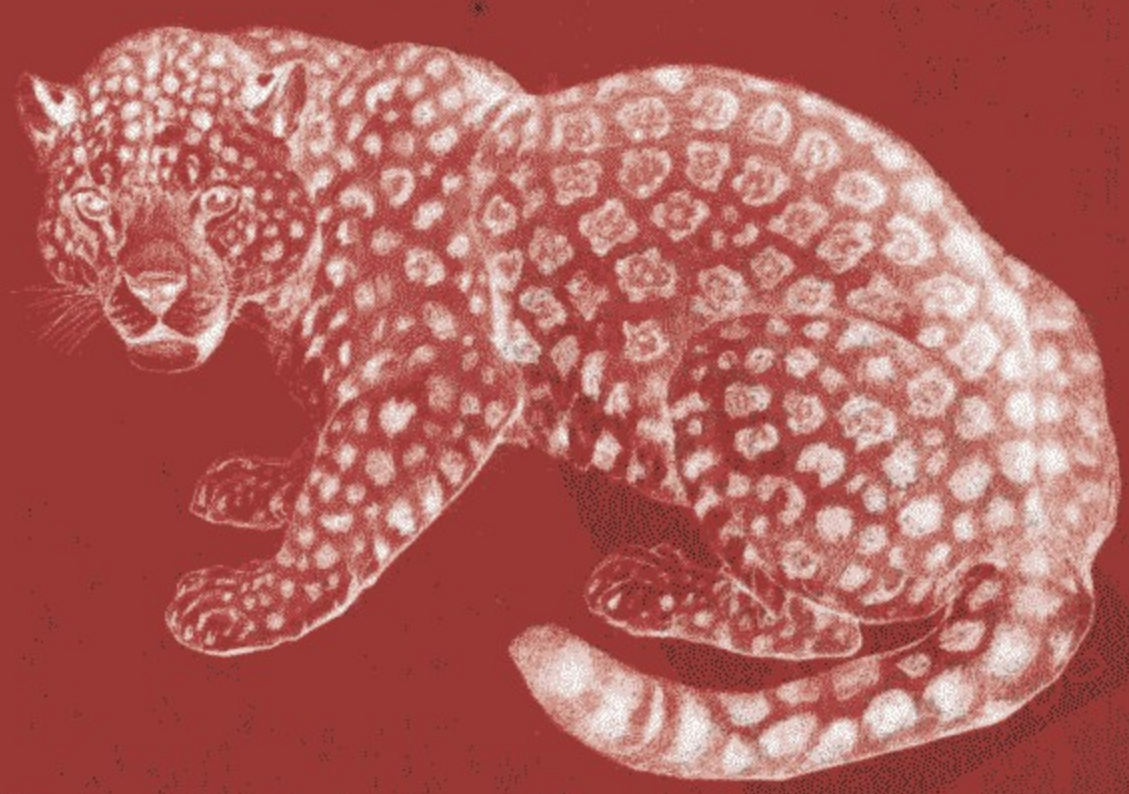
软·件·工·程·师·典·藏

C#

开发

技术大全

明日科技 编著



超量代码·实用工具

人民邮电出版社
POSTS & TELECOM PRESS

软·件·工·程·师·典·藏

一部久享盛誉的程序开发技术大全

技术全面，几乎囊括了使用C#进行程序开发的全部知识

从易到难，逐步深入，循序渐进，适合各个层次的读者学习

层次分明，讲解细致，轻松学习

选择的实例经典实用，有助于程序开发

新技术和热点技术的应用，使本书更贴近实际开发应用

提供编程社区（www.mrbccd.com），远程互动编程，解决开发难题，开拓开发视野

光盘包含所有范例源代码及速查工具

超值配套服务

服务网站：www.mingribook.com

服务社区：bbs.mrbccd.com

QQ：100310063 1026560213

TQ：登录www.mingribook.com直接与客服交流

服务热线：400-675-1066

0431-84978981 /84978982

装帧设计 董志桢

分类建议：计算机/程序设计/C#

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-25870-0

9 787115 258700 >

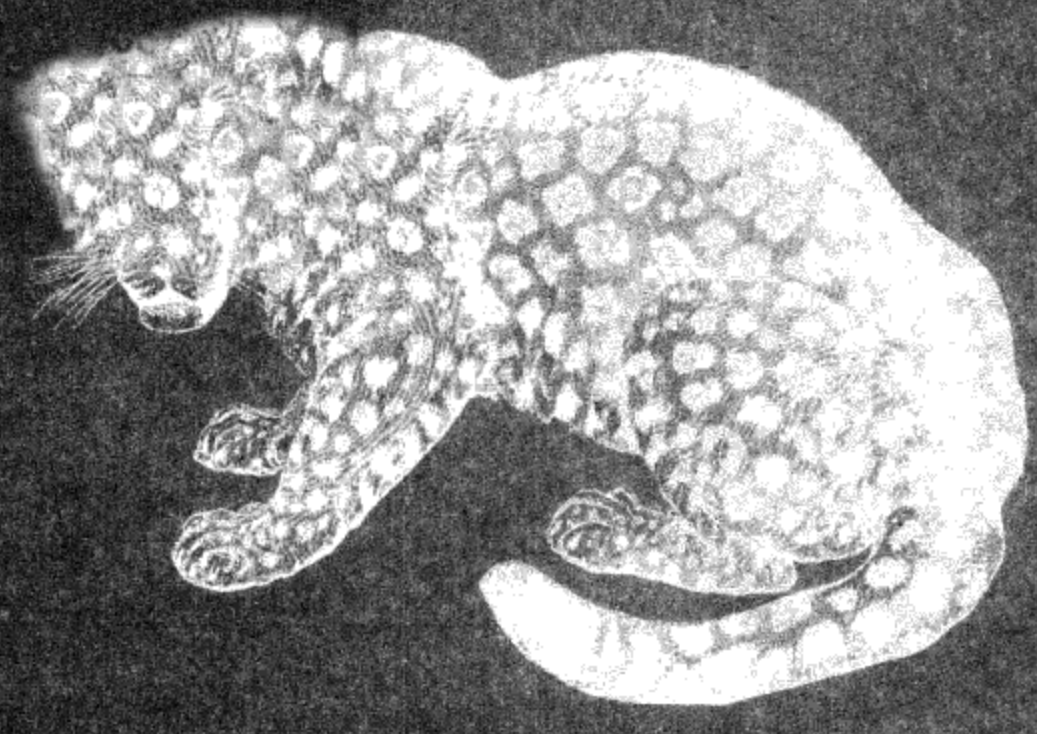
ISBN 978-7-115-25870-0

定价：89.00 元（附光盘）

软·件·工·程·师·典·藏

C#开发技术大全

■ 明日科技 编著



人民邮电出版社
北京

图书在版编目（CIP）数据

C#开发技术大全 / 明日科技编著. -- 北京 : 人民邮电出版社, 2011.11
ISBN 978-7-115-25870-0

Ⅰ. ①C… Ⅱ. ①明… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第141159号

内 容 提 要

本书是一本C#综合开发参考手册，书中几乎囊括了使用C#进行程序开发的全部知识，同时，本书在讲解时结合了大量实用而又有代表性的示例和典型应用。

全书分为7篇，共31章，全面讲解了C#语言基础、Windows窗体设计、ASP.NET网站开发、数据库开发、文件技术、图形图像与多媒体、报表与打印技术。书中讲解的各种技术既相互独立又相互联系，可以逐步引导读者深入学习并掌握C#的编程知识、方法和编程技巧。

本书附有配套光盘。光盘提供了书中示例和典型应用实例的全部源代码，所有代码都经过精心调试，在Windows Server 2003下能够正常运行。

本书内容精炼、重点突出、实例丰富，是程序开发人员必备的参考书，本书也非常适合大、中专院校师生学习、参考。

C#开发技术大全

◆ 编　　著　明日科技
　责任编辑　蒋　佳
◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
　邮编　100061　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京鑫正大印刷有限公司印刷
◆ 开本：787×1092　1/16
　印张：52.25
　字数：1417千字　　2011年11月第1版
　印数：1－3 000册　　2011年11月北京第1次印刷

ISBN 978-7-115-25870-0

定价：89.00元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第0021号

前　言

C#是.NET 技术的核心开发语言，具有简单、高效、功能强大的特点，是 Windows 环境、Web 环境下最优秀的程序设计工具之一。它采用面向对象的程序设计技术，提供了开发 Windows 应用程序最简捷、有效的方法。

作为一名 C#程序开发人员，除了具备必要的开发技能和经验积累外，必备的开发参考工具书也是必不可少的。程序开发参考工具书相当于程序员的“左膀右臂”，全面、系统的参考工具书可以使程序开发事半功倍。

本书内容

本书是一本 Visual C# 综合开发技术大全，书中几乎囊括了使用 C#进行程序开发的全部知识，同时在讲解中结合了大量实用而又有代表性的示例和典型应用。

本书分 7 篇共 31 章。书中各部分技术既相互独立又相互联系，可以逐步引导读者深入学习并掌握 C#的编程知识、方法和技巧，是 C#爱好者和程序开发人员必备的案头参考书。

第 1 篇　C#基础篇

第 1 章介绍了 C#语言基础及应用。

第 2 章介绍了程序设计中的基本控制结构和控制语句。

第 3 章介绍了数组与集合的相关技术及典型应用。

第 4 章介绍了 C#面向对象的编程，并讲解了类、对象、封装、继承和多态理论及其应用。

第 5 章介绍了一些在 C#中经常使用的算法。

第 6 章介绍了一些在 C#中经常使用的数据结构。

第 7 章介绍了程序调试与错误处理。

第 2 篇　Windows 窗体设计

第 8 章介绍了普通窗体、继承窗体、MDI 窗体的创建、设计及其应用。

第 9 章介绍了 Windows 控件与组件的属性、方法、事件及其使用。

第 10 章介绍了用户控件的属性、方法和事件及其使用。

第 11 章介绍了菜单编辑器的使用、菜单编程及工具栏和状态栏的创建和设计。

第 12 章介绍了 C#中的消息对话框和公用对话框的创建和设计。

第 3 篇　ASP.NET 网站开发

第 13 章介绍了 ASP.NET 3.5 运行环境的配置与管理，ASP.NET 网站和 Web 窗体的创建。

第 14 章介绍了 ASP.NET 3.5 常用服务器控件的属性、方法、事件及其应用。

第 15 章介绍了 Web 用户控件的创建及其应用。

第 16 章介绍了 ASP.NET 3.5 内置对象的属性、方法、事件及其应用。

第 17 章介绍了 ASP.NET 3.5 母版页的创建及其使用。

第 18 章介绍了 ASP.NET 3.5 主题的创建及其应用。

第 19 章介绍了站点导航和导航控件的属性、方法、事件及其应用。

第 20 章介绍了 ASP.NET 3.5 缓存技术的相关应用。

第 4 篇　数据库开发

第 21 章介绍了数据库的安装与创建，以及 SQL Server 数据库的备份、还原、附加和分离等。

第 22 章介绍了如何利用 ADO 对象连接 SQL Server 数据库，以及如何利用 ADO 对象对不同类型的数据进行增、删、改、查。

第 23 章介绍了数据绑定控件的属性、方法和事件及其使用。

第 24 章介绍了 LINQ 的语法基础，以及用 Linq 技术对数据库进行添加、修改、删除及查询的操作。

第 25 章介绍了存储过程、视图、触发器和游标的语法格式，以及在 C#中的应用。

第 5 篇　文件技术

第 26 章介绍了文件的创建、移动、删除、复制写入和读取的操作，以及通过文件流对大文件进行复制的操作。

第 27 章介绍了 WinHelp 帮助文件和 HTML 帮助文件的制作及调用。

第 6 篇　图形图像与多媒体

第 28 章介绍了 GDI+基本绘制技术，通过对本章的学习，读者可以掌握绘制圆柱型、饼型以及特殊图型的制作。

第 29 章介绍了多媒体文件的概述、文字处理、音频与视频处理、动画设计以及如何利用 DirectShow 技术进行多媒体的开发。

第 7 篇　报表与打印技术

第 30 章介绍了水晶报表的设计与使用。

第 31 章介绍 Windows 打印组件的属性、方法、事件及其应用。

本书特点

技术全面、系统。本书从开发者实际应用的技术入手，系统地介绍了使用 C#进行程序开发须要掌握的各方面开发技术。具有很强的参考价值。

快速查询与应用。本书针对具体的知识点，提供了大量从实际项目中筛选的示例和典型应用。通过本书，读者不仅可以快速查询相关技术，还能掌握其实际应用的方法和技巧。

本书读者

本书主要面向中级以上水平的程序员。全书力求自成体系，并且各章重点突出、概念清晰。通过阅读本书，读者可以在尽可能短的时间内查找、应用所需技术，创建功能强大的应用程序。同时，还可以通过阅读书中精炼有效的代码来巩固和扩充自己的编程知识，提高自己的编程水平。

本书约定

（1）本书提供了数百个示例和典型应用程序文件，为了读者查找方便，本书在“附录”中按字母顺序提供书中实例的对应章节位置。

（2）随书光盘中提供了书中大部分示例、典型应用的实例源程序，以及相关开发工具的安装、调试、管理和使用视频录像。正文中都给出了具体文件光盘位置。如示例 02-01，书中给出了光盘位置如下：

代码02-01　光盘位置：光盘\mingrisoft\sl\02\01

联系我们

本书由明日科技组织编写。由于C#的功能涉及范围广泛，书中疏漏和错误之处在所难免，敬请广大读者批评指正。

为便于读者和本书作者沟通，明日科技将通过明日科技网站全面为读者提供网上服务和支持。读者使用本书遇到的错误和问题，我们承诺在6个工作日内给您提供答复。

服务网站：www.mingrisoft.com

服务信箱：mingrisoft@mingrisoft.com

客服电话：0431-84978981 84978982 转 802、803

书山有路勤为径，学海无涯苦作舟。软件开发工作是一个漫长而艰苦的过程，需要积累非常丰富的理论知识和开发经验，愿本书能成为您学习编程知识的伴侣、提高技术的良师益友。

编者

2011 年 6 月

目 录

第 1 篇 C#基础篇

第 1 章 C#语言基础 ····· 3
1.1 C#语言简介 ····· 4
1.2 数据类型 ····· 4
1.2.1 值类型 ····· 4
1.2.2 引用类型 ····· 5
1.2.3 装箱和拆箱 ····· 6
1.3 变量和常量 ····· 6
1.3.1 变量 ····· 6
1.3.2 常量 ····· 8
1.4 数据类型转换 ····· 9
1.4.1 隐式类型转换 ····· 9
1.4.2 显示类型转换 ····· 9
1.4.3 使用 System.Convert 转换数据类型 ····· 10
1.5 运算符 ····· 11
1.5.1 算术运算符 ····· 11
1.5.2 赋值运算符 ····· 12
1.5.3 关系运算符 ····· 13
1.5.4 逻辑运算符 ····· 14
1.5.5 位运算 ····· 16
1.5.6 其他特殊运算符 ····· 17
1.5.7 运算符的优先级 ····· 18
1.6 枚举与结构 ····· 19
1.6.1 枚举 ····· 19
1.6.2 结构 ····· 22
1.7 程序编写规范 ····· 25
1.7.1 代码书写规则 ····· 25
1.7.2 命名规范 ····· 26
第 2 章 C#语句控制 ····· 29
2.1 语句 ····· 30
2.2 顺序结构程序设计 ····· 30
2.3 选择结构程序设计 ····· 31
2.3.1 if 语句 ····· 31
2.3.2 switch 语句 ····· 34
2.4 循环结构程序设计 ····· 37
2.4.1 while 语句 ····· 37
2.4.2 do…while 语句 ····· 39
2.4.3 for 语句 ····· 40
2.4.4 foreach 语句 ····· 41
2.5 异常处理语句 ····· 42
2.5.1 throw 语句 ····· 42
2.5.2 try-catch 语句 ····· 43
2.5.3 try-catch-finally 语句 ····· 44
2.6 跳转控制语句 ····· 45
2.6.1 break 语句 ····· 45
2.6.2 continue 语句 ····· 46
2.6.3 goto 语句 ····· 47
2.6.4 return 语句 ····· 48
第 3 章 数组与集合 ····· 49
3.1 数组 ····· 50
3.1.1 一维数组 ····· 50
3.1.2 二维数组 ····· 51
3.1.3 多维数组 ····· 53
3.1.4 数组的基类 Array ····· 53
3.1.5 交错数组 ····· 58
3.2 集合 ····· 60
3.2.1 ArrayList 类 ····· 60
3.2.2 Hashtable 类 ····· 65
3.2.3 其他集合的简单应用 ····· 69
第 4 章 面向对象编程 ····· 73
4.1 面向对象编程概述 ····· 74
4.2 类 ····· 74
4.2.1 类的概念 ····· 74
4.2.2 类的声明 ····· 75

4.3 对象……76
4.3.1 对象的声明和实例化……76
4.3.2 对象的生命周期……76
4.3.3 对象与类的关系……76
4.4 构造函数和析构函数……77
4.4.1 默认的构造函数……77
4.4.2 重载的构造函数……78
4.4.3 析构函数……79
4.5 方法……80
4.5.1 方法的声明……80
4.5.2 方法的类型……80
4.5.3 方法的参数类型……82
4.5.4 方法的返回值类型……85
4.5.5 方法的重载……86
4.6 域和属性……87
4.6.1 域……87
4.6.2 属性……87
4.7 委托……88
4.7.1 委托的用途……88
4.7.2 委托的实现……89
4.8 事件和索引器……90
4.8.1 事件……90
4.8.2 索引器……93
4.9 接口……95
4.9.1 接口的概念……95
4.9.2 接口的声明……95
4.9.3 接口的成员……95
4.9.4 接口的实现……96
4.10 封装、继承、多态……97
4.10.1 封装……97
4.10.2 继承……98
4.10.3 多态……100
4.11 泛型……101
4.11.1 泛型的概念……101
4.11.2 泛型的使用……102
4.11.3 泛型的定义……102
第 5 章 算法……105
5.1 算法概述……106
5.1.1 什么是算法……106
5.1.2 算法的特点……106
5.1.3 算法的描述方法……106
5.2 若干基本数学算法……107
5.2.1 查找最大值和最小值……107
5.2.2 计算最大公约数和最小公倍数……108
5.2.3 累加和累乘……109
5.2.4 求 100 以内素数……109
5.2.5 进制转换……110
5.2.6 杨辉三角……111
5.3 排序算法……112
5.3.1 选择排序……112
5.3.2 冒泡排序……113
5.3.3 插入排序……113
5.3.4 希尔排序……114
5.4 查找算法……115
5.4.1 顺序查找……115
5.4.2 二分法查找……115
5.5 字符处理应用……116
5.5.1 简单的加密解密算法……116
5.5.2 字符串处理……117
5.6 穷举法……117
5.7 迭代法……118
第 6 章 数组结构……121
6.1 单向链表……122
6.1.1 单向链表的定义……122
6.1.2 单链表的操作……122
6.1.3 单链表的制作……123
6.2 双向链表……126
6.2.1 双向链表的定义……126
6.2.2 双向链表的操作……127
6.2.3 双向链表的制作……127
6.3 堆栈……132
6.3.1 堆栈的定义及操作思路……132
6.3.2 堆栈的制作……133
6.4 队列……135
6.4.1 队列的定义及操作思路……135
6.4.2 队列的制作……135
第 7 章 程序调试与错误处理……137
7.1 错误类型……138
7.2 程序调试……138
7.2.1 断点……138
7.2.2 开始执行……139
7.2.3 中断执行……140
7.2.4 停止执行……140

7.2.5 单步执行 ……140
7.2.6 运行到指定位置 ……141
7.3 错误处理 ……141
7.3.1 Web 服务器配置不正确 …… 141
7.3.2 IIS 管理服务没有响应 …… 142
7.3.3 未安装 ASP.NET …… 142

第 2 篇 Windows 窗体设计篇

第 8 章 Windows 窗体介绍 ……145
8.1 设置 Windows 窗体 ……146
8.1.1 创建窗体 ……146
8.1.2 窗体的属性、方法和事件 ……148
8.1.3 调用窗体 ……155
8.1.4 为窗体添加控件 ……157
8.2 创建继承窗体 ……157
8.2.1 继承窗体概述 ……157
8.2.2 创建继承窗体 ……158
8.2.3 继承窗体的应用 ……160
8.3 创建 MDI 窗体 ……163
8.3.1 MDI 窗体概述 ……163
8.3.2 创建 MDI 窗体 ……164
8.3.3 创建 MDI 应用程序 ……166
第 9 章 Windows 控件与组件 ……169
9.1 控件与组件概述 ……170
9.1.1 控件概述 ……170
9.1.2 组件概述 ……170
9.2 文本类型控件 ……170
9.2.1 Label 控件 ……171
9.2.2 LinkLabel 控件 ……174
9.2.3 TextBox 控件 ……176
9.2.4 RichTextBox 控件 ……178
9.3 动作类型控件 ……182
9.4 列表类型控件 ……185
9.4.1 ListBox 控件 ……185
9.4.2 ComboBox 控件 ……190
9.4.3 ListView 控件 ……194
9.4.4 TreeView 控件 ……199
9.4.5 NumericUpDown 控件 ……205
9.5 值类型控件 ……207
9.5.1 CheckBox 控件 ……207
9.5.2 RadioButton 控件 ……211
9.5.3 CheckedListBox 控件 ……212
9.6 图形显示类型控件 ……217
9.6.1 PictureBox 控件 ……217
9.6.2 NotifyIcon 控件 ……219
9.7 日期类型控件 ……223
9.7.1 DateTimePicker 控件 ……223
9.7.2 MonthCalendar 控件 ……226
9.8 其他类型控件 ……231
9.8.1 ToolTip 控件 ……231
9.8.2 ProgressBar 控件 ……234
9.8.3 WebBrowser 控件 ……236
9.8.4 MaskedTextBox 控件 ……243
9.9 容器控件 ……248
9.9.1 FlowLayoutPanel 控件 ……248
9.9.2 Panel 控件 ……252
9.9.3 GroupBox 控件 ……255
9.9.4 SplitContainer 控件 ……257
9.9.5 TabControl 控件 ……259
9.9.6 TableLayoutPanel 控件 ……263
9.10 组件 ……267
9.10.1 BackgroundWorker 组件 ……267
9.10.2 DirectoryEntry 组件 ……270
9.10.3 DirectorySearcher 组件 ……274
9.10.4 ErrorProvider 组件 ……280
9.10.5 EventLog 组件 ……283
9.10.6 FileSystemWatcher 组件 ……287
9.10.7 HelpProvider 组件 ……290
9.10.8 ImageList 组件 ……293
9.10.9 MessageQueue 组件 ……295
9.10.10 PerformanceCounter 组件 ……301
9.10.11 ServiceController 组件 ……305
9.10.12 Timer 组件 ……310
第 10 章 用户控件 ……313
10.1 用户控件的概述 ……314
10.1.1 用户控件的概述 ……314
10.1.2 用户控件的优点 ……314
10.2 用户控件的应用 ……314
10.2.1 组合控件 ……314

10.2.2 自定义控件……317
第 11 章 菜单栏、工具栏和状态栏的设计…319
11.1 菜单栏……320
11.1.1 菜单栏概念和用途……320
11.1.2 菜单栏的设计……320
11.2 工具栏……322
11.2.1 工具栏概念和用途……322
11.2.2 工具栏的设计……322
11.3 状态栏……325
11.3.1 状态栏概念和用途……325
11.3.2 状态栏的设计……325
第 12 章 对话框……327
12.1 “消息”对话框……328
12.1.1 “消息”对话框简介……328
12.1.2 “消息”对话框的应用……328
12.2 “关于”对话框……337
12.2.1 “关于”对话框简介……337
12.2.2 “关于”对话框的应用……337
12.3 “打开”对话框……339
12.3.1 “打开”对话框简介……339
12.3.2 “打开”对话框的应用……339
12.4 “另存为”对话框……342
12.4.1 “另存为”对话框简介……342
12.4.2 “另存为”对话框的应用……342
12.5 “颜色”对话框……343
12.5.1 “颜色”对话框简介……343
12.5.2 “颜色”对话框的应用……343
12.6 “字体”对话框……345
12.6.1 “字体”对话框简介……345
12.6.2 “字体”对话框的应用……346
12.7 “打印”对话框……348
12.7.1 “打印”对话框简介……348
12.7.2 “打印”对话框应用……348
12.8 “页面设置”对话框……350
12.8.1 “页面设置”对话框简介……350
12.8.2 “页面设置”对话框应用……351
12.9 “打印预览”对话框……353
12.9.1 “打印预览”对话框简介……353
12.9.2 “打印预览”对话框应用……353
12.10 “浏览文件夹”对话框……356
12.10.1 “浏览文件夹”对话框简介……356
12.10.2 “浏览文件夹”对话框应用……357

第 3 篇 ASP.NET 网站开发篇

第 13 章 ASP.NET 3.5 网站开发基础……363
13.1 ASP.NET 3.5 运行环境配置与管理……364
13.1.1 安装 IIS 服务器……364
13.1.2 配置 IIS 服务器……366
13.2 ASP.NET 3.5 基本设计步骤……368
13.2.1 创建 ASP.NET 3.5 网站……368
13.2.2 创建 Web 窗体……369
13.2.3 设计 Web 窗体界面……370
13.2.4 编写代码……370
13.2.5 运行程序……371
13.3 ASP.NET 3.5 网页语法……372
13.3.1 ASP.NET 3.5 网页扩展名……372
13.3.2 Page 指令……372
13.3.3 代码注释……372
13.3.4 HTML 服务器控件语法……373
13.3.5 代码块语法……374
13.3.6 数据绑定语法……375
13.3.7 对象标记语法……376
13.4 Web.config 文件配置……378
第 14 章 ASP.NET 3.5 常用服务器控件…381
14.1 标准服务器控件……382
14.1.1 TextBox 控件……382
14.1.2 Button 控件……385
14.1.3 LinkButton 控件……388
14.1.4 ImageButton 控件……389
14.1.5 HyperLink 控件……391
14.1.6 ListBox 控件……392
14.1.7 DropDownList 控件……395
14.1.8 CheckBox 控件……398
14.1.9 RadioButton 控件……400

14.1.10 Calendar 控件……401
14.1.11 FileUpload 控件……403
14.1.12 标准服务器控件典型应用……406
14.2 验证控件……407
14.2.1 RequiredFieldValidator 控件……407
14.2.2 RegularExpressionValidator 控件……408
14.2.3 CompareValidator 控件……410
14.2.4 ValidationSummary 控件……412
14.2.5 验证控件典型应用……413
14.3 登录控件……414
14.3.1 Login 控件……414
14.3.2 CreateUserWizard 控件……416
14.3.3 登录控件典型应用……419
第 15 章 Web 用户控件……421
15.1 用户控件的概述和用途……422
15.1.1 用户控件概述……422
15.1.2 用户控件的优点……422
15.2 创建用户控件……422
15.2.1 创建用户控件……422
15.2.2 将 Web 页转换为 ASP.NET 3.5 用户控件……426
15.3 用户控件的应用……428
第 16 章 ASP.NET 3.5 内置对象……429
16.1 Response 对象……430
16.1.1 Response 对象的属性……430
16.1.2 Response 对象的方法……431
16.1.3 Response 对象的应用……432
16.2 Request 对象……433
16.2.1 Request 对象的属性……433
16.2.2 Request 对象的方法……435
16.2.3 Request 对象的应用……436
16.3 Application 对象……438
16.3.1 Application 对象的属性……438
16.3.2 Application 对象的方法……439
16.3.3 Application 对象的事件……440
16.3.4 Application 对象的应用……441
16.4 Session 对象……442
16.4.1 Session 对象的属性……443
16.4.2 Session 对象的方法……443
16.4.3 Session 对象的事件……444
16.4.4 Session 对象的应用……444
16.5 Server 对象……445
16.5.1 Server 对象的属性……445
16.5.2 Server 对象的方法……446
16.5.3 Server 对象的应用……447
16.6 Cookie 对象……448
16.6.1 Cookie 对象的属性……448
16.6.2 Cookie 对象的应用……449
第 17 章 ASP.NET 3.5 母版页……451
17.1 母版页的概述……452
17.2 创建母版页……454
17.3 创建内容页……456
17.4 嵌套母版页……457
17.5 访问母版页的控件和属性……459
第 18 章 ASP.NET 3.5 主题……465
18.1 主题的概述……466
18.1.1 组成元素……466
18.1.2 文件存储和组织方式……467
18.2 创建主题……467
18.2.1 创建外观文件……467
18.2.2 为主题添加 CSS 样式……469
18.3 应用主题……471
第 19 章 站点导航和导航控件……473
19.1 站点地图概述……474
19.2 SiteMapPath 控件……475
19.2.1 功能……475
19.2.2 属性……475
19.2.3 事件……476
19.3 TreeView 控件……477
19.3.1 功能……477
19.3.2 属性……477
19.3.3 事件……479
19.4 Menu 控件……480
19.4.1 功能……480
19.4.2 属性……481
19.4.3 事件……483
19.5 导航控件的应用……483
19.5.1 TreeView 菜单导航的应用……483
19.5.2 Menu、SiteMapPath 联合制作站点导航……488

第 20 章 ASP.NET 3.5 缓存技术 …… 489
20.1 ASP.NET 3.5 缓存概述 …… 490
20.2 页面输出缓存 …… 490
20.3 页面部分缓存 …… 493
20.3.1 使用@OutputCache 指令 … 493
20.3.2 使用 PartialCaching Attribute 类 …… 493
20.3.3 使用 ControlCache Policy 类 …… 494
20.4 页面数据缓存 …… 496

第 4 篇 数据库开发篇

第 21 章 数据库的安装与创建 …… 501
21.1 Access 数据库 …… 502
21.1.1 安装 Access 数据库 …… 502
21.1.2 创建 Access 数据库 …… 507
21.1.3 创建 Access 数据表 …… 508
21.2 SQL Server 数据库 …… 509
21.2.1 安装 SQL Server …… 509
21.2.2 启动 SQL Server 服务管理器 …… 514
21.2.3 创建 SQL Server 数据库 … 515
21.2.4 创建 SQL Server 数据表 … 516
21.3 配置 ODBC 数据源 …… 517
21.3.1 配置 SQL Server 的 ODBC 数据源 …… 517
21.3.2 配置 Access 的 ODBC 数据源 …… 519
21.4 数据库的连接 …… 519
21.4.1 连接 Access 数据库 …… 519
21.4.2 连接 SQL Server 数据库 … 520
21.4.3 连接 Excel 数据库 …… 521
21.5 数据库备份与恢复 …… 522
21.5.1 数据库备份 …… 522
21.5.2 数据库还原 …… 525
21.6 附加和分离数据库 …… 527
21.6.1 附加数据库 …… 527
21.6.2 分离数据库 …… 530
第 22 章 ADO.NET 技术 …… 533
22.1 ADO.NET 概述 …… 534
22.2 ADO.NET 对象模型 …… 534
22.2.1 Connection 对象 …… 534
22.2.2 Command 对象 …… 536
22.2.3 DataReader 对象 …… 540
22.2.4 DataSet 对象 …… 542
22.2.5 DataAdapter 对象 …… 548
22.3 数据库开发基本操作 …… 551
22.3.1 打开数据库连接 …… 551
22.3.2 向数据库中添加数据 …… 552
22.3.3 修改数据库中数据 …… 553
22.3.4 删除数据库中数据 …… 556
22.3.5 查询数据库中数据 …… 560
22.3.6 使用事务 …… 562
第 23 章 数据绑定控件 …… 565
23.1 DataGridView 控件 …… 566
23.1.1 DataGridView 属性 …… 566
23.1.2 DataGridView 方法 …… 570
23.1.3 DataGridView 事件 …… 572
23.1.4 DataGridView 控件应用实例 …… 577
23.2 GridView 控件 …… 580
23.2.1 GridView 属性 …… 580
23.2.2 GridView 方法 …… 584
23.2.3 GridView 事件 …… 585
23.2.4 GridView 控件应用实例 …… 590
23.3 DataList 控件 …… 592
23.3.1 DataList 属性 …… 592
23.3.2 DataList 方法 …… 596
23.3.3 DataList 事件 …… 596
23.3.4 DataList 控件应用实例 …… 599
23.4 Repeater 控件 …… 601
23.4.1 Repeater 属性 …… 601
23.4.2 Repeater 方法 …… 603
23.4.3 Repeater 事件 …… 604
23.4.4 Repeater 控件应用实例 … 605
第 24 章 LINQ 技术 …… 609
24.1 LINQ 概述 …… 610
24.2 LINQ 语法基础 …… 610
24.2.1 LINQ to Objects …… 610
24.2.2 LINQ to DataSet …… 613

24.2.3 LINQ to SQL ……616
24.2.4 LINQ to XML ……624
第 25 章 高性能数据处理技术 ……629
25.1 存储过程概述 ……630
25.2 存储过程的建立、修改和删除 ……630
25.2.1 存储过程的建立 ……630
25.2.2 存储过程的修改 ……632
25.2.3 存储过程的删除 ……633
25.3 在 C#中调用存储过程 ……634
25.4 视图的概述 ……636
25.5 视图的建立、修改和删除 ……637
25.5.1 视图的建立 ……637
25.5.2 视图的修改 ……638
25.5.3 视图的删除 ……639
25.6 在 C#中调用视图 ……639
25.7 触发器的应用 ……642
25.8 游标的应用 ……643

第 5 篇 文件技术篇

第 26 章 文件处理技术 ……649
26.1 System.IO 命名空间 ……650
26.2 文件的基本操作 ……651
26.2.1 文件操作类 File 和 FileInfo ……651
26.2.2 创建文件 ……652
26.2.3 打开文件 ……654
26.2.4 判断文件是否存在 ……657
26.2.5 复制或移动文件 ……657
26.2.6 删除文件 ……660
26.2.7 获取文件基本信息 ……660
26.3 文件夹的基本操作 ……662
26.3.1 文件夹操作类 Directory 和 DirectoryInfo ……662
26.3.2 创建文件夹 ……663
26.3.3 判断文件夹是否存在 ……663
26.3.4 移动文件夹 ……663
26.3.5 删除文件夹 ……664
26.3.6 遍历文件夹中的子文件夹及文件 ……664
26.4 文件流的操作 ……668
26.4.1 文本文件的读取 ……668
26.4.2 文本文件的写入 ……671
26.4.3 大文件的复制 ……675
第 27 章 帮助文件 ……681
27.1 WinHelp 帮助文件 ……682
27.1.1 安装 Microsoft Help Workshop ……682
27.1.2 编辑 RTF 文件 ……683
27.1.3 制作帮助索引文件 ……686
27.1.4 制作.Hlp 帮助文件 ……688
27.1.5 在 C#中使用帮助文件 ……690
27.2 HTML 帮助文件 ……690
27.2.1 HTML 帮助文件概述 ……690
27.2.2 安装 HTML Help Workshop ……690
27.2.3 HTML 帮助主题文件 ……691
27.2.4 HTML 帮助项目文件 ……691
27.2.5 目录和目录文件 ……694
27.2.6 关键字和索引文件 ……697
27.2.7 制作.chm 文件 ……699
27.2.8 在 C#中使用.chm 帮助文件 ……701

