T++;
        }
    }
}
```

秘笈心法

心法领悟 564：使用 Paint 事件实现窗体或控件的重绘。

当重绘窗体或控件时会引发其 Paint 事件，该事件将 PaintEventArgs 的实例传递给用来处理 Paint 事件的方法，从而实现窗体或控件的重绘。

实例 565　利用多饼型图分析企业人力资源情况

光盘位置：光盘\MR\23\565　　高级　　趣味指数：★★★★★

实例说明

开发企业人力资源管理系统时，经常使用饼型图分析企业中各个阶层的人员情况，为了让用户有更清晰的认识，本实例采用了多饼图表示法实现分析功能。实例运行效果如图 23.15 所示。

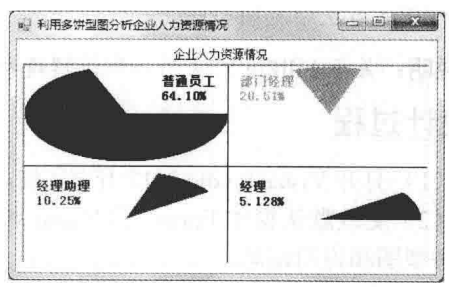


图 23.15　利用多饼型图分析企业人力资源情况

关键技术

本实例在实现时，首先通过 SQL 语句分别统计企业总人数和各个阶层的人数，然后通过统计的比例绘制饼型图，实现过程中主要用到了 Graphics 类的 DrawLine、DrawString 和 FillPie 方法。

说明：（1）关于 DrawLine 方法的详细讲解请参见实例 553 中的关键技术。

（2）关于 DrawString 方法的详细讲解请参见实例 558 中的关键技术。

（3）关于 FillPie 方法的详细讲解请参见实例 563 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MultiCAnalyseHR。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用来显示绘制的多个饼型图。

（3）程序的主要代码如下：

```
private void ShowPic(int Sum)
{
    using (cmd = new SqlCommand("select t_Point,sum(t_Num) from tb_manpower group by t_Point order by sum(t_Num) desc", con))
    {
        Bitmap bmp = new Bitmap(this.panel1.Width, this.panel1.Height);                    //创建画布
        Graphics g = Graphics.FromImage(bmp);                                              //创建 Graphics 对象
        cmd.Connection.Open();
        SqlDataReader dr = cmd.ExecuteReader();                                            //创建 SqlDataReader 对象
        while (dr.Read())                                                                  //开始读取记录
        {
            float f = Convert.ToSingle(dr[1]) / Sum;
            string str = dr[0].ToString();
            drowPic(g, f, str);                                                            //调用 drowPic 方法绘制饼图
        }
        //绘制线条
        g.DrawLine(new Pen(Color.Black), 0, this.panel1.Height / 2, this.panel1.Width, this.panel1.Height / 2);
        g.DrawLine(new Pen(Color.Black), this.panel1.Width / 2, 0, this.panel1.Width / 2, this.panel1.Height);
        this.panel1.BackgroundImage = bmp;                                                 //显示绘制的图形
        dr.Close();                                                                        //关闭 SqlDataReader 对象
        con.Close();                                                                       //关闭数据库连接
    }
}
private void drowPic(Graphics g, float f, string str)                                      //根据要求绘制饼图
{
    if (ConutNum == 0)                                                                     //ConutNum 为 0 时执行
    {
        //绘制扇形
        g.FillPie(new SolidBrush(Color.Black), 0, 0, (this.panel1.Width) / 2, (this.panel1.Height - 10) / 2, 0, 360 * f);
        //绘制文本
        g.DrawString(str, new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Black), (this.panel1.Width) / 2 - 70, 10);
        g.DrawString(Convert.ToString(f * 100).Substring(0, 5) + "%", new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Black),
(this.panel1.Width) / 2 - 70, 25);                                                         //绘制文本
        floatNum = 360 * f;                                                                //计算角度
        ConutNum += 1;                                                                     //使 ConutNum 为 1
    }
    else if (Conu tNum == 1)                                                               //ConutNum 为 1 时执行
    {
        //绘制扇形
        g.FillPie(new SolidBrush(Color.DarkOrange), (this.panel1.Width) / 2, 0, (this.panel1.Width) / 2, (this.panel1.Height - 10) / 2, floatNum, 360 * f);
        //绘制文本
        g.DrawString(str, new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.DarkOrange), (this.panel1.Width) / 2 + 10, 10);
        g.DrawString(Convert.ToString(f * 100).Substring(0, 5) + "%", new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.DarkOrange),
(this.panel1.Width) / 2 + 10, 25);                                                         //绘制文本
        floatNum += 360 * f;                                                               //计算角度
        ConutNum += 1;                                                                     //ConutNum 为 2
    }
    else if (ConutNum == 2)                                                                //ConutNum 为 2 时执行
    {
        g.FillPie(new SolidBrush(Color.Red), 0, (this.panel1.Height - 10) / 2 + 10, (this.panel1.Width) / 2, (this.panel1.Height - 10) / 2, floatNum, 360 * f);
                                                                                           //绘制扇形
        //绘制文本
        g.DrawString(str, new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Red), 10, (this.panel1.Height - 10) / 2 + 20);
        g.DrawString(Convert.ToString(f * 100).Substring(0, 5) + "%", new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Red), 10,
```

```
(this.panel1.Height - 10) / 2 + 35);                                    //绘制文本
            floatNum += 360 * f;                                        //计算角度
            ConutNum += 1;                                              // ConutNum 为 3
        }
        else if (ConutNum == 3)                                         // ConutNum 为 3 时执行
        {
            g.FillPie(new SolidBrush(Color.Blue), (this.panel1.Width) / 2 - 10, (this.panel1.Height - 10) / 2 + 10, (this.panel1.Width) / 2, (this.panel1.Height -
10) / 2, floatNum, 360 * f);                                            //绘制扇形
            g.DrawString(str, new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Blue), (this.panel1.Width) / 2 + 10, (this.panel1.Height - 10) / 2 + 20);
                                                                        //绘制文本
            g.DrawString(Convert.ToString(f * 100).Substring(0, 5) + "%", new Font("宋体", 10, FontStyle.Bold), new SolidBrush(Color.Blue),
(this.panel1.Width) / 2 + 10, (this.panel1.Height - 10) / 2 + 35);      //绘制文本
        }
    }
```

秘笈心法

心法领悟 565：绘制多饼型图的实现原理。

绘制多饼型图时，首先需要计算各个部分所占的比例，然后依次绘制各个部分在饼型图中所占的比例，在绘制过程中，首先从第一部分开始绘制，绘制结束后，记录其绘制的扇形结束位置，并以该位置为起点，开始绘制第二部分，依此类推，直到绘制完最后一部分，这样就实现了绘制多饼型图的功能。

实例 566 在饼型图的外围显示说明文字

光盘位置：光盘\MR\23\566　　高级　　趣味指数：★★★★★

实例说明

饼型图可以很直观地显示一些信息的比例，如果信息过多，则需要在饼型图上绘制说明文字，这样才可以更好地了解饼型图所代表的含义。本实例使用 C#实现了在饼型图的外围绘制说明文字的功能，实例运行效果如图 23.16 所示。

图 23.16　在饼型图的外围显示说明文字

关键技术

本实例实现时主要使用 Graphics 类的 FillPie 方法绘制饼型图，使用 FillRectangle 和 DrawRectangle 方法绘制说明文字的矩形框及矩形框的边，使用 DrawString 方法绘制说明文字。下面对本实例中用到的关键技术进行详细讲解。

说明：（1）关于 FillPie 方法的详细讲解请参见实例 563 中的关键技术。

（2）关于 FillRectangle 方法的详细讲解请参见实例 552 中的关键技术。

（3）关于 DrawString 方法的详细讲解请参见实例 558 中的关键技术。

DrawRectangle 方法用于绘制由坐标对、宽度和高度指定的矩形。语法格式如下：

```
public void DrawRectangle(Pen pen,int x,int y,int width,int height)
```

DrawRectangle 方法中的参数说明如表 23.4 所示。

表 23.4　DrawRectangle 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定矩形的颜色、宽度和样式
x	要绘制的矩形的左上角的 x 坐标
y	要绘制的矩形的左上角的 y 坐标
width	要绘制的矩形的宽度
height	要绘制的矩形的高度

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TextInCaky。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件，用来显示带有说明文字的饼型图。

（3）程序的主要代码如下：

```
public void ProtractArea(Graphics g)                    //绘制饼型图表
{
    AreaValue();                                        //计算最长说明文字的大小
    mypen = new Pen(Color.Black, 1);                    //设置画笔的颜色及大小
    float f = 0;                                        //记录百分比
    float TimeNum = 0;                                  //扇形的绘制度数
    float AXLeft = 0;                                   //设置饼型图的 X 坐标
    float AYUp = 0;                                     //设置饼型图的 Y 坐标
    float AXSize = 0;                                   //设置饼型图的宽度
    float AYSize = 0;                                   //设置饼型图的高度
    float Atop = 0;                                     //设置饼型图的 Y 坐标
    float Aleft = 0;                                    //设置饼型图的 X 坐标
    TimeNum = AreaAngle;                                //设置饼型图的起始度数
    //计算饼型图的初始位置
    XLeft = panel1.Width - (panel1.Width - 5);          //去除边框后饼型图的 X 位置
    XSize = panel1.Width - 10;                          //设置饼型图的宽度
    temXLeft = AXLeft;                                  //记录饼型图的 X 坐标
    AXLeft = XLeft;                                     //记录饼型图的 X 坐标
    AXSize = XSize;                                     //记录饼型图的宽度
    //通过说明文字的大小计算饼型图的位置
    AXLeft = AXLeft + AreaXMaxWidth + Aline;            //设置去除说明文字后的饼型图 X 坐标
    AYUp = AYUp + AreaXMaxHeight;                       //设置去除说明文字后的饼型图 Y 坐标
    AXSize = XSize - AreaXMaxWidth * 2 - Aline * 2;     //设置去除说明文字后的饼型图宽度
    AYSize = YSize - AreaXMaxHeight * 2;                //设置去除说明文字后的饼型图高度
    if (AXSize >= AYSize)                               //如果饼型图的宽度大于等于高度
    {
        Aleft = AXSize - AYSize;                        //记录饼型图的 X 坐标
        AXSize = AYSize;                                //将高度设为宽度
    }
    else
    {
        Atop = AYSize - AXSize;                         //记录饼型图的 Y 坐标
        AYSize = AXSize;                                //将宽度设为高度
    }
    if (Aleft != 0)                                     //如果宽大于高
    {
        AXLeft = AXLeft + Aleft / 2;                    //设置饼型图横向居中
    }
    if (Atop != 0)                                      //如果高大于宽
    {
        AYUp = AYUp + Atop / 2;                         //设置饼型图纵向居中
    }
    temXLeft = XLeft;
    //绘制饼型图
```

```
    if (AXSize > 0 && AYSize > 0)                                          //如果饼型图的宽和高大于 0
    {
        for (int i = 0; i < SzData.Length; i++)                            //遍历数据
        {
            f = SzData[i] / ASum;                                          //获取当前数据的百分比
            //设置当前扇形图的填充颜色
            if (i >= WearColor.Length)                                     //如果没有超出颜色数组
                mybrush = new SolidBrush(WearColor[i - WearColor.Length]);
            else
                mybrush = new SolidBrush(WearColor[i]);
            g.FillPie(mybrush, AXLeft, AYUp, AXSize, AYSize, TimeNum, f * 360); //绘制扇形图
            TimeNum += f * 360;                                            //设置下一个扇形图的度数
        }
        ProAreaSign(g);                                                    //绘制饼型图的说明文字
    }
    else
        return;
}
#endregion
#region 绘制饼型图标识(Area)
public void ProAreaSign(Graphics g)
{
    AreaValue();                                                           //存储最长的名称
    mypen = new Pen(Color.Black, 1);                                       //设置画笔的颜色及大小
    Font LSfont = new System.Drawing.Font("宋体", 8);                      //设置说明文字的字体样式
    SolidBrush Zbrush = new SolidBrush(Color.Black);                       //设置存放说明文字边框的画刷
    SolidBrush ATbrush = new SolidBrush(Color.Khaki);                      //设置存放说明文字方块的背景画刷
    float f = 0;                                                           //记录百分比
    float TimeNum = 0;                                                     //扇形的绘制度数
    float AXLeft = 0;                                                      //设置饼型图的 X 坐标
    float AYUp = 0;                                                        //设置饼型图的 Y 坐标
    float AXSize = 0;                                                      //设置饼型图的宽度
    float AYSize = 0;                                                      //设置饼型图的高度
    float Atop = 0;                                                        //设置饼型图的 Y 坐标
    float Aleft = 0;                                                       //设置饼型图的 X 坐标
    Graphics TitG = panel1.CreateGraphics();                               //创建 Graphics 对象
    SizeF XMaxSize = TitG.MeasureString("", LSfont);                       //将绘制的字符串进行格式化
    float SWidth = 0;                                                      //获取字符串的宽度
    float SHeight = 0;                                                     //获取字符串的高度
    //计算饼型图的初始位置
    XLeft = panel1.Width - (panel1.Width - 5);                             //去除边框后饼型图的 X 位置
    XSize = panel1.Width - 10;                                             //设置饼型图的宽度
    temXLeft = AXLeft;                                                     //记录饼型图的 X 坐标
    AXLeft = XLeft;                                                        //记录饼型图的 X 坐标
    AXSize = XSize;                                                        //记录饼型图的宽度
    //通过说明文字的大小计算饼型图的位置
    AXLeft = AXLeft + AreaXMaxWidth + Aline;                               //设置去除说明文字后的饼型图 X 坐标
    AYUp = AYUp + AreaXMaxHeight;                                          //设置去除说明文字后的饼型图 Y 坐标
    AXSize = XSize - AreaXMaxWidth * 2 - Aline * 2;                        //设置去除说明文字后的饼型图宽度
    AYSize = YSize - AreaXMaxHeight * 2;                                   //设置去除说明文字后的饼型图高度
    if (AXSize >= AYSize)                                                  //如果饼型图的宽度大于等于高度
    {
        Aleft = AXSize - AYSize;                                           //记录饼型图的 X 坐标
        AXSize = AYSize;                                                   //将高度设为宽度
    }
    else
    {
        Atop = AYSize - AXSize;                                            //记录饼型图的 Y 坐标
        AYSize = AXSize;                                                   //将宽度设为高度
    }
    if (Aleft != 0)                                                        //如果宽大于高
    {
        AXLeft = AXLeft + Aleft / 2;                                       //设置饼型图横向居中
    }
    if (Atop != 0)                                                         //如果高大于宽
    {
```

```
            AYUp = AYUp + Atop / 2;                                    //设置饼型图纵向居中
        }
        temXLeft = XLeft;
        //初始化说明文字前横线的变量
        float X1 = 0;
        float Y1 = 0;
        float X2 = 0;
        float Y2 = 0;
        //初始化说明文字位置的变量
        float TX1 = 0;
        float TY1 = 0;
        float TX2 = 0;
        float TY2 = 0;
        float temf = 0;                                                //记录百分比
        double radians = 0;                                            //记录扇形的角度
        temf = (this.AreaAngle * (ASum / 360) / ASum);                 //记录起始位置的度数
        TimeNum = this.AreaAngle;                                      //记录扇形的起始角度
        //绘制说明文字
        if (AXSize > 0 && AYSize > 0)
        {
            for (int i = 0; i < SzData.Length; i++)                    //遍历所有说明文字
            {
                f = SzData[i] / ASum;                                  //获取当前记录的百分比
                if (f == 0)                                            //如果当前值为 0
                    continue;                                          //执行下一次循环
                radians = ((double)((temf + f / 2) * 360) * Math.PI) / (double)180;
                X1 = Convert.ToSingle(AXLeft + (AXSize / 2.0 + (int)((float)(AXSize / 2.0) * Math.Cos(radians))));
                Y1 = Convert.ToSingle(AYUp + (AYSize / 2.0 + (int)((float)(AYSize / 2.0) * Math.Sin(radians))));
                XMaxSize = TitG.MeasureString(AreaText[i].Trim(), LSfont);  //将绘制的字符串进行格式化
                SWidth = XMaxSize.Width;                               //获取字符串的宽度
                SHeight = XMaxSize.Height;                             //获取字符串的高度
                if ((temf + f / 2) * 360 > 90 && (temf + f / 2) * 360 <= 270)
                {
                    X2 = X1 - Aline;
                    TX1 = X2 - 1 - SWidth;
                    TY1 = Y1 - SHeight / 2 - Asash;
                    TX2 = SWidth;
                    TY2 = SHeight + Asash * 2;
                    g.FillRectangle(ATbrush, TX1, TY1, TX2, TY2);          //绘制内矩形
                    g.DrawRectangle(new Pen(Color.Black, 1), TX1, TY1, TX2, TY2);  //绘制矩形
                    g.DrawString(AreaText[i].Trim(), LSfont, Zbrush, new PointF(X2 - SWidth + Asash - 1, Y1 - SHeight / 2));
                }
                else
                {
                    X2 = X1 + Aline;
                    TX1 = X2 + 1;
                    TY1 = Y1 - SHeight / 2 - Asash;
                    TX2 = SWidth;
                    TY2 = SHeight + Asash * 2;
                    g.FillRectangle(ATbrush, TX1, TY1, TX2, TY2);          //绘制内矩形
                    g.DrawRectangle(new Pen(Color.Black, 1), TX1, TY1, TX2, TY2);  //绘制矩形
                    g.DrawString(AreaText[i].Trim(), LSfont, Zbrush, new PointF(X2 + Asash + 1, Y1 - SHeight / 2));
                }
                Y2 = Y1;
                g.DrawLine(new Pen(new SolidBrush(Color.Black), 1), X1, Y1, X2, Y2);
                TimeNum += f * 360;
                temf = temf + f;
            }
        }
        else
            return;
    }
    #endregion
```

秘笈心法

心法领悟 566：使用 DrawString 方法实现水印文字。

对于一些有可能侵权的图片，或者要作为宣传的图片，可以通过使用 DrawString 方法将用到的文字绘制到

图片上，从而得到所谓的“水印文字”。

实例 567　制作一个可以旋转的饼型图

光盘位置：光盘\MR\23\567　　高级　　趣味指数：★★★★★

实例说明

通过本实例绘制的饼型图可以直观地显示公司员工的性别比例，而且还可以通过单击“旋转”按钮使饼型图旋转起来。实例运行效果如图 23.17 所示。

图 23.17　制作一个可以旋转的饼型图

关键技术

本实例在实现时，首先通过 Graphics 类的 FillEllipse 和 FillPie 方法绘制饼型图，然后通过使用 Timer 组件控制饼型图的旋转。下面对本实例中用到的关键技术进行详细讲解。

说明：关于 Graphics 类的 FillEllipse 和 FillPie 方法的详细讲解请参见实例 563 中的关键技术。

Timer 组件表示一个计时器，它可以按用户定义的时间间隔来引发事件，引发的事件一般为周期性的，每隔若干秒或若干毫秒执行一次，其常用的属性、方法及事件如表 23.5 所示。

表 23.5　Timer 组件的常用属性、方法及事件

属性、方法及事件	说　明
Enabled 属性	获取或设置计时器是否正在运行
Interval 属性	获取或设置在相对于上一次发生的 Tick 事件引发 Tick 事件之前的时间（以毫秒为单位）
Start 方法	启动计时器
Stop 方法	停止计时器
Tick 事件	当指定的计时器间隔已过去而且计时器处于启用状态时发生

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 RoundCaky。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Panel 控件，分别用来显示表示男女比例的饼型图和说明信息；添加两个 Button 控件，分别用来执行绘制饼型图和旋转饼型图操作；添加一个 Timer 组件，用来控制饼型图的旋转与停止。

（3）程序的主要代码如下：

```
private void ShowPic(string SexCode, float f)                    //ShowPic 方法用来绘制饼图
{
    this.Controls.Add(panel1);                                   //将 Panel 控件添加到窗体中
    panel1.Width = 230;                                          //设置 Panel 控件的宽度
    panel1.Height = 230;                                         //设置 Panel 控件的高度
    bt = new Bitmap(panel1.Width,panel1.Height);                 //创建画布
    Graphics g = Graphics.FromImage(bt);                         //创建 Graphics 对象
    Pen p = new Pen(new SolidBrush(Color.Blue));                 //创建画笔
    Point p1 = new Point(0, 0);                                  //创建一个 Point 对象
    Size s = new Size(this.panel1.Width, this.panel1.Height);    //创建 Size 对象
    Rectangle trct = new Rectangle(p1, s);                       //创建 Rectangle 对象
    g.FillEllipse(new SolidBrush(Color.Red), trct);              //绘制椭圆
    g.FillPie(new SolidBrush(Color.Blue), trct, flag, f * 360);  //绘制扇形
    bt1 = new Bitmap(panel2.Width,panel2.Height);                //创建画布
    Graphics ginfo = Graphics.FromImage(bt1);                    //创建 Graphics 对象
```

```
    Font font = new Font("宋体", 10, FontStyle.Regular);                                    //设置字体
    ginfo.DrawString(SexCode + " " + f.ToString().Substring(0, 4), font, new SolidBrush(Color.Blue), 0, 5);
    ginfo.DrawString("女" + " " + (1.0 - Convert.ToDouble(f.ToString().Substring(0, 4))).ToString().Substring(0, 4), font, new SolidBrush(Color.Red), 0, 25);
                                                                                            //绘制性别及比率
    panel1.BackgroundImage = bt;                                                            //显示饼图
    panel2.BackgroundImage = bt1;                                                           //显示性别及比率
}
private void button1_Click(object sender, EventArgs e)                                     //单击“显示”按钮显示饼图
{
    Conn();                                                                                 //连接数据库
    using (cmd = new SqlCommand("SELECT sex,COUNT(sex) num FROM tb_sex group by sex", con))
    {
        SqlDataReader dr = cmd.ExecuteReader();                                             //创建 SqlDataReader 对象
        string[] str = new string[2];
        int i = 0;
        while (dr.Read())                                                                   //读取记录
        {
            str[i] = dr[0].ToString() + "," + dr[1].ToString();                             //获取数据库中存储的性别及比例
            i++;
        }
        float N = Convert.ToInt16(str[0].Substring(2)) + Convert.ToInt16(str[1].Substring(2));  //男女性别比例之和
        float f = Convert.ToInt16(str[0].Substring(2)) / N;                                 //男性的比例
        flag = 180;                                                                         //开始绘制的起始角度
        ShowPic(str[0].Substring(0, 1), f);                                                 //调用 ShowPic 方法绘制饼图
    }
    con.Close();
}
private void timer1_Tick(object sender, EventArgs e)                                       //当单击“旋转”按钮时开始旋转饼图
{
    flag += 1;                                                                              //每次转动一度
    Conn();                                                                                 //连接数据库
    using (cmd = new SqlCommand("SELECT sex,COUNT(sex) num FROM tb_sex group by sex", con))
    {
        SqlDataReader dr = cmd.ExecuteReader();                                             //创建 SqlDataReader 对象
        string[] str = new string[2];                                                       //声明数组存储性别及比例
        int i = 0;
        while (dr.Read())
        {
            str[i] = dr[0].ToString() + "," + dr[1].ToString();                             //获取数据库中存储的性别及比例
            i++;
        }
        float N = Convert.ToInt16(str[0].Substring(2)) + Convert.ToInt16(str[1].Substring(2));  //男女性别比例之和
        float f = Convert.ToInt16(str[0].Substring(2)) / N;                                 //男性的比例
        ShowPic(str[0].Substring(0, 1), f);                                                 //调用 ShowPic 方法绘制饼图
    }
    con.Close();                                                                            //关闭连接
}
private void button2_Click(object sender, EventArgs e)                                     //“旋转”按钮
{
    if (button2.Text =="旋转")                                                              //如果按钮显示“旋转”文本
    {
        timer1.Start();                                                                     //开始 Timer 控件
        button2.Text = "停止";                                                              //将按钮的文本设为“停止”
    }
    else                                                                                    //如果按钮的文本为“停止”
    {
        timer1.Stop();                                                                      //停止 Timer 控件
        button2.Text = "旋转";                                                              //将按钮的文本设为“旋转”
    }
}
```

秘笈心法

心法领悟 567：启动和停止计时器技巧。

程序中使用 Timer 计时器时，如果需要启动计时器，Enabled 属性为 true 和 Start 方法的效果是一样的；而停止计时器时，Enabled 属性为 false 和 Stop 方法的效果是一样的。

第24章

动画处理技术

- 图像动画
- 多媒体动画的制作
- 多媒体动画的播放

24.1　图 像 动 画

实例 568	随鼠标移动的图像 光盘位置：光盘\MR\24\568	初级 趣味指数：★★★★☆

实例说明

在浏览较大的图像时，如果窗体比图像小，就不能够完全地显示图像，而如果将较大的图像压缩显示到窗体上，则图像将会模糊不清。本实例设计了一个可以随鼠标移动的图像，首先将图像的一部分显示到窗体上，需要时用鼠标移动图像来浏览其他部分，这样既可以浏览全部的图像，又可以使图像不失真。实例运行效果如图 24.1 所示。

图 24.1　随鼠标移动的图像

关键技术

本实例在显示图像时使用 PictureBox 控件，将该控件放置在一个 Panel 容器控件中，同时设置 Panel 控件的 AutoScroll 属性为 true，以便能够滚动浏览大图片。实现图像随鼠标移动需要用到 MouseDown、MouseMove 和 MouseUp 3 个事件。MouseDown 事件在鼠标指针位于控件上并按下鼠标键时发生，MouseMove 事件在鼠标指针移到控件上时发生，而 MouseUp 事件在鼠标指针在控件上并释放鼠标键时发生。

本实例中，首先定义了一个 bool 型的全局变量 flag，当触发 MouseDown 事件时，将 flag 设置为 true，以标识鼠标已经按下，并且记录鼠标所在位置；当触发 MouseMove 事件时，判断 flag 是否为 true，如果是，则移动 PictureBox 控件的位置，移动距离为当前鼠标位置减去记录的鼠标位置；当触发 MouseUp 事件时，将 flag 设置为 False，标识鼠标已经抬起，这时再移动鼠标，图像将不再跟着移动。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MouseMove。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Panel 控件、一个 PictureBox 控件和一个 Button 控件，并将 Panel 控件的 AutoScroll 属性设置为 true，以便显示图片的全部。PictureBox 控件用来显示选择的图片，Button 控件用来执行打开图片的操作。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //设置文件的类型
    openFileDialog1.Filter = "*.jpg,*.jpeg,*.bmp,*.gif,*.ico,*.png,*.tif,*.wmf|*.jpg;*.jpeg;*.bmp;*.gif;*.ico;*.png;*.tif;*.wmf";
    openFileDialog1.ShowDialog();                                    //打开文件对话框
    //根据文件的路径创建 Image 对象
    Image myImage = System.Drawing.Image.FromFile(openFileDialog1.FileName);
    pictureBox1.Image = myImage;                                     //显示打开的图片
    pictureBox1.Height = myImage.Height;                             //根据图片大小设置 pictureBox1 控件的高度
    pictureBox1.Width = myImage.Width;                               //根据图片大小设置 pictureBox1 控件的宽度
}
```

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    flag = true;                                          //标识，鼠标按下
    x = e.X;                                              //记录鼠标的 X 坐标
    y = e.Y;                                              //记录鼠标的 Y 坐标
}
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    if (flag)
    {
        //设置 pictureBox1 控件的位置
        pictureBox1.Left = pictureBox1.Left + (e.X - x);
        pictureBox1.Top = pictureBox1.Top + (e.Y - y);
    }
}
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    flag = false;
}
```

秘笈心法

心法领悟 568：PictureBox 控件的使用。

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，它所显示的图片由 Image 属性确定，该属性可在运行时或设计时设置。另外，也可以通过设置 ImageLocation 属性，然后使用 Load 方法同步加载图像，或者使用 LoadAsync 方法异步加载图像。

实例 569　十字光标定位

光盘位置：光盘\MR\24\569　　初级　　趣味指数：★★★★☆

实例说明

在一些工程设计软件中，经常会看到一个用来精确定位的十字光标，这个光标在屏幕或地图上垂直相交形成一个十字形状，用此光标可以对一些物体在水平或垂直方向进行衡量，从而达到定位的目的。本实例将在指定图像上实现十字光标定位的功能。实例运行效果如图 24.2 所示。

图 24.2　十字光标定位

关键技术

十字光标其实是两条相交的直线，这两条直线的焦点就是鼠标的位置，画两条经过鼠标焦点的水平和垂直直线即可形成十字光标。

本实例实现十字光标定位功能时，使用了 Graphics 对象的 DrawLine 方法，下面对其进行详细讲解。

DrawLine 方法为重载方法，其常用的两种重载形式如下。

（1）第一种用于绘制一条连接两个 Point 结构的线。

语法如下：

```
public void DrawLine (Pen pen,Point pt1,Point pt2)
```

参数说明

❶ pen：Pen 对象，它确定线条的颜色、宽度和样式。

❷ pt1：Point 结构，它表示要连接的第一个点。

❸ pt2：Point 结构，它表示要连接的第二个点。

（2）第二种用于绘制一条连接由坐标指定的两个点的线条。

语法如下：

```
public void DrawLine (Pen pen,int x1,int y1,int x2,int y2)
```

DrawLine 方法中的参数说明如表 24.1 所示。

表 24.1　DrawLine 方法中的参数说明

参　数	说　明
pen	Pen 对象，它确定线条的颜色、宽度和样式
x1	第一个点的 x 坐标
y1	第一个点的 y 坐标
x2	第二个点的 x 坐标
y2	第二个点的 y 坐标

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 CrossCursor。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Button 控件和一个 PictureBox 控件，其中，Button 控件用来执行打开图片操作，PictureBox 控件用来显示选择的图片。

（3）程序的主要代码如下：

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
    Graphics myGraphics = pictureBox1.CreateGraphics();
    //通过调用 Graphics 对象的 DrawLine 方法实现鼠标十字定位功能
    myGraphics.DrawLine(new Pen(Color.Black, 1), new Point(e.X, 0), new Point(e.X, e.Y));
    myGraphics.DrawLine(new Pen(Color.Black, 1), new Point(e.X, e.Y), new Point(e.X, pictureBox1.Height - e.Y));
    myGraphics.DrawLine(new Pen(Color.Black, 1), new Point(0, e.Y), new Point(e.X, e.Y));
    myGraphics.DrawLine(new Pen(Color.Black, 1), new Point(e.X, e.Y), new Point(pictureBox1.Width - e.X, e.Y));
}
```

秘笈心法

心法领悟 569：使用 DrawLine 方法绘制坐标轴。

一般在绘制坐标轴时，都需要用到 DrawLine 方法。

实例 570　抓取鼠标形状

光盘位置：光盘\MR\24\570　　初级　　趣味指数：★★★★

实例说明

在 Windows 系统中，用户利用 Print Screen SysRq 键抓取窗口时，不能够抓取鼠标形状。但在一些抓图软件中，却能够抓取鼠标形状，该功能是如何实现的呢？本实例中实现了抓取鼠标形状的功能，实例运行效果如图 24.3 所示。

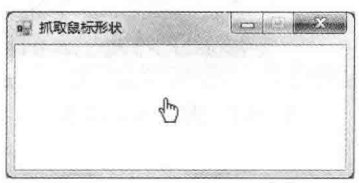


图 24.3　抓取鼠标形状

关键技术

本实例在实现抓取鼠标形状功能时，使用了 Cursor 类的 Draw 方法，该方法用来在指定边界内和指定的表面上绘制光标。语法格式如下：

```
public void Draw ( Graphics g, Rectangle targetRect)
```

参数说明

❶ g：要在其上绘制 Cursor 的 Graphics 表面。

❷ targetRect：代表 Cursor 限定范围的 Rectangle。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchAtMouse。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main。

（3）程序的主要代码如下：

```
private void Frm_Main_MouseUp(object sender, MouseEventArgs e)
{
    Graphics myGraphics = this.CreateGraphics();                    //创建窗体的 Graphics 对象
    Cursor.Draw(myGraphics, new Rectangle(e.X, e.Y, 10, 10));       //调用 Cursor 类的 Draw 方法抓取鼠标形状
}
```

秘笈心法

心法领悟 570：调用窗体的 CreateGraphics 方法创建 Graphics 对象。

在程序中绘图时，可以调用窗体的 CreateGraphics 方法来获取对 Graphics 对象的引用，该对象表示窗体的绘图画面。例如，本实例中创建 Graphics 对象的代码如下：

```
Graphics graphics = this.CreateGraphics( );                         //创建窗体的 Graphics 对象
```

实例 571　图片自动播放

光盘位置：光盘\MR\24\571　　中级　　趣味指数：★★★★★

实例说明

当图片过多时，浏览这些图片就变得很繁琐。为使浏览图片变得更加方便、快捷，开发出本实例用于自动播放图片。本实例提供选择图片目录、设置播放的时间间隔和选择播放模式等功能，实例运行效果如图 24.4 所示。

图 24.4　图片自动播放

关键技术

本实例实现图片自动播放功能时，主要是通过使用 Timer 组件来控制的，下面就对该组件进行详细讲解。

说明：关于 Timer 组件的详细讲解请参见实例 567 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PicPlay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListBox 控件，用来显示图片列表；添加一个 PictureBox 控件，用来显示图片；添加两个 TextBox 控件，分别用来显示图片目录和输入图片切换的时间间隔；添加一个 FolderBrowserDialog 组件，用来选择图片目录；添加一个 ComboBox 控件，用来选择图片播放方式；添加一个 Timer 组件，用来控制图片的切换。

（3）在项目中添加一个新的 Windows 窗体，并将其命名为 Frm_Browser.cs，用来以幻灯片形式浏览图片。

（4）程序的主要代码如下。

在 Frm_Main 窗体中单击“开始播放”按钮，开始自动播放图片，程序会判断是在当前窗口播放还是全屏播放。如果选择全屏播放图片，程序会将播放的图片路径传递给另一个窗体，然后打开这个窗体，使窗体全屏，从而实现全屏播放图片的功能。“开始播放”按钮的 Click 事件代码如下：

```
private void button5_Click(object sender, EventArgs e)
{
    if (Pflag)
    {
        if (txtTime.Text != "")
        {
            if (cbbShowType.SelectedIndex == 1)                    //如果选择“适应窗口大小播放”
            {
                timer1.Interval = int.Parse(txtTime.Text.Trim());  //设定切换的时间间隔
                timer1.Start();                                    //开始图片切换
                button5.Enabled = false;                           //禁用菜单
            }
            else                                                   //否则全屏播放
            {
                Frm_Browser frm2 = new Frm_Browser ();             //创建 Frm_Browser 对象
                frm2.fsi = al;                                     //存储图片路径的集合
                frm2.picPath = txtPicPath.Text.Trim();             //图片路径
                frm2.mytimer = int.Parse(txtTime.Text.Trim());     //切换的时间间隔
                frm2.ShowDialog();                                 //打开窗体 Frm_Browser
            }
        }
    }
}
```

Frm_Browser 窗体中使用 Timer 组件来控制图片的自动播放，实现的思路是设置 Timer 控件的时间间隔后，在其 Tick 事件中编写代码，使图片在 PictureBox 控件中显示，这样在一定的时间间隔内会显示不同的图片。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (MM < listBox1.Items.Count)                                 //如果当前切换的图片小于总的图片数量
    {
        if (txtPicPath.Text.Trim().Length == 3)                    //如果指定的目录是磁盘根目录
        {
            //设置 PictureBox 显示图片
            pictureBox1.Image = Image.FromFile(txtPicPath.Text.Trim() + listBox1.Items[MM].ToString());
            listBox1.SetSelected(MM, true);                        //设置索引等于 MM 的项目被选择
        }
        else                                                       //如果不是磁盘根目录
        {
            //获取路径后还要加上“\\”，然后再加上图片名称
            pictureBox1.Image = Image.FromFile(txtPicPath.Text.Trim() + "\\" + listBox1.Items[MM].ToString());
            listBox1.SetSelected(MM, true);                        //设置索引等于 MM 的项目被选择
        }
    }
    MM++;                                                          //MM 自增
}
```

秘笈心法

心法领悟 571：启动和关闭 Timer 计时器的两种方法。

启动 Timer 计时器时，可以将其 Enabled 属性设置为 true，或者调用其 Start 方法；而关闭 Timer 计时器时，

则需要将其 Enabled 属性设置为 false，或者调用其 Stop 方法。

24.2 多媒体动画的制作

实例 572	利用 Image 制作小动画	初级
	光盘位置：光盘\MR\24\572	趣味指数：★★★★☆

实例说明

本实例实现了用图片控件制作小动画的功能，运行程序，窗体中间会显示一个小动画。实例运行效果如图 24.5 所示。

图 24.5 利用 Image 制作小动画

关键技术

动画主要实现在一定的时间内显示连续的图片，本实例通过添加 Timer 计时器，并在该计时器内显示固定数量的图标来实现动画效果。

说明：关于 Timer 计时器的详细讲解请参见实例 567 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TailorAnimation。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 PictureBox 控件，用来显示图片；添加一个 Timer 计时器组件，用来控制动画显示频率。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    //截取图片所在的文件路径
    strPath = Application.StartupPath.Substring(0, Application.StartupPath.Substring(0, Application.StartupPath.LastIndexOf("\\")).LastIndexOf("\\"));
    pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;                    //设置图片的显示类型
    pictureBox1.Image = Image.FromFile(strPath + @"\image\1.jpg");             //为 pictureBox1 设置显示的图片
}
private void timer1_Tick(object sender, EventArgs e)
{
    Random r = new Random();                                                   //声明一个随机类的对象
    pictureBox1.Image = Image.FromFile(strPath + @"\image\" + r.Next(1, 5) + ".jpg"); //为 pictureBox1 设置显示图片
}
```