第 6 篇 图形图像与多媒体

第 28 章 GDI+图形图像 ……705
28.1 GDI+概述 ……706
28.2 直线、曲线和图形 ……706
28.2.1 向量图形概述 ……706
28.2.2 GDI+中的直线和矩形 ……706
28.2.3 GDI+中的椭圆和弧 ……708
28.2.4 GDI+中的多边形 ……710
28.2.5 GDI+中的基数样条 ……711
28.2.6 GDI+中的贝塞尔样条 ……713
28.2.7 GDI+中的图形路径 ……714

28.2.8 GDI+中的画笔 …… 715
28.2.9 GDI+中曲线的填充 …… 718
28.2.10 GDI+中的区域 …… 718
28.2.11 用直线和曲线消除锯齿 …… 720
28.3 图像、位图和图元文件 …… 721
28.3.1 位图类型 …… 721
28.3.2 GDI+中的元文件 …… 722
28.3.3 在 GDI+中绘制和克隆图像 …… 723
28.3.4 在 GDI+中缩放图像 …… 725
28.4 坐标系和坐标变换 …… 725
28.4.1 坐标系类型 …… 725
28.4.2 变形的矩阵表示形式 …… 726
28.4.3 全局变换和局部变换 …… 728
28.5 GDI+绘制图表的应用 …… 730
28.5.1 绘制柱型图分析商品月销售情况 …… 730
28.5.2 绘制饼图分析男女比例 …… 733
28.5.3 绘制折线图分析网站流量 …… 734
28.5.4 绘制验证码图片 …… 736
第 29 章 多媒体技术 …… 739
29.1 多媒体概述 …… 740
29.2 文字处理 …… 740
29.3 音频与视频处理 …… 741
29.3.1 Windows Media Player 控件 …… 741
29.3.2 DirectShow 设计 …… 772
29.4 动画设计 …… 778
29.4.1 ImageAnimator 类功能 …… 778
29.4.2 ImageAnimator 类方法 …… 778
29.4.3 ImageAnimator 类的应用 …… 779

第 7 篇 报表与打印技术

第 30 章 水晶报表 …… 783
30.1 水晶报表简介 …… 784
30.2 .NET 平台下的 Crystal 报表 …… 784
30.2.1 Crystal Reports.Net 简介 …… 784
30.2.2 创建 Crystal 报表 …… 784
30.2.3 Crystal 报表设计器的环境介绍 …… 786
30.2.4 Crystal 报表区域介绍 …… 786
30.3 Crystal 报表数据源和数据库的操作 …… 788
30.3.1 Visual Studio 2008 中 Crystal Reports 数据源列举 …… 788
30.3.2 Visual Studio 2008 中 Crystal Reports 数据库操作 …… 788
30.3.3 水晶报表的应用 …… 796
第 31 章 Windows 打印对象 …… 799
31.1 PrintDocument 类 …… 800
31.1.1 PrintDocument 类的属性 …… 800
31.1.2 PrintDocument 类的方法 …… 802
31.1.3 PrintDocument 类的事件 …… 803
31.2 PageSetupDialog 类 …… 805
31.2.1 PageSetupDialog 类的属性 …… 805
31.2.2 PageSetupDialog 类的方法 …… 806
31.3 PrintDialog 类 …… 806
31.3.1 PrintDialog 类的属性 …… 806
31.3.2 PrintDialog 类的方法 …… 807
31.4 PrintPreviewControl 类 …… 807
31.4.1 PrintPreviewControl 类的属性 …… 808
31.4.2 PrintPreviewControl 类的方法 …… 810
31.5 PrintPreviewDialog 类 …… 810
31.5.1 PrintPreviewDialog 类的属性 …… 810
31.5.2 PrintPreviewDialog 类的方法 …… 812
31.6 Windows 打印对象的综合应用 …… 812

第1篇

C#基础篇

C#

第 1 章

C#语言基础

- C#语言简介
- 数据类型
- 变量和常量
- 数据类型转换
- 运算符
- 枚举与结构
- 程序编写规范

1.1 C#语言简介

C# 是专门用于.NET 的编程语言，它是为在.NET Framework 上运行的多种应用程序而设计的。C# 简单、功能强大、类型安全，是一种面向对象的语言，是从 C、C++以及 Java 演化而来的，C# 吸取了其他语言的优点，并解决了它们存在的问题。C# 凭借自身的多项创新，实现了应用程序的快速开发，它几乎可以开发出所有的 Windows 程序。

1.2 数 据 类 型

1.2.1 值类型

在 C#中数据类型为 Common Type System（CTS），包括值类型和引用类型。值类型直接存储值，而引用类型存储的是对值的引用。将一个值类型变量赋给另一个值类型变量时，将复制包含的值。这与引用类型变量的赋值不同，引用类型变量的赋值只复制对象的引用，而不复制对象本身。从值类型不可能派生出新的类型。值类型也不包含 null 值，但引用类型可以。值类型与引用类型其结构都可实现接口。

C#中的值类型如表 1.1 所示。

表 1.1　　C#值类型表

名　称	CTS 类型	说　明	范　围
sbyte	System.SByte	8 位有符号整数	−128～127
short	System.Int16	16 位有符号整数	−32 768～32 767
int	System.Int32	32 位有符号整数	-2^{31}～$2^{31}-1$
long	System.Int64	64 位有符号整数	-2^{63}～$2^{63}-1$
byte	System.Byte	8 位无符号整数	0～255
ushort	System.Unit16	16 位无符号整数	0～65 535
unit	System.Unit32	32 位无符号整数	0～$2^{32}-1$
ulong	System.Ulong	64 位无符号整数	0～$2^{64}-1$
float	System.Single	32 位单精度浮点数	$\pm 1.5\times 10^{-45}$～$\pm 3.4\times 10^{38}$
double	System.Double	64 位双精度浮点数	$\pm 5.0\times 10^{-324}$～$\pm 1.7\times 10^{308}$
decimal	System.Decimal	128 位双精度浮点数	$\pm 1.0\times 10^{-28}$～$\pm 7.9\times 10^{28}$

值类型的声明语法如下：

```
Type name;
Name = TypeValue;
```

或者：

```
Type name = new Type();
```

参数说明如下：

- Type：类型，包含表 1.1 中的所有值类型。
- name：值类型的名称。
- TypeValue：相对于类型的值。

下面以整数类型为例，对值类型进行初始化。

例如，声明未进行初始化的整型局部变量。

```
int n;
```

例如，对整型变量进行初始化，其默认值为 0。

```
n = new int();
```

以上代码等同于：

```
myInt = 0;
```

例如，在声明整型变量的同时对其进行初始化。

```
int n = new int();
```

以上代码等同于：

```
int n = 0;
```

在这里要说明的是，在对值类型进行声明时，int 类型的默认值为 0，char 类型的默认值为“0”，bool 类型的默认值为 false，Byte 类型的默认值为 0。

1.2.2 引用类型

引用类型的变量又称为对象，可存储对实际数据的引用。Class、nterface、delegate 可以用于声明引用类型。

C#有两个内置的引用类型，分别为 Object 和 String 类型，下面将对其进行详细讲解。

Object 类型在.NET Framework 中是 Object 的别名。在 C#的统一类型系统中，所有类型（预定义类型、用户定义类型、引用类型和值类型）都是直接或间接从 Object 继承的。可以将任何类型的值赋给 Object 类型的变量。将值类型的变量转换为对象的过程称为“装箱”。将对象类型的变量转换为值类型的过程称为“取消装箱”。

示例 01-01 Object 类型的引用调用

下面通过例子来说明 Object 的引用，主要程序代码如下：

代码 01-01 光盘位置：光盘\mingrisoft\sl\01\01_01

```
namespace ObjectApple
{
    class Program
    {
        class ObjectClass      //定义一个类
        {
            public int i = 60;                          //定义一个整型变量
        }
        static void Main(string[] args)
        {
            object a;                                   //定义 object 类型
            a = 1;                                      //装箱
            Console.WriteLine(a);                       //输出装箱后的值
            Console.WriteLine(a.GetType());             //输出类型
            Console.WriteLine(a.ToString());            //以字符串输出
            Console.WriteLine();                        //换行
            a = new ObjectClass();                      //用 ObjectClass 类进行初始化
            ObjectClass classRef;                       //定义 ObjectClass
            classRef = (ObjectClass)a;                  //拆箱
            Console.WriteLine(classRef.i);              //输出 i 值
        }
    }
}
```

输出结果为：

```
1
System.Int32
1
60
```

说明：可以看到 classRef 只是指向了 a，因为 classRef 并没有实例化，所以它的 classRef.i 等于 60。

string 类型表示 Unicode 字符的字符串。string 是.NET Framework 中的 string 的别名。字符

串是不可变的。字符串对象一旦被创建，其内容就不能更改。

尽管 string 是引用类型，但定义相等运算符（==和!=）是为了比较 string 对象（而不是引用）的值。这使得对字符串相等性的测试更为直观。例如：

```
string a = "hello";                                  //定义字符串
string c = "h";                                      //定义字符串
c = c+"ello";                                        //字符串连
Console.WriteLine(a == c);                           //判断字符串的内容是否相等
Console.WriteLine((object)a == (object)c);           //判断字符串的实例是否相等
```

这时将先显示“True”，然后显示“False”，因为字符串的内容是相同的，但是 a 和 c 引用的不是同一个字符串实例。

string 可以包含转义序列，因为这些转义序列需要一个反斜杠开头，所以如果在字符串中使用这个非转义的反斜杠，则需要用两个“\\”来表示，如 string filepath="C\\load\\file\\test.txt"。但是如果大量使用这样的表示会让人难以理解，在 C#中可以在字符串前面加上“@”符号，字符串里的字符就不会被解释为转义序列了，如 string filepath=@"C\load\file\test.txt"。

1.2.3 装箱和拆箱

装箱就是将值类型转换为引用类型 Object，这使得值类型可以存储在垃圾回收堆中。

拆箱将从对象中提取值类型。拆箱就是将引用类型转换为值类型。例如：

```
int i = 929;
object a = (object) i;                               //装箱
a = 123;
i = (int) a;                                         //拆箱
```

相对于简单的赋值而言，装箱和拆箱过程需要进行大量的计算。对值类型进行装箱时，必须分配并构造一个全新的对象。拆箱所需的强制转换也需要进行大量的计算。因此装箱、拆箱会对性能产生影响。

1.3 变量和常量

1.3.1 变量

变量是指在程序运行过程中其值可以不断变化的量。变量通常用来保存程序运行过程中的输入数据并计算获得的中间结果和最终结果。变量的命名规则必须符合标识符的命名规则，并且变量名要有意义（人性化），以便阅读。

在 C# 中，变量可分为静态变量、实例变量、数组变量、局部变量、值参数、引用参数和输出参数这 7 种类型。下面分别对这 7 种变量进行讲解。

1．静态变量

通过 static 修饰符声明的变量称为静态变量。静态变量只有被创建并加载后才会生效，同样，被卸载后会失效。无论创建多少静态变量，都只有一个副本。例如，声明一个整型的静态变量 a，代码如下：

```
static int n=1024;
```

注意：最好在声明时对静态变量进行赋值。

2．实例变量

声明变量时，没有 static 修饰的变量称为实例变量。当类被实例化时，将生成属于该类的实例变量。当不再对该实例进行引用，而且已执行实例的析构函数后，此实例变量将失效。类中实例变量的初始值为此类型变量的默认值。为了方便进行赋值检查，类中的实例变量应是初始化的。例如，声明一个整型的实例变量 a，代码如下：

```
int a;
```

3．数组变量

数组元素随着数组的存在而存在，当任意一个数组实例被创建时，该数组元素也同时被创建。每个数组元素的初始值都是其数组元素类型的默认值。为了明确赋值检查，数组元素被认为已赋初始值。例如，声明一个整型的数组变量 num，代码如下：

```
int[] num = new int[4];
```

4．局部变量

具有局部作用域的变量，称为局部变量，只在定义它的块内起作用。所谓块指的是大括号“{”和“}”之间的所有内容。块内可以是一条语句，也可以是多条语句或者空语句。局部变量从被声明的位置开始起作用，当块结束时，局部变量也就消失。例如，声明一个整型的局部变量 n，代码如下：

```
public void Test()
{
    int n=0;
}
```

注意：局部变量需要注意初始化问题，局部变量需要人工赋值后才能使用。

5．值参数

声明一个变量时，该变量没有 ref 或 out 修饰，可称此变量为值参数。值参数在其隶属的函数子句被调用时自动生成，同时被赋给调用中的参数值。当函数成员返回时，值参数失效。为了方便赋值检查，所有的值参数都被认为是已被初始化过的。例如，声明一个方法 Test 的参数为整型的值参数变量 a，代码如下：

```
public void Test(int a)
{
}
```

6．引用参数

用 ref 修饰符声明的参数为引用参数。引用参数不创建新的存储位置。引用参数的值总是与基础变量相同。若要使用 ref 参数，则方法定义和调用方法都必须显式使用 ref 关键字。例如，声明一个方法 Test 的参数为整型的引用参数变量 a。

声明具有引用参数的方法：

```
public void Test(ref int a)
{
}
```

调用具有引用参数的方法：

```
int b=0;
Test(ref b);
```

下面讲解一下引用参数的赋值规则。

- 变量作为引用参数在函数成员调用中传递之前必须已明确赋值。
- 在函数成员内部，引用参数被视为初始已赋值。

在结构类型的实例方法或实例访问器内部，this 关键字的行为与该结构类型的引用参数完全相同。

7．输出参数

用 out 修饰符声明的参数是输出参数。输出参数不创建新的存储位置。相反，输出参数表示在对该函数成员调用中被当作“自变量”的变量所表示的同一个存储位置。因此，输出参数的值总是与基础变量相同。例如，声明一个方法 Test 的参数为整型的引用参数变量 a，代码如下：

```
public void Test(out int a)
{
}
```

调用具有输出参数的方法：

```
int b=0;
Test(out b);
```

下面讲解一下输出参数的赋值规则。

- 变量在作为输出参数在函数成员调用中传递之前，不一定要明确赋值。
- 在正常完成函数成员调用之后，每个作为输出参数传递的变量都被认为在该执行路径中已赋值。
- 在函数成员内部，输出参数被视为初始未赋值。
- 函数成员的每个输出参数在该函数成员正常返回前都必须已明确赋值。

在结构类型的实例构造函数内部，this 关键字的行为与结构类型的输出参数完全相同。

1.3.2 常量

常量又叫常数，是在程序运行过程中其值不改变的量。在 C# 中，常量的数据类型有多种，分别是 sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double、decimal、bool、string 等。

常数通过 const 关键字被声明为字段，常数必须在声明时初始化。代码如下：

```
class Calendar1
{
    public const int months = 12;
}
```

在上面的示例中，常数 months 将始终为 12，不能更改，即使是该类自身也不能更改它。常数必须属于整型（sbyte、byte、short、ushort、int、uint、long、ulong、char、float、double、decimal、bool 或 string）、枚举或对 null 的引用。

可以同时声明多个相同类型的常数。代码如下：

```
class Calendar2
{
    const int months = 12, weeks = 52, days = 365;
}
```

只要不会造成循环引用，用于初始化一个常数的表达式就可以引用另一个常数。代码如下：

```
class Calendar3
{
    const int months = 12;                                  //定义整型常量
    const int weeks = 52;
    const int days = 365;
    const double daysPerWeek = days / weeks;                //用常量赋值
    const double daysPerMonth = days / months;
}
```

常数可标记为 public（公有）、private（私有）、protected（类派生）、internal（程序集）或 protectedinternal（类派生的当前程序集）。这些访问修饰符定义类的用户访问该常数的方式。

尽管常数不能使用 static 关键字声明，但可以像访问静态字段一样访问常数。未包含在定义常数的类中的表达式必须使用类名、一个句点和常数名来访问该常数。例如：

```
int birthstones = Calendar3.months;
```

注意：若要创建在运行时初始化的常数值，请使用 readonly 关键字。

示例 01-02 常量与 protected 关键字的应用

下面通过例子来说明用关键字 protected 声明的常量如何在类中进行应用，主要程序代码如下：

```
class A                                         //定义一个类
{
    protected const int x = 123;                //定义一个类派生的常量
}
```

```
class B : A                                          //定义一个 A 的继承类 B
{
    const int y = 72;                                //定义一个常量
    public void C()                                  //定义一个方法
    {
        MessageBox.Show(A.x.ToString() + "    " + B.x.ToString() + "    " + B.y.ToString());
    }
}
private void button1_Click(object sender, EventArgs e)
{
    B b = new B();                                   //实例化类 B
    b.C();                                           //调用自定义方法
}
```

输出结果：

```
123    123    72
```

说明：在以上代码中因为 B 类是 A 类的继承类，所以 A.x 和 B.x 调用的都是 A 类中的 x 常量。

1.4 数据类型转换

1.4.1 隐式类型转换

隐式转换是指将低精度数值转换为高精度数值，可以直接赋值而不用任何转换。隐式转换可能在多种情形下发生，包括在调用方法时或在赋值语句中。

隐式类型转换如表 1.2 所示。

表 1.2　　隐式类型转换表

源 类 型	目 标 类 型
Sbyte	short、int、long、float、double、ordecimal
byte	short、ushort、int、uint、long、ulong、float、double 或 decimal
short	int、long、float、double 或 decimal
ushort	int、uint、long、ulong、float、double 或 decimal
int	long、float、double 或 decimal
uint	long、ulong、float、double 或 decimal
char	ushort、int、uint、long、ulong、float、double 或 decimal
float	double
ulong	float、double 或 decimal
Long	float、double 或 decimal

例如，将整型直接转换成双精度型。代码如下：

```
int i = 123;
double b = i;                          //整型转换成双精度型
```

1.4.2 显示类型转换

显示类型转换是指将高精度数值转换为低精度数值，必须指明将要转换的目标类型。由于数据类型的差异，有可能丢失部分数据。

例如，将双精度型转换为单精度型。代码如下：

```
double d = 23.45D;
float f = (float)d;                    //双精度型转换为单精度型
```

示例 01-02　double 类型 x 的显示类型转换

下面的示例代码演示了如何将 x 进行显示类型转换。

代码 01-02　光盘位置：光盘\mingrisoft\sl\02\02_03

```
class Test
{
    static void Main()
    {
        double x = 1234.7;                    //定义双精度型
        int a;                                //定义整型
        a = (int)x;                           //将双精度型转换成整数
        System.Console.WriteLine(a);
    }
}
```

输出结果：

```
1234
```

1.4.3　使用 System.Convert 转换数据类型

System.Convert 类为支持的转换提供了一整套方法。它提供了一种与语言无关的方法来执行转换，而且可用于针对公共语言运行库的所有语言。不同的语言可能会使用不同的技术来转换数据类型，Convert 类可确保所有的公共转换都可使用一般格式。该类可执行收缩转换以及不相关数据类型的转换。例如，可以从 String 转换为数字类型、从 DateTime 类型转换为 String 类型以及从 String 类型转换为 Boolean 类型。

下面介绍一下 Convert 类的转换方法，如表 1.3 所示。

表 1.3　Convert 类的转换方法

方　　法	说　　明
ToBase64CharArray	将 8 位无符号整数数组的子集转换为用 Base 64 数字编码的 Unicode 字符数组的等价子集
ToBase64String	将 8 位无符号整数数组的值转换为它的等效 String 表示形式（使用 Base 64 数字编码）
ToByte	将指定的值转换为 8 位无符号整数
ToChar	将指定的值转换为 Unicode 字符
ToDateTime	将指定的值转换为 DateTime
ToDecimal	将指定值转换为 Decimal（十进制数）数字
ToDouble	将指定的值转换为双精度浮点数
ToInt16	将指定的值转换为 16 位有符号整数
ToInt32	将指定的值转换为 32 位有符号整数
ToInt64	将指定的值转换为 64 位有符号整数
ToSByte	将指定的值转换为 8 位有符号整数
ToSingle	将指定的值转换为单精度浮点数字
ToString	将指定的值转换为字符串

在对类型进行转换时，可以将要转换的值传递给 Convert 类中的某一相应方法，并将返回的值初始化为新变量。

例如，字符型与数值型的强制转换。代码如下：

```
int n = 5;                                    //定义整数
float f = 10.5F;                              //定义单精度数
double d = 7.258D;                            //定义双精度数
Console.WriteLine(Convert.ToString(n));       //将整数输转换成字符串
Console.WriteLine(Convert.ToString(f));       //将单精度数输转换成字符串
Console.WriteLine(Convert.ToString(d));       //将双精度数输转换成字符串
string sn = "5";                              //定义字符串
string sf = "10.5";
string sd = "7.258";
Console.WriteLine(Convert.ToInt32(sn));       //将字符串转换成 32 位的整数
```

```
Console.WriteLine(Convert.ToSingle(sf));                    //将字符串转换成单精度数
Console.WriteLine(Convert.ToDouble(sd));                    //将字符串转换成双精度数
```

例如，数值型与布尔型的强制转换。代码如下：

```
int n = 1;                                                  //定义整型
int b = 0;
bool b1 = Convert.ToBoolean(n);                             //将数值型转换为布尔型
bool b2 = Convert.ToBoolean(b);
```

在以上代码中，b1 的值为 true，b2 的值为 false，这里要注意的是，在将数值型转换成布尔型时，如果数值型的值为 0，转换后的值为 false；如果不为 0（包含小数和负数），转换后的值为 true。

1.5 运 算 符

1.5.1 算术运算符

算术运算符有+、−、*、/、%等。

1．“+”运算符

C#中为数值类型和字符串类型预定义了二元运算符“+”。对于数值类型，“+”运算符用于计算两个操作数之和。当其中的一个操作数是字符串类型或两个操作数都是字符串类型时，用“+”运算符将操作数的字符串串联在一起。

示例 01-03 “+”运算符的使用

下面的示例代码演示了如何用“+”运算符进行加法和串联字符串。

代码 01-03 光盘位置：光盘\mingrisoft\sl\01\01_03

```
static void Main(string[] args)
{
    Console.WriteLine(15 + 15);                             //两个整数相加
    Console.WriteLine(15 + .5);                             //整数和小数相加
    Console.WriteLine("15" + "15");                         //连接字符串
    Console.WriteLine(5.0 + "15");                          //连接字符串
}
```

输出结果：

```
30
15.5
1515
515
```

2．“−”运算符

“−”运算符既可作为一元运算符也可作为二元运算符。

一元运算符是为所有数值类型预定义的。数值类型的一元运算的结果是操作数的反数。

二元运算符是为所有数值类型和枚举类型预定义的，其功能是从第一个操作数中减去第二个操作数。例如：

```
static void Main(string[] args)
{
    int a = 5;
    Console.WriteLine(-a);                                  //取反数
    Console.WriteLine(a - 1);                               //整数减法运算
    Console.WriteLine(a - .5);                              //小数减法运算
}
```

输出结果：

```
-5
4
4.5
```

3．“*”运算符

乘法运算符（*），用于计算操作数的积。例如：

```
static void Main(string[] args)
{
    Console.WriteLine(5 * 2);                    //整数乘法运算
    Console.WriteLine(-.5 * .2);                 //小数乘法运算
}
```

输出结果：

```
10
-0.1
```

4．“/”运算符

除法运算符（/）用第二个操作数除第一个操作数。所有数值类型都具有预定义的除法运算符。例如：

```
static void Main()
{
    Console.WriteLine(-5 / 2);                   //负数除法运算
    Console.WriteLine(-5.0/ 2);                  //小数除法运算
}
```

输出结果：

```
-2
-2.5
```

5．“%”运算符

模数运算符（%）计算第一个操作数除以第二个操作数后的余数。所有数值类型都具有预定义的模数运算符。例如：

```
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(5 % 2);                //整数取余
        Console.WriteLine(-5 % 2);               //负数取余
        Console.WriteLine(5.0 % 2.2);            //单精度数取余
        Console.WriteLine(5.0m % 2.2m);          //十进制数取余
        Console.WriteLine(-5.2 % 2.0);           //双精度数取余
    }
}
```

输出结果：

```
1
-1
0.6
0.6
-1.2
```

1.5.2 赋值运算符

赋值运算符有=、+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=、??等。

赋值操作符的左操作数必须是一个变量、属性访问器或索引访问器的表达式，赋值结果将一个新的数值存放在左操作数所在的内存空间中。

在 C# 中，可以对变量进行连续赋值，这时赋值操作符是右关联的，这意味着从右向左操作符被分组，如 x=y=z 等价于 x=（y=z）。

如果操作符两侧的操作数类型不一致，则应先进行类型转换。

给变量赋值的表达式称为赋值表达式，赋值表达式的值用于给变量赋值。

赋值操作符的运算对象、运算法则及运算结果如表 1.4 所示。

表 1.4　　赋值操作符的运算规则

名　称	操　作　符	运 算 规 则	意　义	运 算 对 象	运 算 结 果
赋值	=	将表达式赋值给变量	将右边的值给左边	任意类型	任意类型
加赋值	+=	a+=b	a =a+b	数值型（整型、实数型等）	数值型（整型、实数型等）
减赋值	-=	a-=b	a =a-b		
除赋值	/=	a/=b	a =a/b		
乘赋值	*=	a*=b	a =a*b		
模赋值	%=	a%=b	a =a%b	整型	整型
位与赋值	&=	a&=b	a =a&b	整型或字符型	整型或字符型
位或赋值	|=	a|=b	a =a|b		
右移赋值	>>=	a>>=b	a =a>>b		
左移赋值	<<=	a<<=b	a =a<<b		
异或赋值	^=	a^=b	a =a^b		

在 C# 中，所有赋值和自反赋值操作符的优先级都是一样的，比所有其他操作符的优先级都低，是优先级最低的操作符。

示例 01-04　赋值和自反赋值操作符的优先级

下面的示例代码演示了赋值和自反赋值操作符的优先级。

代码 01-04　光盘位置：光盘\mingrisoft\sl\01\01_04

```
static void Main(string[] args)
{
    int i;                                          //定义整型
    double d;                                       //定义双精度型
    i = 5;                                          //赋值
    d = i;
    d += i;                                         //加赋值
    Console.WriteLine("i 是 {0},d 是 {1}", i, d);
}
```

输出结果：

```
i 是 5, d 是 10
```

d +=i;等于 d=d+i; 以下 -= 、 *= 、 /= 、 %= 、&=、|=、^=、 <<= 、>>=与之相同。

“??”是一种比较特殊的赋值运算符，含义是如果“??”运算符的左操作数非空，该运算符将返回左操作数，否则返回右操作数。

示例 01-05　“??”操作符的使用

下面的示例代码演示了如何对 *y* 用“??”运算符进行赋值。

代码 01-05　光盘位置：光盘\mingrisoft\sl\01\01_05

```
static void Main(string[] args)
{
    int x = null;                                   //定义一个值为 null 的整型变量
    int y = x ?? -1;                                //如果 x 是 null, 则 y = -1,否则 y = x
    Console.WriteLine("y 是{0}", y);
}
```

输出结果：

```
y 是-1
```

1.5.3　关系运算符

==、!=、<、>、<=和>=等操作符称为关系操作符，关系操作符都是二元操作符，左右操作

数都是表达式，设左操作数为 a，右操作数为 b，则 a<b 构成一个关系表达式，a 的值若小于 b，则 a<b 关系表达式的值为 true，否则为 false，其他关系如表 1.5 所示。

表 1.5 关系操作符运行结果

a-b	a<b	a<=b	a>b	a>=b
正数	False	False	True	True
零	False	True	False	True
负数	True	True	False	False

例如，获取两个数中的最大值。代码如下：

```
int a = 5;
int b = 3;
int c = a >= b ? a : b;                    //获取两个数中的最大值，并将结果赋值给 c
```

例如，判断两个字符串是否相等。代码如下

```
string str1 = "mr";
string str2 = "mr";
if (str1 == str2)                          //判断两个字符串是否相等（区分大小写）
    Console.WriteLine("相同");
```

在这里要说明的是，字符串之间的比较只能用==和!=运算符。

1.5.4 逻辑运算符

&、^、! 和|操作符称为逻辑操作符，用逻辑操作符把运算对象连接起来使其符合 C#语法的表达式称为逻辑表达式。逻辑操作符“!”和“^”只作用于其后的操作数，故称为一元操作符。而“&&”、“||”、“&”和“|”为二元操作符，这 4 个操作符用于表达式，产生一个 true 或 false 逻辑值。

逻辑操作符对于表达式 a 和 b 的操作如表 1.6 所示。

表 1.6 逻辑操作符运行结果

A	b	a&b 或 a&&b	a|b 或 a||b	!a	a^b
False	False	False	False	True	False
False	True	False	True	True	True
True	False	False	True	False	True
True	True	True	True	False	False

1．| 或 || 运算符

二元 | 运算符是为整型和 bool 类型预定义的。对于整型，| 计算操作数的按位“或”结果。对于 bool 操作数，| 计算操作数的逻辑“或”结果；也就是说，当且仅当两个操作数均为 false 时，结果才为 false。

运算符 || 执行 bool 操作数的逻辑“或”运算时，仅在必要时才计算第二个操作数。

|| 运算符与 | 运算符的区别在于，如果第一个操作数为 true，则不计算第二个操作数（因为不论第二个操作数为何值，“或”操作的结果都为 true）。这被称作为“短路”计算。

例如，对两个整数进行或运算。代码如下：

```
int a = 10;
int b = 7;
int c = b | a;                             //c 的值为 15
```

在以上代码中，c 的值为 15，下面用图 1.1 来说明 10 和 7 或运算的过程。

例如，当 a 大于 b，或者 b 大于 0 时，执行相应的操作。代码如下：

```
int a = 7;
int b = 1;
if (a > b || b > 0)
{
```

```
    Console.WriteLine("条件成立");
}
```

在以上代码中应用的是 || 运算符，当 a>b 为 true 时条件成立，不对 b>0 进行判断。如果用 | 运算符进行或运算，必须对 a > b 和 b > 0 这两个表达式进行判断。

10的二进制为	0 0 0 0 1 0 1 0
7的二进制为	0 0 0 0 0 1 1 1
或运算	0 0 0 0 1 1 1 1

图 1.1　10 和 7 的或运算

2．&或&&运算符

&运算符既可作为一元运算符也可作为二元运算符。&运算符是为整型和 bool 类型预定义的。对于整型，&计算操作数的逻辑按位“与”。对于 bool 操作数，&计算操作数的逻辑“与”；也就是说，当且仅当两个操作数均为 true 时，结果才为 true。

运算符&&执行 bool 操作数的逻辑“与”运算时，仅在必要时才计算第二个操作数。

&&运算符与&运算符的区别在于，如果第一个操作数为 false，则不计算第二个操作数（因为不论第二个操作数为何值，“与”操作的结果都为 false）。这被称作为“短路”计算。

例如，对两个整数进行与运算。代码如下：

```
int a = 5;
int b = 7;
int c = b & a;                                    //c 的值为 5
```

在以上代码中，c 的值为 5，下面用图 1.2 来说明 5 和 7 与运算的过程。

5的二进制为	0 0 0 0 0 1 0 1
7的二进制为	0 0 0 0 0 1 1 1
与运算	0 0 0 0 0 1 0 1

图 1.2　5 和 7 的与运算

例如，判断 a 是否在 0~100 之间。代码如下：

```
int a = 7;
if (a >= 0 && a <= 100)
{
    Console.WriteLine("a 在 0~100 之间");
}
```

在以上代码中应用的是&&运算符，当 a>b 为 false 时条件不成立，不对 b>0 进行判断。如果用&运算符进行与运算，必须对 a > b 和 b > 0 这两个表达式进行判断。

3．!运算符

逻辑非运算符!是对操作数求反的一元运算符。为 bool 定义了该运算符，当且仅当操作数为 false 时才返回 true。

例如，当 a 不等于 0 时，执行相应的操作。代码如下：

```
int a = 7;
if (!(a == 0))
{
    Console.WriteLine("条件成立");
}
```

4．^运算符

二元 ^ 运算符是为整型和 bool 类型预定义的。对于整型，^ 将计算操作数的按位“异或”。对于 bool 操作数，^ 将计算操作数的逻辑“异或”；也就是说，当且仅当只有一个操作数为 true 时，结果才为 true。

例如，获取 5 和 7 的异或值。代码如下：

```
int a = 5;
int b = 7;
int c = a ^ b;                                    //c 的值为 2
```

在以上代码中，c 的值为 2，下面用图 1.3 来说明 5 和 7 的异或过程。

5的二进制为	0 0 0 0 0 1 0 1
7的二进制为	0 0 0 0 0 1 1 1
异或后为	0 0 0 0 0 0 1 0

图 1.3　5 和 7 的异或过程

1.5.5　位运算

位操作符将操作数看作是一个二进制位的集合，每个二进制位可以取 0 或 1。位操作符允许程序员测试或设置单个二进制或一组二进制位。C# 中的位操作符及其功能如表 1.7 所示。

表 1.7　　位操作符

操　作　符	功　　能
~	取补
<<	左移
>>	右移
&	位与
^	位异或
!	位或

1．～运算符

～运算符对操作数执行按位求补运算，其效果相当于反转每一位。按位求补运算符是为 int、uint、long 和 ulong 类型预定义的。

例如，对十进制数进行求补运算。代码如下：

```
int a = 5;
Console.WriteLine(~a);
```

运算结果为：

```
-6
```

2．<< 运算符

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。第二个操作数的类型必须是 int。

例如，将整数 7 向左移二位。代码如下：

```
int a = 7;
int b = a << 2;                                    //b 的值为 28
```

在以上代码中，b 的值为 28，下面用图 1.4 来说明将整数 7 向左移二位的过程。

7的二进制为	0 0 0 0 0 1 1 1
向左移两位	0 0 0 1 1 1 0 0
	16　　12
	28

图 1.4　整数 7 向左移二位

3．>>运算符

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。

例如，将整数 10 向右移二位。代码如下：

```
int a = 10;
int b = a >> 2;                                    //b 的值为 2
```

在以上代码中，b 的值为 2，下面用图 1.5 来说明将整数 10 向右移二位的过程。

10的二进制为	0 0 0 0 1 0 1 0
向右移两位	0 0 0 0 0 0 1 0
	0 2
	2

图 1.5 整数 10 向右移二位

交叉链接：关于位运算&、^和|的介绍可参见 1.5.4 小节。

1.5.6 其他特殊运算符

1．条件运算符

条件运算符（?:）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，则计算第一个表达式并以它的计算结果为准；如果为 false，则计算第二个表达式并以它的计算结果为准。例如：

```
int a = 4;
int b = 2;
int c = a > b ? a / b : b / a;
```

上面的代码首先定义了两个变量，对它们进行条件运算，如果 a>b，则进行 a / b 的运算；否则进行 b / a 的运算，并将结果返回给变量 c。

2．new 运算符

new 运算符用于创建一个新的类型实例，它有如下 3 种形式。

（1）对象创建表达式，用于创建一个类类型或值类型的实例。

（2）数组创建表达式，用于创建一个数组类型的实例。

（3）代表创建表达式，用于创建一个新的代表类型的实例。

例如，用 new 运算符创建一个类。代码如下：

```
class MyClass                                          //自定义的类
{
    public void pp()                                   //自定义方法
    {
        Console.WriteLine("调用类中的方法");
    }
}
static void Main(string[] args)
{
    MyClass myclass = new MyClass();                   //实例化一个类
    myclass.pp();                                      //调用类中的方法
}
```

例如，用 new 运算符对整型变量进行初始化。代码如下：

```
int n = new int();
```

注意：new 运算符表示创建一个类的实例，但不一定动态分配内存。

3．typeof 运算符

typeof 运算符用于获得系统原型对象的类型，也就是 Type 对象。Type 类包含值类型和引用类型的信息。typeof 运算符可以在 C# 各种位置使用，以找出关于引用类型和值类型的信息。

例如，获得 int 类型的类型信息。代码如下：

```
Type myType = typeof(int);
```

以上代码中 myType 的值为 System.Int32。

4．增量、减量运算符

（1）增量,运算符。

增量运算符（++）将操作数加 1。增量运算符可以出现在操作数之前或之后。若为前缀增

量操作，则操作的结果是操作数加 1 之后的值。若为后缀增量操作，则运算的结果是操作数加 1 之前的值。

示例 01-06　增量运算符的使用

下面的示例代码演示了++x 和 x++的区别。

代码 01-06　光盘位置：光盘\mingrisoft\sl\01\01_06

```
static void Main(string[] args)
{
    double x;
    x = 2.5;
    Console.WriteLine(++x);                    //前缀增量操作
    x = 2.5;
    Console.WriteLine(x++);                    //后缀增量操作
    Console.WriteLine(x);
    x = 2.5;
    Console.ReadLine();
}
```

输出结果：

```
3.5
2.5
3.5
```

（2）减量运算符。

减量运算符 (--) 将操作数减 1。减量运算符可以出现在操作数之前或之后。若为前缀减量操作，则操作的结果是操作数减 1 之后的值。若为后缀增量操作，则运算的结果是操作数减 1 之前的值。

示例 01-07　减量运算符的使用

下面的示例代码演示了--x 和 x--的区别。

代码 01-07　光盘位置：光盘\mingrisoft\sl\01\01_07

```
static void Main(string[] args)
{
    double x;
    x = 2.5;
    Console.WriteLine(--x);                    //前缀减量操作
    x = 2.5;
    Console.WriteLine(x--);                    //后缀减量操作
    Console.WriteLine(x);
    x = 2.5;
    Console.ReadLine();
}
```

输出结果：

```
1.5
2.5
1.5
```

1.5.7　运算符的优先级

当表达式包含多个运算符时，运算符的优先级控制着运算符的计算顺序。例如，表达式“x+y*z”按“x+（y*z）”计算，因为运算符“*”的优先级比“+”高。运算符的优先级由运算符的关联语法产生式的定义确定。例如，一个增量表达式由运算符“+”或“-”分隔的乘法表达式组成，因此给运算符“+”和“-”赋予的优先级比运算符“*”、“/”和“%”低。表 1.8 给出了运算符优先级顺序及结合性。

表 1.8 运算符的优先级

类　别	操 作 符	优 先 级	结 合 性
基本	x.y、f(x)、a[x]、x++、x--、new、typeof、checked、unchecked	1	自右向左
单目	+、-、!、~、++、--、(T)x、~	2	自左向右
乘除	*、/、%	3	自左向右
加减	+、−	4	自左向右
移位	<<、>>	5	自左向右
比较	<, >, <=, >=, is, as	6	自左向右
相等	==, !=	7	自左向右
位与	&	8	自左向右
位异或	^	9	自左向右
位或	|	10	自左向右
逻辑与	&&	11	自左向右
逻辑或	||	12	自左向右
条件	?:	13	自右向左
赋值	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	14	自右向左

1.6 枚举与结构

1.6.1 枚举

枚举（enum）是值类型的一种特殊形式，它从 System.Enum 继承而来，并为基础类型的值提供替代名称。枚举类型有名称、基础类型和一组字段。基础类型必须是一个内置的有符号（或无符号）整数类型（如 Byte、Int32 或 UInt64）。字段是静态文本字段，其中的每一个字段都表示常数。同一个值可以分配给多个字段。出现这种情况时，必须将其中某个值标记为主要枚举值，以便进行反射和字符串转换。

用户可以将基础类型的值分配给枚举，反之亦然（运行库不要求强制转换）；也可创建枚举的实例，并调用 System.Enum 的方法以及对枚举基础类型定义的任何方法。

对于枚举还有以下附加限制。

- 枚举不能定义自己的方法。
- 枚举不能实现接口。
- 枚举不能定义属性或事件。

枚举可以分为传统枚举和位域两种形式，下面对这两种形式分别进行说明。

（1）传统枚举。

枚举声明的语法格式如下：

```
enum 枚举名称[:基础类型]
{
    枚举成员 1[=整型变量],
    枚举成员 2[=整型变量|枚举成员 1],
    枚举成员 n[=整型变量|枚举成员 n-1]
}
```

注意：在以上语法格式中，枚举成员可以用中文命名。

在没有指定特定的枚举类型时，默认枚举元素的基础类型为 int，默认状态下，将 0 赋值给枚举类型对象的第一个元素。

例如，定义一个无基础类型，并且没有赋值的枚举变量。代码如下：

```
enum Season
{
    spring,
    summer,
    autumn,
    winter
}
```

Season 枚举的各成员值为 spring=0，summer=1，autumn=2，winter=3。其基础类型为 int。

在定义枚举类型后，可以对枚举成员的值随意设置，也可以用前面的枚举成员设置后面的枚举成员。

例如，定义一个 long 类型的已赋值的枚举变量。代码如下：

```
enum InfoValue : long
{
    A = 1,
    B,
    C,
    D = A,
    E,
    F
}
```

枚举型 Season 的各成员值为 A=1，B=2，C=3，D=1，E=2，F=3。

注意：在设置枚举成员的值时，不可以用后面的枚举成员设置前面的枚举成员，这样做虽然没有语法错误，但没有任何意义，该枚举成员的值将根据前一个枚举成员的值进行设置。

（2）位域。

Flags 属性表示一种特殊的枚举，称为位域。运行库本身不区分传统枚举与位域，但编程语言要区分二者。区分二者时，可以对位域（而不是枚举）使用位操作符以产生未命名的值。枚举一般用于列出惟一的元素，如一周的各天、国家、地区和名称等。位域一般用于列出可能联合发生的质量或数量。

位域通常用于由可组合出现的元素组成的列表，而枚举常数通常用于由互相排斥的元素组成的列表。因此，位域设计为通过按位“或”运算组合来生成未命名的值，而枚举常数则不是。

下面用一个例子来说明枚举常数和位域在使用上的区别。代码如下：

```
enum SingleHue : short                                  //定义一个传统枚举
{
    Black = 0,
    Red = 1,
    Green = 2,
    Blue = 4
};
[FlagsAttribute]                                        //用 FlagsAttribute 属性表示当前定义的是位域
enum MultiHue : short                                   //定义一个位域
{
    Black = 0,
    Red = 1,
    Green = 2,
    Blue = 4
};
static void Main(string[] args)
{
    Console.WriteLine("传统枚举型：");
    for (int val = 0; val <= 8; val++)                  //用循环语句按指定的值输出枚举成员
        Console.WriteLine("{0,3} - {1}", val, ((SingleHue)val).ToString());
    Console.WriteLine("位域：");
    for (int val = 0; val <= 8; val++)                  //用循环语句按指定的值输出位域成员
        Console.WriteLine("{0,3} - {1}", val, ((MultiHue)val).ToString());
    Console.ReadLine();
}
```

运行结果如图 1.6 所示。

图 1.6 运行结果

在图 1.6 中可以观察到，在传统枚举中因为没有设置值为 3、5、6、7、8 的枚举成员，所以并不能输出相应的枚举成员名。而位域则不然，其值可以通过元素成员的组合来获取，如 3 可以通过 1 和 2 来进行组合，5 可以通过 1 和 4 来进行组合，依此类推。

示例 01-08 域和传统枚举的使用

下面的示例代码演示了如何使用位域和传统枚举。

代码 01-08 光盘位置：光盘\mingrisoft\sl\01\01_08

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;                                   //添加 Hashtable 类的命名空间
namespace EnumTest
{
    class Program
    {
        public enum SomeRootVegetables                      //设置枚举型
        {
            山葵,
            萝卜,
            甘蓝
        };
        [Flags]                                             //设置位域
        public enum Seasons
        {
            无 = 0,
            夏天 = 1,
            秋天 = 2,
            冬天 = 4,
            春天 = 8,
            所有季节 = 夏天 | 秋天 | 冬天 | 春天,
        };
        static void Main(string[] args)
        {
            Hashtable AvailableIn = new Hashtable();        //实例化 Hashtable 类
            //设置 Hashtable 类的键/值对
            AvailableIn[SomeRootVegetables.山葵] = Seasons.所有季节;
            AvailableIn[SomeRootVegetables.萝卜] = Seasons.春天;
```

```
            AvailableIn[SomeRootVegetables.甘蓝] = Seasons.春天 | Seasons.秋天;
            //声明一个 Seasons 类型的数组，记录枚举值
            Seasons[] seasons = new Seasons[] { Seasons.冬天, Seasons.春天, Seasons.夏天, Seasons.秋天 };
            for (int i = 0; i < seasons.Length; i++)                    //遍历季节
            {
                Console.WriteLine("\r\n 下面的蔬菜被收获在 " + seasons[i].ToString("G") + ":");
                foreach (DictionaryEntry e in AvailableIn)              //输出指定季节所收获的蔬菜
                {
                    if (((Seasons)e.Value & seasons[i]) > 0)            //如果当前为季节
                                                                        //显示相应的蔬菜名
                        Console.WriteLine("\t" + ((SomeRootVegetables)e.Key).ToString("G"));
                }
            }
            Console.ReadLine();
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 1.7 所示。

图 1.7　运行结果

1.6.2　结构

结构是值类型，该类型将多个不同类型的成员组成一种新的类型，与类相同的是，结构成员的默认可访问级别也是 private（私有），如果需要调用，必须将其声明为 public（公有）。如果从结构创建一个对象并将该对象赋给某个变量，则该变量包含结构的全部成员。复制包含结构的变量时，将复制所有数据，对新副本所做的任何修改都不会改变旧副本的数据。由于结构不使用引用，因此结构没有标识，它无法区分具有相同数据的两个值类型实例。C#中的所有值类型都继承自 ValueType，而副本继承自 Object。

结构可以包含构造函数、常量、字段、方法、属性、索引器、运算符、事件和嵌套类型等。但是，如果同时需要上述几种成员，则应当将类型更改为类。结构的定义和类的定义基本上是一致的，结构是使用 struct 关键字定义的。其语法格式如下：

```
struct 结构名称
{
    //结构成员
}
```

结构具有以下特点。

- 结构是值类型，而类是引用类型。
- 与类不同，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，而且不能作为其他结构或类的基类。
- 结构可以实现接口。

● 结构可用 null 值对结构成员进行赋值。

为了更好地了解结构，定义一个描述公司基本信息的结构 CoConfiguration。代码如下：

```
struct CoConfiguration
{
    public string CoName;
    public Object CoLeaguer;
    public string CoAddress;
    public int CoPhone;
}
```

注意：要想对结构中的成员进行操作，必须将其设置为 public（公有成员），否则将无法调用结构成员。

在实例化结构时，可以通过 new 关键字进行实例化，也可以不通过 new 关键字进行实例化。代码如下：

```
CoConfiguration coconf = new CoConfiguration();        //用 new 关键字进行实例化
CoConfiguration coconf;                                //不用 new 关键字进行实例化
```

如果没有通过 new 关键字对结构进行实例化，结构成员将不被初始化。这时，在使用成员前必需对其进行赋值。代码如下：

```
CoConfiguration coconf;
coconf.CoName = "mrsoft";
Console.WriteLine(coconf.CoName);
```

以下代码是错误的：

```
CoConfiguration coconf;
Console.WriteLine(coconf.CoName);
```

示例 01-09　在结构中使用结构

下面的示例代码演示了如何在结构中使用其他结构。

代码 01-09　光盘位置：光盘\mingrisoft\sl\01\01_09

```
struct CoConfiguration                                          //定义结构
{
    public string CoName;
    public Object CoLeaguer;
    public string CoAddress;
    public int CoPhone;
}
struct Leaguer                                                  //定义结构
{
    public string Mgr;
    public string BranchMgr;
    public string employee;
}
static void Main(string[] args)
{
    CoConfiguration coconf = new CoConfiguration();             //实例化结构
    Leaguer leaguer;                                            //实例化结构
    leaguer.Mgr = "赛某";                                       //设置结构成员值
    leaguer.BranchMgr = "刘某";                                 //设置结构成员值
    leaguer.employee = "孙某";                                  //设置结构成员值
    coconf.CoName = "mrsoft";                                   //设置结构成员值
    coconf.CoLeaguer = leaguer;                                 //在结构中存入子结构
    coconf.CoAddress = "吉林省长春市";
    coconf.CoPhone = 84978982;
    string branchMgr = ((Leaguer)coconf.CoLeaguer).BranchMgr;   //获取子结构中的成员值
    Console.WriteLine(coconf.CoName + "公司，部门经理：" + branchMgr);   //显示结构中的数据
}
```

运行结果为：

```
mrsoft 公司，部门经理：刘某
```

在结构中只能创建有参数的构造函数。代码如下：

```
struct CoConfiguration
{
    public CoConfiguration(string Name, Object Leaguer, string Address, int Phone)      //定义构造函数
    {
        CoName = Name;                                                                  //参数赋值
        CoLeaguer = Leaguer;
        CoAddress = Address;
        CoPhone = Phone;
    }
    public string CoName;                                                               //定义结构成员
    public Object CoLeaguer;
    public string CoAddress;
    public int CoPhone;
}
```

示例 01-10　在结构中定义构造函数

下面的示例代码演示了如何在结构中定义构造函数。

代码 01-10　光盘位置：光盘\mingrisoft\sl\01\01_10

```
struct CoConfiguration
{
    public CoConfiguration(string Name, Object Leaguer, string Address, int Phone)      //定义构造函数
    {
        CoName = Name;                                                          //参数赋值
        CoLeaguer = Leaguer;
        CoAddress = Address;
        CoPhone = Phone;
    }
    public string CoName;                                                       //定义结构成员
    public Object CoLeaguer;
    public string CoAddress;
    public int CoPhone;
}
struct Leaguer
{
    public Leaguer(string M, string BM, string E)                               //定义构造函数
    {
        Mgr = M;                                                                //参数赋值
        BranchMgr = BM;
        Employee = E;
    }
    public string Mgr;                                                          //定义结构成员
    public string BranchMgr;
    public string Employee;
}
static void Main(string[] args)
{
    Leaguer leaguer = new Leaguer("赛某", "刘某", "孙某");                      //在实例化时通过构造函数赋值
    //在实例化时通过构造函数赋值
    CoConfiguration coconf = new CoConfiguration("mrsoft", leaguer, "吉林省长春市", 84978982);
    string branchMgr = ((Leaguer)coconf.CoLeaguer).BranchMgr;                  //获取子结构中的信息
    Console.WriteLine(coconf.CoName + "公司，部门经理：" + branchMgr);          //显示数据
}
```

运行结果为：

```
mrsoft 公司，部门经理：刘某
```

1.7 程序编写规范

1.7.1 代码书写规则

代码的书写规则通常对应用程序的功能没有影响，但它们有助于对源代码的理解。养成良好的编码书写习惯对于软件的开发和维护都是很有益的，下面介绍一些代码书写规则。

- 建立标准的缩进大小（如 4 个空格），并一直使用此标准，用规定的缩进对齐代码节，沿逻辑结构行缩进代码。没有缩进，代码将变得难以理解。
- 避免写过长的代码，如果代码的长度超过一行的宽度，将其写为两行。
- 花括号（{}）需和括号外的代码对齐。
- 把所有系统框架提供的名称空间组织到一起，把第三方提供的名称空间放到系统名称空间的下面。
- 避免通过全限定方式使用类型名称，在程序开始时使用 using 关键字导入命名空间，可以帮助找出哪一段代码产生异常，并给用户发出特定的错误信息。
- 所有的类成员变量应该被声明在类的顶部，并用一个空行把它们和方法以及属性的声明区分开。
- 声明一个局部变量时，局部变量的位置要与使用位置靠近。
- 当一行被分为几行时，应将“+”串联运算符放在每一行的末尾。
- 不要使用 goto 系列语句，除非用在跳出深层循环时。
- if 语句中，总使用一对{}把下面的语句块包含起来，即使只有一条语句也是如此。
- 编写 HTML 时，一种方法是建立标准的标记和属性格式，如所有标记都大写或所有属性都小写；另一种方法是坚持 XHTML 规范，以确保所有 HTML 文档都有效，应使用带引号的属性值和结束标记以方便维护。
- 避免在同一个文件中放置多个类。
- 避免使用大文件。如果一个文件里的代码超过 300 行，必需考虑将代码分开到不同类中。
- 避免写太长的方法。一个典型的方法代码在 1～25 行。如果一个方法的代码超过 25 行，应该考虑将其分解为不同的方法。
- 尽量使用接口，然后使用类实现接口，以提高程序的灵活性。
- 一个方法只完成一个任务。不要把多个任务组合到一个方法中，即使那些任务非常小。
- 避免写超过 5 个参数的方法。如果要传递多个参数，那么可以使用结构。
- 如果捕捉到异常一定要进行异常处理。如果隐藏了一个异常，将永远不知道异常到底发生了没有。
- 发生异常时，提示友好的消息给用户，但要精确记录错误的所有可能细节，包括发生的时间、相关方法和类名等。
- 只捕捉特定的异常，而不是一般的异常。
- 在开发前期不必在所有方法中捕捉异常并进行处理。异常导致程序崩溃将有助于在开发时发现大多数的错误。
- 避免书写代码量过大的 try-catch 模块。如果确实需要，那么可以为每个执行的任务编写单独的 try-catch 模块。
- 如果一个类或方法提供了泛型和非泛型版本，那么优先选择泛型版本。

- 关键的语句（包括声明关键的变量）必需要写注释，注释需和代码对齐。

1.7.2 命名规范

规范的命名有利于程序的开发和维护，下面介绍一套命名规则供读者参考。

- 对于理解应用程序的逻辑流，合理地命名是最好的一种帮助。名称应该说明“什么”而不是“如何”。通过避免使用公开基础实现（它们会发生改变）的名称，可以保留简化复杂性的抽象层。例如，可以使用 GetNextStudent()，而不是 GetNextArrayElement()。
- 因此，提供的名称要符合适用语言的规则和标准，并且长短适当。
- 在解决方案中文件的命名要和文件中的类名一致。
- 用名词或名词短语命名类。
- 用 pascal（大小写命名）规则来命名方法和类型，代码如下：

```
public class TextBox
{
    public void DataBind()
    {
    }
}
```

- 所有的成员变量前加前缀 m_，代码如下：

```
public class Database
{
    public string m_connectionString;
}
```

- 接口的名称加前缀 I。例如，声明一个名为 Icompare 的接口 interface，代码如下：

```
interface ICompare
{
    int compare();
}
```

- 方法的命名，一般将其命名为动宾短语。下面的代码命名了 ShowDialog、ShowDialog 和 GetPath 方法。

```
ShowDialog();
CreateFile();
GetPath();
```

- 局部变量的名称要有意义，不要用 x、y、z 等。例如，一个用户的名字命名如下：

```
sting userName
```

- 用有意义的名字命名 namespace，例如，产品名、公司名。一般都是公司名加产品名，例如，用 MingriSoft.OA 作为明日科技办公自动化系统的命名空间。
- 通过方法名应能看出其作用，尽量不使用容易引起误解的名字。如果名字一目了然，就无需用文档来解释方法的功能了。
- 用有意义的、描述性的词语来命名变量。
- 不用缩写。用 name、address、salary 等代替 nam、addr、sal。
- 不使用单个字母的变量，如 I、n、x 等。使用 index、temp 等，用于循环迭代的变量例外。
- 避免使用很多成员变量。声明局部变量，并传递给方法。不要在方法间共享成员变量。如果在几个方法间共享一个成员变量，那就很难知道是哪个方法在什么时候修改了它的值。
- 为自定义的异常类加上后缀 Exception。
- 使用名词、名词短语或者名词的缩写命名静态字段。
- 在类属性的名称中包含类名是多余的，如 Book.BookTitle，应该使用 Book.Title。
- 在变量名中使用互补对，如 min/max、begin/end 和 open/close。

- 不要将缩写或缩略形式用作标识符名称的组成部分。例如，使用 GetWindow，而不要使用 GetWin。
- 当类是接口的标准执行时，定义这一对类/接口组合就要使用相似的名称。两个名称的不同之处是接口名前有一个 I 前缀。
- 命名控件时，应该将控件名称的前 3 位作为缩写，如 Form 控件命名为 frmAddNew。

第 2 章

C#语句控制

- 语句
- 顺序结构程序设计
- 选择结构程序设计
- 循环结构程序设计
- 异常处理语句
- 跳转控制语句

在 C#程序设计中，语句的顺序表达了操作的执行顺序。本章将介绍如何使用流程控制语句来控制程序的执行，其中将重点介绍顺序结构、选择结构、循环结构、异常处理和跳转语句等 5 种基本结构的代码设计。

2.1 语　　句

语句（Statement）是为实现某一功能而书写的代码（Code）段。对于一条语句只能有一个意思，语句通常情况下包含表达式，表达式是一个包含文本值、简单名称或运算符及其操作数的代码段。

语句是程序指令。除非特别说明，语句都按顺序执行。语句依据其复杂程度可以分为简单语句和复合语句。

简单语句是由表达式和分号组成的。既使分号之前没有任何其他的符号或数据，它也是简单语句，通常情况下这样的语句被称为空语句。

复合语句是以“{”开始，以“}”结束的语句，也称为语句块。复合语句和简单语句在语法上具有同等的地位，即可以把复合语句看成一条语句。复合语句内的每条语句必须以分号结尾，但复合语句本身不用。例如下段代码就是复合语句，功能是交换 x、y 两个变量的值，代码如下：

```
{
    t=x;
    x=y;
    y=t;
}
```

在复合语句内，不但可以包含执行语句，还可以包含声明语句，但是在复合语句中声明的变量在复合语句外不能使用。

当复合语句外的变量和复合语句内的变量重名时，内部的变量在复合语句中有效。

2.2 顺序结构程序设计

顺序结构程序设计是按照自上而下的顺序编写程序，程序中没有结构控制语句，按照语句在程序中的位置自上而下一条一条的执行。其执行流程如图 2.1 所示。

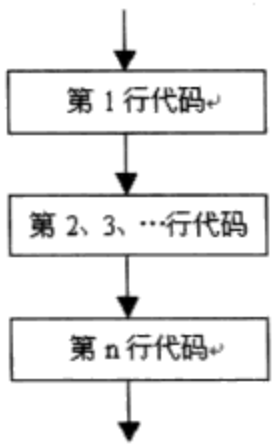


图 2.1　顺序结构的流程

示例 02-01　使用顺序结构交换变量的值

定义 3 个 Integer 类型的变量 x、y、t，动态的为变量 x、y 输入数值并交换变量的值，因 t 作为中间变量，所以不进行赋值操作。运行程序，输入任意的两个整数后显示交换后的结果。

代码如下：

代码 02-01　光盘位置：光盘\mingrisoft\sl\02\01

```
int x, y,t;                                        //定义 3 个整型变量
Console.WriteLine("输入交换前 x，y 的值：");   //输出"输入交换前 x，y 的值："
Console.Write("x=");                               //输出“x=”的信息
x = Convert.ToInt32(Console.ReadLine());           //输入 x 的值
Console.Write("y=");                               //输出“y=”的信息
y = Convert.ToInt32(Console.ReadLine());           //输入 y 的值
t = x;      //将 x 的值赋给 t
x = y;      //将 y 的值赋给 x
y = t;      //将 t 的值赋给 y
Console.WriteLine("输出交换后 x，y 的值："); //输出"输出交换后 x，y 的值："
Console.Write("x=");                               //输出“x=”的信息
Console.WriteLine(x.ToString());                   //输出 x 的值
Console.Write("y=");                               //输出“y=”的信息
Console.Write(y.ToString());                       //输出 y 的值
Console.Read();                                    //表示从标准输入流中读取下一个字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.2 所示。

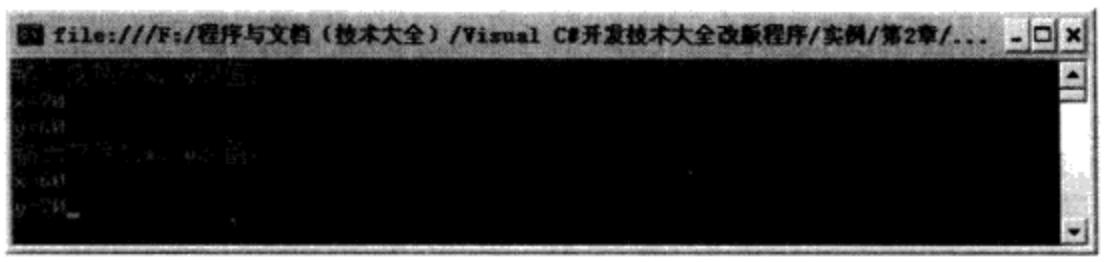


图 2.2　使用顺序结构交换变量的值

2.3　选择结构程序设计

选择结构程序设计是根据特定的条件判断来执行特定的代码，它主要包括两种类型，分别为 if 语句和 switch 语句，本节将分别对这两种类型进行介绍。

2.3.1　if 语句

if 语句用来判断条件语句是否成立，根据判定结果选择执行相对应的语句。同时，它也可以结合 else 子句使用。if 语句的流程如图 2.3 所示。

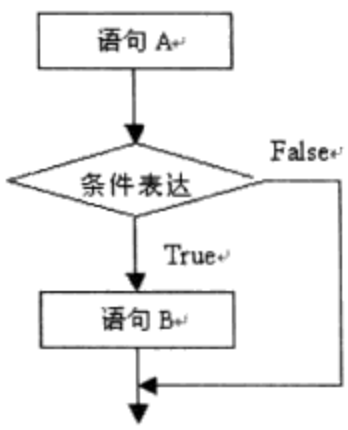


图 2.3　if 语句流程

if 语句有 4 种形式。

● 第 1 种

```
if(表达式)
{
```

```
        语句块
}
```

仅当表达式的值为 true 时，执行语句块。

● 第 2 种

```
if(表达式)
{
        语句 1
}
else
{
        语句 2
}
```

● 第 3 种

```
if(表达式 1)
{
        语句块 1
}
else if(表达式 2)
{
        语句块 2
}
......
else if(表达式 n)
{
        语句块 n
}
else
{
        语句块 n+1
}
```

在第 2 种和第 3 种形式中，如果表达式的值为 true，那么就执行 if 语句后的语句块，否则执行 else 后的语句块，两个语句块不会同时执行。

● 第 4 种

```
if(表达式 1)
{
        if(表达式 2)
        {
            语句块 1
        }
        else
        {
            语句块 2
        }
}
else
{
        if(表达式 3)
        {
            语句块 3
        }
        else
        {
            语句块 4
        }
}
```

第 4 种形式其实就是 if 语句的嵌套，用来进行多条件的判断。

示例 02-02　判断输入整数的奇偶性

定义一个 Integer 类型的变量 x，用来保存任意输入的整数。当 x 为偶数时，输出 x 为偶数

的信息提示；否则，输出 x 为奇数的信息提示。代码如下：

代码 02-02　光盘位置：光盘\mingrisoft\sl\02\02

```
Console.Write("请输入一个整数:\nx=");          //输入提示性信息
int x = Convert.ToInt32(Console.ReadLine());   //输入 x 的值
if (x % 2 == 0)                                //当 x 可以整除 2 时
{
    Console.WriteLine("x 为偶数");             //输出该数为偶数
}
else                                           //当 x 不能被 2 整除时
{
    Console.WriteLine("x 为奇数");             //输出该数为奇数
}
Console.ReadLine();                            //从标准输入流中读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.4 所示。

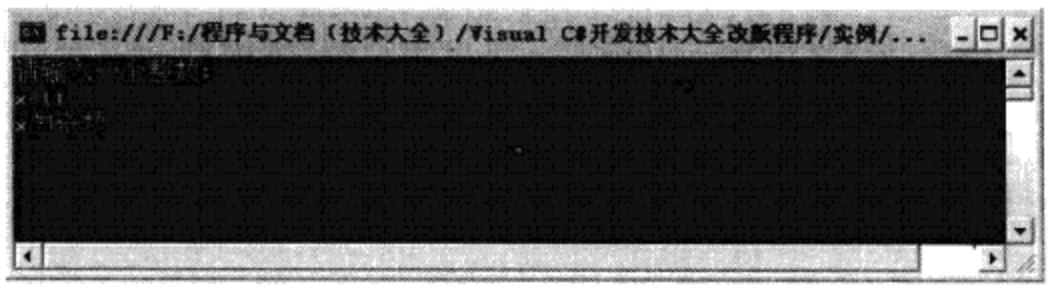


图 2.4　判断输入整数的奇偶性

说明：在嵌套的 if 语句中，else 子句与距离最近的一个没有配对的 if 关键字对应。

示例 02-03　判断是否为闰年

输入任意的一个年份，通过程序中预先设置的条件判断该年份为闰年还是非闰年。代码如下：

代码 02-03　光盘位置：光盘\mingrisoft\sl\02\03

```
Console.Write("请输入一个年份：");                    //输入提示信息
long x = Convert.ToInt64(Console.ReadLine());        //保存输入的年份
if (x % 4 == 0)                                      //当 x 可以被 4 整除时
{
    if (x % 100 != 0)                                //当 x 不能被 100 整除时
    {
        Console.WriteLine("该年份为闰年");           //输出该年份为闰年
    }
    else if (x % 400 == 0)                           //当 x 可以被 400 整除时
    {
        Console.WriteLine("该年份为闰年");           //输出该年份为闰年
    }
    else                                             //当 x 可以被 100 整除时
    {
        Console.WriteLine("该年份为非闰年");         //输出该年份为非闰年
    }
}
else                                                 //当 x 不能被 4 整除时
{
    Console.WriteLine("该年份为非闰年");             //输出该年份为非闰年
}
Console.ReadLine();                                  //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.5 所示。

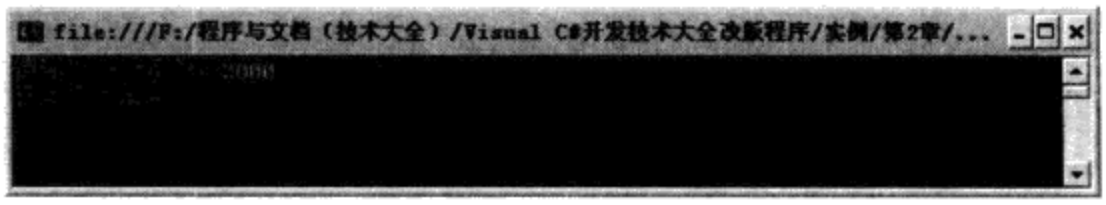


图 2.5　判断是否为闰年

示例 02-04　判断输入字符的类型

运行程序，输入一个字符判断该字符为数字、大写字母还是小写字母。代码如下：

代码 02-04　光盘位置：光盘\mingrisoft\sl\02\04

```
Console.Write("输入任意的一个数字、字母：\nx=");                    //输入提示性信息
char x = Convert.ToChar(Console.ReadLine());                        //用来保存输入的字符
if (Char.IsUpper(x))                                                //当 x 为大写字母时
{
    Console.WriteLine("x 为大写字母");                              //输出 x 为大写字母的提示
}
else if (Char.IsLower(x))                                           //当 x 为小写字母时
{
    Console.WriteLine("x 为小写字母");                              //输出 x 为小写子母的提示
}
else if (Char.IsDigit(x))                                           //当 x 为数字时
{
    Console.WriteLine("x 为数字");                                  //输出 x 为数字的提示
}
else                                                                //若为其他情况
{
    Console.WriteLine("x 既不为数字，也不为字符");                  //输出对应的信息提示
}
Console.ReadLine();                                                 //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.6 所示。

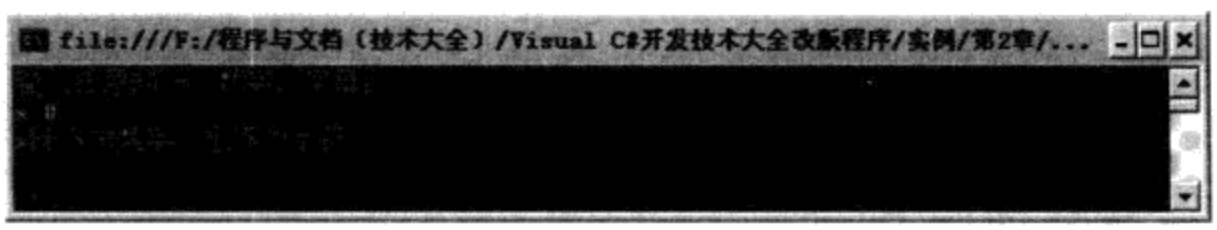


图 2.6　判断输入字符的类型

技巧：避免嵌套太深的 if-else 语句，在许多情况下，程序员理解多于 3～4 级的嵌套 if-else 语句时就会感到吃力。深度嵌套可以通过如下的方法来避免。

- 重新设计布尔表达式，可以使用逻辑操作符来完成。
- 将代码分解成更简单的方法。

2.3.2　switch 语句

虽然使用 if 语句可以处理多选择问题，但是如果嵌套过多，大量的 if…else 语句会增加程序的不可读性，此时应该使用 switch 语句。switch 语句是一个控制语句，它通过将控制传递给其内部的一个 case 语句来处理多个选择和枚举。switch 语句中有很多 case 区段，每一个 case 标记后可以指定一个常数作为标准，不能将一组常数放在一个 case 标记之后。完整的 switch 语句应该包括 switch、case、default 以及 break 等 4 个关键字。switch 语句流程如图 2.7 所示。

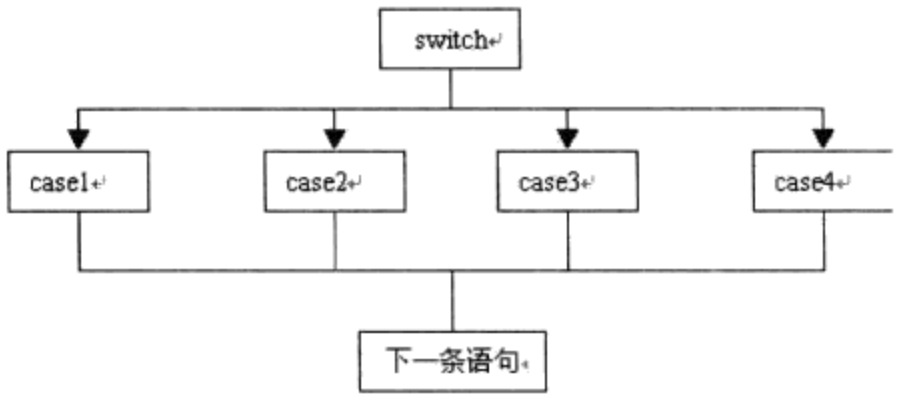


图 2.7 switch 语句流程

switch 语句的语法格式如下：

```
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 n):
        语句块 n
        break;
        default:
        语句块 n+1;
        break;
}
```