秘笈心法

心法领悟 572：Random 类的 Next 方法使用。

Random 类的 Next 方法用来生成随机数，它有以下 3 种重载形式。

（1）Next()：返回非负随机数。

（2）Next(Int32)：返回一个小于所指定最大值的非负随机数。

（3）Next(Int32, Int32)：返回一个指定范围内的随机数。

实例 573	制作家庭影集 光盘位置：光盘\MR\24\573	初级 趣味指数：★★★★

实例说明

许多家庭为了防止照片的损坏，便将图片存入电脑中，为了更方便地浏览图片，本实例制作了一个多图片的浏览程序，运行本实例，单击“上一张”“下一张”按钮，可以对图片进行快速浏览。实例运行效果如图 24.6 所示。

图 24.6　制作家庭影集

关键技术

本实例在实现时，首先通过 DirectoryInfo 类的 GetFileSystemInfos 方法从指定的文件夹中获取图片信息；然后将所有的图片名称取出来放在一个字符串中，并通过 Split 方法分割字符串生成一个存储图片名称的数组，数组的大小就是图片的数量；最后通过一个变量进行增加或减少作为数组下标，从而实现图片的显示。下面对本实例中用到的关键技术进行详细介绍。

（1）DirectoryInfo 类的 GetFileSystemInfos 方法

该方法用来返回表示某个目录中所有文件和子目录的强类型 FileSystemInfo 项的数组。语法格式如下：

```
public FileSystemInfo[] Get FileSystemInfos( )
```

参数说明

返回值：强类型 FileSystemInfo 项的数组。

（2）Split 方法

该方法返回的字符串数组包含字符串中的子字符串（由指定 Unicode 字符数组的元素分隔）。语法格式如下：

```
public string[] Split(char[] separator)
```

参数说明

❶ separator：分隔此实例中子字符串的 Unicode 字符数组，不包含分隔符的空数组或 null。

❷ 返回值：一个数组，其元素包含此实例中的子字符串，这些子字符串由 separator 参数中的一个或多个字符分隔。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 TailorFamilyAlbum。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件和一个 PictureBox

控件，分别用来执行上一张、下一张导向操作以及显示图片信息。

（3）程序的主要代码如下。

在 Frm_Main 窗体中自定义一个 GetAllFiles 方法，用于遍历文件中的所有图片名称。代码如下：

```
public void GetAllFiles(DirectoryInfo dir)
{
    FileSystemInfo[] fileinfo = dir.GetFileSystemInfos( );          //初始化一个 FileSystemInfo 类型的数组
    foreach (FileSystemInfo i in fileinfo)                          //循环遍历 fileinfo 中的每一条记录
    {
        if (i is DirectoryInfo)                                     //当 i 在类 DirectoryInfo 中存在时
        {
            GetAllFiles((DirectoryInfo)i);                          //获取 i 下的所有文件
        }
        else                                                        //当不存在该 i 时
        {
            string str = i.FullName;                                //记录变量 i 的全名
            int b = str.LastIndexOf("\\");                          //在此示例中获取最后一个匹配项的索引
            string strType = str.Substring(b + 1);                  //保存文件的后缀
            //当文件格式为 jpg 或者 bmp 时
            if (strType.Substring(strType.Length - 3) == "jpg" || strType.Substring(strType.Length - 3) == "bmp")
            {
                strInfo += strType+ "#";                            //为变量 strInfo 赋值
            }
        }
    }
}
```

Frm_Main 窗体加载时，调用 GetAllFiles 方法，并将该方法的返回值转化为一个字符串数组，用于存放图片名。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    DirectoryInfo dir = new DirectoryInfo(strPath);                 //创建一个 DirectoryInfo 对象
    GetAllFiles(dir);                                               //获取 dir 下的所有文件
    if (strInfo != "")                                              //当字符串不为空时
    {
        strName = strInfo.Split('#');                               //获取文件名
        Num = 0;                                                    //初始化 Num 的值
        showPic(Num);                                               //显示图片
        Count = strName.Length-1;                                   //记录 Array 中的元素数
    }
    else                                                            //当字符串为空时
    {
        MessageBox.Show("影集里没有照片");                          //弹出信息提示
    }
}
```

秘笈心法

心法领悟 573：妙用 foreach 语句。

foreach 循环专门用来遍历数组和集合的元素，foreach 的工作原理是逐个枚举出数组或集合中的每一个元素，每枚举一个元素就执行一次语句块中的内容，在使用 foreach 遍历的过程中要注意不可以更改遍历中的集合。

实例 574　制作 AVI 播放器　　中级

光盘位置：光盘\MR\24\574　　趣味指数：★★★★☆

实例说明

相信大家都使用过“金山影霸”、RealOne 等影视播放器，其强大的媒体播放功能深受大家的喜爱。是否想过自己设计一个 AVI 播放器呢？本实例中笔者设计了一个 AVI 播放器，实现了对 AVI 文件的基本播放与重

复播放的功能。实例运行效果如图 24.7 所示。

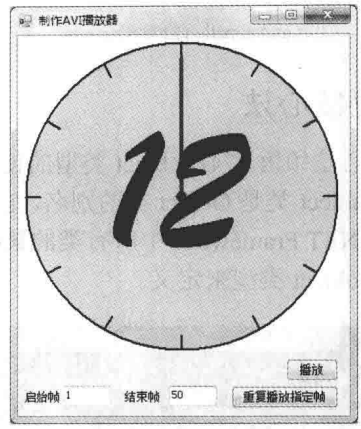


图 24.7　制作 AVI 播放器

关键技术

本实例实现时主要用到了 Microsoft Animation Control Version 6.0 组件，该组件可以一帧一帧地播放 AVI 文件，在使用该组件进行开发程序前，必须将 Microsoft Animation Control Version 6.0 组件添加到工具箱中，步骤如下：

（1）选择工具箱，单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令。

（2）弹出“选择工具箱项”对话框，选择“COM 组件”选项卡。

（3）在“COM 组件”列表中选择 Microsoft Animation Control Version 6.0 选项，再单击“确定”按钮，该组件将会自动添加到工具箱中。

Microsoft Animation Control Version 6.0 组件提供了多个属性、方法用于控制多媒体文件的播放，其常用的属性和方法如表 24.2 所示。

表 24.2　Microsoft Animation Control Version 6.0 组件的常用属性和方法

属性、方法	说　明
AutoPlay 属性	确定程序运行时，媒体播放器是否自动播放
Open 方法	打开一个媒体文件
Stop 方法	停止媒体文件的播放
Play 方法	播放媒体文件

注意： Play 方法可以重载，本实例中使用了它的第一种重载形式，该方法具有 3 个参数，第一个参数是重复播放次数，第二个参数是起始帧的位置，第三个参数是结束帧的位置。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 AviPlay。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 AxAnimation 控件，用来播放 AVI 文件；添加两个 Button 控件，分别用来执行重复播放指定帧操作和播放 AVI 文件操作；添加两个 TextBox 控件，分别用来设置要重复播放的起始帧和结束帧。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.axAnimation1.Open(Application.StartupPath + "//clock.avi");          //加载 AVI 文件
}
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        this.axAnimation1.Stop();                                             //停止播放
        object start = this.textBox1.Text;                                    //保存起始帧中的数据
        object end = this.textBox2.Text;                                      //保存结束帧中的数据
        object time = 20;                                                     //初始化变量 time
        this.axAnimation1.Play(time, start, end);                             //播放指定的帧数
    }
    catch
    {
        MessageBox.Show("请输入正确帧数！");
    }
}
private void button2_Click(object sender, EventArgs e)
```

```
{
    this.axAnimation1.Play();                                    //播放多媒体文件
}
```

秘笈心法

心法领悟 574：object 类型的使用。

object 类是 Object 类的别名，它支持.NET Framework 类层次结构中的所有类，并为派生类提供低级别服务，它是.NET Framework 中所有类的最终基类。一般在程序中定义变量的类型时，如果不确定变量的类型，则可以使用 object 类型来定义。

实例 575	制作电影特效效果 光盘位置：光盘\MR\24\575	中级 趣味指数：★★★★☆

实例说明

随着现代高科技越来越快的发展，电影特效在现代电影中的使用越来越多，而它们这样做的目的无非是为了更好地吸引观看者的眼球。同样地，如果开发人员能在开发的程序中实现一些电影特效效果，也一定可以为程序增色不少。本实例实现了两种常见的电影特效效果，实例运行效果如图 24.8 和图 24.9 所示。

图 24.8　文字特效

图 24.9　图片特效

关键技术

本实例在实现时主要用到了 Label 控件的 Font 属性和 PictureBox 控件的 Size 属性，下面分别对它们进行详细讲解。

（1）Label 控件的 Font 属性

Label 控件表示一个标签控件，其 Font 属性用于获取或设置控件显示的文字的字体，该属性的语法格式如下：

```
public virtual Font Font { get; set; }
```

参数说明

属性值：要应用于由控件显示的文本的 Font。默认为 DefaultFont 属性的值。

（2）PictureBox 控件的 Size 属性

PictureBox 控件用于显示位图、GIF、JPEG、图元文件或图标格式的图形，其 Size 属性用于获取或设置控件的高度和宽度，该属性的语法格式如下：

```
public Size Size { get; set; }
```

参数说明

属性值：表示控件的高度和宽度（以像素为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MovieSpecific。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 Label 控件，用来显示动态缩

放的文字；添加一个 PictureBox 控件，用来显示动态缩放的图片；添加两个 Button 控件，分别用来执行文字和图片缩放操作；添加一个 Timer 计时器组件，用来控制文字和图片的缩放。

（3）程序的主要代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;                                          //启动计时器
    pictureBox1.Visible = false;                                    //隐藏 PictureBox 控件
    label1.Visible = true;                                          //显示 Label 控件
}
private void button3_Click(object sender, EventArgs e)
{
    label1.Visible = false;                                         //隐藏 Label 控件
    timer1.Enabled = true;                                          //启动计时器
    pictureBox1.Visible = true;                                     //显示 PictureBox 控件
}
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Font = new Font(label1.Font.FontFamily, label1.Font.Size + 1);          //使字体逐步加 1
    pictureBox1.Size = new Size(pictureBox1.Size.Width + 5, pictureBox1.Size.Height + 5);  //使图片逐渐增大
}
```

秘笈心法

心法领悟 575：隐藏控件。

控件的基类 Control 类中提供了一个 Visible 属性，在开发程序时，只要将该属性的值设置为 false，即可隐藏相应的控件。

24.3　多媒体动画的播放

实例 576	播放 GIF 动画 光盘位置：光盘\MR\24\576	初级 趣味指数：★★★★

实例说明

GIF 是流行于 Internet 上一种较为特殊的格式，解释为图像交换格式（Graphics InterChange Format），在实际应用中一般都使用动画的 GIF 文件，本实例设计了一个 GIF 动画播放软件。实例运行效果如图 24.10 所示。

图 24.10　播放 GIF 动画

关键技术

本实例主要通过 ImageAnimator 类实现播放 GIF 动画，下面对 ImageAnimator 类的常用方法进行详细讲解。

（1）Animate 方法

该方法表示将多帧图像转化为动画。语法格式如下：

```
public static void Animate (Image image, EventHandler onFrameChangedHandler)
```

参数说明

❶ image：要动画处理的 Image 对象。

❷ onFrameChangedHandler：一个 EventHandler 对象，指定在动画帧发生更改时调用的方法。

（2）CanAnimate 方法

该方法返回一个 bool 值，用来指示图像中是否包含基于时间的帧。语法格式如下：

```
public static bool CanAnimate (Image image )
```

参数说明

❶ image：要测试的 Image 对象。

❷ 返回值：如果指定图像包含基于时间的帧，此方法返回 true；否则返回 false。

（3）StopAnimate 方法

该方法表示终止正在运行的动画。语法格式如下：

```
public static void StopAnimate (Image image, EventHandler onFrameChangedHandler)
```

参数说明

❶ image：要停止动画处理的 Image 对象。

❷ onFrameChangedHandler：一个 EventHandler 对象，指定在动画帧发生更改时调用的方法。

（4）UpdateFrames 方法

该方法使该帧在当前正被动画处理的所有图像中前移，新帧在下一次呈现图像时绘制。语法格式如下：

```
public static void UpdateFrames ( )
```

（5）UpdateFrames 方法

该方法使帧在指定的图像中前移，新帧在下一次呈现图像时绘制。该方法只适用于包含基于时间的帧的图像。语法格式如下：

```
public static void UpdateFrames (Image image )
```

参数说明

image：要为其更新帧的 Image 对象。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PlayGifAnimation。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行播放和停止 GIF 动画操作。

（3）程序的主要代码如下：

```
Bitmap bitmap = new Bitmap(Application.StartupPath+"\\1.gif");                    //创建一个 GDI+绘图图面对象
bool current = false;                                                             //初始化一个 bool 型的变量 current
public void PlayImage( )
{
    if (!current)                                                                 //当该值为 true 时
    {
        ImageAnimator.Animate(bitmap, new EventHandler(this.OnFrameChanged));     //将多帧图片显示为动画图像
        current = true;                                                           //设定 current 的值为 true
    }
}
private void OnFrameChanged(object o, EventArgs e)
{
    this.Invalidate( );                                                           //使控件的整个图片无效并导致重绘事件
}
protected override void OnPaint(PaintEventArgs e)
{
    e.Graphics.DrawImage(this.bitmap, new Point(1, 1));                           //从指定的位置绘制图片
    ImageAnimator.UpdateFrames( );                                                //使该帧在当前正被图画处理的所有图像中前移
}
private void button1_Click(object sender, EventArgs e)
{
    PlayImage( );                                                                 //播放
    ImageAnimator.Animate(bitmap, new EventHandler(this.OnFrameChanged));         //将多帧图像显示为动画
}
private void button2_Click(object sender, EventArgs e)
{
    ImageAnimator.StopAnimate(bitmap, new EventHandler(this.OnFrameChanged));     //停止
}
```

秘笈心法

心法领悟 576：使用 Graphics 类的 DrawImage 方法绘制图像。

DrawImage 方法主要用来绘制图像，此方法在由坐标对指定的位置，使用图形的原始物理大小绘制指定的图像。例如，本实例中使用 Graphics 类的 DrawImage 方法绘制图像的代码如下：

```
e.Graphics.DrawImage(this.bitmap, new Point(1, 1));                    //从指定的位置绘制图片
```

实例 577　播放 Flash 动画

光盘位置：光盘\MR\24\577　　中级　　趣味指数：★★★★★

实例说明

在互联网高度发展的今天，读者一定不会对 Flash 感到陌生，这种格式的媒体文件是由 Macromedia 公司推出的交互式矢量图和 Web 动画的标准。使用此种格式的文件可以创作具有交互性的多媒体动画，并且文件体积非常小。Flash 不仅在网上流行，目前的家用电脑中这种文件也非常多。本实例设计了一个播放 Flash 动画的软件，实例运行效果如图 24.11 所示。

图 24.11　播放 Flash 动画

关键技术

本实例的关键技术是 Flash 插件，首先要确认计算机中是否存在 Flash 插件，也就是 IE 浏览器浏览网页时是否能够播放网页中的 Flash。播放 Flash 时主要是使用了 Macromedia 公司提供的一个 ActiveX 组件，该 ActiveX 组件是 Flash8.OCX。将其引用到项目中的步骤如下：

（1）选择工具箱，单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令。

（2）弹出“选择工具箱项”对话框，选择“COM 组件”选项卡。

（3）在“COM 组件”列表中单击“浏览”按钮，在弹出的对话框中选择 Windows\system32\Macromed\Flash\Flash8.OCX 选项。

（4）在“COM 组件”选项卡的列表视图中会新增 Macromedia Flash Factory Object 选项，选择此选项后，在工具箱中会新增一个 Shockwave Flash Object 控件，之后开发人员就可以像使用 Windows 标准控件一样使用它了。

Shockwave Flash Object 组件的常用属性及说明如表 24.3 所示。

表 24.3　Shockwave Flash Object 组件的常用属性及说明

属　性	说　明
BackgroundColor	用于设置背景颜色
Loop	设置是否循环显示
Menu	是否显示右键菜单
Movie	设置 Flash 动画的文件名，可以动态设置，要关闭一个动画只要把这个属性设置为空即可
Playing	用于播放或暂停一个 Flash

Shockwave Flash Object 组件的常用方法及说明如表 24.4 所示。

表 24.4　Shockwave Flash Object 组件的常用方法及说明

方　法	说　明	方　法	说　明
Play	播放动画	Rewind	播放第一帧动画
Stop	停止动画	SetZoomRet	设置缩放的区域
Back	播放前一帧动画	Zoom	按指定百分比缩放
Forward	播放后一帧动画	LoadMovie	加载网络上的动画

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 playflash。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 OpenFileDialog 控件，用来打开 Flash 文件；添加一个 Panel 控件，用来添加播放 Flash 的控件；添加一个 Timer 组件，用来控制菜单栏中各个菜单项的状态。

（3）程序的主要代码如下。

在 Frm_Main 窗体的后台代码中，首先定义程序中要使用的全局对象，该对象用于播放 Flash 文件。代码如下：

```
AxShockwaveFlashObjects.AxShockwaveFlash ax;                    //创建 AxShockwaveFlash 实例
```

自定义一个 AddFlash 方法用于创建 AxShockwaveFlash 对象，并将该对象添加到 Panel 控件中。代码如下：

```
private void AddFlash()
{
    ax = new AxShockwaveFlashObjects.AxShockwaveFlash();        //创建 AxShockwaveFlash 对象
    panel1.Controls.Add(ax);                                    //添加到 Panel 控件中
    ax.Dock = DockStyle.Fill;                                   //设置填充模式
    ax.ScaleMode = 1;
    ax.Stop();                                                  //停止，不播放
}
```

当窗体加载时，会调用 AddFlash 方法加载播放 Flash 文件的控件。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    AddFlash();                                                 //窗体加载时添加播放器
    ax.Visible = false;                                         //隐藏播放器
    ControlState(0);                                            //设置菜单状态
}
```

选择菜单栏中的“文件”/“打开”命令，弹出选择 Flash 文件的窗口，选择某个 Flash 文件后，单击“确定”按钮，获取 Flash 文件的路径，并赋值给 AxShockwaveFlash 对象的 Movie 属性值，实现播放 Flash 文件的功能。代码如下：

```
private void 打开 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)        //如果选择了 Flash 文件
    {
        ax.Visible = true;                                      //显示播放器
        string flashPath = openFileDialog1.FileName;            //获取 Flash 文件路径
        ax.Movie = flashPath;                                   //设置播放器的 Movie 属性
        panel1.Visible = true;                                  //显示 Panel 控件
        ControlState(1);                                        //激活菜单
    }
}
```