执行 switch 语句时，会将“表达式”的值和常量表达式的值进行比较，当某个 case 语句中“常量表达式”的值与“表达式”的值相等时，就执行该 case 语句后的语句块。当遇到 break 语句时，退出 switch 语句。如果所有 case 语句中常量表达式的值都不等于表达式的值，则执行 default 语句后的语句块。其中，default 语句是一个可选项，可以选择也可以不选择。

switch 语句可以包括任意数目的 case 实例，但是任何两个 case 语句都不能具有相同的值。在每一个 case 块的后面，都必须有一个跳转语句（如 break）。如果当前 case 语句中无代码时，（与 C++中的 switch 语句不同,C#不支持从一个 case 标签显式贯穿到另一个 case 标签）程序将执行下一条语句。

示例 02-05 判断月份所在的季节

输入任意的月份判断其属于哪个季节。代码如下：

代码 02-05 光盘位置：光盘\mingrisoft\sl\02\05

```
Console.Write("输入一个月份：\nX=");                    //输入提示性信息
int X = Convert.ToInt32(Console.ReadLine());            //实例化一个变量用来保存输入的月份
switch (X)
{
    case 1://当为 1 月份时
    Console.WriteLine(X.ToString() + "月份属于冬季");    //输出该月份所在的季节为冬季
    break;
   case 2://当为 2 月份时
   Console.WriteLine(X.ToString() + "月份属于冬季");     //输出该月份所在的季节为冬季
   break;
   case 3:                                              //当为 3 月份时
   Console.WriteLine(X.ToString() + "月份属于春季");     //输出该月份所在的季节为春季
   break;
```

```
case 4:                                                    //当为 4 月份时
Console.WriteLine(X.ToString() + "月份属于春季");          //输出该月份所在的季节为春季
break;
case 5:                                                    //当为 5 月份时
Console.WriteLine(X.ToString() + "月份属于春季");          //输出该月份所在的季节为春季
break;
case 6:                                                    //当为 6 月份时
Console.WriteLine(X.ToString() + "月份属于夏季");          //输出该月份所在的季节为夏季
break;
case 7:                                                    //当为 7 月份时
Console.WriteLine(X.ToString() + "月份属于夏季");          //输出该月份所在的季节为夏季
break;
case 8:                                                    //当为 8 月份时
Console.WriteLine(X.ToString() + "月份属于夏季");          //输出该月份所在的季节为夏季
break;
case 9:                                                    //当为 9 月份时
Console.WriteLine(X.ToString() + "月份属于秋季");          //输出该月份所在的季节为秋季
break;
case 10:                                                   //当为 10 月份时
Console.WriteLine(X.ToString() + "月份属于秋季");          //输出该月份所在的季节为秋季
break
case 11:                                                   //当为 11 月份时
Console.WriteLine(X.ToString() + "月份属于秋季");          //输出该月份所在的季节为秋季
break;
case 12:                                                   //当为 12 月份时
Console.WriteLine(X.ToString() + "月份属于冬季");          //输出该月份所在的季节为冬季
break;
default:                                                   //当为其他情况时
Console.WriteLine("你查询的月份不存在，请确认输入信息");   //输出提示性信息
break;
}
Console.ReadLine();                                        //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.8 所示。

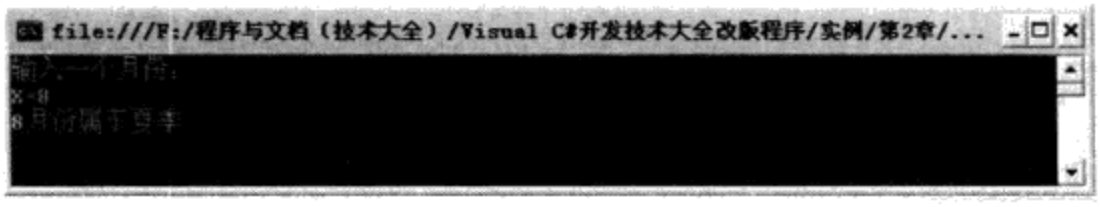


图 2.8　判断月份所在的季节

示例 02-06　在 case 标签内 goto 语句的使用

在 case 标签内使用 goto 语句跳转到另一个 case 标签，判断用户输入的值，进而完成费用的计算。程序的流程如图 2.9 所示。

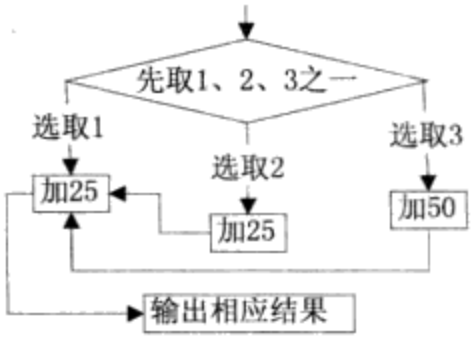


图 2.9　在 case 标签内 goto 语句流程

代码如下：

代码 02-06　光盘位置：光盘\mingrisoft\sl\02\06

```
Console.WriteLine("输出结果为：\n 选对号码: 1=小 2=大 3=很大");        //输出提示性信息
Console.Write("请输入你的选择：");                                     //输出提示性信息
string s = Console.ReadLine();                                         //从键盘输入一个字符
int n = int.Parse(s);                                                  //将字符串型转化为整型
int cost = 0;                                                          //初始化变量 cost 为 0
switch (n)
{
    case 1:                                                            //当输入的值为 1 时
    cost += 25;                                                        //将 cost 的值累加 25
    break;
    case 2:                                                            //当输入的值为 2 时
    cost += 25;                                                        //将 cost 的值累加 25
    goto case 1;                                                       //跳转到 case 1
    case 3:                                                            //当输入的值为 3 时
    cost += 50;                                                        //将 cost 的值累加 50
    goto case 1;                                                       //跳转到 case 1
    default:                                                           //当输入的值为其他情况时
    Console.WriteLine("请按要求输入信息");                              //输出错误提示信息
    break;
}
if (cost != 0)                                                         //当 cost 的值不为 0 时
{
    Console.WriteLine("应付费用为 {0}", cost);                         //输出应付费用的值
}
Console.Read();                                                        //从标准输入流读取下一个字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.10 所示。

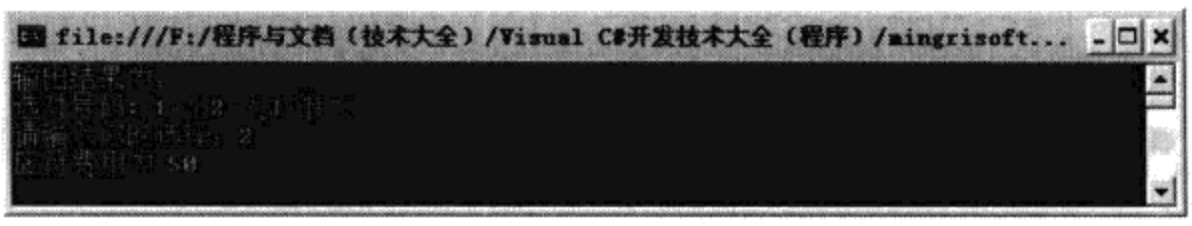


图 2.10　case 标签内使用 goto 语句

技巧：一般情况下，应将 default（缺省）语句放在最后。虽然 case 语句和 defalut 语句的出现顺序没有限制，但将 default 语句放在 switch 语句块的结尾是最常用的方式。

2.4　循环结构程序设计

循环结构程序设计在很大程度上简化了顺序结构程序设计，使程序看起来更加清晰易懂。在 C#语言中有 4 种语句可以实现循环，分别是 while 语句、do…while 语句、for 语句以及 foreach 语句。下面分别对这 4 种语句进行介绍。

2.4.1　while 语句

while 语句是用于重复执行程序代码的语句。在使用 while 语句时，只要条件表达式的值为 true，while 循环就会一直执行，直到条件表达式的值为 false 时结束。其语法格式如下：

```
while(布尔表达式)
{
    循环体
}
```

while 语句的流程如图 2.11 所示。

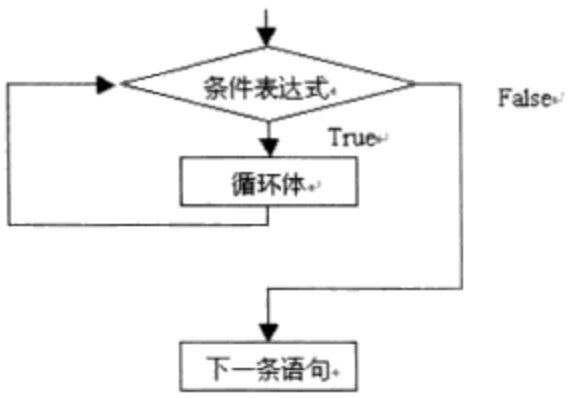


图 2.11　while 语句流程图

while 语句的语法格式如下：

```
while(条件表达式)
{
        循环体
}
```

仅当条件表达式为 true 时，程序才会重复执行 while 循环体中的语句。由于 while 表达式的判断是在每次执行循环前发生，因此 while 循环可能执行 0 次或多次；与 while 循环相似的 do 循环不同，do 循环的运行机制是先运行循环体语句，然后再进行表达式值的判断，也就是说无论表达式的值成立与否，循环体都要执行一次，因此 do 循环执行 1 次或多次。

示例 02-07　输出 50~100 的所有素数

利用 while 语句循环输出 50~100 的所有素数。代码如下：

代码 02-07　光盘位置：光盘\mingrisoft\sl\02\07

```
int Max = 100, Min = 50;                          //定义 2 个临界值 50、100
int Num = 2;                                      //设置除数的初始值
double temp;                                      //定义一个中间变量
Console.WriteLine("输出 50-100 间的所有素数：");   //输出提示性信息
while (Min <= Max)                                //当 Min 的值不大于 Max 的值时
{
    temp = Math.Sqrt(Min);                        //保存 Min 的平方根的值
    while (Num <= temp)                           //当除数的值不大于 temp 的值时
    {
        if (Min % Num == 0)                       //当 Min 不能被 Num 整除时
        {
            break;                                //跳出循环
        }
        Num++;                                    //递增变量 Num 的值
    }
    if (Num > temp)                               //当 Num 的值大于 temp 的值时
    {
        Console.Write(Min.ToString() + "   ");    //输出 Min 的值
    }
    Num = 2;                                      //重新为变量 Num 赋值
    Min += 1;                                     //使变量 Min 的值累加 1
}
Console.ReadLine();                               //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.12 所示。

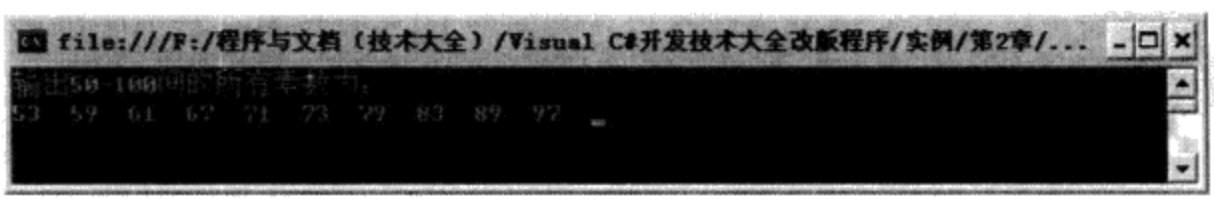


图 2.12　while 语句

技巧：当 break 语句、goto 语句、return 语句或 throw 语句将控制权转移到 while 循环之外时，可以终止该循环。若要将控制权传递给下一次迭代但不退出循环，可以通过 continue 语句实现。

2.4.2 do…while 语句

do…while 循环与 while 循环类似，只要布尔表达式为 true，循环体就会不断地重复执行，但 do…while 语句与 while 语句也有不同之处，它首先执行循环体，然后再进行条件表达式的判断，无论条件成立与否，循环体至少执行一次。

do…while 语句流程如图 2.13 所示。

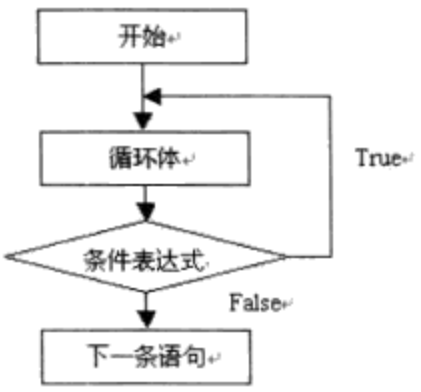


图 2.13 do…while 语句

do…while 语句的语法格式如下：

```
do
{
    循环体
}
while(布尔表达式);
```

注意：while 布尔表达式后的分号一定要写，否则出现语法错误。

示例 02-08 输出 0~100 的所有偶数

利用 do-while 语句输出 0~100 间的所有偶数，代码如下：

代码 02-08 光盘位置：光盘\mingrisoft\sl\02\08

```
Console.WriteLine("输出 0-100 间的所有偶数：");          //输出提示性信息
int Num = 0;                                            //初始化变量 Num 的值为 0
do
{
    if (Num % 2 == 0)                                   //当 Num 可以被 2 整除时
    {
        Console.Write(Num.ToString() + "    ");          //输出 Num 的值
    }
    Num++;                                              //递增 Num 的值
} while (Num <= 100);                                   //当 Num 的值不大于 100 时
Console.ReadLine();                                     //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.14 所示。

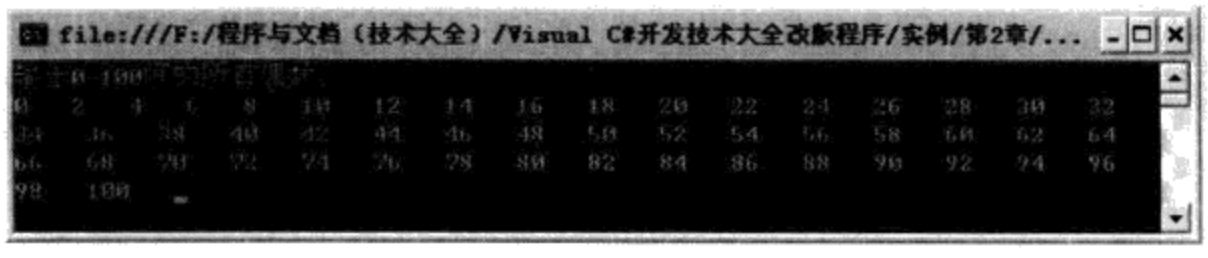


图 2.14 do…while 语句

技巧：选择 while 循环还是 do…while 循环取决于所解决的实际问题，但大部分情况下应选择 while 循环。例如，如果循环用于遍历一个数字列表，此列表可能为空，在这种情况下，循环体语句不应该被执行。如果此时使用 do…while 就会导致程序错误。

2.4.3 for 语句

for 语句是最常用的循环语句之一，它不同于 while 语句，它可以确定循环次数。在使用 for 语句时需要确定循环变量的初始值和终止值。循环变量可以是任何的合法标识符，循环变量的类型可以是整型、布尔型、字符类型、枚举类型等。

for 语句流程如图 2.15 所示。

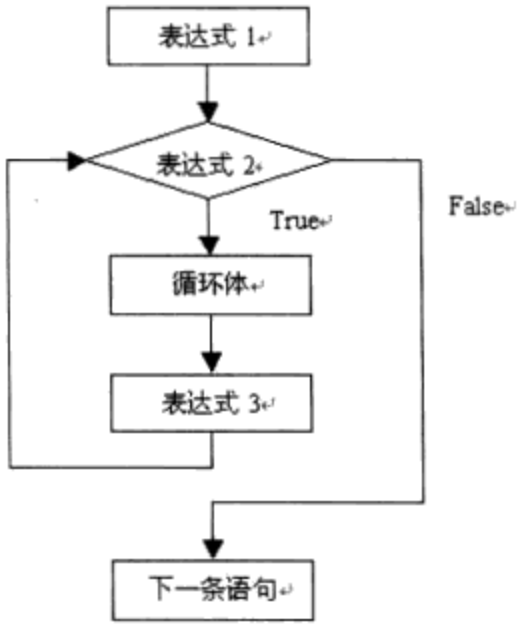


图 2.15　for 语句流程图

for 语句的语法格式如下：

```
for(表达式 1；表达式 2；表达式 3)
{
    循环体
}
```

其中 3 个表达式的内容如下。

- 表达式 1：设定 for 语句内各变量在循环开始前的初始值。
- 表达式 2：决定循环是否继续执行的条件，如果表达式 2 的值为 true，那么继续循环，否则退出循环。
- 表达式 3：每次循环结束前执行的语句，通常是使一个变量的值递增或递减，用来改变表达式 1 的值。

注意：可以使用逗号来分隔多于一个的初始语句。如果更新语句多于一个，同样也可以用逗号来分隔。分号用于分隔循环语句与循环条件、循环条件与更新语句。初始化语句、循环条件和更新语句都是可选的。例如，for(;；){}构建了一个有效的 for 循环，但必须包括两个分号。

将两个或多个表达式压缩到一个语句行中的语言符号就是逗号操作符，逗号操作符的语法格式如下。

```
for（i=0,j=10;i<=j;i++,j--）
```

逗号操作运算符将两个或多个语句组合，可将它们只看作一条语句。所以，它所放置的位置只能放置一条语句。例如，应避免将逗号操作符与下面应用场合混淆。最左边的逗号既可以是一个逗号分隔符，也可以是一个逗号操作符，编译器将无法分辨。

```
for（int i=0,j=10;i<=j;i++,j--）                    //合法
for（int i=0,int j=10;i<=j;i++,j--）                //不合法
```

示例 02-09 显示九九乘法表

利用 for 语句循环输出九九乘法表。代码如下：

代码 02-09 光盘位置：光盘\mingrisoft\sl\02\09

```
Console.WriteLine("输出九九乘法表:");                                    //输出提示性信息
for (int i = 1; i < 10; i++)                                            //从 1 循环至 9
{
    for (int j = 1; j <= i; j++)                                        //从 1 循环至 i
    {
        Console.Write(i.ToString() + "*" + j.ToString() + "=" + (i * j).ToString() + " ");   //输出公式
    }
    Console.WriteLine();                                                //输出换行符
}
Console.ReadLine();                                                     //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.16 所示。

图 2.16 九九乘法表

技巧：循环初始化和循环更新只用于属于循环条件部分的变量。通过逗号操作符可以在循环初始和循环更新处插入无限多的表达式。另外，还可以在这里填充大量的初始代码和更新语句，而不是与循环条件中的变量直接连接。由于条件表达式的测试在循环执行之前发生，因此 for 语句执行 0 次或多次。

2.4.4 foreach 语句

foreach 语句是为简化对数组或集合的循环访问而设计的，它为数组或集合中的每个元素重复地嵌入一个语句体。另外，它还可用于循环访问集合以获取所需信息，但不能用于更改集合内容以避免产生不可预知的副作用。

foreach 语句的流程如图 2.17 所示。

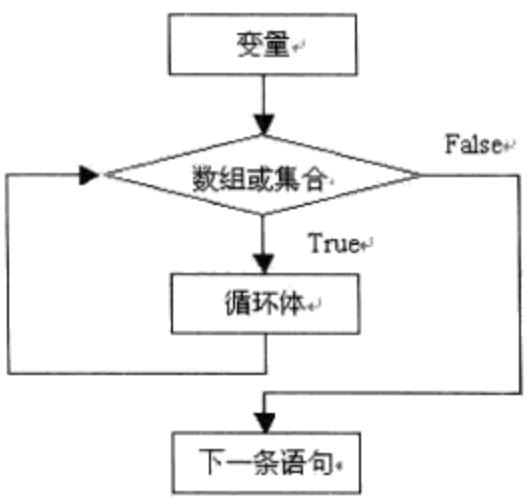


图 2.17 foreach 语句

foreach 语句语法格式如下：

```
foreach(类型  变量名  in  集合对象)
{
    语句体
}
```

嵌入语句用来控制数组或集合中的每个元素，当为集合中的所有元素完成循环后，其控制权传递给 foreach 块之后的下一条语句。

示例 02-10　输出整数数组中的全部内容

利用 foreach 输出整数数组的内容。代码如下：

代码 02-10　光盘位置：光盘\mingrisoft\sl\02\10

```
int[] Array = { 0,2,4,6,8};                          //初始化一个整型数组
Console.WriteLine("输出数组中的每一个元素:");          //输出提示性信息
foreach (int temp in Array)                          //循环遍历 Array 中的每一个元素
{
    Console.Write(temp.ToString() + "   ");          //输出数组中每一个元素的值
}
Console.ReadLine();                                  //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.18 所示。

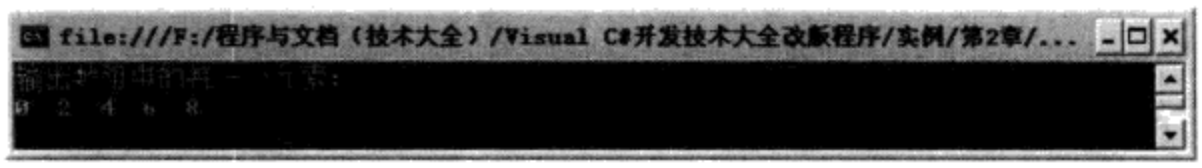


图 2.18　输出整数数组的所有内容

技巧：对具有相同特性的对象操作时，可以使用 foreach 语句，如对同一容器（groupBox）内的控件（Button）的统一操作。

2.5　异常处理语句

异常是指程序执行时遇到的错误情况或意外行为，常见的异常包括代码或调用的代码（如共享库）中有错误、操作系统资源不可用、公共语言运行库遇到意外情况（如无法验证代码）、文件操作失败、除数为 0 等。当异常发生时，用户不能消除它，只能确定如何解决它。

异常处理是 C#的一种特性，它使程序按照有序、有组织的方式捕获和处理异常。通常，异常处理用一段程序来感知和抛出异常信息，用另一段程序来处理错误。本节将帮助读者掌握异常处理语句，以便更好地解决在设计程序时发生的异常情况。

一个异常处理语句可以有如下情况。

- 有一个 finally 块，无 catch 块。
- 有一个或多个相关的 catch 块，但是无 finally 块。
- 包含一个或多个 catch 块，同时有一个 finally 块。

注意：异常类继承层次结构为 System.Object→System.Exception→System.SystemException→派生类。

基类 Exception 存在以下两种异常。

- 从 SystemException 类派生的预定义公共语言运行库异常。
- 从 ApplicationException 类派生的用户定义的应用程序异常。

2.5.1　throw 语句

throw 语句用于发出在程序执行期间出现反常情况（异常）的信号。throw 语句通常与

try-catch 或 try-finally 语句一起使用。可以使用 throw 语句显式引发异常(这里引发自定义异常)。创建用户自定义异常，可以用“Exception”作为用户自定义异常类名的结尾。

示例 02-11 判断输入数字的临界值

通过 throw 语句显式地引发异常。代码如下：

代码 02-11 光盘位置：光盘\mingrisoft\sl\02\11

```
Console.WriteLine("要求：请输入 0-10 之间的数字");          //输入提示性信息
int Num = int.Parse(Console.ReadLine().ToString());        //输入一个整型的数值
if (Num > 10 || Num < 0)                                   //当输入的数大于 10 或者小于 0 时
{
    throw new ArgumentOutOfRangeException();               //抛出异常
}
Console.WriteLine("输入数字在正常范围内");                  //给出提示性信息
Console.ReadLine();                                        //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.19 所示。

图 2.19 判断输入数字的临界值

当出现异常时，效果如图 2.20 所示。

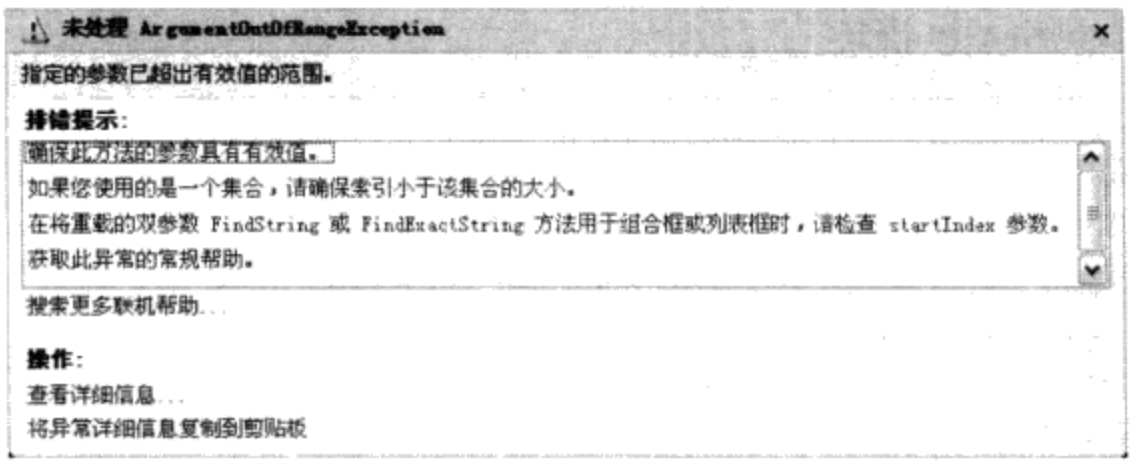


图 2.20 出现异常情况

2.5.2 try-catch 语句

try-catch 异常处理语句允许将任何可能发生异常情形的程序代码放置在 try{}代码块中进行监控，在 catch{}块内放置处理异常的程序代码，一个 try{}块即可对应一个 catch{}块又可对应多个 catch{}块，finally{}块只能有一个，同时它为可选项。

示例 02-12 try-catch 语句中写入多个 catch 块的使用

通过两个 catch 语句进行捕获异常，它们分别是 ArgumentNullException 异常和 Exception 异常。代码如下：

代码 02-12 光盘位置：光盘\mingrisoft\sl\02\12

```
static void ProcessString(string str)
{
    if (str == null)                                       //当 str 为空时
    {
        throw new ArgumentNullException();                 //抛出异常
    }
```

```
    }
    static void Main()
    {
        Console.WriteLine("输出结果为：");                          //输出提示性信息
        try
        {
            string str = null;                                      //设置 str 为空
            ProcessString(str);                                     //调用自定义方法
        }
        catch (ArgumentNullException e)                             //捕获字符串为空的异常
        {
            Console.WriteLine(e.Message);                           //输出异常信息
        }
        catch (Exception e)                                         //捕获异常
        {
            Console.WriteLine(e.Message);                           //输出异常提示信息
        }
        Console.ReadLine();                                         //从标准输入流读取下一行字符
    }
```

按<Ctrl+F5>组合键运行程序，效果如图 2.21 所示。

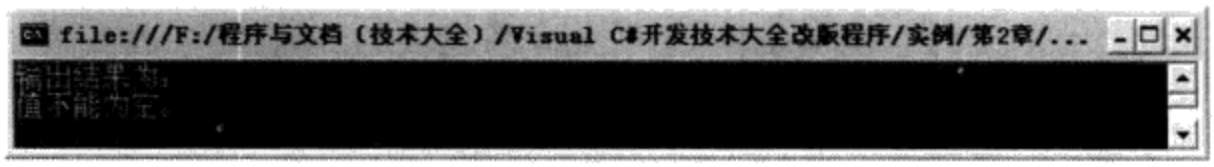


图 2.21　try-catch 语句

2.5.3　try-catch-finally 语句

异常发生时，必须处理异常，同时将控制权交给最近的异常处理语句，这就意味着不能按正常的设计思想执行。而有些资源清理程序（如关闭文件）必须执行，为了实现该功能，可以使用 finally 块。即便有异常情况发生，finally 块也可以清除 try 块中分配的任何资源，以及运行必须执行的代码。最后控制权总会交给 finally 块，与 try 块的退出方式无关。

示例 02-13　try-catch-finally 语句的使用

本示例中有一个导致异常的无效转换语句，当运行程序时，用户会收到一条运行出错的信息，但 finally 子句仍继续执行并显示输出。代码如下：

代码 02-13　光盘位置：光盘\mingrisoft\sl\02\13

```
int i = 0;                                                          //初始化一个整型变量
string s = Console.ReadLine();                                      //从键盘输入一个字符串
try
{
    i += int.Parse(s);                                              //将输入的值转化为整型和 i 相加
}
catch
{
    Console.Write("出现异常，异常提示为：\n");                       //输出提示性信息
    Console.WriteLine("输入内容必须为数字");                         //输出异常原因
}
finally
{
    Console.Write("输出结果为：\n" + "i = {0}", i);                  //输出最终结果
}
Console.ReadLine();                                                 //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.22 所示。

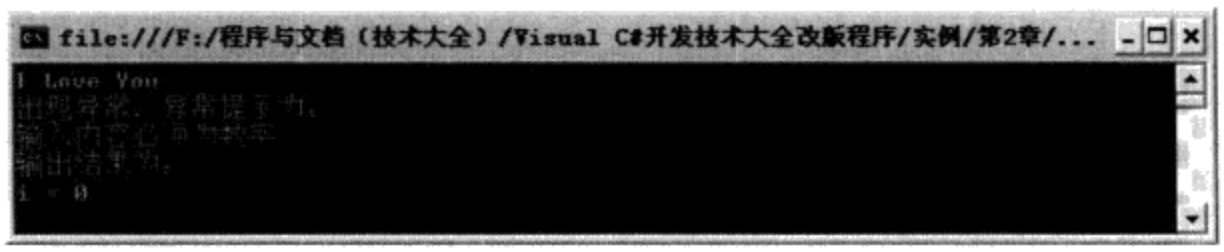


图 2.22 try-catch-finally 语句

2.6 跳转控制语句

跳转语句一般情况下用于循环中，主要是当满足某一条件时，退出循环(或者退出此次循环进入下一次循环)。使用跳转语句执行分支，将传递程序的控制权。

跳转语句主要包括以下 4 种不同的类型。

- break 语句；
- continue 语句；
- goto 语句；
- return 语句。

2.6.1 break 语句

break 语句可以用来终止一条循环语句，使控制流程跳转到循环体外的下一条语句。break 语句可以放在循环体的任意位置来实现这一目的，若要终止整个嵌套循环，可以使用 return 语句。

示例 02-14 累加数字求和

本示例主要实现的功能为：输入任意数字，从 1 累加至当前值；当输入 0 时，退出累加过程。代码如下：

代码 02-14 光盘位置：光盘\mingrisoft\sl\02\14

```
Console.WriteLine("输入进行累加的数字：");                    //输出提示输入的信息
int Num = int.Parse(Console.ReadLine());                      //从键盘中输入一个数字
int Sum = 0, temp = 0;                                        //初始化两个整型变量
while (true)                                                  //无限循环
{
    if (temp <= Num)                                          //当临时变量的值不大于输入的值时
    {
        Sum += temp;                                          //累加该值
    }
    if (Num == 0)                                             //当输入的值等于 0 时
    {
        break;                                                //退出循环
    }
    if (temp > Num)                                           //当临时变量的值大于输入的值时
    {
        break;                                                //退出循环
    }
    temp++;                                                   //递增变量 temp 的值
}
Console.WriteLine("输出进行累加的结果：");                    //输出输出时的提示性信息
Console.Write("Sum=" + Sum.ToString());                       //输出最终累加结果
Console.ReadLine();                                           //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.23 所示。

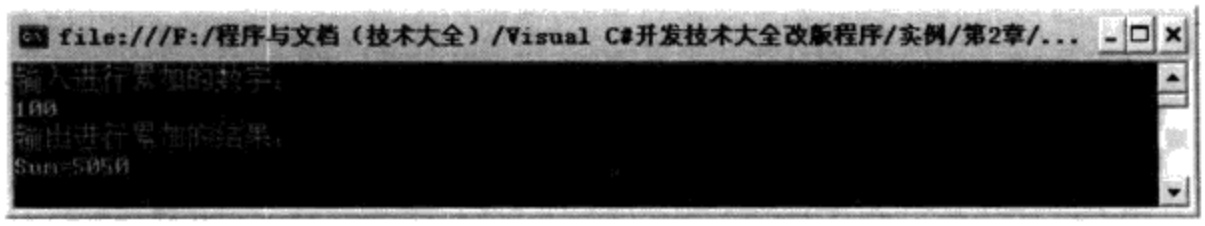


图 2.23　break 语句

示例 02-15　break 语句在 switch 语句中的使用

通过判断用户输入的值显示相应信息，代码如下：

代码 02-15　光盘位置：光盘\mingrisoft\sl\02\15

```
Console.WriteLine("请输入 1、2、3 或者 4：");                    //输出与输入有关的信息提示
int temp = int.Parse(Console.ReadLine());                        //输入一个整型字符
switch (temp )
{
    case 1:                                                      //当输入的为 1 时
    Console.WriteLine("恭喜你选择《三国演义》，谢谢合作！");      //输出用户的选择为《三国演义》
    break;
    case 2:                                                      //当输入的为 2 时
    Console.WriteLine("恭喜你选择《西游记》，谢谢合作！");        //输出用户的选择为《西游记》
    break;
    case 3:                                                      //当输入的为 3 时
    Console.WriteLine("恭喜你选择《水浒传》，谢谢合作！");        //输出用户的选择为《水浒传》
    break;
    case 4:                                                      //当输入的为 4 时
    Console.WriteLine("恭喜你选择《红楼梦》，谢谢合作！");        //输出用户的选择为《红楼梦》
    break;
    default:                                                     //当为其他情况时
    Console.WriteLine("输入信息不正确，请确认你的输入信息！");    //输出错误的信息提示
    break;
}
Console.ReadLine();                                              //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.24 所示。

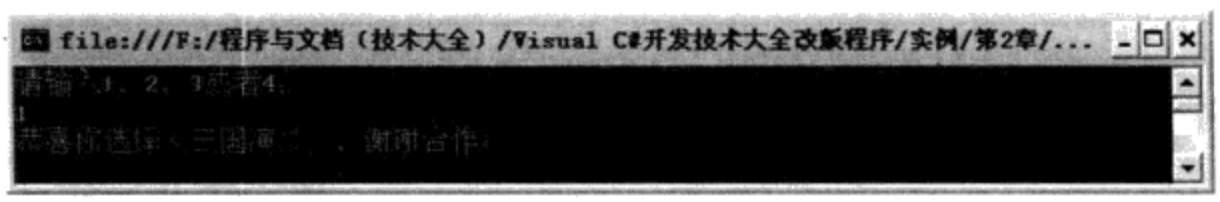


图 2.24　break 语句在 switch 语句中的使用

2.6.2　continue 语句

continue 语句的功能是结束本次循环，返回条件判断部分，重新开始下一次循环。continue 语句只能用在循环结构中。

示例 02-16　输出指定范围内的数字

计数器最初是从 1～6 进行计数，但通过将 continue 语句与表达式（i<4）一起使用，可以跳过 continue 与 for 循环体末尾之间的语句，此时仅输出 4～6 之间的数字。代码如下：

代码 02-16　光盘位置：光盘\mingrisoft\sl\02\16

```
Console.WriteLine("输出结果为：");                               //输出提示性信息
for (int i = 1; i <= 6; i++)                                     //从 1 循环至 6
{
    if (i < 4)                                                   //当 i 小于 4 时
    {
```

```
        continue;                                          //跳出本次循环进入下一层循环
    }
    Console.WriteLine(i);                                  //输出 i 的值
}
Console.ReadLine();                                        //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.25 所示。

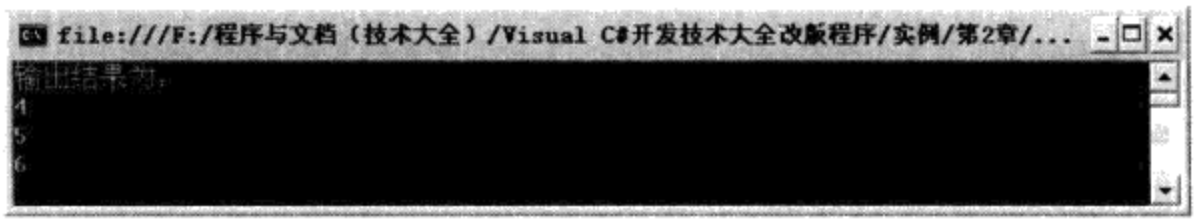


图 2.25　continue 语句

注意：continue 语句不会像 break 语句那样干涉循环体重复的次数，它只是影响某特定循环中的流程控制。

2.6.3　goto 语句

goto 语句属于无条件的跳转语句，因为 C#允许在代码行加上标签，这样就可以用 goto 语句直接跳转到这些代码行上，在代码行上必须有其他语句，否则这样做是错误的。该语句有其优缺点，优点在于可以方便地控制何时去执行指定代码，缺点在于过多地使用这个语句将降低代码的可读性。

示例 02-17　查找数组中是否存在指定内容

通过使用 goto 语句跳出嵌套循环完成数字的查找，代码如下：

代码 02-17　光盘位置：光盘\mingrisoft\sl\02\17

```
int x = 200, y = 4;                                        //初始化二维数组的行和列的长度
int count = 0;                                             //初始化一个整型变量
string[,] array = new string[x, y];                        //实例化一个二维字符串数组
for (int i = 0; i < x; i++)                                //循环遍历数组的每一行
{
    for (int j = 0; j < y; j++)                            //循环遍历数组的每一列
    {
        array[i, j] = (++count).ToString();                //为数组赋值
    }
}
Console.Write("输入查找的数字:");                          //输出查找的提示性信息
string myNumber = Console.ReadLine();                      //从键盘中输入内容
for (int i = 0; i < x; i++)                                //遍历数组中的每一行
{
    for (int j = 0; j < y; j++)                            //遍历数组中的每一列
    {
        if (array[i, j].Equals(myNumber))                  //当数组中存在输入数字时
        {
            goto Found;                                    //跳转到查找到标签
        }
    }
}
Console.WriteLine("数字{0} 无法找到.", myNumber);          //输出查找失败的信息
goto Finish;                                               //跳转到结束标签
Found:                                                     //查找到标签
Console.WriteLine("数字{0} 存在.", myNumber);              //输出查找到的结果
Finish:                                                    //结束标签
Console.WriteLine("结束查找.");                            //输出查找完成时的结果
Console.ReadLine();                                        //从标准输入流读取下一行字符
```

按<Ctrl+F5>组合键运行程序，效果如图 2.26 所示。

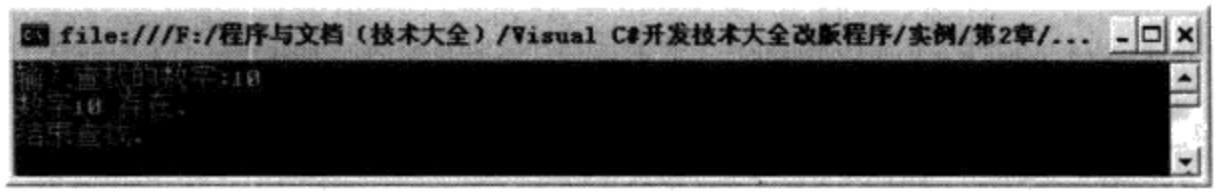


图 2.26　查找数组中是否存在指定内容

2.6.4　return 语句

return 语句用于终止它所在的方法的执行，并将控制权返回给调用方法。另外，它还可以返回一个可选值。如果方法为 void 类型，则可以省略 return 语句。

示例 02-18　求任意半径下的面积

通过键盘输入半径的值，用 CalculateArea 方法实现对面积的求解。代码如下：

代码 02-18　光盘位置：光盘\mingrisoft\sl\02\18

```
static double CalculateArea(int r)                                    //自定义计算面积方法
{
    double area = r * r * Math.PI;                                    //求面积的表达式
    return area;                                                      //返回面积的值
}
static void Main()
{
    Console.WriteLine("输入圆的半径为：");                            //输出提示输入半径的信息
    int radius = int.Parse(Console.ReadLine());                       //输入一个半径的值
    Console.WriteLine("输出圆的面积为：\n{0:0.00}", CalculateArea(radius));  //输出所在半径下的面积值
    Console.ReadLine();                                               //从标准输入流读取下一行字符
}
```

按<Ctrl+F5>组合键运行程序，效果如图 2.27 所示。

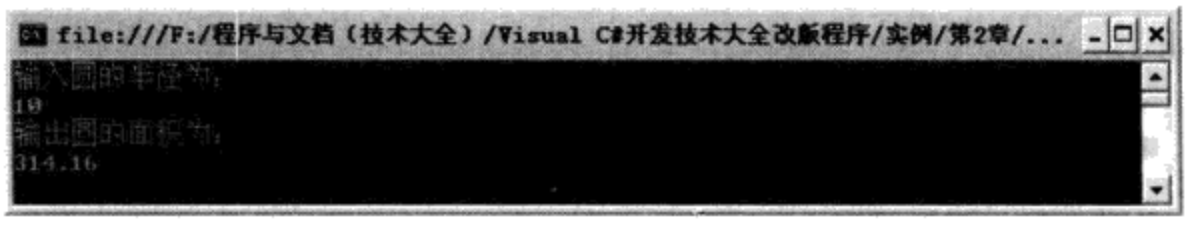


图 2.27　求任意半径下的面积

技巧：在写程序时，可以通过 return 语句判断这个方法是否需要执行下去。return 语句可以返回任何类型，也可以无返回结果。

注意：return 语句常用于方法中，有时候为了终止事件中的一些操作，也可以在事件中应用 return 语句。

第 3 章

数组与集合

- 一维数组
- 二维数组
- 多维数组
- Array 类
- 交错数组
- ArrayList 类
- Hashtable 类
- 其他集合的应用

3.1 数　　组

数组是包含若干相同类型元素的一组变量。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。数组的维数即数组的秩。数组中的每个元素都具有惟一的索引与其相对应。数组的索引从零开始。

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、二维、多维和交错数组。本章将会对一维数组、二维数组、多维数组及交错数组进行详细介绍。

3.1.1 一维数组

1．一维数组的声明

（1）声明。

一维数组即数组的维数为 1。

语法：

```
type[] arrayName;
```

参数说明如下：

type：存储数据的数据类型。

arrayName：数组名称。

例如，声明一个没有初始化的整型一维数组。代码如下：

```
int[] arr;
```

说明：数组的长度不是声明的一部分，而且数组必须在访问前初始化。数组的类型可以是基本数据类型，也可以是枚举或其他类型。

（2）初始化。

数组的初始化有很多形式。可以通过 new 运算符创建数组并将数组元素初始化为它们的默认值。例如：

```
int[] arr =new int[5];                    //arr 数组中的每个元素都初始化为 0
```

可以在声明数组时将其初始化，并且初始化的值为用户自定义的值。例如：

```
int[] arr=new int[5]{1,2,3,4,5};
```

说明：数组大小必须与大括号中的元素个数相匹配，否则会产生编辑时错误。

声明一个数组变量时可以不对其初始化，但在对数组初始化时必须使用 new 运算符。

例如：

```
string[] arrStr;
arrStr=new string[7]{"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
```

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计算数组长度，并创建数组。例如：

```
string[] arrStr={"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
```

2．一维数组的使用

当需要存储多个值时，可以使用一维数组，并通过 foreach 语句或数组的下标将值读出来。如图 3.1 所示，包括 5 个元素的数组。

示例 03-01　一维数组的使用

示例将用户输入的一组数存入数组，并进行降序排序，排序后使用 foreach 遍历数组并将数组中的元素输出。程序代码如下：

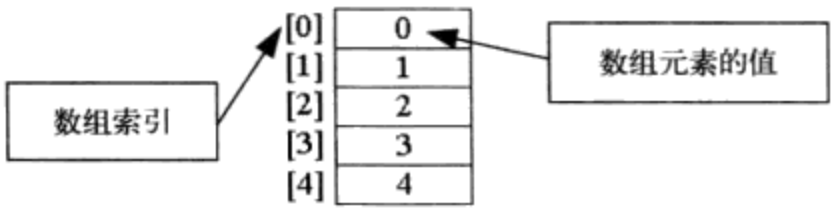


图 3.1　存储 5 个元素的数组结构

代码 03-01　光盘位置：光盘\mingrisoft\sl\03\03_01

```
static void Main(string[] args)
{
    int[] arr = new int[5];                                  //定义一个具有 5 个元素的数组
    Console.WriteLine("请输入一组数：");
    for (int i = 0; i < 5; i++)                              //向数组中输入值
    {
        arr[i] = Convert.ToInt32(Console.ReadLine());
    }
    for (int i = 0; i < 5; ++i)                              //遍历数组中的元素
    {
        int temp = arr[i];                                   //获取当前的元素值
        int j = i;
        while ((j > 0) && (arr[j - 1] > temp))               //对数组中的数据进行排序
        {
            arr[j] = arr[j - 1];
            --j;
        }
        arr[j] = temp;
    }
    Console.WriteLine("排序后结果为：");
    foreach (int n in arr)                                   //显示排序后的数组元素
    {
        Console.WriteLine("{0}", n);
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.2 所示。

图 3.2　一维数组的使用

说明：foreach 语句声明一个迭代变量，自动获取数组中每个元素的值。它是遍历一维数组的首选方式。如果要以其他方式遍历数组或修改数组中的元素，那么应该采用 for 语句，因为 foreach 语句中的迭代变量是数组的每个元素的只读副本。

3.1.2　二维数组

1．二维数组的声明

（1）声明。

二维数组即数组的维数为 2，它相当于一个表格。

语法：

```
type[,] arrayName;
```

参数说明如下。

- type：数组存储数据的数据类型。
- arrayName：数组名称。

例如，声明一个3行2列的数组。代码如下：

```
int[,] arr=new int[3,2];
```

（2）初始化。

例如，在声明二维数组的同时对其进行初始化。代码如下：

```
int[,] arr2=new int[3,2]{{1,2},{3,4},{5,6}};
```

程序代码也可以写为：

```
int[,] arr2=new int[,]{{1,2},{3,4},{5,6}};
```

2．二维数组的使用

需要存储表格式的数据时，可以使用二维数组，图3.3所示为包括4行3列的二维数组的存储结构。

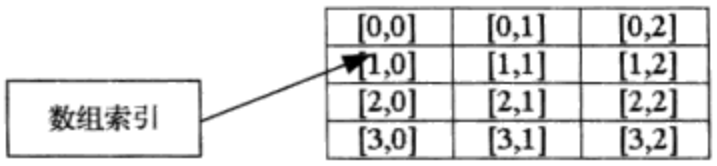


图3.3　二维数组的存储结构

示例03-02　二维数组的使用

示例中，将用户输入的行值和列值作为二维数组的长度，并将数组行索引和列索引相同的数组元素输出为“*”，其他元素输出为“@”。程序代码如下：

代码03-02　光盘位置：光盘\mingrisoft\sl\03\03_02

```
static void Main(string[] args)
{
    Console.Write("请输入定义数组的行数:");
    int row = Convert.ToInt32(Console.ReadLine());          //获取输入的行数
    Console.Write("请输入定义数组的列数:");
    int col = Convert.ToInt32(Console.ReadLine());          //获取输入的列数
    int[,] arr2 = new int[row, col];                        //根据行数和列数实例化二维数组
    Console.WriteLine("结果:");
    for (int i = 0; i < row; i++)                           //遍历数组的行数
    {
        for (int j = 0; j < col; j++)                       //遍历数组的列数
        {
            if (i == j)                                     //如果行数和列数相等
            {
                Console.Write("*");                         //输出*号
            }
            else
            {
                Console.Write("@");                         //输出@号
            }
        }
        Console.WriteLine();                                //换行显示
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图3.4所示。

图3.4　二维数组的使用

3.1.3 多维数组

1．多维数组的声明

（1）声明。

多维数组即数组的维数为 n。

语法：

```
type[,(,……)] arrayName;
```

参数说明如下。

- type：数组存储数据的数据类型。
- arrayName：数组名称。

例如，声明一个三维的数组，代码如下：

```
int[,,] arr3;
```

（2）初始化。

以三维数组为例，代码如下：

```
int[, ,] arr3 = new int[2, 1, 3] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
```

代码也可以为：

```
int[, ,] arr3 = new int[, , ] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
```

2．多维数组的使用

在多维数组中，最常使用的是二维数组，其次是三维数组。下面以三维数组为例对多维数组的创建及读取做简单介绍。

示例 03-03 多维数组的使用

示例将定义一个三维数组，并使用 foreach 语句将该数组输出。程序代码如下：

代码 03-03 光盘位置：光盘\mingrisoft\sl\03\03_03

```
static void Main(string[] args)
{
    int[, ,] arr3;                                          //声明一个三维数组
    arr3 = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };   //初始化该三维数组，维数为[2,1,3]
    foreach (int i in arr3)                                 //使用 foreach 语句遍历数组并输出
    {
        Console.WriteLine(i);
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.5 所示。

图 3.5 多维数组的使用

3.1.4 数组的基类 Array

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做所有数组的基类。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，多维 Array 类的各个维度可以有不同的界限。

1．Array 类的常用属性

（1）Length 属性。该属性用于获得一个 32 位整数，该整数表示 Array 类的所有维数中元素的总数。

语法：

```
public int Length { get; }
```

返回值：一个 32 位整数，表示 Array 的所有维数中元素的总数。

例如，获取一维数组的长度。代码如下：

```
string[] str = { "H", "e", "l", "l", "o", ",", "B", "e", "i", "J", "i", "n", "g", "." };
int n = str.Length.ToString();                                    //获取字符串的长度
```

在以上代码中，*n* 的值为 14。

例如，获取二维数组中元素的总数。代码如下：

```
int[,] str = { { 1, 2, 3, 4, 5, 6 }, { 7, 8, 9, 10, 11, 12 } };
int n = str.Length.ToString();
```

在以上代码中，*n* 的值为 12。

（2）Rank 属性。该属性用于获取 Array 类的秩，也就是维数。

语法：

```
public int Rank { get; }
```

属性值：Array 的秩（维数）。

例如，获取一维数组的维数。代码如下：

```
int[] ArrayInt=new int[] { 0, 1, 2, 3, 4 };
int n=ArrayInt.Rank;                                              //获取字符串的维度
```

以上代码中，*n* 的值为 1，如果获取的是二维数组的维数，*n* 的值为 2。

2．Array 类的常用方法

（1）BinarySearch 方法。

该方法是一个重载方法，主要使用二进制搜索算法在一维排序的 Array 中搜索值，如果有则返回该值的索引号，否则返回负数，在对值进行查找时，区分大小写。

语法：

```
public static int BinarySearch ( Array array, Object value )
public static int BinarySearch ( Array array, int index, int length, Object value )
```

参数说明如下。

- array：要搜索的已排序一维 Array。
- index：要搜索的范围的起始索引。
- length：要搜索的范围的长度。
- value：要搜索的对象。

例如，在字符串数组中查找字符串 B。代码如下：

```
string[] str = { "H", "e", "l", "l", "o", ",", "B", "e", "i", "J", "i", "n", "g", "." };
string myObject = "B";
int n = Array.BinarySearch(str, myObject);                        //获取指定的元素的索引号
```

在以上代码中，*n* 的值为 6。

例如，在字符串数组索引号 2~5 中查找指定的字符串。代码如下：

```
string[] str = { "H", "e", "l", "l", "o", ",", "B", "e", "i", "J", "i", "n", "g", "." };
Object myObject = "B";
int n = Array.BinarySearch(str,2,5, myObject);                    //查找元素是否在指定范围内
```

在以上代码中，因为在指定的范围内没有查找到字符串 B，所以 *n* 的值为一个负数。

（2）Clone 方法。

创建 Array 的浅表副本。Array 的浅表副本仅复制 Array 的元素（无论它们是引用类型还是值类型），但不复制这些引用所引用的对象。这里要说明的是，在新 ArrayArray 中的引用与原始 Array 中的引用指向相同的对象。

语法：

```
public Object Clone ()
```

返回值：Array 的浅表副本

例如，复制一个一维数组的浅表副本。代码如下：

```
int[] ArrayInt = new int[] { 0, 1, 2, 3 };                    //定义一个数组
int[] TemArrayInt = (int[])ArrayInt.Clone();                  //复制一个浅表副本
foreach (int i in TemArrayInt)                                //显示副本
    Console.Write(i.ToString());
```

在这里要说明的是，在对数组的浅表副本进行操作时，不影响原数组。

（3）Copy 方法。

从指定的源索引开始，复制 Array 中的一系列元素，将它们粘贴到另一 Array 中（从指定的目标索引开始）。长度和索引指定为 64 位或 32 位整数。下面以 32 位整数为例说明语法格式：

```
public static void Copy(Array sourceArray,Array destinationArray,int length)
public static void Copy(Array sourceArray,int sourceIndex,Array destinationArray,int destinationIndex,int length)
```

参数说明如下。

- sourceArray：Array，它包含要复制的数据。
- sourceIndex：一个 32 位整数，它表示 sourceArray 中复制开始处的索引。
- destinationArray：Array，它接收数据。
- destinationIndex：一个 32 位整数，它表示 destinationArray 中存储开始处的索引。
- length：一个 32 位整数，它表示要复制的元素数目。

例如，将两个数组复制到一个数组中。代码如下：

```
int[] ArrayInt1 = new int[] { 0, 1, 2, 3 };                   //实例化一个数组
int[] ArrayInt2 = new int[] { 4, 5, 6, 7 };                   //实例化一个数组
int[] TemArrayInt = new int[ArrayInt1.Length + ArrayInt2.Length];   //实例化一个长度是前两个数组长度总和的数组
//从 TemArrayInt 数组的起始位置开始，将 ArrayInt1 数组中的前 4 位复制到 TemArrayInt 数组中
Array.Copy(ArrayInt1, TemArrayInt, 4);
//从 TemArrayInt 数组的第 4 位开始，将 ArrayInt2 数组中的前 4 位复制到 TemArrayInt 数组中
Array.Copy(ArrayInt2,0, TemArrayInt,4, 4);
foreach (int i in TemArrayInt)                                //显示 TemArrayInt 数组中的元素
  Console.Write(i.ToString());
```

以上代码中输出结果为：01234567。

（4）Find 方法。

搜索与指定谓词定义的条件匹配的元素，然后返回整个 Array 中的第一个匹配项。

语法：

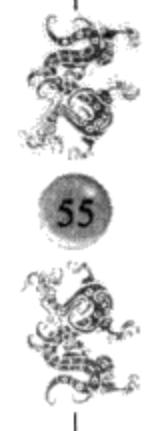
```
public static T Find<T> (T[] array,Predicate<T> match)
```

参数说明如下。

- T：数组元素的类型。
- array：要搜索的从零开始的一维 Array。
- match：Predicate，定义要搜索的元素的条件。

例如，在一维数组中查找符合条件的第一个元素的元素值。代码如下：

```
static void Main(string[] args)
{
    int[] ArrayInt = new int[] {1, 3, 5, 7 };        //实例化数组
    int n = Array.Find(ArrayInt, Findterm);          //查找符合条件的元素
    Console.WriteLine(n.ToString());                 //显示符合条件的元素
}
public static bool Findterm(int n)                   //具有布尔型返回值的自定义方法，判断值是否在指定的区间内
{
    if (n > 1 && n < 5)                              //如果当前参数在 1~5 内（不包含 1 和 5）
        return true;
```



```
    else
        return false;
}
```

在 Array 类中还有 FindLast 和 FindAll 方法， 这两个方法的操作与 Find 方法基本相同，唯一的区别在于，Find 方法从前向后进行查找，返回满足条件的第一个元素的元素值；FindLast 方法从后向前进行查找，返回满足条件的最后一个元素的元素值；FindAll 方法是将符合条件的所有元素存储在一维数组中。

（5）GetUpperBound 方法。

获取 Array 中指定维度的上限。

语法：

```
public int GetUpperBound(int dimension)
```

参数说明如下。

- dimension：Array 的从零开始的维度，其上限需要确定。

返回值：Array 中指定维度的上限。

例如，用 GetUpperBound 方法显示动态二维数组中的所有元素值。代码如下：

```
int[,] ArrayInt = { { 1, 3, 5 }, { 7, 9, 11 }, { 2, 4, 6 }, { 8, 10, 12 } };
for (int i = 0; i < (ArrayInt.Length / (ArrayInt.GetUpperBound(ArrayInt.Rank - 1) + 1)); i++)//遍历行
{
    for (int j = 0; j <= ArrayInt.GetUpperBound(ArrayInt.Rank - 1); j++)          //遍历列
        Console.Write(ArrayInt[i, j].ToString() + "    ");                        //输出二维数组中的元素
    Console.WriteLine();
}
```

例如，获取交错数组中的各元素值。代码如下：

```
int[][] StaggerArray = new int[3][];                                              //实例化交错数组
StaggerArray[0] = new int[] { 1, 3, 5, 7, 9 };                                    //实例化交错数组中的数组
StaggerArray[1] = new int[] { 0, 2, 4, 6 };
StaggerArray[2] = new int[] { 11, 22 };
for (int i = 0; i < StaggerArray.Length; i++)                                     //遍历交错数组中的元素
{
    for (int j = 0; j <= StaggerArray[i].GetUpperBound(StaggerArray[i].Rank - 1); j++)   //遍历元素中的数组
        Console.Write(StaggerArray[i][j].ToString() + "    ");                    //输出交错数组中的元素
    Console.WriteLine();
}
```

在以上代码中，因为交错数组中的每个元素都是一个数组，所以 Length 属性获取的元素个数，并不包含数组中的数组元素。

在 Array 类中有 GetLowerBound 和 GetLength 方法，与 GetUpperBound 方法的区别在于 GetLowerBound 方法获取的是指定维度的下线，一般是当前维度的最小索引值，也就是 0；GetLength 方法获取的是当前维度中元素的总数，一般比维度的上线大 1。

（6）Reverse 方法。

反转整个或部分一维 Array 中元素的顺序。

语法：

```
public static void Reverse(Array array)
public static void Reverse(Array array,int index,int length)
```

参数说明如下。

- array：要反转的一维 Array。
- index：要反转的部分的起始索引。
- length：要反转的部分中的元素数。

例如，将指定的一维数组进行反转。代码如下：

```
int[] ArrayInt = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };                    //定义整型数组
Array.Reverse(ArrayInt);                                              //对数组中的元素进反转
```

```
foreach (int i in ArrayInt)
    Console.Write(i.ToString());
```

输出结果为：9876543210。

（7）Sort 方法。

对一维 Array 的部分或全部元素进行排序，也可以用 IComparer 实现降序排序。

语法：

```
public static void Sort(Array array)
public static void Sort(Array array,IComparer comparer)
public static void Sort(Array array,int index,int length)
public static void Sort(Array array,int index,int length,IComparer comparer)
```

参数说明如下。

- array：要排序的一维 Array。
- index：排序范围的起始索引。
- length：排序范围内的元素数。
- comparer：比较元素时要使用的 IComparer 实现。

例如，对指定的一维数组进行升序排序。代码如下：

```
String[] myArr = { "The", "QUICK", "BROWN", "FOX", "jumps", "over", "the", "lazy", "dog" };
Array.Sort(myArr);                                                    //对数组进行降序排序
foreach(string i in myArr)
    Console.Write(i.ToString()+" ");
```

输出结果为：BROWN dog FOX jumps lazy over QUICK the The。

例如，利用 IComparer 对指定的一维数组进行降序排序。代码如下：

```
public class myReverserClass : IComparer                              //设置 IComparer 接口的类
{
    int IComparer.Compare(Object x, Object y)
    {
        return ((new CaseInsensitiveComparer()).Compare(y, x));       //不区分大小写进行比较
    }
}
static void Main(string[] args)
{
    String[] myArr = { "The", "QUICK", "BROWN", "FOX", "jumps", "over", "the", "lazy", "dog" };   //实例化数组
    IComparer myComparer = new myReverserClass();                     //实例化 IComparer 接口的类
    Array.Sort(myArr, myComparer);                                    //对数组进行降序排序
    foreach (string i in myArr)                                       //输出显示排序后的数组
        Console.WriteLine(i.ToString());
}
```

输出结果为：The the QUICK over lazy jumps FOX dog BROWN。

3．Array 类的使用

在对数组进行添加、修改、删除操作时，为了保护数组的原始内容，一般都创建一个数组副本，所有操作都在副本中进行。下面以二维数组为例，介绍如何生成二维数组的副本，以及遍历二维数组。

示例 03-04　生成二维数组的副本

将一个二维数组存入到另一个二维数组中。程序代码如下：

代码 03-04　光盘位置：光盘\mingrisoft\sl\03\03_04

```
static void Main(string[] args)
{
    int[,] ArrayInt1 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 }, { 13, 14, 15 } };   //定义二维数组
    int row = ArrayInt1.Length / (ArrayInt1.GetUpperBound(ArrayInt1.Rank - 1) + 1);   //获取二维数组的行数
    int col = ArrayInt1.GetLength(ArrayInt1.Rank - 1);                                //获取二维数组的列数
    int[,] ArrayInt2 = new int[row, col];                                             //声明一个二维数组
    ArrayInt2 = ArrayInt1;                                                            //实现二维数组的赋值
```

```
        for (int i = 0; i < row; i++)                                    //遍历行数
        {
            for (int j = 0; j < col; j++)                                //遍历列数
                Console.Write(ArrayInt2[i, j].ToString() + "    ");      //输出赋值后的二维数组的元素值
            Console.WriteLine();
        }
    }
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.6 所示。

图 3.6　生成二维数组的副本

3.1.5　交错数组

1．交错数组的声明

（1）声明。

交错数组是指元素为数组的数组。交错数组元素的维度和大小可以不同。交错数组有时称为“数组的数组”。

语法：

```
type[][] jaggedArray[ = new type[n][]];
```

参数说明如下。

- type：数组存储数据的数据类型。
- jaggedArray：交错数组名称。

例如，声明一个没有初始化的整型交错数组。代码如下：

```
int[][] jaggedArray1;
```

（2）初始化。

交错数组的初始化有两种形式，一是先声明一个交错数组，然后对其进行初始化。二是在声明的同时直接对其进行初始化。在这里要说明的是，在使用交错数组前，必须对其进行初始化。

例如，初始化一个具有默认值的整型交错数组。代码如下：

```
int[][] jaggedArray2 = new int[][]
{
    new int[7],
    new int[5],
    new int[2]
};
```

例如，声明一个交错数组，然后对其进行初始化。代码如下：

```
int[][] jaggedArray3;
jaggedArray3 = new int[][]
{
    new int[] { 1, 3, 5, 7, 9 },
    new int[] { 0, 2, 4, 6 },
    new int[] { 11, 22 }
};
```

例如，在声明交错数组的同时，对其进行初始化。代码如下：

```
int[][] jaggedArray4 = new int[][]
{
    new int[] { 1, 3, 5, 7, 9 },
    new int[] { 0, 2, 4, 6 },
    new int[] { 11, 22 }
};
```

2．交错数组的使用

有时，在制作应用程序时，为了节省更多的内存，可以将非矩型的二维数组定义成交错数组，这样可以节省各行的元素个数。

示例 03-05　用交错数组显示一个心形图像

将一个用“*”号组成的心形图像存入到交错数组中，以节省内存空间。程序代码如下：

代码 03-05　光盘位置：光盘\mingrisoft\sl\03\03_05

```
        static void Main(string[] args)
        {
            string[][] jaggedArray = new string[][]                                    //实例化交错数组
            {
              //实例化交错数组中的元素数组
                new string[]{" " " " " " " " " " " " "*" "*" "*" "*" "*" "*" " " " " " " " " " " " " " " " " " "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " " " " " " "*" "*" "*" "*" "*"
"*" "*" "*" "*" "*"},
                new string[]{" " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" " " "*" "*" "*" "*" "*" "*"
"*" "*" "*" "*" "*" "*" "*"},
                new string[]{"*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
"*" "*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{"*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
"*" "*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{"*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
"*" ‘"*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
‘"*" "*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" ‘"*"
"*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" ’ "*" "*" "*"
"*" "*" "*" "*" "*"},
                new string[]{" " " " " " " " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " " " " " " " " " "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " " " " " " " " " " " "*" "*" "*" "*" "*" "*" "*"},
                new string[]{" " " " " " " " " " " " " " " " " " " " " " " " "*" "*" "*"},
                new string[]{" " " " " " " " " " " " " " " " " " " " " " " " " " " "*"}
            };
            for (int i = 0; i < jaggedArray.Length; i++)                               //遍历交错数组中的元素
            {
                for (int j = 0; j <= jaggedArray[i].GetUpperBound(jaggedArray[i].Rank - 1); j++)    //遍历元素数组
                    Console.Write(jaggedArray[i][j].ToString());                       //输出元素值
                Console.WriteLine();
            }
            Console.ReadLine();
        }
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.7 所示。

图 3.7　用交错数组显示一个心形图象

3.2 集　　合

.NET 框架的基类库提供了一组“集合”，这是一组通用的数据类型。这些数据类型可以对数据进行集中存储和操作。

集合是一组组合在一起的类似的类型化对象。这些集合都在 System.Collections 命名空间中。每个内建的集合都实现了 Icollection 接口。

3.2.1 ArrayList 类

1．ArrayList 类的概念

ArrayList 类位于 System.Collections 命名空间下，它可以动态地添加和删除元素。可以将 ArrayList 类看做是扩充了功能的数组，但它并不等同于数组。

与数组相比，ArrayList 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。

2．ArrayList 类的声明

ArrayList 类主要用于对一个数组中的元素进行各种处理。

语法 1：

初始化 ArrayList 类的新实例，该实例为空并且具有默认初始容量。

```
public ArrayList()
```

例如，声明一个空的 ArrayList 类。代码如下：

```
ArrayList arraylist = new ArrayList();
```

向 ArrayList 添加元素时，将通过重新分配内部数组，根据需要自动增大容量。

语法 2：

初始化 ArrayList 类的新实例，该实例包含从指定集合复制的元素并且具有与所复制的元素数相同的初始容量。

```
public ArrayList(ICollection c)
```

参数说明如下。

- c：类型，System.Collections.ICollection，它的元素被复制到新列表中。

例如，根据指定的数组实例化 ArrayList 类。代码如下：

```
string[] StrArray = { "one", "two", "three", "four" };
ArrayList arraylist = new ArrayList(StrArray);
```

语法 3：

初始化 ArrayList 类的新实例，该实例为空并且具有指定的初始容量。

```
public ArrayList(int capacity)
```

参数说明如下。

- capacity：类型，System.Int32 新列表最初可以存储的元素数。

例如，根据指定的元素个数实例化 ArrayList 类。代码如下：

```
int n = 4;
ArrayList arraylist = new ArrayList(n);
```

注意：在声明 ArrayList 类时其默认大小为 0，在向 ArrayList 类添加元素时，可通过重新分配内部数组，根据需要自动增大容量。也就是说，其容量大小将根据添加元素的个数

而改变。

3．ArrayList 类的添加

向 ArrayList 类中添加元素时，可以使用 ArrayList 类提供的 Add 和 Insert 方法，下面对这两个方法进行详细介绍。

（1）Add 方法。该方法用来将对象添加到 ArrayList 集合的结尾处。

语法：

```
public virtual int Add (Object value)
```

参数说明如下。

- value：要添加到 ArrayList 的末尾处的 Object，该值可以为空引用。

返回值：ArrayList 索引，已在此处添加了 value。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 Add 方法为 ArrayList 对象添加元素 7。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };          //实例化数组
ArrayList List = new ArrayList(arr);                 //通过数组实例化 ArrayList 类
List.Add(7);                                         //在 ArrayList 对象的尾部添加元素
```

添加后的结果为：1 2 3 4 5 6 7

（2）Insert 方法。该方法用来将元素插入 ArrayList 集合的指定索引处。

语法：

```
public virtual void Insert (int index,Object value)
```

参数说明如下。

- index：从零开始的索引，应在该位置插入 value。
- value：要插入的 Object，该值可以为空引用。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 Insert 方法在该 ArrayList 对象的指定索引处添加一个元素。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
List.Insert(3, 7);
```

添加后的结果为：1 2 3 7 4 5 6

示例 03-06　ArrayList 类的添加操作

用一个 int 类型的一维数组实例化一个 ArrayList 对象，然后分别使用 ArrayList 对象的 Add 方法和 Insert 方法向 ArrayList 集合的结尾处和指定索引处添加元素。程序代码如下：

代码 03-06　光盘位置：光盘\mingrisoft\sl\03\03_06

```
static void Main(string[] args)
{
    int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
    ArrayList List = new ArrayList(arr);                    //使用声明的一维数组实例化一个 ArrayList 对象
    Console.WriteLine("原 ArrayList 集合：");
    foreach (int i in List)                                 //遍历 ArrayList 集合并输出
        Console.Write(i + " ");
    Console.WriteLine();
    for (int i = 1; i < 5; i++)
        List.Add(i + arr.Length);                           //在 ArrayList 集合的尾部追加元素
    Console.WriteLine("使用 Add 方法添加：");
    foreach (int i in List)                                 //遍历添加元素后的 ArrayList 集合并输出
        Console.Write(i + " ");
    Console.WriteLine();
    List.Insert(6, 9);                                      //在 ArrayList 集合的第 6 个元素后添加 9
    Console.WriteLine("使用 Insert 方法添加：");
    foreach (int i in List)                                 //遍历最后的 ArrayList 集合并输出
        Console.Write(i + " ");
```

```
        Console.WriteLine();
        Console.ReadLine();
    }
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.8 所示。

图 3.8　ArrayList 类的添加操作

在 ArrayList 类中插入一个数组时，只需要在上面代码中重新声明一个一维数组，并将 List.Insert(6, 6)语句改为 List.InsertRange(6, 一维数组名)即可。

4．ArrayList 类的修改

在 ArrayList 集合中删除元素时，可以使用 ArrayList 类提供的 Clear 方法、Remove 方法、RemoveAt 方法和 RemoveRange 方法，下面对这 4 个方法进行详细介绍。

（1）Clear 方法。该方法用来从 ArrayList 中移除所有元素。

语法：

```
public virtual void Clear ()
```

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 Clear 方法清除 ArrayList 中的所有元素。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
List.Clear();
```

（2）Remove 方法。该方法用来从 ArrayList 中移除特定对象的第一个匹配项。

语法：

```
public virtual void Remove (Object obj)
```

参数说明如下。

- obj：要从 ArrayList 移除的 Object，该值可以为空引用。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 Remove 方法从声明的 ArrayList 对象中移除与“3”匹配的元素。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
List.Remove(3);
```

移除后的结果为：1 2 4 5 6

（3）RemoveAt 方法。该方法用来移除 ArrayList 的指定索引处的元素。

语法：