选择菜单栏中的“控制”/“第一帧”命令，调用 AxShockwaveFlash 对象的 Rewind 方法使 Flash 从第一帧开始播放。代码如下：

```
private void 第一帧 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    ax.Rewind();                                                // Rewind 方法播放第一帧
}
```

选择菜单栏中的“控制”/“向前”命令，调用 AxShockwaveFlash 对象的 Back 方法播放 Flash 的上一帧。代码如下：

```
private void 向前 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    ax.Back();                                                  // Back 方法播放上一帧
}
```

选择菜单栏中的“控制”/“向后”命令，调用 AxShockwaveFlash 对象的 Forward 方法播放 Flash 的下一帧。代码如下：

```
private void 向后 ToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    ax.Forward();                                    // Forward 方法播放下一帧
}
```

秘笈心法

心法领悟 577：添加 Shockwave Flash Object 组件时的注意事项。

在 C#程序中添加 Shockwave Flash Object 组件时，必须保证本地计算机上已经正确安装了 Macromedia Flash 软件。

实例 578　播放 FLV 文件

光盘位置：光盘\MR\24\578　　高级　　趣味指数：★★★★★

实例说明

随着网络的普及，很多网民会选择在线观看影片或者其他的视频节目。由于这些视频文件本身体积很大，而且还不利于网络传播。所以现在各大网络视频网站都选择将视频文件转换成 FLV 格式，然后放到网站上供网民们点击观看。这样不仅大大减小了视频文件的体积，而且还有利于网络传播，使视频播放更加流畅。很多网民为了观看方便，有时会将 FLV 视频文件下载到自己的计算机中，这样就需要在本地计算机中提供 FLV 文件播放器。本实例通过 C#与 Flash 相结合开发出 FLV 文件播放器。实例运行效果如图 24.12 所示。

图 24.12　播放 FLV 文件

关键技术

本实例的要点是如何播放 FLV 文件，在 C#中没有提供播放 FLV 文件的控件，所以只能借助其他工具来实现播放 FLV 文件的功能，Macromedia Flash 8 针对 FLV 增加了一个非常好用的 FLVPlayback 组件。FLVPlayback 组件包含自定义用户界面控件，用于播放、停止、暂停和回放视频，这些控件包括 BackButton、ForwardButton、PauseButton、PlayButton、PlayPauseButton、SeekBar 和 StopButton 等，可以将它们拖动到舞台上并分别进行自定义。

FLVPlayback 组件具有下列功能：

- 提供一组预制的外观，可用于自定义用户界面。
- 使高级用户可以创建自己的自定义外观。
- 提供指令点，可用于将 Flash 应用程序中的动画、文本和图形同步。
- 提供自定义的实时预览。
- 保持合理的 SWF 文件大小以便于下载。

下面介绍如何通过 Macromedia Flash 8 制作 FLV 播放器，具体步骤如下：

（1）新建一个空白 Flash 文档，宽度设为 320，高度设为 258，背景设为黑色。

（2）在“组件”面板中单击 FLV Playback - Player 8 项目前面的加号（+）按钮，如图 24.13 所示。

（3）将 FLVPlayback 组件拖动到舞台上。

（4）选中舞台上的 FLVPlayback 组件，在“组件”检查器的“参数”选项卡上找到 skin 参数，单击其右侧的放大镜按钮，打开“选择外观”对话框。在该对话框中选择所需要的皮肤，单击“确定”按钮，完成对 skin 参数的设置，如图 24.14 所示。

图 24.13 “组件”面板

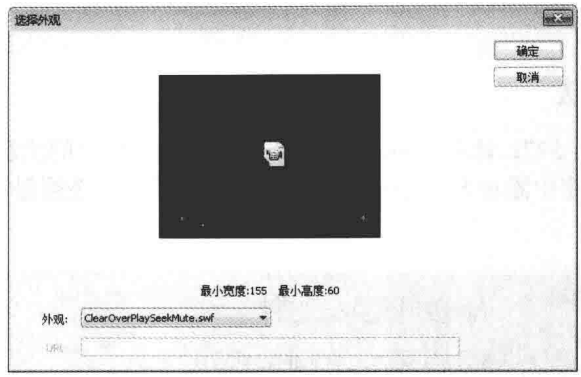


图 24.14 “选择外观”对话框

（5）选中舞台上的 FLVPlayback 组件，在“组件”检查器的“参数”选项卡上找到 contentPath 参数，单击其右侧的放大镜按钮，输入属于以下内容的路径：

- 指向 FLV 文件的本地路径。
- 指向 FLV 文件的 URL。
- 指向 XML 文件的 URL，该文件说明如何播放 FLV 文件。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 Playflv。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ListView 控件，用来显示播放列表；添加一个 ContextMenuStrip 控件，用来作为播放器的右键快捷菜单；添加两个 Panel 控件，分别用来添加 Flash 播放器和显示、隐藏播放列表；添加一个 OpenFileDialog 控件，用来打开 FLV 文件。

（3）程序的主要代码如下。

在 Frm_Main 窗体中自定义一个 ChangeFlv 方法，用于修改 XML 文件中的节点，该节点记录要播放的 FLV 文件的路径。代码如下：

```
private void ChangeFlv(string path)
{
    //获取 XML 文件的路径
    xmlPath = Application.StartupPath.ToString();
    xmlPath = xmlPath.Substring(0, xmlPath.LastIndexOf("\\"));
    xmlPath = xmlPath.Substring(0, xmlPath.LastIndexOf("\\"));
    xmlPath += @"\FLVPlayer";
    xmlPath += @"\list.xml";
    XmlDocument doc = new XmlDocument();                          //创建 XmlDocument 实例
    doc.Load(xmlPath);                                            //加载 XML 文件
    XmlNode nodePath = doc.SelectSingleNode("flvLists/item");     //打开节点
    XmlElement xe = (XmlElement)nodePath;                         //创建 XML 节点
    xe.SetAttribute("title", path);                               //设置元素的属性
    doc.Save(xmlPath);                                            //保存
}
```

创建一个 playFLV 方法用于播放 FLV 文件，在此方法中，首先在 C 盘创建一个 flvVidio 文件夹，然后将要播放的 FLV 文件复制到该文件夹中。由于播放 FLV 的组件不支持中文路径，所以需要对 FLV 文件重新命名，最后将更改后的路径赋值给 AxShockwaveFlash 对象的 Movie 属性，开始播放 FLV 文件。代码如下：

```
private void playFLV(string path)                                 //播放 FLV 文件的方法
{
    FileInfo fi2 = new FileInfo(path);                            //创建 FileInfo 对象
    if (fi2.Exists)                                               //如果文件存在
    {
        Directory.CreateDirectory("c:\\flvVidio");                //新建文件夹
        //随机生成文件名
        string newPath="c:\\flvVidio\\"+DateTime.Now.Year+DateTime.Now.Second+".flv";
        File.Copy(path, newPath);                                 //将原 FLV 文件复制到新建的文件夹中
```

```
        ChangeFlv(newPath);                                          //修改 XML 文件中的播放地址
        this.Text = listView1.SelectedItems[0].SubItems[0].Text;     //显示正在播放的文件名称
        ax.Dispose();                                                //释放
        AddFlash();                                                  //重新添加播放器
        ax.Movie = strg;                                             //设置 Movie 属性
    }
}
```

在播放列表上单击鼠标右键，在弹出的快捷键菜单中选择“打开文件”命令，弹出选择 FLV 文件的窗口。选择 FLV 文件之后，会根据选择的路径获取该文件的名称、路径和大小等信息，然后将这些信息添加到 ListView 控件中。代码如下：

```
private void 打开文件 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)                 //选择文件
    {
        listView1.Items.Clear();                                         //清空 ListView 控件
        string[] files = openFileDialog1.FileNames;                      //声明数组存储选择的文件
        for (int i = 0; i < files.Length; i++)                           //遍历数组
        {
            string flvPath = files[i];                                   //获取路径
            string flvName = flvPath.Substring(flvPath.LastIndexOf("\\") + 1, flvPath.Length - flvPath.LastIndexOf("\\") - 1);
                                                                         //获取文件名称
            fi = new FileInfo(flvPath);                                  //创建 FileInfo 对象
            flv[0] = flvName;                                            //文件名称
            flv[1] = flvPath;                                            //文件路径
            flv[2] = Convert.ToString(fi.Length / 1024) + "KB";          //文件大小
            ListViewItem lvi = new ListViewItem(flv);                    //创建 ListViewItem 实例
            listView1.Items.Add(lvi);                                    //添加到 ListView 控件中
        }
    }
}
```

将 FLV 文件添加到列表之后，双击列表中的某一项即可播放该文件。实现的思路是双击后获取选择项的路径，然后将这个路径作为参数传递给 playFLV 方法播放该文件。代码如下：

```
private void listView1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    try
    {
        if (listView1.SelectedItems.Count > 0)                           //判断是否添加了要播放的文件
        {
            string path = listView1.SelectedItems[0].SubItems[1].Text;   //获取 FLV 文件的路径
            playFLV(path);                                               //调用 playFLV 方法播放 FLV 文件
        }
    }
    catch { }
}
```

秘笈心法

心法领悟 578：如何使用 FLVPlayback 组件播放 FLV 文件？

在 C#中没有提供播放 FLV 文件的控件，所以只能借助 Macromedia Flash 8 的 FLVPlayback 组件播放 FLV 文件。C#只对 XML 文件进行读写，修改 XML 文件中的 FLV 文件路径，当通过 FLVPlayback 组件制作的 Flash 加载时首先读取 XML 文件中的 FLV 路径。

实例 579	产品电子报价	中级
	光盘位置：光盘\MR\24\579	趣味指数：★★★★☆

实例说明

本实例是通过电子产品的名称来查看该电子产品的图片。运行程序，在“海报列表”下拉列表框中选择电

子产品的名称，单击“查询”按钮，即可在产品电子报价栏中显示该产品的图片信息。实例运行效果如图 24.15 所示。

图 24.15　产品电子报价

关键技术

通过对本实例的分析，由于要将所有海报名添加到下拉列表框中，海报的名称是唯一的，但是海报的格式（如 JPG、BMP 等）不一定相同，所以要通过名称以最快的方式定位到图片上，笔者在这里使用了 Hashtable 类。Hashtable 类用于表示键/值对的集合，这些键/值对可以根据键的哈希代码进行组织。Hashtable 类的常用方法及说明如表 24.5 所示。

表 24.5　Hashtable 类的常用方法及说明

方　法	说　明
Add	将带有指定键和值的元素添加到 Hashtable 中
Clear	从 Hashtable 中移除所有元素
Contains	确定 Hashtable 是否包含特定键
ContainsKey	确定 Hashtable 是否包含特定键
ContainsValue	确定 Hashtable 是否包含特定值
CopyTo	将 Hashtable 元素复制到一维 Array 实例中的指定索引位置
GetEnumerator	返回循环访问 Hashtable 的 IDictionaryEnumerator
Remove	从 Hashtable 中移除带有指定键的元素

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ProductElectronQuote。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ComboBox 控件，用来选择产品名称列表；添加一个 Button 控件，用来执行查找功能；添加一个 PictureBox 控件，用来显示产品海报。

（3）程序的主要代码如下：

```
Hashtable ht = new Hashtable();                          //初始化一个哈希表对象
string str;                                              //初始化一个字符串 str
public Frm_Main()
{
    InitializeComponent();
    str = Application.StartupPath + "\\Image";           //保存商品图片所在的路径
}
private void Form1_Load(object sender, EventArgs e)
{
    DirectoryInfo dir = new DirectoryInfo(str);          //创建一个 DirectoryInfo 类的对象
    GetAllFiles(dir);                                    //获取指定路径下的所有文件
    foreach (DictionaryEntry de in ht)                   //循环哈希表中的所有数据
        this.comboBox1.Items.Add(de.Key);                //向 comboBox1 中添加内容
}
public void GetAllFiles(DirectoryInfo dir)
{
```

```
    FileSystemInfo[] fileinfo = dir.GetFileSystemInfos();                    //创建一个 FileSystemInfo 类型的对象
    foreach (FileSystemInfo i in fileinfo)                                   //循环遍历 fileinfo 下的所有内容
    {
        if (i is DirectoryInfo)                                              //当在 DirectoryInfo 中存在 i 时
        {
            GetAllFiles((DirectoryInfo)i);                                   //获取 i 下的所有文件
        }
        else                                                                 //当在 DirectoryInfo 中不存在 i 时
        {
            string str = i.FullName;                                         //记录 i 的绝对路径
            int b = str.LastIndexOf("\\");                                   //获取字符串与指定项匹配的最后一个索引
            string strType = str.Substring(b + 1);                           //获取文件的后缀名
            //当图片类型为 jpg 或者 bmp 时
            if (strType.Substring(strType.Length - 3).ToLower() == "jpg" || strType.Substring(strType.Length - 3).ToLower() == "bmp")
            {
                ht.Add(strType.Substring(0, strType.Length - 4), strType);   //向哈希表中添加内容
            }
        }
    }
}
private void showPic(string name)
{
    this.pictureBox1.ImageLocation = str + "\\" + name;                      //设置在 pictureBox1 中显示图片的路径
}
```

秘笈心法

心法领悟 579：如何获取一个文件夹中的所有子文件夹及文件？

获取一个文件夹中的所有子文件夹及文件时，可以使用 DirectoryInfo 类的 GetFileSystemInfos 方法来实现，该方法用来返回表示某个目录中所有文件和子目录的强类型 FileSystemInfo 项的数组。例如，本实例中使用该方法的代码如下：

```
str = Application.StartupPath + "\\Image";                                   //保存商品图片所在的路径
DirectoryInfo dir = new DirectoryInfo(str);                                  //创建一个 DirectoryInfo 类的对象
FileSystemInfo[] fileinfo = dir.GetFileSystemInfos();                        //创建一个 FileSystemInfo 类型的对象
```

实例 580	产品滚动展示程序 光盘位置：光盘\MR\24\580	中级 趣味指数：★★★★☆

实例说明

很多商家为了宣传自己的产品，在网络上制作网页，通过互联网将自己的产品推向全国，甚至全世界。本实例将模拟网页来制作一个窗体，将一系列产品的图片在窗体上滚动显示。实例运行效果如图 24.16 所示。

图 24.16　产品滚动展示程序

关键技术

本实例主要使用 Timer 计时器组件来控制产品图片的滚动显示。实现过程中，主要利用 Timer 计时器组件使 Panel 控件的 Left 属性在一定的时间内减 10，从而达到产品图片滚动的效果。下面对本实例中用到的关键技术进行详细讲解。

Panel 控件是一个容器控件，其 Left 属性用来获取或设置控件左边缘与其容器的工作区左边缘之间的距离（以像素为单位），该属性的语法格式如下：

```
[BrowsableAttribute(false)]
public int Left { get; set; }
```

参数说明

属性值：表示控件左边缘与其容器的工作区左边缘之间的距离（以像素为单位）。

说明：关于 Timer 计时器的详细讲解请参见实例 567 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ProductTumblingShown。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 PictureBox 控件，用来显示产品图片；添加 3 个 ImageList 组件，用来存放产品图片；添加两个 Timer 组件，分别用来控制产品图片以指定的速度向左和向右移动。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    left = 10;                                          //初始化变量 left 的值
    this.panel1.Left += left;                           //设置 panel1 控件距容器左边缘之间的距离
    int width = this.Width - this.panel1.Width;         //初始化变量 width
    if (this.panel1.Left > width)                       //当 panel1 控件距容器左边缘之间的距离大于 width 时
    {
        this.timer1.Enabled = false;                    //禁用计时器 timer1
        this.timer2.Enabled = true;                     //启用计时器 timer2
        this.pictureBox1.Image = this.imageList1.Images[0];  //设置 pictureBox1 控件中显示的图片
        this.pictureBox2.Image = this.imageList2.Images[0];  //设置 pictureBox2 控件中显示的图片
        this.pictureBox3.Image = this.imageList3.Images[0];  //设置 pictureBox3 控件中显示的图片
    }
}
private void Frm_Main_Load(object sender, EventArgs e)
{
    this.pictureBox1.Image = this.imageList1.Images[0];  //设置 pictureBox1 控件中显示的图片
    this.pictureBox2.Image = this.imageList2.Images[0];  //设置 pictureBox2 控件中显示的图片
    this.pictureBox3.Image = this.imageList3.Images[0];  //设置 pictureBox3 控件中显示的图片
}
private void timer2_Tick(object sender, EventArgs e)
{
    left = -10;                                         //初始化 left 的值
    this.panel1.Left += left;                           //设置 panel1 距容器左边缘之间的距离
    if (this.panel1.Left < 0)                           //当 panel1 距容器左边缘之间的距离小于 0 时
    {
        this.timer1.Enabled = true;                     //启用计时器 timer1
        this.timer2.Enabled = false;                    //禁用计时器 timer2
        this.pictureBox1.Image = this.imageList1.Images[1];  //设置 pictureBox1 控件中显示的图片
        this.pictureBox2.Image = this.imageList2.Images[1];  //设置 pictureBox2 控件中显示的图片
        this.pictureBox3.Image = this.imageList3.Images[1];  //设置 pictureBox3 控件中显示的图片
    }
}
```

秘笈心法

心法领悟 580：启动和关闭 Timer 计时器的两种方法。

启动 Timer 计时器时，可以将其 Enabled 属性设置为 true，或者调用其 Start 方法；而关闭 Timer 计时器时，则需要将其 Enabled 属性设置为 false，或者调用其 Stop 方法。

第25章

音频与视频控制

- CD、VCD 播放
- MP3、WAV 播放
- 媒体控制
- 多媒体应用

25.1 CD、VCD 播放

实例 581	播放指定的 avi-mid-wav 文件	初级
	光盘位置：光盘\MR\25\581	趣味指数：★★★★★

实例说明

AVI、MID 和 WAV 文件都是很流行的多媒体文件，AVI 文件是带有视频的多媒体文件，而 MID 和 WAV 都是声音文件，MID 文件是一种简单的 MIDI 格式文件，这种文件非常小，文件中只标记乐器和节奏等信息，目前的移动电话都支持此种格式作为铃声。本实例将制作一个播放 avi-mid-wav 文件的实例，实例运行效果如图 25.1 所示。

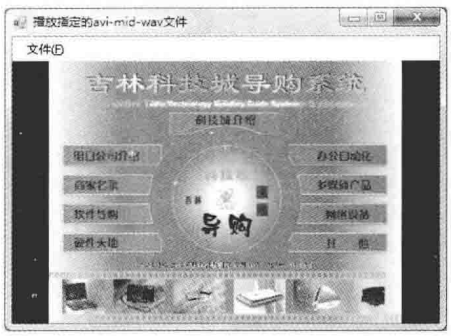


图 25.1 播放指定的 avi-mid-wav 文件

关键技术

本实例主要使用 Windows Media Player 组件实现播放 avi-mid-wav 文件的功能，下面对其进行详细讲解。

Windows Media Player 是一个媒体播放组件，该组件可以播放 Windows 中的多种媒体文件格式。在使用 Windows Media Player 组件进行开发程序前，必须添加 Windows Media Player 组件，将其引用到项目中的步骤如下：

（1）选择工具箱，单击鼠标右键，在弹出的快捷菜单中选择“选择项”命令。

（2）弹出“选择工具箱项”对话框，选择“COM 组件”选项卡。

（3）在列表中选择 Windows Media Player 选项，然后单击“确定”按钮返回工具箱，此时在工具箱中会新增一个 Windows Media Player 组件，之后开发人员就可以像使用 Windows 标准控件一样使用它了。

Windows Media Player 组件的常用属性及说明如表 25.1 所示。

表 25.1 Windows Media Player 组件的常用属性及说明

属　性	说　明
URL	string 类型，指定媒体位置，本机或网络地址
mute	bool 类型，设置是否静音
playState	int 类型，播放状态，1=停止，2=暂停，3=播放，6=正在缓冲，9=正在连接，10=准备就绪
currentPositionString	string 类型，表示获取当前进度，例如“00:23”
duration	double 类型，获取媒体总长度
currentPosition	double 类型，获取当前进度
volume	int 类型，获取或设置音量，范围为 0~100

Windows Media Player 组件的常用方法及说明如表 25.2 所示。

表 25.2 Windows Media Player 组件的常用方法及说明

方　法	说　明
play	播放媒体
pause	暂停播放媒体
stop	停止播放媒体

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 PlayAVI、MIDAndWAVCluster。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 MenuStrip 控件，用来作为窗体的菜单栏；添加一个 OpenFileDialog 控件，用来显示选取文件对话框；添加一个 Windows Media Player 组件，用于播放媒体文件。

（3）程序的主要代码如下：

```
private void 选取文件_Click(object sender, EventArgs e)
{
    this.optFile.ShowDialog();                                    //打开“打开”对话框
    this.axWindowsMediaPlayer1.newMedia(this.optFile.FileName);   //加载选择的文件
}
private void 播放文件_Click(object sender, EventArgs e)
{
    this.axWindowsMediaPlayer1.URL = this.optFile.FileName;       //播放文件
}
private void 停止_Click(object sender, EventArgs e)
{
    this.axWindowsMediaPlayer1.close();                           //停止播放
}
```

秘笈心法

心法领悟 581：音频的主要分类。

音频主要分为 3 种类型，即波形音频、CD Audio 和 MIDI，其详细说明如下。

（1）波形音频：存放在计算机文件中的电子数字音频，文件中保存的是音频的波形信息，其特点是声音质量好，但文件通常比较大。

（2）CD Audio：通过计算机上的 CD-ROM 控制并播放的 CD 音乐，通常被称为红皮书音频标准，CD Audio 需要大量的存储空间。

（3）MIDI：电子合成音乐，是音乐与计算机结合的产物，MIDI 不是把音乐的波形进行数字化采样和编码，而是将数字式电子乐器的弹奏过程记录下来。

实例 582　获取多媒体详细信息列表

光盘位置：光盘\MR\25\582　　初级　　趣味指数：★★★★

实例说明

近年来多媒体已经深入到千家万户，每台计算机中都会安装媒体播放器这类软件，如何制作一款媒体播放器呢？本实例实现一个简单的媒体播放器，使用它可以播放多种媒体文件。运行本实例，首先单击“打开”按钮选择一个文件，然后单击“播放”按钮即可完成媒体播放的功能，同时本软件还为使用者提供了歌曲的详细信息。实例运行效果如图 25.2 所示。

图 25.2　获取多媒体详细信息列表

关键技术

本实例在获取多媒体文件的详细信息时，主要用到了 IWMPMedia 类的 getItemInfo 方法，下面对该方法进行详细讲解。

IWMPMedia 类是微软提供的一个多媒体播放类，其 getItemInfo 方法用来获取当前播放媒体的歌手名、歌名等详细信息。

```
string getItemInfo(string bstrItemName);
```

参数说明

❶ bstrItemName：要获取的媒体信息。

❷ 返回值：获取到的媒体详细信息。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchDetailsInfoList。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 ToolStrip 控件，用来作为窗体的工具栏，主要实现打开媒体文件、开始播放、暂停播放、停止播放和显示详细信息等功能；添加一个 OpenFileDialog 控件，用来显示选取文件对话框；添加一个 Windows Media Player 组件，用于播放媒体文件。

（3）程序的主要代码如下：

```
private void ButInfo_Click(object sender, EventArgs e)
{
    if (this.optFile.FileName != "optFile")                    //当打开文件的文件名不为 optFile 时
    {
        c = new WMPLib.WindowsMediaPlayerClass();              //创建 WindowsMediaPlayerClass 类的对象
        m = c.newMedia(this.optFile.FileName);                 //为 IWMPMedia 对象赋值
        //获取媒体详细信息
        MessageBox.Show("歌手名:" + m.getItemInfo("Author") + "\r\n" + "歌　名:" + m.getItemInfo("Title"));
    }
}
```

秘笈心法

心法领悟 582：什么是多媒体？

多媒体是指利用计算机技术，把多种媒体综合在一起，使之建立起逻辑上的联系，并能对它们进行各种处理的一种方法。多媒体主要包括文本、图形、影像、声音、视频和动画等各种形式，并与先进的计算机、通信和广播电视技术相结合，形成一个可组织、存储、操作和控制多媒体信息的集成环境和交互系统。

25.2　MP3、WAV 播放

实例 583	获取 MP3 文件的播放时间	初级
	光盘位置：光盘\MR\25\583	趣味指数：★★★★

实例说明

每一首歌曲都有不同长度的播放时间，如果对歌曲的播放时间做到心中有数，那么在听歌的过程中就能更好地欣赏歌曲。本实例可以获取音乐文件的播放时间，实例运行效果如图 25.3 所示。

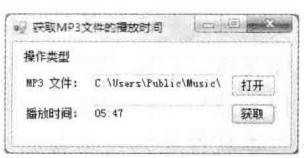


图 25.3　获取 MP3 文件的播放时间

关键技术

本实例实现时主要用到了 Windows Media Player 组件的 currentMedia.duration 属性，下面对其进行详细讲解。

Windows Media Player 组件的 currentMedia 属性用来获取或设置 Windows Media Player 组件的当前播放媒体，而且 currentMedia.duration 属性用来获取当前媒体的总播放长度，其声明格式如下：

```
double duration {get;}
```

参数说明

属性值：double 类型，表示当前媒体的总播放长度（以秒为单位）。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchPlayTime。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示 MP3 文件的路径及播放时间；添加两个 Button 控件，分别用来打开 MP3 文件和获取 MP3 文件的播放时间；添加一个 Windows Media Player 组件，用来加载 MP3 文件。

（3）程序的主要代码如下：

```
private void unfold_Click(object sender,EventArgs e)
{
    OpenFileDialog MP3Dialog = new OpenFileDialog();                    //声明一个提示用户打开文件的对象
    MP3Dialog.Filter = "MP3 文件(*.MP3)|*.MP3";                         //获取或设置当前文件名筛选器字符串
    if(MP3Dialog.ShowDialog() == DialogResult.OK)                       //当选定文件之后单击“打开”按钮时
    {
        filePath.Text = MP3Dialog.FileName;                             //在文本框中显示打开文件的文件路径
        axWindowsMediaPlayer1.URL = MP3Dialog.FileName;                 //为 WindowsMediaPlayer 组件指定媒体
    }
}
private void snatch_Click(object sender,EventArgs e)
{
    //在文本框中显示歌曲的播放时间
    playTime.Text = GetFileTime(Convert.ToInt32(axWindowsMediaPlayer1.currentMedia.duration)*1000);
}
#region  获取文件的播放时间，并按指定格式进行显示
/// <summary>
/// 获取文件的播放时间，并按指定格式进行显示
/// </summary>
/// <param Millisecond="int">毫秒数</param>
public string GetFileTime(int Millisecond)
{
    string Tem_Time = "";                                               //用来保存歌曲的播放时间
    double Tem_min = 0;                                                 //用来保存歌曲播放时间的分钟部分
    double Tem_sec = 0;                                                 //用来保存歌曲播放时间的秒部分
    double Tem_millisec = 0;                                            //用来保存歌曲播放时间的毫秒部分
    Tem_min = Millisecond / 1000;                                       //将当前时间转化为以秒为单位的数据类型
    Tem_min = Tem_min / 60.0;                                           //将当前时间转化为以分为单位的数据类型
    Tem_sec = Tem_min - (int)Tem_min;                                   //保存歌曲播放时间的小数部分（当以分为单位时）
    Tem_min = (int)Tem_min;                                             //将 double 型变量 Tem_min 转化为 int 型变量
    Tem_sec = (60 * Tem_sec) / 100.0;                                   //将获得的小数转化为以秒为单位的数据
    Tem_sec = (int)(Tem_sec * 100);                                     //将数据类型转化为 int 型
    //将歌曲播放的时间转换为以秒为单位存储
    Tem_millisec = (int)((Millisecond - Tem_min * 60 * 1000 - Tem_sec * 1000) / 1000 * 100);
    if(Tem_min >= 100)                                                  //当 Tem_min 的值大于等于 100 时
    {
        Tem_Time = Tem_min.ToString("000") + ":" + Tem_sec.ToString("00");//设置时间的显示格式
    }
    else                                                                //当 Tem_min 的值小于 100 时
        Tem_Time = Tem_min.ToString("00") + ":" + Tem_sec.ToString("00"); //设置时间的显示格式
    return Tem_Time;                                                    //返回变量 Tem_Time
}
#endregion
```

秘笈心法

心法领悟 583：数值类型如何格式化？

如果要把一个数字格式化为 4 位数、货币形式等，这时可以使用数值类型的 ToString 方法来对其进行格式化。例如，要将一个 int 类型的数字 8 格式化为 4 位数，即 0008 这种格式，那么可以使用下面的代码来实现：

```
int i = 8;                                                              //定义 int 类型变量
string str = i.ToString("0000");                                        //将 int 类型的变量格式化为 4 位数字
```

实例 584　多功能 MP3 播放器

光盘位置：光盘\MR\25\584　　高级　　趣味指数：★★★★★

实例说明

在过去，都是通过唱片或磁带存储音乐，使用起来非常不便，而且唱片和磁带容易损坏。随着网络的流行，现在听音乐大多都是从网络上下载。音乐文件的格式有很多，如 WMA、RM 和 MP3 等，比较常用的是 MP3 格式的文件。当从网络上将 MP3 格式的音乐下载到本地计算机后，需要通过播放器进行播放，通过本实例可以播放 MP3 格式的音乐。实例运行效果如图 25.4 所示。

图 25.4　多功能 MP3 播放器

关键技术

本实例实现时主要是使用 Windows Media Player 组件实现播放 MP3 文件的功能。

说明：关于 Windows Media Player 组件的详细讲解请参见实例 581 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 MP3Player。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 OpenFileDialog 控件，用来显示打开对话框；添加一个 Timer 组件，用来显示播放状态及控制播放进度条的滚动；添加两个 HScrollBar 控件，分别用来作为播放进度条和音量控制器；添加 7 个 PictureBox 控件，分别用来实现窗体最小化、关闭窗体、拖动窗体、播放 MP3、暂停 MP3、停止 MP3 和调节音量功能；添加一个 Windows Media Player 组件，用于播放 MP3 文件。

（3）程序的主要代码如下。

如果想播放 MP3 文件，可以单击“播放”按钮，弹出选择 MP3 文件的窗口。选择之后，将文件的路径赋给 axWindowsMediaPlayer1 对象的 URL 属性，实现自动播放 MP3 的功能。代码如下：

```
private void pictureBox4_Click(object sender, EventArgs e)                    //打开播放
{
    if (!flag)                                                                //标记是暂停播放器还是选择打开文件
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)                  //选择文件
        {
            axWindowsMediaPlayer1.URL = openFileDialog1.FileName;             //将选择文件的路径赋值给 URL 属性
            m = 1;
            lblSongTitle.Text=" 歌曲名称：" + axWindowsMediaPlayer1.currentMedia.getItemInfo("Title");
        }
    }
    else
    {
        axWindowsMediaPlayer1.Ctlcontrols.play();                             //如果是暂停播放器则调用 play 方法播放
    }
}
```

如果想暂停播放 MP3 文件，可以单击“暂停”按钮。此时，会调用 axWindowsMediaPlayer1 对象的 pause 方法，实现暂停播放 MP3 的功能。代码如下：

```
private void pictureBox5_Click(object sender, EventArgs e)
{
    axWindowsMediaPlayer1.Ctlcontrols.pause();                                //暂停
    flag = true;                                                              //设置 flag 为 true，单击“播放”按钮直接播放
}
```

如果想停止播放 MP3 文件，可以单击“停止”按钮。此时，会调用 axWindowsMediaPlayer1 对象的 stop 方

法，实现停止播放 MP3 的功能。代码如下：

```
private void pictureBox6_Click(object sender, EventArgs e)
{
    axWindowsMediaPlayer1.Ctlcontrols.stop();                    //停止
    flag = false;                                                //设置 flag 为 false，单击“播放”按钮选择新文件
}
```

如果想静音，可以单击“静音”按钮。将 axWindowsMediaPlayer1 对象的 mute 属性设为 true，则静音。设为 false 则取消静音。代码如下：

```
private void pictureBox7_Click(object sender, EventArgs e)                    //静音
{
    if (MM)
    {
        pictureBox7.Image = (Image)Properties.Resources.音量按钮变色;          //设置按钮图标
        axWindowsMediaPlayer1.settings.mute = true;                           //静音
        MM = false;                                                           //MM 设为 false，用于控制按钮的图标切换
    }
    else                                                                      //如果 MM 为 false
    {
        pictureBox7.Image = (Image)Properties.Resources.音量按钮;              //恢复原来的图标
        axWindowsMediaPlayer1.settings.mute = false;                          //取消静音
        MM = true;                                                            //MM 设为 true
    }
}
```

当播放 MP3 文件时，程序提供一个进度条，用于显示歌曲播放的进度，这个功能是在 Timer 组件的 Tick 事件中实现的。在此事件中，除了显示歌曲播放的进度外，还实现了获取当前播放状态的功能。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    int i =(int) axWindowsMediaPlayer1.playState;                             //获取播放器状态
    switch(i)                                                                 //通过 switch 语句判断
    {
        case 1: lblStauts.Text = "状态：停止"; break;                          //1 表示停止
        case 2: lblStauts.Text = "状态：暂停"; break;                          //2 表示暂停
        case 3: lblStauts.Text = "状态：播放"; break;                          //3 表示播放
        case 6: lblStauts.Text = "状态：正在缓冲"; break;                      //6 表示正在缓冲
        case 9: lblStauts.Text = "状态：正在连接"; break;                      //9 表示正在连接
        case 10: lblStauts.Text = "状态：准备就绪"; break;                     //10 表示准备就绪
    }
    lbljindu.Text = axWindowsMediaPlayer1.Ctlcontrols.currentPositionString;  //显示播放的进度信息
    if (m == 1)
    {
        //设置进度条的最大值是音乐的总的播放时间
        hScrollBar1.Maximum = (int)axWindowsMediaPlayer1.currentMedia.duration;
        hScrollBar1.Minimum = 0;                                              //最小值为 0
        //当前值是音乐的当前播放进度
        hScrollBar1.Value = (int)axWindowsMediaPlayer1.Ctlcontrols.currentPosition;
        hScrollBar2.Value=axWindowsMediaPlayer1.settings.volume;              //音量控制
    }
}
```

秘笈心法

心法领悟 584：多媒体播放器介绍。

多媒体播放器，其实是一种能够播放视频文件的程序，它一般理解为多种媒体的综合，现在市场上最流行的多媒体播放器主要有酷我播放器和暴风影音播放器等。多媒体播放器最常见、最主要的功能如下：

（1）可以播放 MP3、WAV、AVI、DAT、RM、RMVB、MP4 等各种格式的音频/视频文件。

（2）暂停、播放、快进和快退播放文件。

（3）设置音频、视频大小。

（4）设置播放文件的播放模式（如单曲播放、顺序播放、循环播放等）。

（5）实现字幕滚动，并根据选择的字幕移动播放进度条。

实例 585 获取 MP3 文件的歌词

光盘位置：光盘\MR\25\585　　中级　　趣味指数：★★★★★

实例说明

在上网时，有很多播放器都可以实现一边听音乐一边看歌词的功能，这样做可以帮助听者更好地欣赏音乐，同时还可以强化对歌词的记忆进而达到背诵，为听者以后独唱歌曲做铺垫。本实例实现了获取歌曲歌词的功能。实例运行效果如图 25.5 所示。

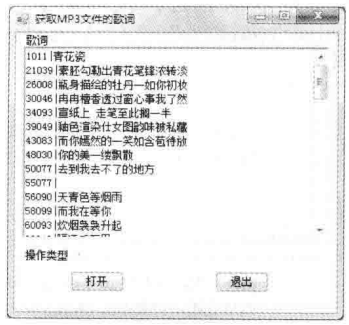


图 25.5　获取 MP3 文件的歌词

关键技术

本实例实现时主要用到了 String 类的 Substring 和 IndexOf 方法，下面分别对它们进行详细讲解。

（1）Substring 方法

该方法表示从此实例检索子字符串。语法格式如下：

```
public string Substring(int startIndex,int length)
```

参数说明

❶ startIndex：表示此实例中子字符串的起始字符位置（从零开始）。

❷ length：表示子字符串中的字符数。

❸ 返回值：一个 String，它等于此实例中从 startIndex 开始的长度为 length 的子字符串，如果 startIndex 等于此实例的长度且 length 为零，则为 Empty。

（2）IndexOf 方法

该方法表示报告 String 或一个或多个字符在此字符串中的第一个匹配项的索引。语法格式如下：

```
public int IndexOf(char value)
```

参数说明

❶ value：表示要查找的 Unicode 字符。

❷ 返回值：表示如果找到该字符，则为 value 的索引位置；如果未找到，则为-1。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ListLyric。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 RichTextBox 控件，用来显示 MP3 文件的歌词；添加两个 Button 控件，分别用来打开 MP3 歌词文件和退出本程序。