```
public virtual void RemoveAt (int index)
```

参数说明如下。

- index：要移除的元素的从零开始的索引。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 RemoveAt 方法从声明的 ArrayList 对象中移除索引为 3 的元素。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
List.RemoveAt(3);
```

移除后的结果为：1 2 3 5 6

（4）RemoveRange 方法。该方法用来从 ArrayList 中移除一定范围的元素。

语法：

```
public virtual void RemoveRange (int index,int count)
```

参数说明如下。

- index：要移除的元素的范围从零开始的起始索引。

● count：要移除的元素数。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后在 ArrayList 对象中使用 RemoveRange 方法从索引 3 处删除两个元素，代码如下。

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
List.RemoveRange(3,2);
```

移除后的结果为：1 2 3 6

例如，创建一个控制台应用程序，使用 RemoveRange 方法删除 ArrayList 集合中的一批元素。程序代码如下。

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
ArrayList List = new ArrayList(arr);
Console.WriteLine("原 ArrayList 集合：");
foreach (int i in List)                          //遍历 ArrayList 集合中的元素并输出
    Console.Write(i.ToString() + " ");
Console.WriteLine();
List.RemoveRange(0, 5);                          //从 ArrayList 集合中移除指定下标位置的元素
Console.WriteLine("删除元素后的 ArrayList 集合：");
foreach (int i in List)                          //遍历删除元素后的 ArrayList 集合并输出其元素
    Console.Write(i.ToString() + " ");
Console.WriteLine();
```

移除后的结果为：

```
原 ArrayList 集合：
1 2 3 4 5 6 7 8 9
删除元素后的 ArrayList 集合：
6 7 8 9
```

5．ArrayList 类的遍历

ArrayList 集合的遍历与数组类似，都可以使用 for 或 foreach 语句，下面通过一个实例说明如何遍历 ArrayList 集合中的元素。

例如，创建一个控制台应用程序，其中实例化了一个 ArrayList 对象，并使用 Add 方法向 ArrayList 集合中添加了两个元素，然后使用 foreach 语句遍历 ArrayList 集合中的各个元素并输出。代码如下。

```
static void Main(string[] args)
{
    ArrayList list = new ArrayList();            //实例化一个 ArrayList 对象
    list.Add("TM");                              //向 ArrayList 集合中添加元素
    list.Add("C# 2.0 从入门到应用开发");
    foreach (string str in list)                 //遍历 ArrayList 集合中的元素并输出
    {
        Console.WriteLine(str);
    }
}
```

6．ArrayList 类的查找

在 ArrayList 集合中查找元素时，可以使用 ArrayList 类提供的 Contains 方法、IndexOf 方法和 LastIndexOf 方法，下面对其进行详细介绍。

（1）Contains 方法。该方法用来确定某元素是否在 ArrayList 集合中。

语法：

```
public virtual bool Contains (Object item)
```

参数说明如下。

● item：要在 ArrayList 中查找的 Object，该值可以为空引用。

返回值：如果在 ArrayList 中找到 item，则为 true；否则为 false。

例如，声明一个包含 6 个元素的一维数组，并使用该数组实例化一个 ArrayList 对象，然后使用 Contains 方法判断数字 2 是否在 ArrayList 集合中。代码如下：

```
int[] arr = new int[] { 1, 2, 3, 4, 5, 6 };
ArrayList List = new ArrayList(arr);
Console.Write(List.Contains(2));
```

运行结果为：true。

（2）IndexOf 方法。该方法用于搜索指定的对象（Object），并返回 ArrayList 集合中第一个匹配项的索引（从零开始）。

语法：

```
public virtual int IndexOf(Object value)
public virtual int IndexOf(Object value,int startIndex)
public virtual int IndexOf(Object value,int startIndex,int count)
```

参数说明如下。

- value ：System.Object，在 ArrayList 中查找的 Object。该值可以为 null 引用。
- startIndex：Int32，从零开始搜索的起始索引。
- count：Int32，要搜索的部分中的元素数。

返回值：Int32，第一个匹配项的索引号。

例如，在 ArrayList 类中查找字符串 the，并返回第一个匹配项的索引号。代码如下：

```
ArrayList List = new ArrayList();                    //实例化 ArrayList 类
List.Add("Return");                                  //添加元素
List.Add("the");
List.Add("XOR");
List.Add("of");
List.Add("the");
List.Add("x");
List.Add("and");
List.Add("y");
List.Add("fields");
int n = List.IndexOf("the");                         //获取指定元素的索引号
Console.WriteLine(n);
```

运行结果为：1。

LastIndexOf 方法与 IndexOf 方法基本相同，只是 LastIndexOf 方法返回的是最后一个匹配项的索引号。

7．ArrayList 类的使用

ArrayList 类的容量是根据需要自动扩展的，该类提供了添加、插入和移除功能对集合进行相应的操作。

示例 03-07　根据其他数组删除 ArrayList 类中的指定元素

将 ArrayList 类中包含的另一个数组的元素删除。程序代码如下：

代码 03-07　光盘位置：光盘\mingrisoft\sl\03\03_07

```
static void Main(string[] args)
{
    string[] ArrayStr = new string[] { "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r",
"s", "t", "u", "v" };                                //实例化字符串数组
    string[] ArrayStr1 = new string[] { "m", "r", "s", "o", "f", "t" };   //实例化字符串数组
    ArrayList list = new ArrayList(ArrayStr);        //通过数组实例化 ArrayList 类
    for (int i = 0; i < ArrayStr1.Length; i++)       //遍历 ArrayStr1 类
    {
        list.RemoveAt(list.IndexOf(ArrayStr1[i], 0, list.Count));   //删除 ArrayStr1 数组中的元素
    }
```

```
    foreach (string str in list)                    //遍历删除后的 ArrayList 类
        Console.Write(str);
    Console.ReadLine();
}
```

按<Ctrl+F5>组合键运行程序，运行结果为：

```
abcdeghijklnpquv
```

3.2.2 Hashtable 类

1．Hashtable 类的概述

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在 DictionaryEntry 对象中的键/值对。键不能为空引用，但值可以。在 Hashtable 中插入一对 Key/Value 时，自动将 Key 值映射到 Value，并允许获取与一个指定的 Key 相关联的 value。

Hashtable 的构造函数有多种，这里介绍两种最常用的。

（1）使用默认的初始容量、加载因子、哈希代码提供程序和比较器来初始化 Hashtable 类的新的空实例。

语法：

```
public Hashtable ()
```

例如，声明一个默认初始容量的 Hashtable 类。代码如下：

```
Hashtable hashtable = new Hashtable();
```

（2）使用指定的初始容量、默认加载因子、默认哈希代码提供程序和默认比较器来初始化 Hashtable 类的新的空实例。

语法：

```
public Hashtable (int capacity)
```

参数说明如下。

- capacity：Hashtable 对象最初可包含的元素的近似数目。

例如，声明一个指定初始容量的 Hashtable 类。代码如下：

```
Hashtable hashtable = new Hashtable(5);
```

注意：在使用 Hashtable 类时要引用命名空间 System.Collections。

2．Hashtable 类的添加

向哈希表中添加元素时，可以使用 Hashtable 类提供的 Add 方法，下面对该方法进行详细介绍。

Add 方法用来将带有指定键和值的元素添加到哈希表中。

语法。

```
public virtual void Add (Object key,Object value)
```

参数说明如下。

- key：要添加的元素的键。
- value：要添加的元素的值，该值可以为空引用。

例如，实例化一个 Hashtable 对象，然后使用 Add 方法为 Hashtable 对象添加 3 个元素。代码如下：

```
Hashtable hashtable = new Hashtable();                  //实例化 Hashtable 对象
hashtable.Add("id", "BH0001");                          //向 Hashtable 哈希表中添加元素
hashtable.Add("name", "刘某");
hashtable.Add("sex", "男");
Console.WriteLine(hashtable.Count);                     //获得 Hashtable 哈希表中的元素个数
```

运行结果为：3。

3．Hashtable 类的删除

在哈希表中删除元素时，可以使用 Hashtable 类提供的 Clear 方法和 Remove 方法，下面对

这两个方法进行详细介绍。

（1）Clear 方法。该方法用来从 Hashtable 中移除所有元素。

语法：

```
public virtual void Clear ()
```

例如，实例化一个 Hashtable 对象，同时使用 Add 方法为该对象添加 3 个元素，然后使用 Clear 方法移除 Hashtable 中的所有元素。代码如下：

```
Hashtable hashtable = new Hashtable();                    //实例化对象
hashtable.Add("id", "BH0001");                            //向 Hashtable 哈希表中添加元素
hashtable.Add("name", "刘某");
hashtable.Add("sex", "男");
hashtable.Clear();                                        //移除 Hashtable 哈希表中的元素
Console.WriteLine(hashtable.Count);                       //显示元素个数
```

运行结果为：0。

（2）Remove 方法。该方法用来从 Hashtable 中移除带有指定键的元素。

语法：

```
public virtual void Remove (Object key)
```

参数说明如下。

- key：要移除的元素的键。

例如，创建一个控制应用程序，其中实例化 Hashtable 一个对象，同时使用 Add 方法为该 Hashtable 对象添加 3 个元素，然后使用 Remove 方法移除 Hashtable 哈希表中键为“sex”的元素。代码如下。

```
Hashtable hashtable = new Hashtable();                    //实例化 Hashtable 对象
hashtable.Add("id", "BH0001");                            //向 Hashtable 哈希表中添加元素
hashtable.Add("name", "刘某");
hashtable.Add("sex", "男");
hashtable.Remove("sex");                                  //移除 Hashtable 哈希表中的指定元素
Console.WriteLine(hashtable.Count);//显示元素个数
```

运行结果为：2。

4．Hashtable 类的遍历

Hashtable 哈希表的遍历与数组类似，都可以使用 foreach 语句，这里需要注意的是，由于 Hashtable 中的元素是一个键/值对，因此需要使用 DictionaryEntry 类型来进行遍历。DictionaryEntry 类型表示一个键/值对的集合。下面通过一个实例说明如何遍历 Hashtable 哈希表中的元素。

例如，实例化了一个 Hashtable 对象，并使用 Add 方法向 Hashtable 哈希表中添加 3 个元素，然后使用 foreach 语句遍历 Hashtable 哈希表中的各个键/值对并输出。程序代码如下。

```
Hashtable hashtable = new Hashtable();                    //实例化 Hashtable 对象
hashtable.Add("id", "BH0001");                            //向 Hashtable 哈希表中添加元素
hashtable.Add("name", "刘某");
hashtable.Add("sex", "男");
Console.WriteLine("\t 键\t 值");
//遍历 Hashtable 哈希表中的元素并输出其键值对
foreach (DictionaryEntry dicEntry in hashtable)
    Console.WriteLine("\t " + dicEntry.Key + "\t " + dicEntry.Value);
Console.WriteLine();
```

5．Hashtable 类的查找

在 Hashtable 哈希表中删除元素时，可以使用 Hashtable 类提供的 Contains 方法、ContainsKey 方法和 ContainsValue 方法，下面主要对这 3 个方法进行详细介绍。

（1）Contains 方法。该方法用来确定 Hashtable 中是否包含特定键。

语法：

```
public virtual bool Contains (Object key)
```

参数说明如下。

- key：要在 Hashtable 中定位的键。

返回值：如果 Hashtable 包含具有指定键的元素，则为 true；否则为 false。

例如，实例化一个 Hashtable 对象，同时使用 Add 方法为该 Hashtable 对象添加 3 个元素，然后使用 Contains 方法判断键 id 是否在 Hashtable 哈希表中。代码如下：

```
Hashtable hashtable = new Hashtable();                    //实例化 Hashtable 对象
hashtable.Add("id", "BH0001");                            //向哈希表中添加元素
hashtable.Add("name", "刘某");
hashtable.Add("sex", "男");
//判断 Hashtable 哈希表中是否包含指定的键，如果有则显示键值
Console.WriteLine(hashtable.Contains("id") ? hashtable["id"] : "");
```

运行结果为：BH0001。

（2）ContainsValue 方法。该方法用来确定 Hashtable 中是否包含特定值。

语法：

```
public virtual bool ContainsValue (Object value)
```

参数说明如下。

- value：要在 Hashtable 中定位的值，该值可以为空引用。

返回值：如果 Hashtable 包含带有指定的 value 的元素，则为 true；否则为 false。

例如，实例化一个 Hashtable 对象，同时使用 Add 方法为 Hashtable 对象添加 3 个元素，然后使用 ContainsValue 方法判断值“BH0002”是否在 Hashtable 哈希表中。代码如下：

```
Hashtable hashtable = new Hashtable();                    //实例化 Hashtable 对象
hashtable.Add("id", "BH0001");                            //向哈希表中添加元素
hashtable.Add(“name”, “TM”);
hashtable.Add(“sex”, “男”);
//判断 Hashtable 哈希表中是否包含指定的键值
Console.WriteLine(hashtable.ContainsValue("BH0002"));
```

运行结果为：false。

6．Hashtable 哈希表的使用

示例 03-08　将货币转换大写

用 Hashtable 哈希表实现货币的大写转换。程序代码如下：

代码 03-08　光盘位置：光盘\mingrisoft\sl\03\03_08

```
static void Main(string[] args)
{
    Hashtable Figure = new Hashtable();                   //实例化 Hashtable 对象
    Figure.Add("0", "零");                                //添加数字的大写
    Figure.Add("1", "壹");
    Figure.Add("2", "贰");
    Figure.Add("3", "叁");
    Figure.Add("4", "肆");
    Figure.Add("5", "伍");
    Figure.Add("6", "陆");
    Figure.Add("7", "柒");
    Figure.Add("8", "捌");
    Figure.Add("9", "玖");
    Hashtable Digit = new Hashtable();                    //实例化 Hashtable 对象
    Digit.Add("1", "元");                                 //添加位数
    Digit.Add("2", "拾");
    Digit.Add("3", "佰");
    Digit.Add("4", "仟");
    Digit.Add("5", "万");
    Digit.Add("6", "亿");
```

```
Digit.Add("7", "兆");
Console.WriteLine("请输入单个数字：");
string Str = Console.ReadLine();                                    //记录输入的数据
ArrayList ArrayValue = new ArrayList();                             //实例化 ArrayList 类
for (int i = 0; i < Str.Length; i++)                                //以数组的型式记录数据
    ArrayValue.Add(Str.Substring(i, 1));
ArrayValue.Reverse();                                               //将数组中的元素进行反转
bool Zero = false;                                                  //判断是否有重复的零
bool Again = false;                                                 //判断万亿兆是否连续输出
int n = 0;                                                          //记录当前数字的位数
string Caps = "";                                                   //记录转换后的字符串
string TemStr = "";                                                 //记录指定数字的大写
int Div = 0;                                                        //记录除数
int Res = 0;                                                        //记录余数
for (int i = ArrayValue.Count - 1; i >= 0; i--)                     //对数组中的数据进行遍历
{
    n = i + 1;                                                      //记录当前数据的位数
    TemStr = Convert.ToString(Figure.Contains(Convert.ToString(ArrayValue[i])) ? Figure[ArrayValue[i]] :
" ");                                                               //记录当前数据的大写
    if (TemStr == "零")                                             //如果当前数据为 0
        Zero = true;
    else
        Zero = false;
    if (Zero)                                                       //如果为 0
    {
        if (Caps.Length > 0)                                        //如果转换的字符串长度大于 0
        {
            if (Caps.Substring(Caps.Length - 1, 1) != "零")     //如果转换后的字符串的最后一个字符不是零
            {
                Caps += "零";                                       //在字符串的尾部加零
            }
            Div = Math.DivRem(n, 4, out Res);                       //获取当前数字的位数
            if (Div > 0 && Res == 1)                                //如果当前数字是万亿兆
            {
                for (int j = 5; j <= Digit.Count; j++)              //遍历万亿兆
                {
                    string tem = Caps.Substring(Caps.Length - 2, 1);    //获取字符串中倒属第二个字符
                    if (tem == Convert.ToString(Digit[j.ToString()]))    //如果字符是万、亿或兆
                    {
                        Again = true;
                        break;
                    }
                }
                if (!Again)                                     //如果字符串中倒属第二个字符不是万、亿或兆
                    //在零的前面插入万、亿或兆
                    Caps = Caps.Insert(Caps.Length - 1, Digit[Convert.ToString(Div + 4)].ToString());
                Again = false;
            }
        }
        continue;
    }
    else
    {
        Caps += TemStr;                                         //在字符串的尾部添加当前数字的大写
    }
    Div = Math.DivRem(n, 4, out Res);                               //获取当前数字的位数
    if (Div == 0)                                                   //如果是万以内的位数
    {
        Caps += Digit[Res.ToString()].ToString();                   //添加位数
    }
    else                                                            //万以上的位数（包含万位）
    {
        if (Res == 1)                                               //如果当前是万位
```

```
                Caps += Digit[Convert.ToString(Div + 4)].ToString();          //在字符串尾部添加万
            else
            {
                if (Res == 0)                                                 //如果当前是仟位
                    Caps += Digit[Convert.ToString(4)].ToString();            //在字符串尾部添加仟
                Else                                                          //如果当前是拾位和佰位
                    Caps += Digit[Convert.ToString(Res)].ToString();          //在字符串尾部添加拾或佰
            }
        }
    }
    Console.WriteLine(Caps);
    Console.ReadLine();
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.9 所示。

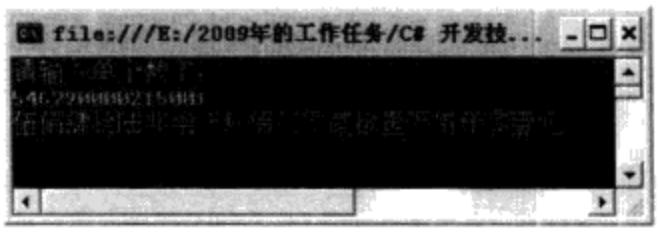


图 3.9 将货币转换成大写

说明：Hashtable 不能包含重复的 Key。如果调用 Add 方法来添加一个 keys 数组中已有的 Key，就会抛出异常。为了避免这种情况，可以使用 ContainsKey 方法来测试哈希表中是否包含一个特定的 Key。

3.2.3 其他集合的简单应用

1．Stack 类

Stack（堆栈）类主要实现了一个 LIFO（Last In First Out，后进先出）的机制。元素从栈的顶部插入（入栈操作），从堆的顶部移除（出栈操作）。在 Stack 中主要使用 Push、Pop、Peek 这 3 种方法对栈进行操作。Push 方法用于将对象插入 Stack 的顶部，Pop 方法用于移除并返回位于 Stack 顶部的对象，Peek 方法用于返回位于 Stack 顶部的对象但不将其移除。

示例 03-09 Stack 的使用

示例将介绍如何创建一个 Stack，如何添加项、移除项以及如何遍历 Stack。程序代码如下：

代码 03-09 光盘位置：光盘\mingrisoft\sl\03\03_09

```
using System.Collections;                                                     //引入命名空间
namespace StackUse
{
    class Program
    {
        static void Main(string[] args)
        {
            Stack stack = new Stack();                                        //实例化 Stack 类的对象
            for (int i = 1; i < 6;i++)                    //入栈,使用 Pust 方法向 Stack 对向中添加元素
            {
                stack.Push(i);
                Console.WriteLine("{0}入栈",i);
            }
            Console.WriteLine ("当前栈顶元素为：{0}",stack.Peek().ToString ());   //返回栈顶元素
            Console.WriteLine("移除栈顶元素：{0}", stack.Pop().ToString());      //出栈
            Console.WriteLine("当前栈顶元素为：{0}", stack.Peek().ToString());   //返回栈顶元素
            Console.WriteLine("遍历栈");
            foreach (int i in stack)                                          //遍历栈
            {
                Console.WriteLine(i);
            }
            while(stack .Count!=0)                                            //清空栈
            {
```

```
                int s = (int)stack.Pop();
                Console.WriteLine("{0}出栈",s);
            }
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.10 所示。

图 3.10　Stack 类的使用

2．Queue 类

Queue（队列）类主要实现了一个 FIFO 的机制。元素在队列的尾部插入（入队操作），并从队列的头部移出（出队操作）。在 Queue 中主要使用 Enqueue、Dequeue、Peek 这 3 种方法对队列进行操作。Enqueue 方法用于将对象添加到 Queue 的结尾处，Dequeue 方法移除并返回位于 Queue 开始处的对象，Peek 方法用于返回位于 Queue 开始处的对象但不将其移除。

示例 03-10　Queue 的使用

示例将介绍如何创建一个 Queue，如何添加项、移除项以及如何遍历 Queue。程序代码如下：

代码 03-10　光盘位置：光盘\mingrisoft\sl\01\03_10

```
using System.Collections;                                   //引入命名空间
namespace QueueUse
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue queue = new Queue();                      //实例化 Queue 类的对象
            for (int i = 1; i < 6; i++)                     //入栈,使用 Push 方法向 Stack 对向中添加元素
            {
                queue .Enqueue(i);
                Console.WriteLine("{0}入队", i);
            }
            Console.WriteLine("当前队开始处元素为：{0}", queue.Peek().ToString());//返回队开始处的元素
            Console.WriteLine("遍历队");
            foreach (int i in queue)                        //遍历队
            {
                Console.WriteLine(i);
            }
            while (queue.Count != 0)                        //清空栈
            {
                int q = (int)queue.Dequeue ();
                Console.WriteLine("{0}出队", q);
            }
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.11 所示。

3．SortedList 类

SortedList 类也是键/值对的集合，但与哈希表不同的是这些键/值对是按键排序，并可以按照键和索引访问。在 SortedList 中主要使用 Add、Remove、RemoveAt 这 3 种方法对 SortedList 进行操作。Add 方法用于将带有指定键和值的元素添加到 SortedList 中，Remove 方法用于从 SortedList 中移除带有指定键的元素，RemoveAt 方法用于移除 SortedList 的指定索引处的元素。

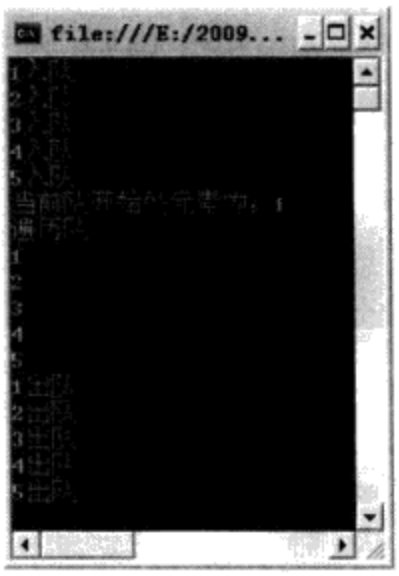


图 3.11 Queue 类的使用

示例 03-11 SortedList 的使用

示例将介绍如何创建一个 SortedList，如何添加项、移除项以及如何遍历 SortedList。程序代码如下：

代码 03-11 光盘位置：光盘\mingrisoft\sl\03\03_11

```
using System.Collections;                                   //引入命名空间
namespace SortedListUse
{
    class Program
    {
        static void Main(string[] args)
        {
            SortedList student = new SortedList();          //实例化 SortedListTest 类的对象
            student.Add("S1001", "Tom");                    //向 SortedList 中添加元素
            student.Add("S1003", "Jim");
            student.Add("S1002", "Lily");
            student.Add("S1004", "Lucy");
            foreach (DictionaryEntry element in student)    //遍历 SortedList
            {
                string id = element.Key.ToString();
                string name = element.Value.ToString();
                Console.WriteLine("学生的 ID：{0}    学生姓名：{1}", id, name);
            }
            student.Remove("S1003");                        //移除 SortedList 中 Key 为“S1003”的元素
            Console.WriteLine("移除学生 ID 为：S1003");
            student.RemoveAt(0);            //移除 SortedList 中索引为 0 的元素，即第一个元素
            Console.WriteLine("移除第一个元素");
            Console.WriteLine("显示移除后的信息：");
            foreach (DictionaryEntry element in student)
            {
                string id = element.Key.ToString();         //获取键
                string name = element.Value.ToString();     //获取值
                Console.WriteLine("学生的 ID：{0}    学生姓名：{1}", id, name);
            }
            Console.ReadLine();
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 3.12 所示。

说明：同样，SortedList 也不能包含重复的 Key，而且使用 foreach 语句遍历 SortedList 对象时，会返回 DictionaryEntry 对象。该对象将根据 Key 属性，以排序后的顺序返回。

图 3.12　SortedList 类的使用

第 4 章

面向对象编程

- 面向对象编程概述
- 类
- 对象
- 构造函数和析构函数
- 方法
- 域和属性
- 委托
- 事件和索引器
- 接口
- 封装、继承和多态
- 泛型

为了解决结构化程序设计的一些弊端和缺点，面向对象技术应运而生。它将数据和对数据的操作看成一个不可分割的整体，力求将现实中的不同问题简化处理。这样既符合人们的思维习惯，同时也可以增加软件开发后期的可维护性。因为面向对象编程可以提高程序员的开发效率，目前它已经成为当下最流行的开发方式之一。

4.1 面向对象编程概述

在现实生活中，任何事物都有各自的状态和行为。把事物的状态抽象到计算机语言中以后，就成为某一实例的属性，把事物的行为抽象到计算机语言中，就成了具体的方法，或者被称为函数。面向对象的思想就出自这里。

面向对象编程（Object-Oriented Programming，OOP）技术是开发计算机应用程序的一种新方法、新思想。过去的面向过程编程常常会导致所有的代码都包含在几个模块中，使程序难以阅读，同时很难进行维护。对一些程序进行修改时常常牵一动百，使后期的维护工作很难进行。

使用 OOP 技术开发软件，也可以将软件的开发分为各个模块，每个模块都只提供特定的功能。它们之间彼此独立，同时又相互联系，每个模块都可以重复使用。这样大大增加了代码的重用率，有利于软件的开发、维护和升级。

在面向对象思想中，将算法与数据结构看做是一个整体，称作对象。关于算法，就是解决问题的方法和策略，数据结构就是事物中各个属性之间的联系和组织形式。在现实世界中，任何类的对象都具有一定的属性和功能，也总能用数据结构与算法将它们的属性和功能合起来描述。因此可以用下面的等式来定义对象和程序。

对象=（算法+数据结构）

程序=（对象+对象+……+对象）

从上面的等式可以看出，程序就是许多对象在计算机中的集合，而对象则是一个又一个的程序实例化。对象与算法和数据结构的关系就像鸟类和鸟的特征和行为的关系一样,整个鸟类就是由鸟的这些特征和行为组成的，如图 4.1 所示。

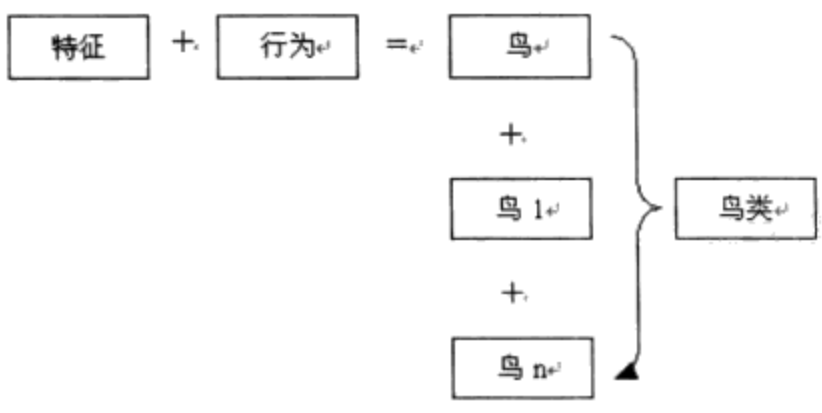


图 4.1 鸟类的抽象

4.2 类

4.2.1 类的概念

通常把具有相同性质和功能的事物构成的集合体叫做类。类描述了一系列在概念上有相同含义的对象，为这些对象统一定义了编程语言上的属性和方法。例如，苹果、香蕉、人、花、鸟、虫、鱼等都可以看做是类。所有的水果可以看做水果类，所有的蔬菜可以看做蔬菜类，而

具体的某一种蔬菜都是蔬菜类的一个实例，在 OOP 的术语中，叫做对象，如图 4.2 所示。

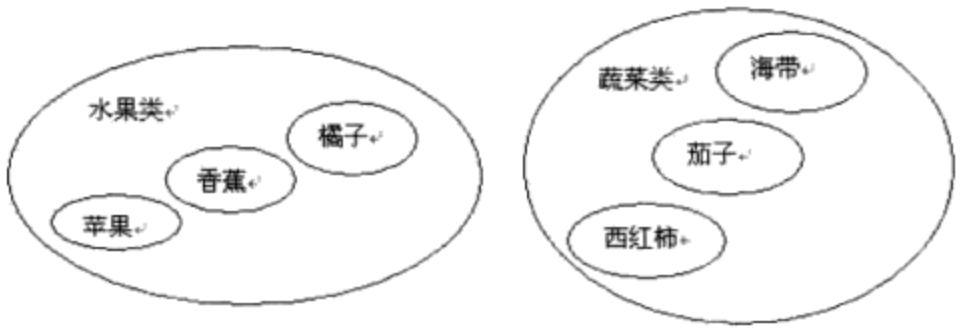


图 4.2　类

4.2.2　类的声明

类是 C#中功能最为强大的数据类型。类定义了数据类型的数据和行为。在 C#中，类的声明格式如下：

```
类的属性集　类的修饰符　关键字　class　类名　继承方式　基类名
{
    类的成员
}
```

在上面显示的格式中，除 class 关键字和类名外，其他的都是可选项，可以根据实际情况进行选择。例如在下面的类中，是一个关于车的类，代码如下：

```
01  public class Car                                        //定义一个车类
02  {
03      int Length,Width;                                   //公有成员，车的长和宽
04      private float Height;                               //私有成员，车的高度
05      public Car(int length,int width,int height)         //重载车类的构造函数
06      {
07          Length=length;                                  //成员变量，车的长度
08          Width=width;                                    //成员变量，车的宽度
09          Height=height;                                  //成员变量，车的高度
10      }
11      public void Information()                           //成员方法
12      {
13          Console.WriteLine("我有{0}高",Height);          //输出车类的高度
14          Console.WriteLine("我有{0}长",Length);          //输出车类的长度
15          Console.WriteLine("我有{0}宽",Width);           //输出车类的宽度
16      }
17      public void Speed()                                 //成员方法，车类的功能
18      {
19          Console.WriteLine("我可以加速");                //输出车类可以加速的功能
20      }
21  }
```

在上述实例程序中，第 01 行代码就是声明 Car 类。第 03 行、04 行代码定义它的成员变量 Length、Width、Height，分别代表车的长度、宽度和高度，这里可以理解为 Car 类的 3 个属性。代码第 05 行是构造函数，这是一种面向对象的术语，它的作用是初始化 Car 类的一些数据，例如长度、宽度和高度。第 05 行的构造函数在这里被称为重载的构造函数，它有 3 个参数，用来传递接收的成员变量，就如第 07 行、08 行以及 09 行代码一样。在第 17 行处的代码，叫做成员方法。就像人类具有说话的功能一样，这个成员方法就是 Car 类的功能。在类中，成员变量和成员方法统称为类的成员。

public 是类的修饰符之一，每一个类都具有各自不同的属性和方法，因此它们可以实现不同的功能，决定不同的类之间是否可以访问是通过修饰符来实现的。下面介绍几个常用的修饰符。

（1）new：仅允许在嵌套类声明时使用，表明类中隐藏了由基类中继承而来的并与基类中

同名的成员。

（2）public：表示不限制对该类的访问。

（3）protected：表示只能从其所在类和所在类的子类中进行访问。

（4）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。

（5）private：只能在类的内部进行访问。

（6）abstract：抽象类，不允许创建类的实例。

（7）sealed：密封类，不允许被其他类继承。

4.3 对　　象

4.3.1 对象的声明和实例化

对象是 OOP 应用程序的一个组成部件，该部件可以是一个过程、一些数据或者一些更为抽象的实体。C#中的对象是从类型中创建的，就像声明变量一样，对象的类型在 C#中被称为“类”。声明类的一个对象称为是该类的实例化，“类的实例”和对象表示相同的含义。在声明一个类时，并没有为其分配内存空间，只有在实例化一个对象时，才为对象分配空间。声明一个对象和声明一个变量类似。

下面的代码声明了一个整型变量 i 和一个 MyPet 类对象 Dog。

```
int i;                                  //声明一个整型变量
MyPet Dog;                              //声明一个类 Dog
Dog=new MyPet();                        //实例化 Dog 类
```

4.3.2 对象的生命周期

在每个对象中，或者说在类的每个实例中，至少有一个属性具有状态。对象的状态是指在某一时刻对象中属性的值，实际上，每个对象都有一个明确定义的生命周期，除了使用时的正常状态外，还包含如下两个重要的阶段。

- 构造阶段：对象的初始化过程，由构造函数来完成。
- 析构阶段：对象在被删除时的一些清理工作，如释放内存等，由析构函数来完成。

在构造阶段，对象的初始化过程是自动完成的，编程人员不需要专门设置一个存储该新对象的内存空间，但必须初始化对象存储的数据，构造函数的作用便是初始化这些存储数据。

在析构阶段，编程人员不需要专门编写代码，而由.NET Framework 提供的默认析构函数来完成。但是，由于析构函数只是在删除对象时才对该对象占用的资源进行清理，当某个对象长期不再使用时，它将一直占用其拥有的系统资源，如果这些资源十分重要，这将引起重大问题。解决这个问题的方法就是使用来自 IDisposable 接口的 Dispose 方法，该方法为对象执行必要的清理。

知识扩展：在解决释放非托管资源这类问题时，可以使用 Using 语句。Using 语句提供能确保正确使用 IDisposable 对象的方便语法。

4.3.3 对象与类的关系

类就是具有同样性质和功能的东西构成的集合。在计算机语言中，它是一种抽象的数据类型，但是抽象的程度可能不同，而对象就是一个类的实例。

例如，将树木设计为一个类，杨树和柳树各为一个实例。从这里可以看出，杨树和柳树有很多共同点，它们都需要阳光、雨露，每天都要进行光合作用。对于这样相似的对象就可以将其抽象为一个数据类型，在这里抽象为树木。这样，只要将树木这个数据类型编写好，程序中

就可以方便的创建杨树、柳树这样的实例。在代码需要改动时，只需改动树木这个基类就可以实现目的。

4.4 构造函数和析构函数

4.4.1 默认的构造函数

构造函数是类中比较特殊的成员函数，它主要用来为对象分配内存空间，完成初始化操作。一般来说，构造函数一旦开始执行，对象的生命周期就开始了，最终它将以析构函数结束。如果一个类含有构造函数，在实例化该类的对象时该函数就会被调用。在 C#编程语言中，类的构造函数具有如下一些特征。

- 构造函数和类的名称一致。
- 当类没有构造函数时，系统将会创建一个默认的构造函数来对类进行初始化。
- 在大多说情况下，构造函数的访问修饰符是 public，但有时候也会有 private，此时表示该类不能进行实例化，通常用于只含有静态成员的类中。
- 构造函数没有返回值类型，空值也不可以。
- 构造函数可以进行重载。