（3）程序的主要代码如下：

```
#region  获取 MP3 的歌词
/// <summary>
///获取 MP3 的歌词
/// </summary>
/// <param FileName="string">歌词路径</param>
public string[] LRC_Lyric(string FileName)
{
    ArrayList ArrLyric = new ArrayList();                                    //声明一个使用大小可动态增加的数组对象
    //定义一个公开以文件为主的 Stream，既支持同步读写操作，也支持异步读写操作的对象
    FileStream fs = new FileStream(@FileName,FileMode.Open,FileAccess.Read,FileShare.None);
    StreamReader sr = new StreamReader(fs,System.Text.Encoding.Default);     //声明一个读取数据的对象
    sr.BaseStream.Seek(0,SeekOrigin.Begin);                                  //从指定的起始点开始读取数据
    string Tem_strLine = sr.ReadLine();                                      //读取当前流中的一行数据
    string Tem_Str = "";                                                     //定义一个 string 型临时变量 Tem_Str
    string sp = "";                                                          //定义一个 string 型临时变量 sp
    int Tem_p = 0;                                                           //记录当前“[”的位置
```

```
    int Tem_q = 0;                                                    //记录当前“]”的位置
    string Tem_Time = "";                                             //记录播放时间
    string Tem_Slyrec = "";                                           //记录歌词
    bool Tem_bool = true;                                             //循环条件
    while(Tem_strLine != null)                                        //只要当前流中还存在数据就继续循环
    {
        Tem_bool = true;                                              //设定 Tem_bool 的值为 true
        Tem_Str = Tem_strLine;                                        //为变量 Tem_Str 赋值
        sp = Tem_strLine;                                             //为变量 sp 赋值
        //当“[”字符在此字符串中的第一个匹配项的索引为-1 或当从当前 String 对象的开始和末尾移除所有空白字符后保留的字符串为空值时
        if(sp.IndexOf(Convert.ToChar("[")) == -1 || sp.Trim() == "")
        {
            Tem_strLine = sr.ReadLine();                              //继续从流中读取数据
            continue;                                                 //跳出此层循环
        }
        //从此实例检索子字符串。子字符串从指定的字符位置开始且具有指定的长度
        sp = sp.Substring(sp.IndexOf(Convert.ToChar("[")) + 1,1);
        Tem_Slyrec = Tem_Str.Substring(Tem_Str.LastIndexOf(Convert.ToChar(")")) + 1,Tem_Str.Length - (Tem_Str.LastIndexOf(Convert.ToChar(")")) + 1));
                                                                      //截取歌词中“[”和“]”中的内容
        if(EstimateFig(sp))                                           //当字符串中存在数字时
        {
            while(Tem_bool)                                           //只要 Tem_bool 为真就不断地循环
            {
                Tem_p = Tem_Str.IndexOf(Convert.ToChar("["));         //获取字符“[”的索引
                Tem_q = Tem_Str.IndexOf(Convert.ToChar("]"));         //获取字符“]”的索引
                Tem_Time = Tem_Str.Substring(Tem_p + 1,Tem_q - Tem_p - 1);     //获取当前行的播放时间
                ArrLyric.Add(Tem_Time + "|" + Tem_Slyrec);            //在播放时间和歌词之间添加“|”
                if(Tem_q != Tem_Str.LastIndexOf(Convert.ToChar("]")))     //当没有循环到字符串的结尾时
                    //保存字符串的值
                    Tem_Str = Tem_Str.Substring(Tem_q + 1,Tem_Str.Length - Tem_q - 1);
                else                                                  //当循环到结尾时
                    Tem_bool = false;                                 //设置 Tem_bool 的值为 false
            }
        }
        Tem_strLine = sr.ReadLine();                                  //从当前流中读取一行数据
    }
    sr.Dispose();                                                     //释放由 sr 对象使用的所有资源
    fs.Dispose();                                                     //释放由 fs 对象使用的所有资源
    int Tem_DatetimeUp;                                               //记录前一个时间
    int Tem_DstetimeDown;                                             //记录后一个时间
    string Tem_taxisTime = "";                                        //记录截取后的时间字符串
    string Tem_Transitorily = "";                                     //排序时临时存储的字符串
    string[] ArrayStr = new string[ArrLyric.Count];                   //定义一个 string 型的数组 ArrayStr
    for(int i = 0; i < ArrayStr.Length; i++)                          //循环遍历数组 ArrayStr 中的每一个元素
    {
        Tem_Str = ArrLyric[i].ToString();                             //记录数组中的元素
        //截取歌词的播放时间
        Tem_taxisTime = Tem_Str.Substring(0,Tem_Str.LastIndexOf(Convert.ToChar("|")));
        Tem_taxisTime = BuildMillisecond(Tem_taxisTime);              //保存歌词的播放时间总数
        ArrayStr[i] = Tem_taxisTime + "|" + Tem_Str.Substring(Tem_Str.LastIndexOf(Convert.ToChar("|")) + 1,Tem_Str.Length - Tem_Str.LastIndexOf
(Convert.ToChar("|")) - 1);                                           //将时间和歌词内容赋值给字符数组 ArrayStr
    }
    string iStr = "";                                                 //定义一个 string 型的变量 iStr
    string jStr = "";                                                 //定义一个 string 型的变量 jStr
    for(int i = 0; i < ArrayStr.Length - 2; i++)                      //循环遍历数组 ArrayStr 中的每一行数据
    {
        for(int j = 0; j < ArrayStr.Length - 1 - i; j++)              //循环遍历数组 ArrayStr 中的每一行数据
        {
            iStr = ArrayStr[j];                                       //记录数组 ArrayStr 中的当前数据
            jStr = ArrayStr[j + 1];                                   //记录数组 ArrayStr 中的下一条数据
            //截取当前行中的时间部分
            Tem_taxisTime = iStr.Substring(0,iStr.LastIndexOf(Convert.ToChar("|")));
            Tem_DatetimeUp = Convert.ToInt32(Tem_taxisTime);          //将时间类型转化为整型
            //截取下一行中的时间部分
            Tem_taxisTime = jStr.Substring(0,jStr.LastIndexOf(Convert.ToChar("|")));
            Tem_DstetimeDown = Convert.ToInt32(Tem_taxisTime);        //将时间类型转化为整型
```

```
            if(Tem_DstetimeDown < Tem_DatetimeUp)                    //当下一行的时间小于当前行的时间
            {
                Tem_Transitorily = ArrayStr[j];                      //记录当前行的时间
                ArrayStr[j] = ArrayStr[j + 1];                       //为当前行的时间重新赋值
                ArrayStr[j + 1] = Tem_Transitorily;                  //为下一行的时间赋值
            }
        }
    }
    return ArrayStr;                                                 //返回数组 ArrayStr
}
#endregion
```

秘笈心法

心法领悟 585：使用 StreamReader 类对文本文件进行操作。

以文本模式对文本文件进行读操作一般用 StreamReader 类实现，该类是专门用来读取文本文件的类，它可以从底层 Stream 对象创建 StreamReader 对象的实例，而且也能指定编码规范参数。

实例 586　学校体操定时音乐播放

光盘位置：光盘\MR\25\586　　中级　　趣味指数：★★★★☆

实例说明

学校的广播体操，在第二节课下课时，学生到操场上集合，开始做广播体操，这时广播室内便会响起音乐。其实，这种定时播音的功能是由程序控制的，当系统时间为指定的时刻，程序就会播放音乐。本实例中笔者设计了一个定时播放音乐的程序。实例运行效果如图 25.6 所示。

图 25.6　学校体操定时音乐播放

关键技术

本实例在实现时主要是利用 Timer 组件定时读取系统时间，如果系统时间为指定的时刻，则利用 Windows Media Player 组件播放音乐。

说明：（1）关于 Timer 组件的详细讲解请参见实例 567 中的关键技术。

（2）关于 Windows Media Player 组件的详细讲解请参见实例 581 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ClockingPlayImplement。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 Button 控件，分别用来执行等待播放、显示选择对话框和退出应用程序功能；添加两个 TextBox 控件，分别用来设置播放时间和音乐文件路径；添加一个 Windows Media Player 组件，用来播放媒体文件。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    //当当前的时间与文本框中的内容一致时
    if (DateTime.Now.ToShortTimeString( ) == this.textBox2.Text.Trim( ).ToString( ))
    {
```

```
            this.axWindowsMediaPlayer1.URL = this.textBox1.Text;          //设置音乐文件的播放路径
            this.axWindowsMediaPlayer1.Ctlcontrols.play( );               //播放多媒体文件
            this.timer1.Enabled = false;                                  //关闭 timer1 计时器
            this.timer2.Enabled = true;                                   //启动 timer2 计时器
            this.timer2.Interval = 1000;                                  //设置 timer2 计时器的 Tick 事件间隔为 1 秒
            this.Show( );                                                 //显示该窗体
        }
    }
    private void timer2_Tick(object sender, EventArgs e)
    {
        if (this.axWindowsMediaPlayer1.status != "正在播放")              //当多媒体处于正在播放的状态时
        {
            this.timer2.Enabled = false;                                  //关闭 timer2 计时器
            MessageBox.Show("本次播放已完成，谢谢使用！！！");            //弹出信息提示
        }
    }
```

秘笈心法

心法领悟 586：if 语句的使用技巧。

使用 if 语句时，一定要注意判断的条件必须是一个布尔表达式，例如，if(this.axWindowsMediaPlayer1.status == "正在播放")是正确的，但 if (this.axWindowsMediaPlayer1.status = "正在播放")是错误的，因为“=”是一个赋值运算符，它不能构成布尔表达式。

实例 587　M3U 歌词文件的创建及删除

高级

光盘位置：光盘\MR\25\587

趣味指数：★★★★☆

实例说明

一般情况下，欣赏一首歌曲，只需把它拖入音乐播放器即可；但是，当向音乐播放器中导入大量歌曲时，这种方法就比较麻烦。此时，如果借助于 M3U 文件，那么这个问题就很好解决。本实例将详细讲解有关 M3U 文件的创建及删除。实例运行效果如图 25.7 所示。

图 25.7　M3U 歌词文件的创建及删除

关键技术

本实例在创建 M3U 文件时用到类 File 的 Create 方法，在向 M3U 文件中写入时用到类 StreamWriter 对象的 Write 方法，在删除 M3U 文件时用到类 File 的 Delete 方法。下面对本实例中用到的关键技术进行详细讲解。

（1）Create 方法

该方法用来在指定路径中创建或覆盖文件。声明格式如下：

```
public static FileStream Create(string path)
```

参数说明

❶ path：要创建的文件的路径及名称。

❷ 返回值：一个 FileStream，它提供对 path 中指定的文件的读/写访问。

（2）Write 方法

该方法用来使用与 String.Format 相同的语义写出格式化的字符串。声明格式如下：

```
public virtual void Write(string format,Object arg0)
```

参数说明

❶ format：表示格式化字符串。

❷ arg0：要写入格式化字符串的对象。

（3）Delete 方法

该方法用来删除指定的文件。如果指定的文件不存在，则不引发异常。声明格式如下：

```
public static void Delete(string path)
```

参数说明

path：要删除的文件的名称。

📖 **说明**：程序中用到与文件有关操作的类时，首先需要在命名空间区域添加 System.IO 命名空间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 EstablishAndExpunctionM3U。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 5 个 TextBox 控件，分别用来显示创建提示、MP3 文件名、MP3 文件路径、要写入的 M3U 文件路径和要删除的 M3U 文件路径；添加 6 个 Button 控件，分别用来创建 M3U 文件、打开 MP3 文件、打开要写入的 M3U 文件、在 M3U 文件中写入内容、打开要删除的 M3U 文件和删除 M3U 文件。

（3）程序的主要代码如下。

当文件中不存在 M3U 文件时，单击“创建”按钮时即可创建一个新的 M3U 文件。代码如下：

```
private void found_Click(object sender,EventArgs e)
{
    SaveFileDialog M3UDialog = new SaveFileDialog();                    //声明一个提示用户选择文件的保存位置对象
    //获取或设置当前文件名筛选器字符串，该字符串决定对话框的“另存为文件类型”或“文件类型”中出现的选择内容
    M3UDialog.Filter = "M3U 文件(*.M3U)|*.M3U";
    if(M3UDialog.ShowDialog() == DialogResult.OK)                       //当选定保存位置，单击“保存”按钮时
    {
        m3uCreate(M3UDialog.FileName);                                  //创建一个 M3U 文件
        M3UName.Text = "恭喜你，创建成功！";                             //在文本框中显示提示信息
        M3UName.ForeColor = Color.Red;                                  //设定文本框中文字的颜色
        M3UName.BackColor = Color.Black;                                //设定文本框的背景颜色
        found.Enabled = false;                                          //设定“创建”按钮为不可用状态
    }
    else                                                                //当单击“取消”按钮时
    {
        found.Enabled = true;                                           //设置“创建”按钮为可用状态
        M3UName.Text = "对不起，创建失败！";                             //在文本框中显示提示信息
        M3UName.ForeColor = Color.Red;                                  //设定文本框中字体的颜色
        M3UName.BackColor = Color.Black;                                //设定文本框中的背景颜色
    }
}
```

在上面的代码中用到 m3uCreate 方法，该方法用来创建一个 M3U 文件。代码如下：

```
public void m3uCreate(string FileDir)
{
    //声明一个公开以文件为主的 Stream 对象，既支持同步读写操作，也支持异步读写操作
    FileStream fs;
    Byte[] info;                                                        //声明一个字节数组
    if(File.Exists(FileDir))                                            //如果文件存在，则退出操作
    {
        MessageBox.Show("文件已存在，请重新设置文件名！");               //弹出信息提示
        return;                                                         //直接返回
    }
    else                                                                //如果文件不存在，则创建 File.CreateText 对象
    {
```

```
            fs = File.Create(FileDir);                          //创建 M3U 文件
        }
        info = new UTF8Encoding(true).GetBytes("#EXTM3U");      //定义 M3U 文件的编码方式
        fs.Write(info,0,info.Length);                           //向数据流中写入内容
        fs.Close();                                             //关闭 FileStream 对象
        fs.Dispose();                                           //释放 FileStream 对象所占用的资源
    }
```

当存在 M3U 文件后，这时需要向其中添加歌曲，单击“写入”栏中的“打开”按钮实现歌曲的写入。代码如下：

```
private void openMusic_Click(object sender,EventArgs e)
{
    OpenFileDialog MusicDialog = new OpenFileDialog();          //声明一个提示用户打开文件的对象
    //获取或设置当前文件名筛选器字符串，该字符串决定对话框的“另存为文件类型”或“文件类型”中出现的选择内容
    MusicDialog.Filter = "MP3 文件(*.MP3)|*.MP3|WAV 文件(*.WAV)|*.WAV|WMA 文件(*.WMA)|*.WMA";
    if(MusicDialog.ShowDialog() == DialogResult.OK)             //当选定文件之后单击“打开”按钮时
    {
        musicPath.Text = MusicDialog.FileName;                  //保存打开文件的文件路径
        //保存打开的文件的文件名
        musicName.Text = System.IO.Path.GetFileNameWithoutExtension(musicPath.Text);
    }
}
```

当歌曲显示成功后，单击“M3U 路径”文本框后面的“打开”按钮显示 M3U 文件的路径。代码如下：

```
private void openM3U_Click(object sender,EventArgs e)
{
    M3UPath.Text = ListM3UFile();                               //打开一个 M3U 文件
}
```

在上面的代码中用到 ListM3Ufile 方法，该方法用来显示 M3U 文件。代码如下：

```
private string ListM3UFile()
{
    OpenFileDialog M3U_Dialog = new OpenFileDialog();           //声明一个提示用户打开文件的对象
//获取或设置当前文件名筛选器字符串，该字符串决定对话框的“另存为文件类型”或“文件类型”中出现的选择内容（继承自 FileDialog）
    M3U_Dialog.Filter = "M3U 文件(*.M3U)|*.M3U";
    if(M3U_Dialog.ShowDialog() == DialogResult.OK)              //当选定文件之后单击“打开”按钮时
    {
        temp = M3U_Dialog.FileName;                             //保存所打开的文件名
    }
    return temp;                                                //返回保存的值
}
```

当歌曲和 M3U 文件的路径都存在时，单击“写入”按钮向 M3U 文件中写入内容。代码如下：

```
private void writeIn_Click(object sender,EventArgs e)
{
    m3uWrite(musicName.Text,musicPath.Text,M3UPath.Text);       //向 M3U 文件中写入内容
}
```

上面的代码中用到 m3uWrite 方法，该方法用来向 M3U 文件中写入内容。代码如下：

```
public void m3uWrite(string FName,string FDir,string FileDir)
{
    if(!File.Exists(FileDir))                                   //当不存在文件路径时
    {
        MessageBox.Show("文件不存在！");                        //弹出信息提示
        return;                                                 //直接返回
    }
    //定义实现一个 TextWriter 对象，使其以一种特定的编码向流中写入字符
    StreamWriter ASW = new StreamWriter(FileDir,true,Encoding.Default);
    ASW.WriteLine();                                            //将行结束符写入文本流
    ASW.Write(FDir,Encoding.Default);                           //将数据流中的文件以特定的编码方式写入指定路径中的文件
    ASW.Flush();                                                //清理当前编写器的所有缓冲区，并使所有缓冲数据写入基础流
    ASW.Close();                                                //关闭当前的 StreamWriter 对象和基础流
    ASW.Dispose();                                              //释放由此 TextWriter 对象使用的所有资源
}
```

当有不需要的 M3U 文件时，在“删除”栏中单击“打开”按钮，打开所要删除的文件。代码如下：

```
private void unfold_Click(object sender,EventArgs e)
{
    fileName.Text = ListM3UFile();                              //打开一个 M3U 文件
}
```

当所要删除的文件路径显示在“删除”栏中时，单击“删除”按钮即可删除该文件。代码如下：

```
private void Expunction_Click(object sender,EventArgs e)
{
    m3uDelete(fileName.Text);                                    //删除指定的文件
    temp = null;                                                 //清空变量 temp 中保存的值
    fileName.Clear();                                            //清空文本框中的内容
    MessageBox.Show("祝贺你，删除成功！","信息提示",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
}
```

在上面的代码中用到了 m3uDelete 方法，该方法用来删除文件。代码如下：

```
public void m3uDelete(string FileDir)
{
    if(!File.Exists(FileDir))                                    //当不存在文件路径时
    {
        MessageBox.Show("文件不存在!");                          //弹出信息提示
        return;                                                  //直接返回
    }
    File.Delete(FileDir);                                        //删除指定路径下的文件
}
```

秘笈心法

心法领悟 587：使用 StreamWriter 类的 Write 方法向文件中写入内容。

StreamWriter 类实现了 TextWriter 抽象类，它主要使用一种特定的编码向流中写入字符，该类的 Write 方法用来以指定的编码方式向指定文件中写入内容。

实例 588　异步加载并播放声音文件

光盘位置：光盘\MR\25\588　　中级　　趣味指数：★★★★☆

实例说明

一般情况下，在加载某些文件时，如果都进行同步加载，那么将浪费很多不必要的时间，因为有些文件并不是用户想要的，因此应该实现异步加载，只加载需要的文件。本实例实现了异步加载并播放声音的功能。实例运行效果如图 25.8 所示。

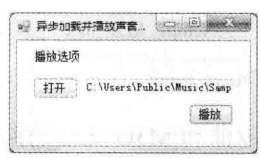


图 25.8　异步加载并播放声音文件

关键技术

本实例实现时主要用到了 SoundPlayer 类，该类用来控制.wav 文件中的声音播放。在播放.wav 文件前，首先定位该文件的位置，此时用到该类的 SoundLocation 属性，该属性的功能是获取或设置要加载的.wav 文件的文件路径或 URL。然后用到该类的 LoadAsync 方法，该方法的作用是使用新线程从流中或 Web 资源中加载.wav 文件。下面对本实例中用到的关键技术进行详细讲解。

（1）SoundLocation 属性

该属性值的类型是 System.String，它的作用是从中加载.wav 文件的文件路径或 URL，如果不存在文件路径，则为 String.Empty。默认为 String.Empty。语法格式如下：