在开发程序时，程序员可以定义一个默认的构造函数，只要构造函数没有任何参数类型就可以。

示例 04-01　利用默认构造函数初始化类的成员变量

下面的代码定义了一个 Car 类，在该类中定义了一个默认的构造函数用来初始化 Car 类的成员变量。代码如下：

代码 04-01　光盘位置：光盘\mingrisoft\sl\04\01

```
namespace ConstructFunction
{
    class Program
    {
        static void Main(string[] args)
        {
            Car MicroBus=new Car();                                  //初始化一个 Car 类的实例
            Console.WriteLine("输出该车的车轮个数：");                //输出提示性信息
            Console.WriteLine("该车有{0}个车轮", MicroBus.Wheels);    //输出该车的车轮数
            Console.ReadLine();                                      //从标准输入流读取下一行字符
        }
    }
    class Car                                                        //自定义一个 Car 类
    {
        public Car()                                                 //自定义的默认构造函数
        {
            Weight = 200;                                            //为成员变量 Weight 赋值
            Wheels = 4;                                              //为成员变量 Wheels 赋值
        }
        ~Car()                                                       //析构函数
        {
            Weight = 0;                                              //清空 Weight 的值
            Wheels = 0;                                              //清空 Wheels 的值
        }

        public int Weight;                                           //定义一个 int 型变量 Weight
        public int Wheels;                                           //定义一个 int 型变量 Wheels
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.3 所示。

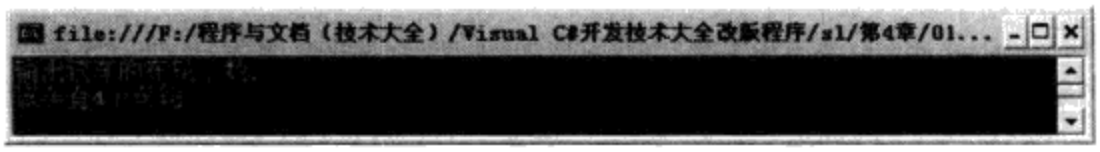


图 4.3　利用默认构造函数初始化类的成员变量

4.4.2　重载的构造函数

构造函数与其他的成员函数一样，可以被重载。在调用构造函数时，编译器会根据参数的不同而执行不同的构造函数。

示例 04-02　利用自定义构造函数初始化类的成员变量

下面的代码定义了一个 Car 类，在该类中定义了一个默认的构造函数和一个重载的构造函数，它们均用来初始化 Car 类的成员变量。在对类进行实例化时，依据参数决定使用哪一个构造函数。代码如下：

代码 04-02　光盘位置：光盘\mingrisoft\sl\04\02

```
namespace BurdenConstructFunction
{
    class Program
    {
        static void Main(string[] args)
        {
            Car MicroBus = new Car(400,8);                          //初始化一个 Car 的实例
            Console.WriteLine("输出该车的车轮个数：");              //输出提示性信息
            Console.WriteLine("该车有{0}个车轮", MicroBus.Wheels);  //输出该车的车轮数
            Console.ReadLine();                                     //从标准输入流读取下一行字符
        }
    }
    class Car                                                       //自定义一个 Car 类
    {
        public Car()                                                //自定义的默认构造函数
        {
            Weight = 200;                                           //为成员变量 Weight 赋值
            Wheels = 4;                                             //为成员变量 Wheels 赋值
        }
        public Car(int weight, int wheels)                          //重载构造方法
        {
            Weight = weight;                                        //为成员变量 Weight 赋值
            Wheels = wheels;                                        //为成员变量 Wheels 赋值
        }
        ~Car()                                                      //析构函数
        {
            Weight = 0;                                             //清空 Weight 的值
            Wheels = 0;                                             //清空 Wheels 的值
        }
        public int Weight;                                          //定义一个 int 型变量 Weight
        public int Wheels;                                          //定义一个 int 型变量 Wheels
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.4 所示。

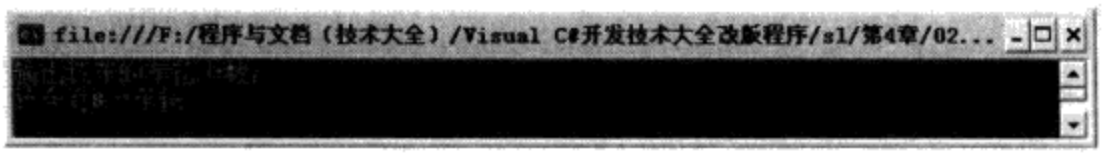


图 4.4　利用自定义构造函数初始化类的成员变量

4.4.3 析构函数

析构函数与构造函数相对，它是在对象被撤销后清除并释放所分配的内存，并且它与构造函数的名字相同。但是，析构函数需要在名字前加上一个波浪号（～）。当实例化一个类的对象后，这个对象在销毁时就会自动调用析构函数，例如释放对象占用的内存空间等。析构函数具有以下特点。

析构函数不能带有参数。

析构函数不能拥有访问修饰符，例如，不能加上 public 等。

析构函数的名字是在类名前加上一个波浪号(～)。例如 Car 类的析构函数为～Car()。

析构函数与构造函数的调用方式类似，它是在对象被销毁时自动被调用。

说明：在实际的项目开发过程中，有时候看到调用系统资源，但没有看到使用析构函数。其实，这个时候对象销毁的过程依然存在，只是这个过程由系统自动完成。

示例 04-03 使用析构函数

下面的代码定义了一个 Car 类，在该类中定义了一个默认的构造函数用来初始化 Car 类的成员变量。代码如下：

代码 04-03 光盘位置：光盘\mingrisoft\sl\04\03

```
namespace ExtractFunction
{
    class Program
    {
        static void Main(string[] args)
        {
            Car MicroBus;                                   //声明一个 Car 类对象
            MicroBus = new Car();                           //实例化一个 Car 类对象
            MicroBus = new Car(400, 8);                     //实例化一个 Car 类对象
        }
    }
    class Car                                               //自定义一个 Car 类
    {
        public Car()                                        //自定义的默认构造函数
        {
            Weight = 200;                                   //为成员变量 Weight 赋值
            Wheels = 4;                                     //为成员变量 Wheels 赋值
        }
        public Car(int weight, int wheels)                  //重载构造方法
        {
            Weight = weight;                                //为成员变量 Weight 赋值
            Wheels = wheels;                                //为成员变量 Wheels 赋值
        }
        ~Car()                                              //析构函数
        {
            Weight = 0;                                     //清空 Weight 的值
            Wheels = 0;                                     //清空 Wheels 的值
            Console.WriteLine("资源释放完毕");              //输出释放资源完成的信息
        }
        public int Weight;                                  //定义一个 int 型变量 Weight
        public int Wheels;                                  //定义一个 int 型变量 Wheels
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.5 所示。

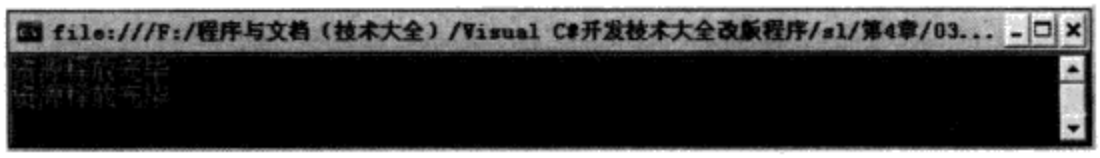


图 4.5　使用析构函数

4.5　方　　法

方法是一种对象或类的成员，用来实现具体的某一个功能。一个类的方法决定了通过这个类创建的实例的行为，一个类的所有方法决定了这个类所具有的功能。另外，有时候讲方法其实也就是函数，本节将对方法进行详细的讲解。

4.5.1　方法的声明

方法是类功能的实现方式，一个标准的方法需要指定访问级别，例如是公有还是私有等；还要注意写明方法的返回值类型，例如是整型还是字符型；另外，还要写清楚方法的名称以及方法的参数，例如方法名以及参数的类型和参数的个数等。方法参数写在括号中，用逗号分隔开。括号中如果没有参数，表示该方法为无参方法。方法的声明格式如下：

```
方法的访问属性 返回值 类型方法名（参数列表）
{
    方法的具体实现；
}
```

下面的代码中，类 MyClass 只有一个方法，该方法用来返回一个字符串，代码如下：

```
Class MyClass
{
        public string GetString()
        {
            return "It is a string"
        }
}
```

说明：当方法被关键字 static 修饰时，如果需要使用该方法必须用类名进行引用；如果声明一个 void 类型的方法，return 语句可以省略，如果是一个非 void 类型方法，则方法中必须有 return 语句。

方法被不同的修饰限定符修饰就会具有不同的特性，常见的修饰限定符如下。

- Virtual：可以用于修饰方法、属性、索引器，并使它们可以在派生类中被重写。例如，此方法可被任何继承它的类重写。
- Abstract：该修饰符可以和类、方法、属性、索引器及事件一起使用，如果在声明一个类时使用该修饰符，那么该类只能被其他的类继承，而类中的成员必须通过从抽象类派生的类来实现。
- Override：该修饰符用于扩展或修改继承的方法、属性、索引器或事件的抽象实现或虚实现。
- Extern：该修饰符用于声明在外部实现的方法，例如在使用 Interop 服务调入非托管代码时与 DllImport 属性一起使用。在这种情况下，还必须将方法声明为 static，例如：

```
[DllImport("avifil32.dll")]
private static extern void AVIFileInit();
```


4.5.2　方法的类型

通常情况下，在面向对象技术中，方法有两种：静态方法和非静态方法。在声明一个方法时，如果有 static 关键字修饰，那么该方法为静态方法；否则，该方法为非静态方法。静态方

法不能对类的实例进行操作，同时也不能用 this 关键字修饰；而非静态方法可以对类的实例进行操作，同时也可以用 this 关键字修饰。

示例 04-04 使用静态方法

下面的代码定义了一个 StaticClass 类，在该类中定义了一个静态方法 Measure，该方法用来返回一个字符串。代码如下：

代码 04-04 光盘位置：光盘\mingrisoft\sl\04\04

```
namespace StaticMeasure
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("方法的调用类型为：");                    //输出提示性信息
            Console.WriteLine(StaticClass.Measure());                   //调用自定义静态方法
            Console.ReadLine();                                         //从标准输入流读取下一行字符
        }
    }
    class StaticClass
    {
        public static string Measure()                                  //定义一个静态方法
        {
            string Content = "静态方法调用";                            //定义一个字符串
            return Content;                                             //返回字符串的值
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.6 所示。

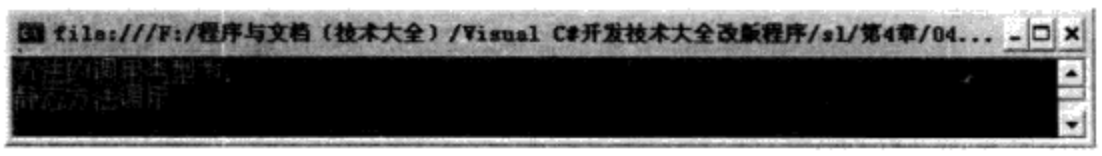


图 4.6 使用静态方法

示例 04-05 使用非静态方法

下面的代码定义了一个 StaticClass 类，在该类中定义了一个返回字符串的非静态方法 Measure 和一个用来叠加字符串的 Splice 方法。在类 Program 中定义了一个返回字符串方法 Onestring，该方法是一个非静态方法。代码如下：

代码 04-05 光盘位置：光盘\mingrisoft\sl\04\05

```
namespace UnStaticMeasure
{
    class Program
    {
        public string OneString()                                       //自定义一个返回字符串方法
        {
            string str1 = "这是一个";                                   //定义一个字符串
            return str1;                                                //返回字符串的内容
        }
        static void Main(string[] args)
        {
            StaticClass Static1 = new StaticClass();                    //实例化类 StaticClass 的一个实例
            Static1.Splice();                                           //调用非静态方法 Splice
            Console.ReadLine();                                         //从标准输入流读取下一行字符
        }
    }
```

```
    class StaticClass
    {
        public string Measure()                                    //定义一个非静态方法
        {
            string Content = "非静态方法调用";                      //定义一个字符串
            return Content;                                        //返回字符串的值
        }
        public void Splice()                                       //自定义的字符串叠加函数
        {
            Program p1 = new Program();                            //实例化类 Program 的一个实例
            Console.WriteLine(p1.OneString() + this.Measure());    //输出叠加的内容
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.7 所示。

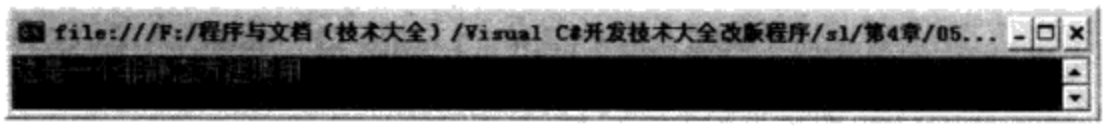


图 4.7　使用非静态方法

4.5.3　方法的参数类型

如果方法要更改数值，有时需要传递值给方法并从方法获得返回值。下面是传递参数时的 4 种情况。

（1）值参数：定义值类型参数的方式很简单，只要注明参数类型和参数名即可。当该方法被调用时，便会为每个值类型参数分配一个新的存储空间，然后将对应的表达式运算的值复制到该内存空间。在方法中更改参数的值不会影响到这个方法外的变量。

示例 04-06　使用值参数传递

下面的代码定义了一个静态方法 NotSwap，该方法用来交换传递的值。代码如下：

代码 04-06　光盘位置：光盘\mingrisoft\sl\04\06

```
namespace ValueParameterPass
{
    class Program
    {
        static void NotSwap(int x, int y)                          //自定用于进行交换的方法 NotSwap
        {
            Console.WriteLine("进入 NotSwap 时：x={0},y={1}", x, y); //输出提示性信息
            int temp = x;                                          //将 x 的值赋给 temp
            x = y;                                                 //将 y 的值赋给 x
            y = temp;                                              //将 temp 的值赋给 y
            Console.WriteLine("退出 NotSwap 时：x={0},y={1}", x, y); //输出提示性信息
        }
        static void Main(string[] args)
        {
            int i = 1, j = 2;     //初始化两个变量 i、j
            Console.WriteLine("执行 NotSwap 前：i={0},j={1}", i, j);      //输出提示性信息
            NotSwap(i, j);                                                //调用方法 NotSwap
            Console.WriteLine("执行 NotSwap 后：i={0},j={1}", i, j);      //输出提示性信息
            Console.ReadLine();                                           //从标准输入流读取下一行字符
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.8 所示。

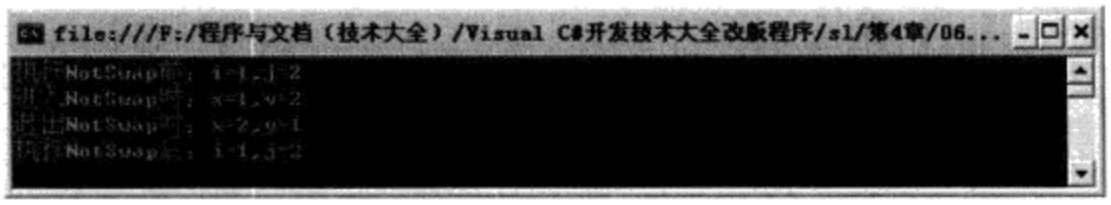


图 4.8　使用值参数传递

（2）引用参数：与传递值类型参数不同，引用类型的参数并没有再分配内存空间，实际上传递的是指向原变量的指针，即引用参数和原变量保存在同一地址。

示例 04-07　使用引用参数传递

下面的代码定义了一个静态方法 NotSwap，该方法用来交换传递的值。代码如下：

代码 04-07　光盘位置：光盘\mingrisoft\sl\04\07

```
namespace QuoteParameterPass
{
    class Program
    {
        static void NotSwap(ref int x,ref int y)                          //自定义用于进行交换的方法 NotSwap
        {
            Console.WriteLine("进入 NotSwap 时：x={0},y={1}", x, y);  //输出提示性信息
            int temp = x;                                                 //将 x 的值赋给 temp
            x = y;                                                        //将 y 的值赋给 x
            y = temp;                                                     //将 temp 的值赋给 y
            Console.WriteLine("退出 NotSwap 时：x={0},y={1}", x, y);  //输出提示性信息
        }
        static void Main(string[] args)
        {
            int i = 1, j = 2;                                             //初始化两个变量 i、j
            Console.WriteLine("执行 NotSwap 前：i={0},j={1}", i, j);   //输出提示性信息
            NotSwap(ref i,ref j);                                         //调用方法 NotSwap
            Console.WriteLine("执行 NotSwap 后：i={0},j={1}", i, j);   //输出提示性信息
            Console.ReadLine();                                           //从标准输入流读取下一行字符
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.9 所示。

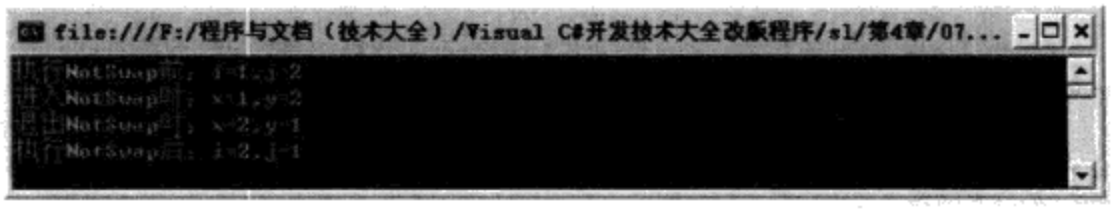


图 4.9　使用引用参数传递

（3）输出参数：在传递参数前加 out 关键字即可将该传递参数当作一个输出参数。输出参数用来返回一个结果。它和引用参数的区别是不必先初始化变量。

说明：当方法被关键字 static 修饰时，如果需要使用该方法必须用类名进行引用；如果声明一个 void 类型的方法，return 语句可以省略，如果是一个非 void 类型方法，则方法中必须有 return 语句。

示例 04-08　使用输出参数传递

下面的代码定义了一个静态方法 NotSwap，该方法用来实现输出参数传递。代码如下：

代码 04-08　光盘位置：光盘\mingrisoft\sl\04\08

```
namespace OutputParameterPass
{
```

```
    class Program
    {
        static void NotSwap(out int x, out int y)                    //自定义一个方法 NotSwap
        {
            x = 5;                                                   //实例化 x 的值
            y = 6;                                                   //实例化 y 的值
        }
        static void Main(string[] args)
        {
            int i = 3, j = 4;                                        //实例化 i，j 的值
            Console.WriteLine("执行 NotSwap 前：i={0},j={1}", i, j);  //输出 i，j 的值
            NotSwap(out i, out  j);                                  //调用自定义方法 NotSwap
            Console.WriteLine("执行 NotSwap 后：i={0},j={1}", i, j);  //输出 i，j 的值
            Console.ReadLine();                                      //从标准输入流读取下一行字符
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.10 所示。

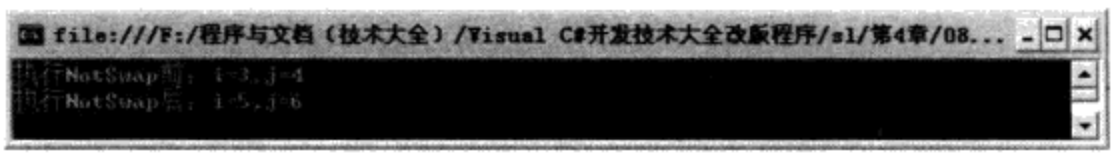


图 4.10　使用输出参数传递

（4）参数数组：参数数组必须用 params 修饰词明确指定。在方法的参数列表中只允许出现一个参数数组，而且在方法同时具有固定参数和参数数组的情况下，参数数组必须放在整个参数列表的最后，并且参数数组只允许是一维数组，不允许将 params 修饰符与 ref 修饰符和 out 修饰符组合起来使用。与参数数组对应的实参可以是同一类型的数组名，也可以是任意多个与该数组的元素属于同一类型的变量，若实参是数组则按引用传递，若实参是变量或表达式则按值传递。

示例 04-09　使用参数数组传递

下面的代码定义了一个静态方法 Print，该方法用来实现参数数组传递。代码如下：

代码 04-09　光盘位置：光盘\mingrisoft\sl\04\09

```
namespace ParamsArrayPass
{
    class Program
    {
        static void Print(params int[] args)                         //自定义输出内容方法 Print
        {
            Console.WriteLine("调用 Print：数组包含{0}个元素", args.Length);//输出数组的长度
            for (int i = 0; i < args.Length; i++)                    //循环遍历数组中的每一个元素
            {
                Console.Write("{0}  ", args[i]);                     //输出数组中的每一个元素
            }
            Console.WriteLine();                                     //将当前行中止符写入标准输出流
        }
        static void Main(string[] args)
        {
            int[] arr = new int[3] { 1, 2, 3 };                      //初始化一个一维整型数组
            Print(arr);                                              //输出数组 arr 的内容
            Print(10, 20, 30, 40);                                   //输出对应的内容
            Print();                                                 //空值传递
            Console.ReadLine();                                      //从标准输入流读取下一行字符
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.11 所示。

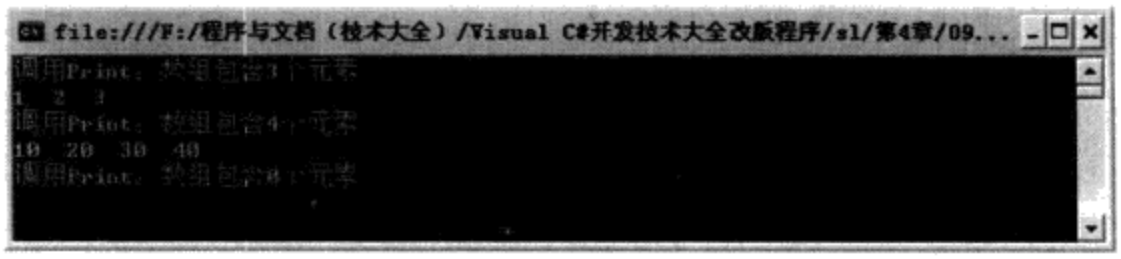


图 4.11 使用参数数组传递

4.5.4 方法的返回值类型

方法可以向调用方返回值。如果返回类型（方法名称前列出的类型）不为空，则可以使用 return 关键字向调用方返回相应的值。如果返回类型为 void，则可使用没有值的 return 语句来停止方法的执行。如果没有 return 关键字，方法执行到代码块末尾时即结束执行。

示例 04-10 求两个数相加后的数的平方

下面的代码定义了一个类 Math，在该类的内部有计算两个数之和的 AddTwoNumbers 方法和求一个数平方的 SquareANumber 方法。代码如下：

代码 04-10 光盘位置：光盘\mingrisoft\sl\04\10

```
namespace SimpleMath
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("请输入任意的两个整数：");                    //输出提示性信息
            int i = int.Parse(Console.ReadLine());                          //输入第一个整数
            int j = int.Parse(Console.ReadLine());                          //输入第二个整数
            //输出所输入的整数信息
            Console.WriteLine("你输入的第一个数为：{0}\n 你输入的第二个数为：{1}", i, j);
            Math SumAndSquare = new Math();                                 //实例化类 Math 的一个实例
            int result = SumAndSquare.AddTwoNumbers(i, j);                  //调用方法 AddTwoNumbers
            Console.WriteLine("两个数的和为：{0}", result);                 //输出两个数的和
            //输出两个数之和的平方
            Console.WriteLine("和的平方为：{0}", SumAndSquare.SquareANumber(result));
            Console.ReadLine();                                             //从标准输入流读取下一行字符
        }
    }
    class Math
    {
        public int AddTwoNumbers(int number1, int number2)      //自定义两数相加的方法 AddTwoNumbers
        {
            return number1 + number2;                           //返回两数相加的和
        }
        public int SquareANumber(int number)                    //自定义求平方的方法 SquareANumber
        {
            return number * number;                             //返回求平方后的值
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.12 所示。

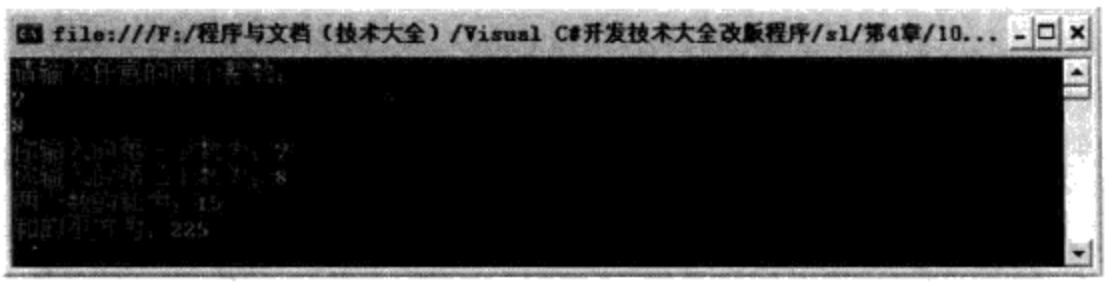


图 4.12　求两个数相加后的数的平方

说明：为进行方法重载，方法的返回类型不是方法签名的一部分。但是，在确定委托和委托所指向方法之间的兼容性时，返回类型是方法签名的一部分。

4.5.5　方法的重载

方法重载是指同样的一个方法名，有多种不同的实现方法。方法重载的格式是在一个类中两次或多次定义同名的方法，这些同名的方法包括从基类继承而来的方法，这些方法名称相同，但每个方法的参数类型或个数不同，从而便于在用户调用方法时系统能够自动识别应调用的方法。

示例 04-11　方法重载的应用

下面的代码定义了一个 Add 方法，该方法因为参数类型的不同而实现不同的功能。代码如下：

代码 04-11　光盘位置：光盘\mingrisoft\sl\04\11

```
namespace MeasureBurden
{
    class Program
    {
        public static int Add(int i, int j)                          //自定义一个求两数之和的 Add 方法
        {
            return i + j;                                            //返回两个数的和
        }
        public static string Add(string s1, string s2)               //自定义一个拼接字符串的 Add 方法
        {
            return s1 + s2;                                          //返回两个字符串拼接的结果
        }
        public static long Add(long x)                               //自定义一个求和的 Add 方法
        {
            return x + 5;                                            //返回该值与 5 的和
        }
        static void Main(string[] args)
        {
            Console.WriteLine("输入任意的两个整数：");               //输出提示性信息
            int i = int.Parse(Console.ReadLine());                   //输入一个整数
            int j = int.Parse(Console.ReadLine());                   //输入一个整数
            Console.WriteLine("输出两个数的和：");                   //输出提示性信息
            Console.WriteLine(Add(i, j));                            //输出两个数的和
            Console.WriteLine("输入任意的两个字符串：");             //输出提示性信息
            string str1 = Console.ReadLine();                        //输入一个字符串 str1
            string str2 = Console.ReadLine();                        //输入一个字符串 str2
            Console.WriteLine("输出两个字符串相加的结果：");         //输出提示性信息
            Console.WriteLine(Add(str1, "    " + str2));             //输出两个字符串拼接的结果
            Console.WriteLine("输入任意的一个整数：");               //输出提示性信息
            int p = int.Parse(Console.ReadLine());                   //输入一个整数 p
            Console.WriteLine("输出该数加 5 后的结果：");            //输出提示性信息
            Console.WriteLine(Add(p));                               //输出相加后的结果
            Console.ReadLine();                                      //从标准输入流读取下一行字符
```

```
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.13 所示。

图 4.13　方法重载的应用

4.6　域和属性

4.6.1　域

域也叫成员变量，它表示存储位置，可以用来保存类的各种数据信息。域是 C#中不可缺少的一部分，相当于 C#语言中的简单成员变量，但是它表示与对象或类相关联的变量。它还有一个重要的用途，即引入一个或多个给定类型的域。

域本质上讲就是一种变量，所以它的声明和变量是一样的，声明时可用的修饰符有 new、public、protected、internal、private、static、readonly、const 等。一个声明了多个域的域声明和单个域的多个域声明作用相同。例如：

```
private int a;
private int b;
private int c;
```

等同于：

```
private int a,b,c;
```

域又分为静态域和实例域。当域的声明中包含 static 修饰符时，该域就是静态域。如果域的声明不包含 static 修饰符，则该域就是实例域。实例域属于具体的对象，为特定的对象专有。当创建一个对象时，它的实例变量赋予对应于此对象的值，而且这些值在对象的生存期间与对象保持一致。

静态域属于类，是所有对象共同拥有的。C#中严格规定实例域只能通过对象来获取，静态域只能通过类来获得。

4.6.2　属性

属性是一种用于访问对象或类的特性的成员。属性可以包括字符串的长度、字体的大小、窗体的标题、客户的名称等。属性是成员的自然扩展，二者都是关联类型的命名成员。访问成员和属性的语法是相同的。

与成员不同的是，属性不表示存储位置。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。属性提供了一种机制，它把读取和写入对象的某些特性与一些操作关联起来。这种机制用来读取、编写或计算私有成员的值，可以像使用公共数据成员一样使用属性，这使得数据在可被轻松访问的同时仍能提供方法的安全性和灵活性。

属性结合了字段和方法的多个方面。对于对象的用户，属性显示为字段，访问该属性需要完全相同的语法。对于类的实现者，属性是一个或两个代码块，表示一个 get 访问器或一个 set 访问器。当读取属性时，执行 get 访问器的代码块；当向属性分配一个新值时，执行 set 访问器的代码块。不具有 set 访问器的属性被视为只读属性，不具有 get 访问器的属性被视为只写属性，同时具有这两个访问器的属性是读写属性。

与成员不同，属性不作为变量来分类。因此，不能将属性作为 ref 参数或 out 参数传递。

属性具有多种用法。它们可在允许更改前验证数据；可透明地公开某个类上的数据，该类的数据实际上是从其他源（例如数据库）检索到的；当数据被更改时，它们可采取行动，例如，引发事件或更改其他字段的值。

属性在类模块内是通过以下方式声明的：指定字段的访问级别、属性的类型、属性的名称，然后声明 get 访问器或 set 访问器的代码模块。其声明格式如下：

```
访问级别   属性类型   属性名称
{
        get
        {
            读取模块
        }
        set
        {
            写入模块
        }
}
```

例如，下面在类中声明了一个属性，代码如下：

```
public class Date
{
    private int Day = 7;                          //定义一个私有变量
    public int day                                //定义一个属性
    {
        get                                       //get 访问器
        {
            return Day;                           //返回 Day 的值
        }
        set                                       //set 访问器
        {
            if ((value > 0) && (value < 8))       //当 value 的值大于 0 并且小于 8 时
            {
                Day = value;                      //为 Day 赋值
            }
        }
    }
}
```

get 访问器体与方法体相似，必须返回属性类型的值。执行 get 访问器相当于读取字段的值。例如，当正在从 get 访问器返回私有变量并且启用了优化时，对 get 访问器方法的调用由编译器进行内联，因此不存在方法调用的系统开销。然而，由于在编译时编译器不确定在运行时实际调用哪个方法，无法内联虚拟 get 访问器。

set 访问器类似于返回类型为 void 的方法。它使用称为 value 的隐式参数，此参数的类型是属性的类型。

4.7 委　　托

4.7.1 委托的用途

委托是一种引用方法的类型。一旦为委托分配了方法，委托将与该方法具有完全相同的行

为。委托方法的调用可以像其他任何方法一样，具有参数和返回值。在定义了委托后，就可以声明该委托类型的变量。接着把这个变量初始化为与委托有相同签名的函数引用。然后，就可以使用委托变量调用这个函数，此时对变量的调用就等同于对函数的调用。下面的语句声明了一个委托：

```
public delegate int PerformCalculation(int x, int y);
```

- 委托的功能非常强大，它具有以下特点。
- 委托类似于 C++函数指针，但它们是类型安全的。
- 委托允许将方法作为参数进行传递。
- 委托可用于定义回调方法。
- 委托可以链接在一起，例如，可以对一个事件调用多个方法。
- 方法不必与委托签名完全匹配。

4.7.2 委托的实现

委托是一种可以把引用存储为函数的类型。委托的实现非常的类似于函数，但不带函数体，且要使用 delegate 关键字。委托的实现指定了一个函数签名，其中包含一个返回类型和参数列表。委托的一般声明格式如下：

```
public delegate void Del<T>(T item);
public void Notify(int i) { }
Del<int> d1 = new Del<int>(Notify);
```

在委托的实现过程中，委托的签名必须与方法的签名保持一致，这些签名包括返回值、参数类型、参数个数以及修饰符。

示例 04-12 利用委托实现四则运算

本实例运用委托实现两个数的四则运算，其中 Sum 方法用来实现加法运算，Subtract 方法用来实现减法运算，Multiply 方法用来实现乘法运算，Divide 方法用来实现除法运算。代码如下：

代码 04-12 光盘位置：光盘\mingrisoft\sl\04\12

```
namespace DelegateApplication
{
    class Program
    {
        public delegate int DelegateApplication(int param1,int param2);    //定义一个委托对象
        public static int Sum(int param1, int param2)                      //自定义一个计算两个数之和的方法
        {
            return param1 + param2;                                        //返回两个数的和
        }
        public static int Subtract(int param1, int param2)                 //自定义一个计算两个数之差的方法
        {
            return Math.Abs(param1 - param2);                              //返回两个数的差
        }
        public static int Multiply(int param1, int param2)                 //自定义一个计算两个数之积的方法
        {
            return param1 * param2;                                        //返回两个数的积
        }
        public static int Divide(int param1, int param2)                   //自定义一个计算两个数之商的方法
        {
            return param1 / param2;                                        //返回两个数的商
        }
        static void Main(string[] args)
        {
            DelegateApplication Operate;                                   //声明一个委托对象
            Console.Write("输入两个整数 x,y:\n");                          //输出提示性信息
            int x = int.Parse(Console.ReadLine());                         //输入一个整数 x
```

```
            int y = int.Parse(Console.ReadLine());                          //输入一个整数 y
            Console.WriteLine("输出输入的两个数：");                          //输出提示性信息
            Console.WriteLine("x={0}\ny={1}", x, y);                        //输出 x，y 的值
            Console.Write("输入要进行的计算：");                              //输出提示性信息
            string operate = Console.ReadLine();                            //从键盘输入将进行的计算类型
            if (operate == "+")                                             //当输入的为加法运算式时
            {
                Operate = new DelegateApplication(Sum);                     //实例化一个委托对象
                Console.WriteLine("输出两数的计算结果：{0}", Operate(x, y));     //输出计算后的结果
            }
            else if (operate == "-")                                        //当输入的为减法运算式时
            {
                Operate = new DelegateApplication(Subtract);                //实例化一个委托对象
                Console.WriteLine("输出两数的计算结果：{0}", Operate(x, y));     //输出计算后的结果
            }
            else if (operate == "*")                                        //当输入的为乘法运算式时
            {
                Operate = new DelegateApplication(Multiply);                //实例化一个委托对象
                Console.WriteLine("输出两数的计算结果：{0}", Operate(x, y));     //输出计算后的结果
            }
            else if (operate == "/")                                        //当输入的为除法运算式时
            {
                Operate = new DelegateApplication(Divide);                  //实例化一个委托对象
                Console.WriteLine("输出两数的计算结果：{0}", Operate(x, y));     //输出计算后的结果
            }
            else                                                            //当为其他输入情况时
            {
                Console.WriteLine("对不起，请确认你的输入参数，谢谢合作");      //输入提示性信息
            }
            Console.ReadLine();                                             //从标准输入流读取下一行字符
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.14 所示。