```
public string SoundLocation { get; set; }
```

（2）LoadAsync 方法

该方法用来从流中或者 Web 资源中异步加载文件。语法格式如下：

```
public void LoadAsync()
```

说明：程序中使用 SoundPlayer 类时，首先需要在命名空间区域添加 System.Media 命名空间。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为

AsynchronismLoadPlayVoice。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示所要播放的文件路径；添加两个 Button 控件，分别用来打开.wav 文件和播放音乐文件。

（3）程序的主要代码如下：

```
private void play_Click(object sender,EventArgs e)
{
    try
    {
        SoundPlayer player = new SoundPlayer();                    //声明一个控制 WAV 文件的声音播放文件对象
        player.SoundLocation = path.Text;                          //指定声音文件的路径
        player.LoadAsync();                                        //设置播放的方法
        player.Play();                                             //播放声音文件
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

秘笈心法

心法领悟 588：使用 Exception 类捕捉异常。

所有.NET 中的异常都可被称为 Exception。虽然说异常多种多样，但基本上它们都是 Exception。这个说法可以打一个简单比喻，就像是无论黑猫、白猫、花猫，它们都是“猫”一样。

25.3　媒 体 控 制

实例 589　获取声音设备的名称及 PNPDeviceID　　初级

光盘位置：光盘\MR\25\589　　趣味指数：★★★☆

实例说明

如果要想使电脑的寿命增长，那么就要学会爱护它的部件。不同的部件有不同的爱护方法，但是如何知道声音设备的名称和 PNPDeviceID 呢？本实例可以解决这个问题。实例运行效果如图 25.9 所示。

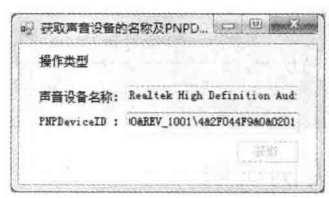


图 25.9　获取声音设备的名称及 PNPDeviceID

关键技术

本实例实现时主要用到了 WMI 管理类中的 ManagementObjectSearcher 类和 ManagementObject 类，下面分别对它们进行详细讲解。

（1）ManagementObjectSearcher 类

该类是基于指定的查询检索管理对象的集合，是用于检索管理信息的较为常用的入口点之一。例如，它可以用于枚举系统中的所有磁盘驱动器、网络适配器、进程及更多管理对象，或者用于查询所有处于活动状态的网络连接以及暂停的服务等。ManagementObjectSearcher 类的 Get 方法用于调用指定的 WMI 查询并返回结果集合。语法格式如下：

```
public ManagementObjectCollection Get()
```

参数说明

返回值：一个 ManagementObjectCollection，它包含匹配指定查询的对象。

（2）ManagementObject 类

该类表示 WMI 实例，本实例中用到该类的 Item 属性，开发人员需要通过[]符获取对该属性值的访问。

ManagementObject 类的 Item 属性的语法格式如下：

```
public Object Item[string propertyName] { get; set; }
```

参数说明

❶ propertyName：相关属性的名称。

❷ 属性值：返回一个 Object 值，该值包含特定类属性的管理对象。

说明：程序中用到 WMI 管理类时，首先需要在命名空间区域添加 System.Management 命名空间，下面再遇到类似的情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchVoiceDeviceName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示声音设备的名称及其 PNPDeviceID；添加一个 Button 控件，用来获取声音设备的名称及其 PNPDeviceID。

（3）程序的主要代码如下：

```
private void snatch_Click(object sender,EventArgs e)
{
    //声明一个用于检索设备管理信息的对象
    ManagementObjectSearcher VoiceDeviceSearcher = new ManagementObjectSearcher("select * from Win32_SoundDevice");
    foreach(ManagementObject VoiceDeviceObject in VoiceDeviceSearcher.Get())          //循环遍历 WMI 实例中的每一个对象
    {
        VoiceDeviceName.Text = VoiceDeviceObject["ProductName"].ToString();           //显示声音设备的名称
        aristotle.Text = VoiceDeviceObject["PNPDeviceID"].ToString();                 //显示声音设备的 PNPDeviceID
    }
    snatch.Enabled = false;                                                           //设置“获取”按钮为不可用状态
}
```

秘笈心法

心法领悟 589：巧用 WMI 管理类。

程序开发中，通过 WMI 管理类可以获取计算机硬件相关的信息，如获取声卡、显卡设备，获取 CPU、硬盘信息等。

实例 590　检测是否安装声卡

光盘位置：光盘\MR\25\590　　初级　　趣味指数：★★★★

实例说明

声卡是一种硬件设备，该设备可以对音频信号进行处理，实现一种数字到模拟和模拟到数字的转换工作，对计算机内的数字音频转换为模拟信号后就能够在音响中播放。许多软件在运行过程中需要播放声音，计算机中不存在声卡不但浪费了资源而且可能出现错误。本实例可以检测计算机中是否安装了声卡，实例运行效果如图 25.10 所示。

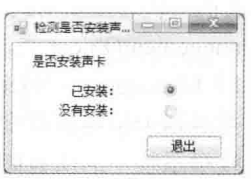


图 25.10　检测是否安装声卡

关键技术

本实例实现时主要用到了 Windows 系统提供的 API 函数 waveOutGetNumDevs，下面对其进行详细讲解。

waveOutGetNumDevs 函数能够返回系统中音频输出设备的数量，该函数被包含在 winmm.dll 库中。声明格式如下：

```
[DllImport("winmm.dll", EntryPoint = "waveOutGetNumDevs")]
public static extern int waveOutGetNumDevs();
```

参数说明

返回值：如果该函数返回 0，表示系统中没有声卡；非 0 表示存在声卡。

说明：程序中使用系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间，下面再遇到类似的情况时将不再提示。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 InspeckWhetherBuildinClamour。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 RadioButton 控件，分别用来显示是否安装声卡；添加一个 Button 控件，用来执行退出应用程序操作。

（3）程序的主要代码如下：

```
[DllImport("winmm.dll", EntryPoint = "waveOutGetNumDevs")]
public static extern int waveOutGetNumDevs();
private void Frm_Main_Load(object sender, EventArgs e)
{
    if (waveOutGetNumDevs() != 0)                    //当安装声卡时
    { this.radioButton1.Select(); }                  //设定 radioButton1 按钮为选定状态
    else                                             //当没有安装声卡时
    { this.radioButton2.Select(); }                  //设定 radioButton2 按钮为选定状态
}
```

秘笈心法

心法领悟 590：RadioButton 的 Select 方法使用。

Select 方法是 Control 基类提供的一个方法，主要用来激活控件，但使用 RadioButton 调用时，它则起到选中单选按钮的作用。

实例 591　打开和关闭 CDROM

中级

光盘位置：光盘\MR\25\591

趣味指数：★★★★

实例说明

CDROM 是光盘驱动器，它是一种光存储设备，只有装配 CDROM 的计算机才能够称为是多媒体计算机。CDROM 存储量大，传输速度快，目前已经得到广泛的应用，其传输媒体是一种以反射光信号的介质存储数据的盘片，称为光盘，目前这种盘片已经比较常见。首先将 CDROM 的盘舱弹出，放入光盘，然后关闭盘舱，这样就可以在计算机中访问光盘中的内容了，其中打开和关闭盘舱主要是使用 CDROM 上的按钮来操作的，但是如果经常使用按钮，则非常容易损坏。本实例使用软件来控制 CDROM 盘舱的弹出和关闭。运行程序，单击“打开 CD_ROM”按钮，即可弹出盘舱；单击“关闭 CD_ROM”按钮，即可关闭盘舱。实例运行效果如图 25.11 所示。

图 25.11　打开和关闭 CDROM

关键技术

本实例实现时主要用到了 Windows 提供的 API 函数 mciSendString，下面对其进行详细讲解。

mciSendString 函数用来向媒体控制接口设备发送命令。声明格式如下：

```
[DllImport("winmm.dll", EntryPoint = "mciSendString")]
public static extern int mciSendString(string lpstrCommand, string lpstrReturnString, System.UInt16 uReturnLength, System.IntPtr HwndCallback);
```

mciSendString 函数中的参数说明如表 25.3 所示。

表 25.3　mciSendString 函数中的参数说明

参　数	说　明
lpstrCommand	表示指令字符串
lpstrReturnString	表示返回信息字符串
uReturnLength	表示返回信息字符长度
HwndCallback	表示回调窗口的句柄
返回值	表示函数的执行状态

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 OpenUpOrClosureClamourDevice。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Button 控件，分别用来执行打开和关闭 CDROM 操作。

（3）程序的主要代码如下：

```
[DllImport("winmm.dll", EntryPoint = "mciSendString")]
public static extern int mciSendString(string lpstrCommand, string lpstrReturnString, System.UInt16 uReturnLength, System.IntPtr HwndCallback);
public Frm_Main()
{
    InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
    mciSendString("Set cdaudio door open wait", "", 0, this.Handle);          //打开 CD_ROM
}
private void button2_Click(object sender, EventArgs e)
{
    mciSendString("Set cdaudio door closed wait", "", 0, this.Handle);        //关闭 CD_ROM
}
```

秘笈心法

心法领悟 591：有效使用系统 API 函数。

系统 API 函数中封装了很多常用的功能，在 C#程序中可以通过重写来方便地使用它，从而快速实现指定的功能。C#程序中重写系统 API 函数时，首先需要在命名空间区域添加 System.Runtime.InteropServices 命名空间。

实例 592　控制 PC 喇叭发声

光盘位置：光盘\MR\25\592　　中级　　趣味指数：★★★★☆

实例说明

PC 喇叭是 PC 上的一种发声设备，该设备由计算机中的硬件控制，需要发声时使用脉冲控制声音的声调和延时。软件中经常需要提示音来提醒用户的操作，目前通常是用声卡来完成的，而一些 PC 上没有安装声卡，这样就用到 PC 喇叭发出声音来提醒用户。本实例中设计了一个控制 PC 喇叭发声的软件，用户通过按键便可使 PC 喇叭发出不同的声音。实例运行效果如图 25.12 所示。

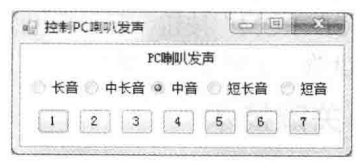


图 25.12　控制 PC 喇叭发声

关键技术

本实例在实现控制 PC 喇叭发声功能时，主要用到了 Console 类的 Beep 方法，下面对其进行详细讲解。

Console 类表示控制台应用程序的标准输入流、输出流和错误流，其 Beep 方法用来通过控制台扬声器播放

提示音，该方法有两种重载形式，分别如下：

```
[HostProtectionAttribute(SecurityAction.LinkDemand, UI = true)]
public static void Beep()
[HostProtectionAttribute(SecurityAction.LinkDemand, UI = true)]
public static void Beep(int frequency,int duration)
```

参数说明

❶ frequency：提示音的频率，介于 37～32767 赫兹之间。

❷ duration：提示音的持续时间，以毫秒为单位。

📖 **说明**：五线谱中的音符分别与 220、247、262、294、330、349 和 392 对应。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 DominateHornVocalization。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 5 个 RadioButton 控件，分别用来表示长音、中长音、中音、短长音和短音；添加 7 个 Button 控件，分别用来表示五线谱中的 7 个音符。

（3）程序的主要代码如下：

```
protected enum Tone                                                    //枚举五线谱
{
    REST = 0,
    A = 220,
    B = 247,
    C = 262,
    D = 294,
    E = 330,
    F = 349,
    G = 392,
}
protected enum Duration                                                //枚举发音时间
{
    WHOLE = 1200,
    HALF = WHOLE / 2,
    QUARTER = HALF / 2,
    EIGHTH = QUARTER / 2,
    SIXTEENTH = EIGHTH / 2,
}
protected struct Note
{
    Tone toneVal;                                                      //初始化一个 Tone 对象
    Duration durVal;                                                   //初始化一个 Duration 对象
    public Note(Tone frequency, Duration time)                         //定义一个 Note 方法
    {
        toneVal = frequency;                                           //为变量 toneVal 赋值
        durVal = time;                                                 //为变量 durVal 赋值
    }
    public Tone NoteTone { get { return toneVal; } set { toneVal = value; } }          //定义一个 Tone 类型的属性
    public Duration NoteDuration { get { return durVal; } set { durVal = value; } }    //定义一个 Duration 类型的属性
}
protected void Play(Note tune)
{
    if (tune.NoteTone == Tone.REST)                                    //当没有选择 RadioButton 按钮时
        Thread.Sleep((int)tune.NoteDuration);                          //将当前线程挂起指定的时间
    else                                                               //当选定某一个 RadioButton 按钮时
        //通过控制台扬声器播放具有指定频率和持续时间的声音
        Console.Beep((int)tune.NoteTone, (int)tune.NoteDuration);
}
```

秘笈心法

心法领悟 592：构造函数的使用。

C#中的构造函数是一个特殊的函数，它的名称与类名相同，每个类都有一个默认的空构造函数，一般的初

始化数据操作都在构造函数中进行。

实例 593 获取显示设备的名称及 PNPDeviceID

光盘位置：光盘\MR\25\593　　初级　　趣味指数：★★★☆

实例说明

一般情况下，当需要知道计算机的显示设备名称和 PNPDeviceID 时，如果按照原有的步骤既繁琐又费力，那么通过对本实例的学习，获取显示设备的名称和 PNPDeviceID 将是很容易的事。实例运行效果如图 25.13 所示。

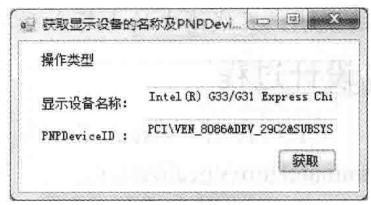


图 25.13　获取显示设备的名称及 PNPDeviceID

关键技术

本实例实现时，首先使用 WMI 管理类中的 ManagementObjectSearcher 类生成一个设备管理信息对象，然后使用 ManagementObject 类遍历设备管理信息对象，从而得到显示设备的名称及 PNPDeviceID。

说明：关于 ManagementObjectSearcher 类和 ManagementObject 类的详细讲解请参见实例 589 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchDisplayDeviceName。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 TextBox 控件，分别用来显示显示设备的名称及其 PNPDeviceID；添加一个 Button 控件，用来获取显示设备的名称及其 PNPDeviceID。

（3）程序的主要代码如下：

```
private void snatch_Click(object sender,EventArgs e)
{
    //声明一个用于检索设备管理信息的对象
    ManagementObjectSearcher FlashDevice = new ManagementObjectSearcher("select * from win32_VideoController");
    foreach(ManagementObject FlashDeviceObject in FlashDevice.Get())          //循环遍历 WMI 实例中的每一个对象
    {
        printName.Text = FlashDeviceObject["name"].ToString();                //在文本框中显示显示设备的名称
        aristotle.Text = FlashDeviceObject["PNPDeviceID"].ToString();         //在文本框中显示显示设备的 PNPDeviceID
    }
}
```

秘笈心法

心法领悟 593：显示设备信息有什么用处？

在实现软件的注册功能时，可以获取显示设备的相关信息，并从中截取部分内容作为注册功能中用到的机器码，以防止软件被破解。

实例 594 获取显示设备的最大、最小及当前刷新率

光盘位置：光盘\MR\25\594　　初级　　趣味指数：★★★☆

实例说明

一般情况下，不同的屏幕大小适合不同的刷新率，只有当两者相协调时，画面质量才能达到最佳。本实例通过代码实现了对当前系统的最大、最小以及当前刷新率的获取。实例运行效果如图 25.14 所示。

关键技术

本实例实现时，首先使用 WMI 管理类中的 ManagementObjectSearcher 类生成一个设备管理信息对象，然后使用 ManagementObject 类遍历设备管理信息对象，从而得到显示设备的最大、最小及当前刷新率。

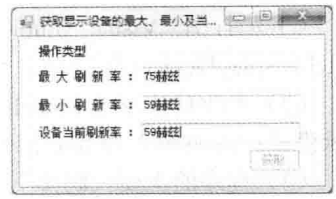


图 25.14　获取显示设备的最大、最小及当前刷新率

说明：关于 ManagementObjectSearcher 类和 ManagementObject 类的详细讲解请参见实例 589 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchDisplayDeviceRefresh。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加 3 个 TextBox 控件，分别用来显示显示设备的最大、最小及当前刷新率；添加一个 Button 控件，用来获取显示设备的最大、最小及当前刷新率。

（3）程序的主要代码如下：

```
private void snatch_Click(object sender,EventArgs e)
{
    //声明一个用于检索设备管理信息的对象
    ManagementObjectSearcher RefreshSearcher = new ManagementObjectSearcher("select * from Win32_VideoController");
    foreach(ManagementObject RefreshObject in RefreshSearcher.Get())                    //循环遍历 WMI 实例中的每一个对象
    {
        maxRefresh.Text = RefreshObject["MaxRefreshRate"].ToString() + "赫兹";          //显示最大刷新率
        minRefresh.Text = RefreshObject["MinRefreshRate"].ToString() + "赫兹";          //显示最小刷新率
        nowRefresh.Text = RefreshObject["CurrentRefreshRate"].ToString() + "赫兹";      //显示当前刷新率
    }
    snatch.Enabled = false;                                                             //设置“获取”按钮为不可用状态
}
```

秘笈心法

心法领悟 594：如何设置按钮的不可用状态？

按钮的不可用状态可以通过设置其 Enabled 属性为 false 来实现，该属性用来获取或设置一个值，该值指示控件是否可以对用户交互作出响应。

实例 595　获取显示设备的当前显示模式

光盘位置：光盘\MR\25\595　　初级　　趣味指数：★★★☆

实例说明

对于电脑爱好者来说，在进行不同的工作时，电脑屏幕显示的图片质量不一样，这是由于改变了显示器的显示模式所造成的，本实例将讲解如何获取显示设备的当前显示模式。实例运行效果如图 25.15 所示。

图 25.15　获取显示设备的当前显示模式

关键技术

本实例实现时，首先使用 WMI 管理类中的 ManagementObjectSearcher 类生成一个设备管理信息对象，然后使用 ManagementObject 类遍历设备管理信息对象，从而得到显示设备的当前显示模式。

说明：关于 ManagementObjectSearcher 类和 ManagementObject 类的详细讲解请参见实例 589 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SnatchDisplayDeviceListMode。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来显示当前显示模式；添加一个 Button 控件，用来获取显示设备的当前显示模式。

（3）程序的主要代码如下：

```
private void snatch_Click(object sender,EventArgs e)
{
    //声明一个用于检索设备管理信息的对象
    ManagementObjectSearcher ListModeSearcher = new ManagementObjectSearcher("select * from Win32_VideoController");
    foreach(ManagementObject ListModeObject in ListModeSearcher.Get())                    //循环遍历 WMI 实例中的每一个对象
    {
        unhideMode.Text = ListModeObject["VideoModeDescription"].ToString();              //显示设备的当前显示模式
    }
    snatch.Enabled = false;                                                               //设定“获取”按钮为不可用状态
}
```

秘笈心法

心法领悟 595：巧用 foreach 语句。

foreach 语句专门用来遍历数组和集合的元素，foreach 的工作原理是逐个枚举出数组或集合中的每一个元素，每枚举一个元素就执行一次语句块中的内容，在使用 foreach 遍历的过程中要注意不可以更改遍历中的集合。

实例 596　收听网络电台

光盘位置：光盘\MR\25\596　　中级　　趣味指数：★★★★☆

实例说明

现在网络已普及到千家万户，使人们的生活变得多姿多彩。在上网生活中，可以听音乐、看电影、和亲朋好友聊天，还有重要的一项娱乐节目就是收听网络电台，本实例通过程序的形式收听网络电台。实例运行效果如图 25.16 所示。

图 25.16　收听网络电台

关键技术

本实例在实现网络电台收听功能时，主要用到 Windows 系统中的 Windows Media Player 组件。在使用 Windows Media Player 组件播放网络电台时，只需将其 URL 属性设置为网络电台地址即可。

说明：关于 Windows Media Player 组件的详细讲解请参见实例 581 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为

ListenToNetWorkStation。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用来输入网络电台地址；添加一个 Button 控件，用来执行收听网络电台操作；添加一个 Windows Media Player 组件，用来播放网络电台。

（3）程序的主要代码如下：

```
private void snatch_Click(object sender,EventArgs e)
{
    try
    {
        this.axWindowsMediaPlayer1.URL = path.Text;                    //设置 Windows Media Player 的 URL
    }
    catch(Exception ex)                                                //捕获异常
    {
        MessageBox.Show(ex.Message);                                   //显示异常信息
    }
}
```

秘笈心法

心法领悟 596：使用 try…catch 语句捕获异常。

在大部分情况下，开发人员不希望异常状况发生导致程序结束，因此，可以使用 try…catch 程序语句块捕捉程序中的 Exception 对象，再使用自定义的程序逻辑处理异常状况。如果有需要，用户也可以使用多重 try…catch 语句块（如一个 try 块后跟一个或多个 catch 子句构成）。

25.4　多媒体应用

实例 597	制作开机祝福程序 光盘位置：光盘\MR\25\597	中级 趣味指数：★★★★★

实例说明

运行 Windows 的应用程序有很多种方法，双击程序的快捷方式、双击程序的可执行文件、在命令行模式中输入程序文件名、双击程序的关联文件等都可以运行目标程序，然而 Windows 中有许多程序需要在开机后自动运行。当然这样的程序最好不要很多，因为太多会影响 Windows 启动的速度。本实例将利用 C#语言制作一个开机祝福程序。实例运行效果如图 25.17 所示。

图 25.17　制作开机祝福程序

关键技术

本实例实现的关键是如何在系统启动时自动运行程序，下面就对其进行详细讲解。

系统启动时自动运行程序有很多种方法，最为常用的一种方法就是利用系统注册表。在注册表中的 HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN 键下可以添加 Windows

启动后自动运行的程序，这种方法需要对注册表进行操作。C#中可以使用 Registry 类来实现，使用该类可以方便地对注册表进行操作，该类中的 SetValue 方法能够向注册表中添加或修改一个字符串值。语法格式如下：

```
public static void SetValue (string keyName, string valueName,Object value)
```

参数说明

❶ keyName：以有效注册表根（如 HKEY_CURRENT_USER）开头的键的完整注册表路径。

❷ valueName：名称/值对的名称。

❸ value：要存储的值。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 BootStrapBeatitude。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加两个 Label 控件，分别用来显示当前日期和时间；添加两个 Button 控件，分别用来显示信息和执行退出程序操作。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    GraphicsPath gp = new GraphicsPath();                                    //初始化一个表示一系列相互连接的直线和曲线的类
    //初始化一个矩形操作区域
    Rectangle rect = new Rectangle(new Point(0, 0), new Size(this.Width, this.Height));
    gp.AddEllipse(rect);                                                     //向当前指定的路径下添加一个椭圆
    this.Region = new Region(gp);                                            //设置与此控件关联的窗口区域
    this.label3.Text = DateTime.Now.ToShortDateString();                     //在 label3 控件中显示当前的日期
    this.label5.Text = DateTime.Now.ToShortTimeString();                     //在 label5 中显示当前的时间
    GraphicsPath gpstirng = new GraphicsPath();                              //初始化一个表示一系列相互连接的直线和曲线的类
    FontFamily family = new FontFamily("宋体");                              //初始化一个字体样式类
    int fontStyle = (int)FontStyle.Italic;                                   //设置字体的样式类型
    int emSize = 25;                                                         //初始化一个 emSize 变量
    Point origin = new Point(0, 0);                                          //初始化一个有序实数对的实例
    StringFormat format = StringFormat.GenericDefault;                       //创建一个包含文本布局信息的对象
    gpstirng.AddString("开开心心每一天", family, fontStyle, emSize, origin, format);  //向指定的路径添加字符串
    this.button1.Region = new Region(gpstirng);                              //设置与 button1 控件关联的窗口区域
    Registry.LocalMachine.CreateSubKey(@"SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\RUN").SetValue("MyAngel",
Application.StartupPath + "\\Ex05_13.exe", RegistryValueKind.String);        //打开注册表中的现有项并设置其中的键值类型
}
```

秘笈心法

心法领悟 597：注册表的使用。

注册表的层次结构非常类似于文件系统，它记录了用户账号、服务器硬件以及应用程序的设置信息等。同 INI 文件相比，注册表可以控制的数据更多，而且不仅限于处理字符串类型的数据。注册表也包含了一些系统配置的信息，这些信息根据操作系统的不同而不同。用户选择“开始”/“运行”命令，在弹出对话框的“打开”文本框中输入“regedit”后，单击“确定”按钮即可打开注册表编辑器。

实例 598　电子相册屏幕保护程序

光盘位置：光盘\MR\25\598　　中级　　趣味指数：★★★★☆

实例说明

本实例实现一个具有个性化的电子相册屏幕保护程序。屏幕保护是由系统启动的、特殊的可执行文件，所以要将本实例应用程序的 exe 扩展名改为 src，并将程序放到系统的 system32 文件夹下（与程序相关的文件也要求放到系统的 system32 文件夹下，如图片），然后通过设置显示属性选中这个屏幕保护程序。实例运行效果如图 25.18 所示。

图 25.18　电子相册屏幕保护程序

关键技术

本实例实现时主要用到了 Graphics 对象的 DrawImage 方法，下面对其进行详细讲解。

Graphics 对象的 DrawImage 方法用来在指定位置并且按照指定的大小绘制指定的 Image 图像，该方法有多种重载形式，本实例中用到的重载形式如下：

```
public void DrawImage(Image image,int x,int y)
```

参数说明

❶ image：要绘制的 Image 图像。

❷ x：所绘制图像的左上角的 x 坐标。

❸ y：所绘制图像的左上角的 y 坐标。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 ElectronAlbumShroudAegis。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并设置其 FormBorderStyle 属性为 None。在 Frm_Main 窗体中添加一个 Timer 计时器组件，用来控制屏保上图片变换的频率。

（3）程序的主要代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    strpath = Application.StartupPath + "\\Image";                    //获取可执行文件的路径
    Cursor.Hide();                                                    //隐藏当前鼠标指针
    this.timer1.Enabled = true;                                       //启动 timer1 计时器
    width = this.Width;                                               //保存当前窗体的宽度
    heigh = this.Height;                                              //保存当前窗体的高度
    drowImage();                                                      //绘制图片
}
private void timer1_Tick(object sender, EventArgs e)
{
    this.timer1.Interval = new Random().Next(500, 2000);              //设置 Timer 控件的 Interval 属性来控制图片更改的频率
    drowImage();                                                      //绘制图片
}
private void drowImage()
{
    Graphics myGraphics = this.CreateGraphics();                      //创建一个 GDI+绘图图面类
    myGraphics.Clear(Color.Black);                                    //清空原有的绘图图面并以指定的背景色填充
    //创建一个 GDI+位图对象
    Bitmap myBitmap = new Bitmap(strpath + "\\" + new Random().Next(1, 5).ToString() + ".jpg")
    //绘制图片
    myGraphics.DrawImage(myBitmap, new Random().Next(0, width - myBitmap.Width), new Random().Next(0, heigh - myBitmap.Height));
}
```

秘笈心法

心法领悟 598：如何清空窗体的背景色？

清空窗体背景色时，可以使用 Graphics 类的 Clear 方法来实现。例如，本实例中清空默认窗体的背景色的实现代码如下：

```
Graphics myGraphics = this.CreateGraphics();                    //创建一个 GDI+绘图图面类
myGraphics.Clear(Color.Black);                                  //清空原有的绘图图面并以指定的背景色填充
```

实例 599　歌曲播放屏幕保护程序

光盘位置：光盘\MR\25\599　　中级　　趣味指数：★★★★☆

实例说明

本实例实现一个具有个性化的歌曲播放屏幕保护程序。屏幕保护是由系统启动的、特殊的可执行文件，所以要将本实例应用程序的 exe 扩展名改为 src，并将程序放到系统的 system32 文件夹下（与程序相关的文件也要求放到系统文件夹下，如音乐文件），然后通过设置显示属性选中这个屏幕保护程序。实例运行效果如图 25.19 所示。

图 25.19　歌曲播放屏幕保护程序

关键技术

本实例实现的关键是如何播放屏幕保护歌曲，这里通过调用 Windows Media Player 组件来实现。

说明：关于 Windows Media Player 组件的详细讲解请参见实例 581 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SongPlayShroudAegis。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，并设置其 FormBorderStyle 属性为 None。在 Frm_Main 窗体中添加一个 Timer 计时器组件，用来控制屏保上文字位置的变换频率；添加一个 Windows Media Player 组件，用来播放屏幕保护歌曲。

（3）程序的主要代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        drowInfo();                                             //绘制屏保信息
        this.timer1.Interval = new Random().Next(800, 1600);    //设置 timer1 的时间间隔
        string strname = new Random().Next(1, 4).ToString();    //记录随机数
        //当播放器处于空或者已停止时
        if (this.axWindowsMediaPlayer1.status == "" || this.axWindowsMediaPlayer1.status == "已停止")
        {
            string strUrl = strpath + "\\" + strname + ".mp3";  //获取多媒体文件所处的路径
            this.axWindowsMediaPlayer1.URL = strUrl;            //设置播放文件的 URL
        }
    }
    catch (Exception ex)                                        //捕获异常
    {
        timer1.Enabled = false;                                 //关闭计时器 timer1
        MessageBox.Show(ex.Message);                            //弹出异常信息提示
    }
}
```

```
private void drowInfo()
{
    Graphics myGraphics = this.CreateGraphics();                    //创建一个 GDI+绘图对象
    myGraphics.Clear(Color.Black);                                  //清空原有绘图面并以指定的背景色填充
    string strinfo = "歌曲播放屏幕保护";                              //保存显示信息
    int x = new Random().Next(0, width - 250);                      //设置显示地点的 X 坐标
    int y = new Random().Next(50, heigh - 20);                      //设置显示地点的 Y 坐标
    myGraphics.DrawString(strinfo, new Font("华文形楷", 72, FontStyle.Bold), new SolidBrush(Color.FromArgb(new Random().Next(50, 255), new
Random().Next(70, 255), new Random().Next(36, 255))), x, y);        //绘制内容
}
```

秘笈心法

心法领悟 599：Exception 类的含义。

Exception 类表示在应用程序执行过程中发生的错误，该类是所有异常的基类。当发生错误时，系统或当前执行的应用程序将通过引发包含错误信息的异常来报告该错误。引发异常之后，将由应用程序或默认异常处理程序处理该异常。

实例 600	开发一个语音计算器	高级
	光盘位置：光盘\MR\25\600	趣味指数：★★★★★

实例说明

在使用计算器对数字进行计算时，有时会输入错误的数字进行计算，但操作者并不知道，这样会给工作带来不必要的麻烦。本实例将制作一个带语音的计算器，运行本程序，效果如图 25.20 所示。当用户在语音计算器的主窗体上单击鼠标右键时，可以在弹出的快捷菜单中选择“设置声音”命令，弹出“语音设置”对话框，如图 25.21 所示。在该对话框中可以对各个按键的声音进行设置。

图 25.20　语音计算器

图 25.21　“语音设置”对话框

关键技术

本实例主要使用 API 函数 mciSendString 来实现语音文件的播放与关闭。

说明：关于 mciSendString 函数的详细讲解请参见实例 591 中的关键技术。

设计过程

（1）打开 Visual Studio 2012 开发环境，新建一个 Windows 窗体应用程序，并将其命名为 SoundCalculator。

（2）更改默认窗体 Form1 的 Name 属性为 Frm_Main，在该窗体中添加一个 TextBox 控件，用于显示输入的数字及运算结果；添加 23 个 Button 控件，用来设置计算器中的按键。

（3）在项目中添加一个新的 Windows 窗体，并将其命名为 Frm_Set.cs，该窗体用来对各个按键的声音进行设置。

（4）程序的主要代码如下：

```
public void sound(string FileName)
{
    if (FileName == null)                                          //如果文件为空
        return;                                                    //退出操作
    if (FileName.IndexOf(" ") == -1)                               //如果路径中没有空格
    {
        if (tem_FileName.Length!=0)                                //如果有播放的文件
            mciSendString("close " + tem_FileName, null, 0, 0);    //关闭当前文件的播放
        n=mciSendString("open " + FileName , null, 0 , 0);         //打开要播放的文件
        n=mciSendString("play " + FileName, null, 0, 0);           //播放当前文件
        tem_FileName = FileName;                                   //记录播放文件的路径
    }
}
private void pict_Back_Click(object sender, EventArgs e)
{
    tem_Value = ((PictureBox)sender).AccessibleName;               //获取当前按钮的标识
    switch (tem_Value)
    {
        case "0": num(tem_Value); sound(VoxPath[0]); break;        //实现按钮的语音功能
        case "1": num(tem_Value); sound(VoxPath[1]); break;
        case "2": num(tem_Value); sound(VoxPath[2]); break;
        case "3": num(tem_Value); sound(VoxPath[3]); break;
        case "4": num(tem_Value); sound(VoxPath[4]); break;
        case "5": num(tem_Value); sound(VoxPath[5]); break;
        case "6": num(tem_Value); sound(VoxPath[6]); break;
        case "7": num(tem_Value); sound(VoxPath[7]); break;
        case "8": num(tem_Value); sound(VoxPath[8]); break;
        case "9": num(tem_Value); sound(VoxPath[9]); break;
        case "+": js(tem_Value); sound(VoxPath[10]); break;
        case "-": js(tem_Value); sound(VoxPath[11]); break;
        case "*": js(tem_Value); sound(VoxPath[12]); break;
        case "/": js(tem_Value); sound(VoxPath[13]); break;
        case "=": js(tem_Value); sound(VoxPath[14]); break;
        case "C": Aclose(); sound(VoxPath[15]); break;
        case "CE": ce(); sound(VoxPath[16]); break;
        case "Back": backspace(); sound(VoxPath[17]); break;
        case "%": bai(); sound(VoxPath[18]); break;
        case "X": ji(); sound(VoxPath[19]); break;
        case ".": dian(); sound(VoxPath[20]); break;
        case "+-":
            {
                zf();
                if (Convert.ToInt32(textBox1.Text) > 0)            //如果当前为正数
                    sound(VoxPath[21]);                            //实现正数发音
                else
                    sound(VoxPath[22]);                            //实现负数发音
                break;
            }
        case "Sqrt": kfang(); sound(VoxPath[23]); break;
    }
    textBox1.Select(textBox1.Text.Length, 0);
}
```

秘笈心法

心法领悟 600：switch 语句的使用注意事项。

使用 switch 语句时需要注意以下 3 点：

（1）每个 case 后面的“常量表达式”的值必须是与“表达式”的类型相同的一个常量，不能是变量。

（2）同一个 switch 语句中的两个或多个 case 标签中指定同一个常数值，会导致编译出错。

（3）一个 switch 语句中最多只能有一个 default 标签，并且每一个标签后面都需要一个 break 语句跳过 switch 语句的其他标签。

“软件工程师开发大系”简介

“软件工程师开发大系”是一套软件开发“实例类”图书。它分为7个方向，每个方向又分为“基础卷”和“提高卷”两册，每册约600个实例，两册共有约1200个实例，内容极为丰富，从基础学习到高级应用分门别类包罗万象。对每个实例按照实例说明、关键技术、设计过程（代码实现）、详尽注释、秘笈心法的顺序进行了详尽分析。大多数实例及源代码来自现实开发中，很多可以移植应用。配书光盘给出了实例完整的源代码，部分实例还给出了讲解视频。

“软件工程师开发大系”丛书适合工作应用速查、项目开发参考、学习实战练习使用。“软件工程师开发大系”丛书具体品种如下：

- C#开发实例大全（基础卷）
- C#开发实例大全（提高卷）

- Java开发实例大全（基础卷）
- Java开发实例大全（提高卷）

- Java Web开发实例大全（基础卷）
- Java Web开发实例大全（提高卷）

- PHP开发实例大全（基础卷）
- PHP开发实例大全（提高卷）

- ASP.NET开发实例大全（基础卷）
- ASP.NET开发实例大全（提高卷）

- Visual C++开发实例大全（基础卷）
- Visual C++开发实例大全（提高卷）

- Visual Basic开发实例大全（基础卷）
- Visual Basic开发实例大全（提高卷）

清华大学出版社数字出版网站
WQBook 书文局泉
www.wqbook.com

ISBN 978-7-302-38438-0
9 787302 384380 >
定价：128.00元

（附光盘1张，含实例源代码、部分实例视频、实例配置说明等）

[General Information]
书名=C#开发实例大全　基础卷
作者=软件开发技术联盟编著
页数=888
SS号=13913763
DX号=
出版日期=2016.01
出版社=清华大学出版社