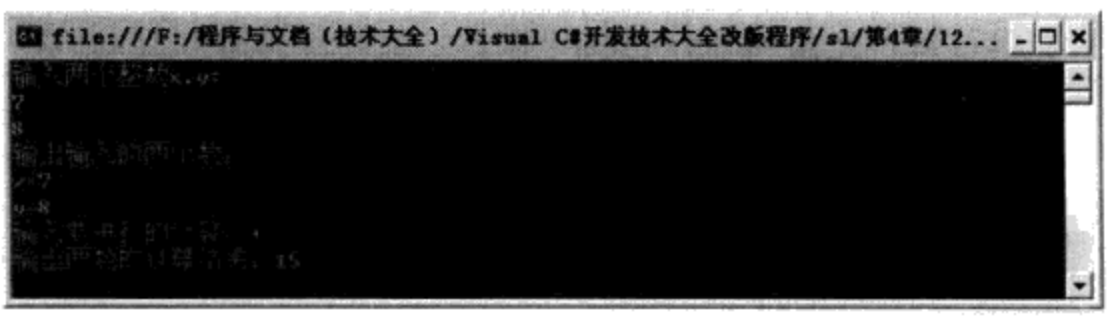


图 4.14　利用委托实现四则运算

4.8　事件和索引器

4.8.1　事件

事件是类在发生其关注的事情时用来提供通知的一种方式。例如，封装用户界面控件的类可以定义一个在用户单击该控件时发生的事件。控件类不关心单击按钮时发生了什么，但它需要告知派生类单击事件已发生，然后，派生类可选择如何响应。

C#是一个完全面向对象的开发语言，所以允许编写自定义的事件。实际上，事件就是一个回呼（callback）的函数指针，在 C#中是通过 delegate 实现的。

在.NET 的事件模型中，主要的角色有事件发行者和事件订阅者。事件的发行者是触发事件的对象，而事件的订阅者指的是在某种事件发生时被通知者。

当发生与某个对象相关的事件时，类和结构会使用事件将这一对象通知给用户，这种通知即称为“引发事件”。引发事件的对象称为事件的源或发送者。对象引发事件的原因很多，响应对象数据的更改、长时间运行的进程完成或服务中断。例如，一个对象在使用网络资源时如果丢失网络连接，则会引发一个事件。用户界面元素的对象通常会引发事件来响应用户操作，如按钮单击或菜单选择。

如果要在程序中使用事件，就必须先声明事件。声明事件的方法如下。

- 引发名为 EventName 事件的类具有以下成员：

```
public event EventNameEventHandler EventName;
```

- EventName 事件的事件委托是 EventNameEventHandler，它具有以下签名：

```
public delegate void EventNameEventHandler(object sender, EventNameEventArgs e);
```

- 要使用 EventName 事件，事件处理程序必须与事件委托具有相同签名：

```
void EventHandler(object sender, EventNameEventArgs e) {}
```

在.NET Framework 中的事件签名，通常第一个参数为引用事件源的对象，第二个参数为一个传送与事件相关的数据的类。但是，在 C#中并不强制使用这种形式，只要事件签名返回 void，其他方面可以与任何有效的委托签名一样。

向类中添加事件需要使用 event 关键字，并提供委托类型和事件名称。例如：

```
public class EventSend
{
    public event EventNameEventHandler TestEvent;
    private void OneTestEvent() { /* ... */ }
}
```

事件可标记为 public、private、protected、internal 或 protected internal，这些访问修饰符定义类的用户访问事件的方式。

若要引发事件，类可以调用委托，并传递所有与事件有关的参数。然后，委托调用已添加到该事件的所有处理程序。如果该事件没有任何处理程序，则该事件为空。因此在引发事件之前，事件源应确保该事件不为空以避免 NullReferenceException。若要避免争用条件（最后一个处理程序会在空检查和事件调用之间被移除），在执行空检查和引发事件之前，事件源还应创建事件的一个副本。

每个事件都可以分配多个处理程序来接收该事件。这种情况下，事件自动调用每个接收器；无论接收器有多少，引发事件只需调用一次该事件。

要接收某个事件的类，可以创建一个方法来接收该事件，然后向类事件自身添加该方法的一个委托，这个过程称为“订阅事件”。

首先，接收类必须具有与事件自身具有相同签名（如委托签名）的方法。然后，该方法（称为事件处理程序）可以采取适当的操作来响应该事件。例如：

```
public class EventReceiver
{
    public void ReceiveTestEvent(object sender, System.EventArgs e)
    {
        System.Console.Write("Event received from ");
        System.Console.WriteLine(sender.ToString());
    }
}
```

每个事件可以有多个处理程序，多个处理程序由源按顺序调用。如果一个处理程序引发异常，则还未调用的处理程序没有机会接收事件。因此，建议事件处理程序迅速处理事件并避免引发异常。

若要订阅事件，接收器必须创建一个与事件具有相同类型的委托，并使用事件处理程序作为委托目标。然后，接收器必须使用加法赋值运算符（+=）将该委托添加到源对象的事件中。例如：

```
public void Subscribe(EventSend send)
{
    TestEventDelegate temp = new TestEventDelegate(ReceiveTestEvent);
    send.TestEvent+= temp;
}
```

若要取消订阅事件，接收器可以使用减法赋值运算符（-=）从源对象的事件中移除事件处理程序的委托。例如：

```
public void UnSubscribe(EventSource send)
{
    TestEventDelegate temp = new TestEventDelegate(ReceiveTestEvent);
    send.TestEvent-= temp;
}
```

说明：.NET Framework 中的事件委托命名为 EventNameEventHandler，而文档中的术语“事件处理程序”指的是事件处理方法。命名方案后的逻辑是 EventNameEventHandler 委托指向实际处理事件的事件处理程序（方法）。

示例 04-13　自定义事件的应用

本实例主要向 Timer 的 Elapsed 事件中添加自定义的内容。在本示例中定义了两个类：类 Connection 和类 Display。在类 Connection 中，定义了一个事件 MessageArrived，该事件当有满足条件的消息传递时发生。在类 Display 中，定义了一个 DisplayMessage 方法用来显示信息。代码如下：

代码 04-13　光盘位置：光盘\mingrisoft\sl\04\13

```
using System.Timers;                                          //添加该命名空间
namespace User_DefinedAffair
{
    class Program
    {
        public delegate void MessageHandler(string MessageContent);//自定义事件的委托
        static void Main(string[] args)
        {
            Connection UsConnection = new Connection();        //实例化一个类 Connection 的对象
            Display display = new Display();                   //实例化一个类 Display 的对象
            UsConnection.MessageArrived += new MessageHandler(display.DisplayMessage);//添加自定义事件
            UsConnection.Connect();                            //调用连接信息
            Console.ReadKey();                                 //获取用户按下的下一个字符或功能键
        }
        public class Connection
        {
            public event MessageHandler MessageArrived;        //声明一个自定义事件
            private Timer timer;                               //声明一个 Timer 类的实例
            public Connection()                                //类 Connection 的构造方法
            {
                timer = new Timer(100);                        //实例化 timer 对象
                timer.Elapsed += new ElapsedEventHandler(timer_Elapsed);//向 timer 的事件 Elapsed 中添加内容
            }
            private static Random random = new Random();       //实例化一个随机类的对象 random
            void timer_Elapsed(object sender, ElapsedEventArgs e) //新创建的事件方法
            {
                Console.WriteLine("正在查找信息…");             //输出提示性信息
                if (random.Next(9) == 0 && (MessageArrived != null))//当随机数等于 9 且事件的内容不为空时
                {
                    MessageArrived("Hello,Event!");            //输出“Hello，Event!”
                }
            }
            public void Connect()                              //用于连接的方法
            {
                timer.Start();                                 //启动计时器
            }
```

```
            public void DisConnect()                          //用于断开连接的方法
            {
              timer.Stop();                                   //停止计时器
            }
        }
        public class Display                                  //实例化一个用于显示信息的类 Display
        {
            public void DisplayMessage(string message)        //自定义一个显示信息的方法
            {
                Console.WriteLine("有信息传递：{0}", message); //输出信息
            }
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.15 所示。

图 4.15　自定义事件的应用

4.8.2　索引器

索引器允许类或结构的实例按照与数组相同的方式进行索引。索引器类似于属性，不同之处在于索引器的访问器采用参数。它不仅可以实现像数组那样对对象使用下标，而且提供了通过索引方式方便地访问类的数据信息的方法。

声明类或结构上的索引器时，可以使用 this 关键字，例如：

```
public int this[int index]      //声明索引器
{
    // get and set 访问
}
```

索引器的修饰符有 new、public、protected、internal、private、virtual、sealed、override、abstract 和 extern。当索引器声明包含 extern 修饰符时，称该索引器为外部索引器。因为外部索引器声明不提供任何实际的实现，所以它的每个访问器声明都由一个分号组成。

索引器的签名由其形参的数量和类型组成，不包括索引器类型或形参名。如果在同一类中声明一个以上的索引器，则它们必须具有不同的签名。索引器的值不归属于变量，因此，不能将索引器的值作为 ref 参数或 out 参数来传递。

C#并不将索引类型限制为整数。例如，对索引器使用字符串可能是有用的。通过搜索集合内的字符串并返回相应的值，可以实现此类索引器的功能。由于访问器可被重载，字符串和整数版本可以共存。

示例 04-14　利用索引器输出数组中的内容

下面的示例说明如何声明私有数组字段、temps 和索引器。使用索引器可直接访问实例 tempRecord[i]。代码如下：

代码 04-14　光盘位置：光盘\mingrisoft\sl\04\14

```
namespace IndexMachine
{
    class Program
    {
        class RecordItem//自定义一个类 RecordItem
        {
            //自定义一个浮点型数组 temps
            private float[] temps = new float[10] { 56.2F, 56.7F, 56.5F, 56.9F, 58.8F,
61.3F, 65.9F, 62.1F, 59.2F, 57.5F };
            public int Length
            {
                get { return temps.Length; }                    //只读属性表示数组的长度
            }
            public float this[int index]                        //自定义索引器
            {
                get
                {
                    return temps[index];                        //返回 temps 数组中当前索引下的元素
                }
                set
                {
                    temps[index] = value;                       //设置 temps 数组中当前索引下的元素
                }
            }
        }
        class MainClass
        {
            static void Main()
            {
                RecordItem Record = new RecordItem();           //实例化一个类 RecordItem 的实例
                Record[3] = 58.3F;                              //修改第 3 条记录
                Record[5] = 60.1F;                              //修改第 5 条记录
                for (int i = 0; i < 10; i++)                    //从 0 循环遍历至 9
                {
                    if (i < Record.Length)                      //当 i 的值小于数组的长度时
                    {
                        System.Console.WriteLine("元素#{0} = {1}", i, Record[i]);      //输出数组中的内容
                    }
                    else                                        //当 i 的值不小于数组的长度时
                    {
                        System.Console.WriteLine("{0}超出数组的索引", i);              //弹出信息提示
                    }
                }
                Console.ReadKey();                              //获取用户按下的下一个字符或功能键
            }
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.16 所示。

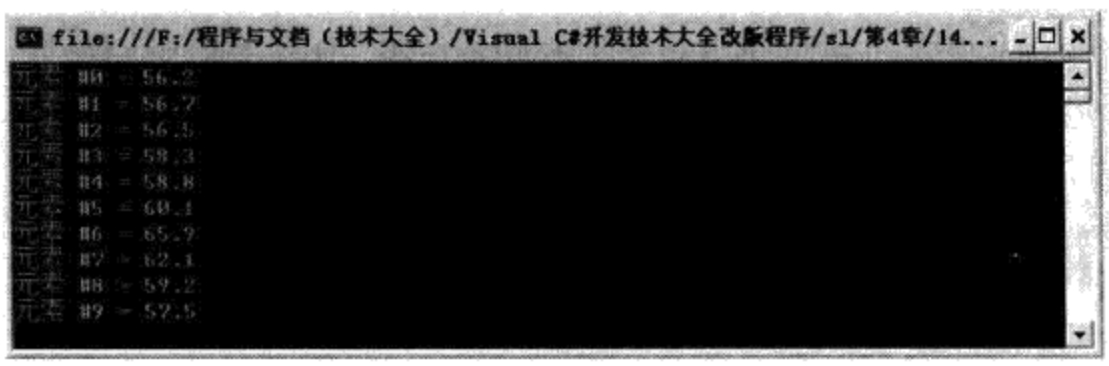


图 4.16　索引器的应用

4.9 接　　口

C#不支持多重继承，但是客观世界出现多重继承的情况又比较多。为了避免传统的多重继承给程序带来的复杂性等问题，C#提出了接口的概念。通过接口可以实现多重继承的功能。

4.9.1 接口的概念

在 C#编程语言中，接口是声明类具有的方法和成员的、自身不能被实例化的、只能被继承的特殊类。它只用于描述事物的特性而实现特性的任务却由其他类负责。一个类可以直接继承多个接口，但只能直接继承一个类，包括抽象类在内。例如下面的这段代码表示声明一个接口。

```
public delegate void MyDelegate();
public interface MyTest
{
        int Theme
        {
            get;
        }
        void Test();
        event MyDelegate OnDelegateEventHandler;
        int this[int index]
        {
            get;
            set;
        }
}
```

4.9.2 接口的声明

接口声明的语法格式如下。

```
修饰符 interface 接口名称：继承的接口列表
{
    接口内容;
}
```

其中，除 interface 和接口名称外，其他的都是可选项，下面是一个定义接口的例子，代码如下：

```
public interface iSomeInterface
{
    void UpdateDataBase();
}
```

一个接口可以从一个或多个基接口继承。例如：

```
interface iOthrerInterface :iSomeInterface,iCom
{
  int CompareTo(object obj);
}
```

4.9.3 接口的成员

接口可由方法、属性、事件、索引器或这 4 种成员类型的任何组合构成，接口不能包含字段。此外，接口成员一定是公共的。特别要注意的是在声明接口成员时，如果包含任何修饰符，都会出现错误。

例如，下面的代码声明了一个接口，该接口的成员涵盖了所有可能作为接口成员的种类，即方法、属性、事件和索引器。

```
public delegate void MessageArrived();
public interface Message
{
        int Content
        {
```

```
            get;
        }
        void Read();
        event MessageArrived OnPressEventHandler;
        int this[int index]
        {
            get;
            set;
        }
    }
```

4.9.4 接口的实现

实现接口的类或结构要与接口的定义严格一致。接口描述可属于任何类或结构的一组相关行为。类和结构可以像类继承基类或结构一样从接口继承，而且可以继承多个接口。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。类的属性和索引器可以为接口上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。

接口可以继承其他接口。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。

示例 04-15　利用接口输出指定内容

下面的示例说明如何使用接口实现字符串的输出。代码如下：

代码 04-15　光盘位置：光盘\mingrisoft\sl\04\15

```
namespace InterfaceApplication
{
    class Program
    {
        interface IMyFace                                   //自定义接口
        {
            string Str                                      //自定义接口的属性
            {
                get;                                        //获取属性值的声明
                set;                                        //设置属性值的声明
            }
            void PrintMethod();                             //输出内容方法
        }
        class MyClass:IMyFace
        {
            string str = "Visual C#  开发  技术大全";       //自定义字符串
            public string Str                               //自定义属性
            {
                get                                         //获取属性的值
                {
                    return str;                             //返回属性的值
                }
                set                                         //设置属性的值
                {
                    str = value;                            //向属性中写入值
                }
            }
            public void PrintMethod()                       //输出内容方法
            {
```

```
                Console.WriteLine(this.str);                    //输出指定的内容
            }
        }
        static void Main(string[] args)                         //主函数
        {
            IMyFace MyInterface = new MyClass();                //实例化一个接口
            MyInterface.PrintMethod();                          //调用输出指定内容的方法
            Console.ReadKey();                                  //获取用户按下的下一个字符或功能键
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.17 所示。

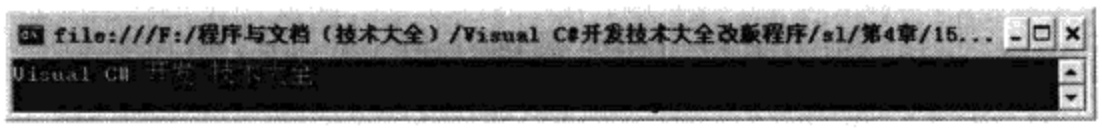


图 4.17　接口的应用

4.10　封装、继承、多态

数据封装、继承和多态构成了面向对象程序设计思想的基础。通过这些基础可以在最大范围内实现代码的重用，有效地降低软件的复杂性；通过多态可以逼真地描述客观世界，使得应用程序满足用户的要求。

4.10.1　封装

在 C#中可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。除此之外，还可以控制数据的存取方式。

面向对象程序设计中一般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。在设计类时，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。如此就可以达到封装数据的目的，方便以后维护、升级，也可以在操作数据时多一层判断，提高安全性。

封装还可以解决数据存取权限问题，使用封装可以将数据隐藏起来，形成一个封闭的空间，用户可以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，则有些用户可以访问，有些用户却不能访问。如果不对这些数据的访问加以限制，那么后果是很严重的。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。

示例 04-16　利用封装实现字符串的输出

下面的示例说明通过对方法的访问权限的设置实现字符串的输出。代码如下：

代码 04-16　光盘位置：光盘\mingrisoft\sl\04\16

```
namespace PackagedApplication
{
    class Program
    {
        class Demo
        {
            public void PrintData()                                 //自定义输出数据方法
            {
                Console.WriteLine("《Visual C# 开发 技术大全》");   //输出相应的数据
            }
        }
```

```
        static void Main(string[] args)
        {
            Demo MyDemo = new Demo();                    //实例化一个对象
            MyDemo.PrintData();                          //调用输出数据的方法
            Console.ReadKey();                           //获取用户按下的下一个字符或功能键
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.18 所示。

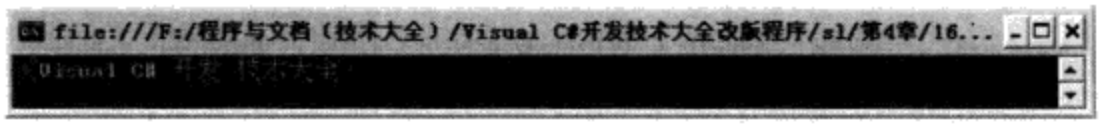


图 4.18　封装的应用

4.10.2　继承

继承（Inheritance）是 OOP 最重要的特性之一。任何类都可以从另外一个类继承，即这个类拥有它所继承类的所有成员。在 OOP 中，被继承的类称为父类或基类。C#提供了类的继承机制，但 C#只支持单继承，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。

从基类继承出一个新子类的语法格式如下：

```
class 子类名：基类名
{
        //派生代码
}
```

利用继承机制，用户可以通过增加、修改或替换类中方法对这个类进行扩充，以适应不同的应用要求。利用继承，程序开发人员可以在已有类的基础上构造新类。继承使得类支持分类的概念。在日常生活中很多东西具有条理，那是因为它们有着很好的层次分类。如果不用层次分类，则要对每个对象定义其所有的性质。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。

在 C#中，接口允许多继承，可以通过继承多个接口来实现类似于 C++中的多重继承。

在继承一个基类时，成员的可访问性是一个重要的问题。子类不能访问基类的私有成员，但是可以访问其公共成员。子类和外部代码都可以访问公共成员。这就是说，只使用这两个可访问性，就可以让一个成员被基类和子类访问，同时也可以被外部的代码访问。

为了解决基类成员只能被派生类访问的问题，C#还提供了第 3 种可访问性：protected。只有派生类才能访问 protected 成员，基类和外部代码都不能访问 protected 成员。

除了成员的保护级别外，用户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，外部代码就访问基类中成员的执行代码。

虚拟成员不能是私有成员，因为私有成员不能被子类重写，也不能访问它。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。

示例 04-17　利用继承计算矩形的面积

下面的示例说明如何使用继承，在本示例中声明了一个基类 Shape 以及派生类 Rectangle。代码如下：

代码 04-17　光盘位置：光盘\mingrisoft\sl\04\17

```
namespace InheritApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(4, 6);                    //调用 Rectangle 类的构造方法
            rect.Name = "矩形";                                       //调用基类的 Name 属性
            int area = rect.ComputeArea();                           //调用 Rectangle 类的 ComputeArea 方法
            rect.Draw();                                             //调用基类的画图方法
            Console.WriteLine(rect.Name);                            //输出继承基类的属性内容
            Console.WriteLine("该矩形的面积为：{0}", area);           //输出矩形的面积
            Console.ReadKey();                                       //获取用户按下的下一个字符或功能键
        }
    }
    class Shape                                                      //自定义的基类
    {
        protected string name;                                       //自定义字段
        public string Name                                           //自定义属性
        {
            get
            {
                return name;                                         //返回自定义属性的值
            }
            set
            {
                name = value;                                        //设置自定义属性的值
            }
        }
        public Shape()                                               //类 Shape 的构造方法
        {
        }
        public Shape(string shapeName)                               //重载类 Shape 的构造方法
        {
            name = shapeName;                                        //重新为字段赋值
        }
        public void Draw()                                           //画图方法
        {
            Console.Write("画");                                     //输出对应的内容
        }
    }
    class Rectangle : Shape                                          //继承基类 Shape
    {
        protected int width;                                         //定义矩形的宽
        protected int height;                                        //定义矩形的高
        public int Width                                             //定义矩形的宽度属性
        {
            get
            {
                return width;                                        //返回矩形的宽度
            }
        }
        public int Height                                            //定义矩形的高度属性
        {
            get
            {
                return height;                                       //返回矩形的高度
            }
        }
        public Rectangle()                                           //类 Rectangle 的构造方法
        {
        }
        public Rectangle(int cx, int cy)                             //类 Rectangle 构造方法的重载方法
        {
```

```
            width = cx;                                    //为宽度字段赋值
            height = cy;                                   //为高度字段赋值
        }
        public int ComputeArea()                           //自定义计算面积方法
        {
            return width * height;                         //返回宽度与高度的积
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.19 所示。

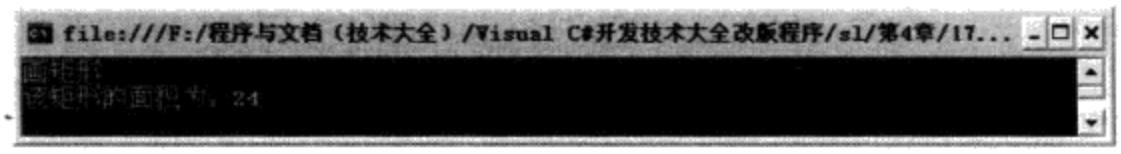


图 4.19　继承的应用

4.10.3　多态

多态是指在具有继承关系的类中，不同对象的相同的函数成员可以具有不同的实现，因而会产生不同的执行结果。

多态有两种：一种是编译时的多态，也就是指方法的重载；另一种是运行时的多态，通过继承和虚成员来实现的。运行时的多态性是指系统在编译时不确定选用哪个重载方法，而是直到程序运行时，才根据实际情况决定采用哪个方法。

多态使得子类的实例可以直接赋予基类的变量，不需要进行强制类型转换，直接就可以通过这个变量调用基类的方法。

在派生于同一个类的不同对象上执行任务时，多态是一种极为有效的技巧，使用的代码最少。可以把一组对象放到一个数组中，然后调用它们的方法，这些对象不必是相同类型的对象，在这种情况下多态的作用就体现出来了。当然如果它们都继承自某个类，可以把这些派生类都放到一个数组中。如果这些对象都有同名方法，可以调用每个对象的同名方法。

示例 04-18　利用多态输出各类中的内容

下面的示例说明如何使用多态，在本示例中声明了一个基类 Book 以及派生类 SanGuoYanYi、HongLouMeng、XiYouJi 以及 ShuiHuZhuan。代码如下：

代码 04-18　光盘位置：光盘\mingrisoft\sl\04\18

```
namespace PolymorphismApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Book [] MyBook = new Book[4];                  //实例化一个 Book 类的数组
            MyBook[0] = new SanGuoYanYi();                 //实例化数组中的第 1 个元素
            MyBook[1] = new HongLouMeng();                 //实例化数组中的第 2 个元素
            MyBook[2] = new ShuiHuZhuan();                 //实例化数组中的第 3 个元素
            MyBook[3] = new XiYouJi();                     //实例化数组中的第 4 个元素
            foreach (Book Obj in MyBook)                   //循环遍历数组中的每一个对象
            {
                Obj.Interest();                            //调用对象的 Interest 方法
            }
        }
    }
    class Book                                             //自定义一个 Book 类
    {
        public virtual void Interest()                     //自定义一个表示兴趣的方法
```

```
        {
            Console.WriteLine("是我最喜爱的图书之一");          //输出提示性信息
            Console.ReadLine();                                 //从标准输入流读取下一行字符
        }
    }
    class SanGuoYanYi : Book                                    //类 SanGuoYanYi 继承类 Book
    {
        public override void Interest()                         //重写基类的表示兴趣的方法
        {
            Console.Write("《三国演义》");                       //输出相应的内容
            base.Interest();                                    //调用基类的表示兴趣的方法
        }
    }
    class ShuiHuZhuan : Book                                    //类 ShuiHuZhuan 继承类 Book
    {
        public override void Interest()                         //重写基类的表示兴趣的方法
        {
            Console.Write("《水浒传》");                         //输出相应的内容
            base.Interest();                                    //调用基类的表示兴趣的方法
        }
    }
    class HongLouMeng : Book                                    //类 HongLouMeng 继承类 Book
    {
        public override void Interest()                         //重写基类的表示兴趣的方法
        {
            Console.Write("《红楼梦》");                         //输出相应的内容
            base.Interest();                                    //调用基类的表示兴趣的方法
        }
    }
    class XiYouJi : Book                                        //类 XiYouJi 继承类 Book
    {
        public override void Interest()                         //重写基类的表示兴趣的方法
        {
            Console.Write("《西游记》");                         //输出相应的内容
            base.Interest();                                    //调用基类的表示兴趣的方法
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.20 所示。

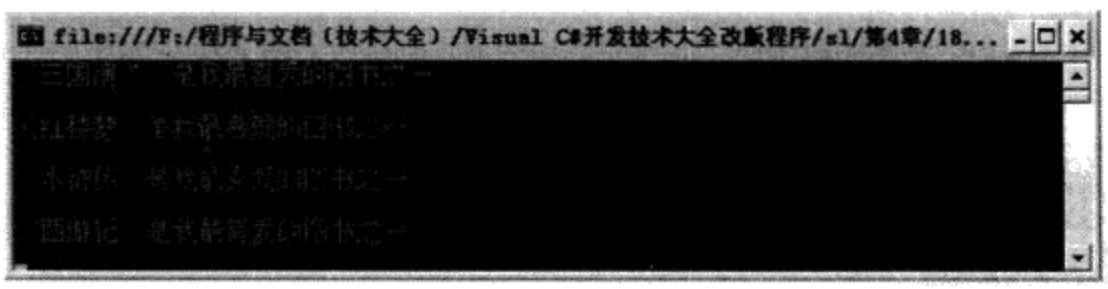


图 4.20　多态的应用

4.11　泛　　型

4.11.1　泛型的概念

泛型是 C#中的一种面向对象编程技术，它的作用与 C++中的 STL(标准模板库)相似。在 C#中，方法的参数可以通过变量来传递，但这种传递仅限于变量的值，而在某些特殊的情况下，编程人员需要将变量的类型作为参数进行传递，泛型的作用就体现在此处。通过使用泛型，可以达到以下两个目的：类型安全和提高性能。

每次创建需要包含在集合中的新类时，就必须执行下列任务之一。

- 使用某个集合类，该类已经定义为可以包含新类型的项。
- 创建一个新的集合类，它可以包含新类型的项，实现所需要的方法。

一般情况下，创建一个新类型都具有新的功能，而依据现实情况并不需要新的集合类，并且创建集合类也会耗费大量时间。

通过对泛型的学习可以简化这个过程。泛型类是以实例化过程中提供的类型或类为基础建立的，可以毫不费力地对对象进行强类型转化。对于集合，创建“T 类型对象的集合”只需把它用一行代码写出来即可。不建议使用下面的代码：

```
CollectionItem Col=new CollectionItem();
Col.Add(new ItemClass);
```

建议使用下面的代码：

```
CollectionItem<ItemClass> Col=new CollectionItem<ItemClass>();
Col.Add(new ItemClass);
```

上面代码中尖括号的作用就是把变量的类型传递给泛型类型。

4.11.2 泛型的使用

下面使用泛型类定义一个类。用一个通用的数据类型 T 做为一个占位符，在实例化时用一个实际的类型来代替它，从而将变量的数据类型作为参数进行传递。代码如下：

```
public class MyStack<T>
{
        private T[ ] Item;
        public T pop()
        {
            出栈方法体
        }
        public void push(object item)
        {
            入栈方法体
        }
        public MyStack(int i)
        {
            this.Item=new T [i];
        }
}
```

下面的代码说明这个类的调用方法：

```
MyStack<int> IntStack=new MyStack<int>(100);          //实例化一个只能保存 int 类型的类
IntStack.push(10);                                    //向 IntStack 中存入 10
//该行代码将编译出错，因为对象 IntStack 只能接收 int 类型的数据
IntStack.push("ABC");
int x=IntStack.pop();                                 //定义一个标记
Mystack<string> StrStack=new MyStack<string>(100);    //实例化一个只能保存 string 类型的类
//该行代码将编译出错，因为对象 StrStack 只能接受 String 类型的数据
StrStack.push(10);                                    //使指定的元素 10 出栈
StrStack.push("ABCDE");                               //向 StrStack 中存入 ABCDE
```

4.11.3 泛型的定义

下面将以定义泛型类为代表讲解怎样定义泛型。要创建泛型类，只需在类定义中包括尖括号语法：

```
class GenericClass<T>
{
        类体
}
```

其中 T 可以是任意的标识符，只要遵循 C#的命名规则就可以，例如，不以数字开头，以字符或者下划线开头等。

泛型类可以在其定义中包含任意多个类型，它们之间用逗号分隔开，例如：

```
class GenericClass<T1,T2,T3>
{
        类体
}
```

示例 04-19 利用泛型输出指定的内容

下面的示例说明如何使用泛型。代码如下：

代码 04-19 光盘位置：光盘\mingrisoft\sl\04\19

```
namespace CommanTypeApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, string> dic = new Dictionary<string, string>();      //实例化一个泛型类
            dic.Add("First", "Hello");                                              //向该类中添加内容
            dic.Add("Second", "World");
            dic.Add("Third", "!");
            Dictionary<string, string>.KeyCollection dicKeys = dic.Keys;            //保存泛型集合类的值
            Dictionary<string, string>.ValueCollection dicValues = dic.Values;
            foreach (string s in dicKeys)                                           //循环遍历集合中的每一个数据键
            {
                Console.Write("{0}  ", s);                                          //输出数据键的名称
            }
            foreach (string s in dicValues)                                         //循环遍历集合中的每一个数据记录
            {
                Console.Write("{0}  ", s);                                          //输出数据记录
            }
            Console.ReadKey();                                                      //获取用户按下的下一个字符或功能键
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，效果如图 4.21 所示。

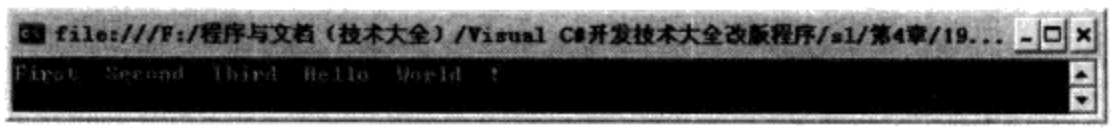


图 4.21 泛型的应用

第 5 章

算法

- 算法概述
- 若干基本数学算法
- 排序算法
- 查找算法
- 字符处理应用
- 穷举法
- 迭代法

5.1 算法概述

5.1.1 什么是算法

算法是解决问题的方法和步骤。对于计算机程序设计语言的初学者，掌握算法是必要的，因为它是学习程序设计的基础，也可以说是程序设计的入门知识。掌握算法可以帮助读者快速理清程序设计的思路，找出多种解决方法，从而选择最合适的解决方案。

5.1.2 算法的特点

一个算法应该具有以下的特点。

（1）有穷性：一个算法必须在执行有穷多个计算机步骤后终止。

（2）确定性：一个算法给出的每个计算步骤，必须是有精确定义、无二义性的。

（3）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。

（4）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。

（5）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。

5.1.3 算法的描述方法

为了让算法清晰易懂，需要选择一种好的描述方法。算法的描述方法有很多，有自然语言、传统流程图、N-S 结构化流程图等。自然语言就是用人们日常使用的语言描述解决问题的方法和步骤。这种描述方法通俗易懂，但比较繁琐，并且对程序流向等描述不明了、不直观。针对这些缺点产生了传统流程图和 N-S 结构化流程图。

下面分别介绍这两种流程图。

1．传统流程图

传统流程图，使用不同的几何图形来表示不同性质的操作，使用流程线来表示算法的执行方向，具有直观形象、逻辑清楚、易于理解等特点，但它占用篇幅较大，流程随意转向，较大的流程图不易读懂。

传统流程图的基本流程图符号及说明如表 5.1 所示。

表 5.1　　流程图符号及说明

流程图符号	名　称	说　明
	起止框	表示算法的开始和结束
	处理框	表示完成某种操作，如初始化或运算赋值等
	判断框	表示根据一个条件成立与否，决定执行两种不同操作的其中一个
	输入输出框	表示数据的输入输出操作
	流程线	用箭头表示程序执行的流向
	连接点	用于流程分支的连接

2．N-S 结构化流程图

N-S 结构化流程图是 1973 年美国学者 I · Nassi 和 B · Shneiderman 首次提出的一种描述算法的图形方法。N-S 结构化流程图将传统流程图中的流程线去掉，将全部算法写在一个矩形框里，这个框包含其他从属于它的从属框。当程序算法比较繁锁时，一般采用 N-S 结构化流程

图。另外，N-S 结构化流程图更有利于程序设计的结构化。

以上介绍的两种流程图各有特点，但对于初学者和编写较小的程序时，笔者建议使用传统流程图来描述算法。

5.2 若干基本数学算法

5.2.1 查找最大值和最小值

计算一组数的最大值和最小值是数值应用中经常用到的，也是程序设计中解决某些问题的基础，例如排序问题就涉及到比较数值大小的算法。下面给出查找最大值最小值的算法思路。

（1）假设一组数中的第一个数为最大数（或最小数）*M*。

（2）将给定的这组数中的每个数依次与 *M* 进行比较，若某个数大于（或小于）*M*，则将这个数假设为新的最大数（或最小数）*M*。

（3）全部比较完毕后，*M* 即为所求。

查找最大值的算法流程图如图 5.1 所示。

示例 05-01 求一组数中最大值和最小值

本示例将在给定的一组数据中找出最大值和最小值。运行结果如图 5.2 所示。

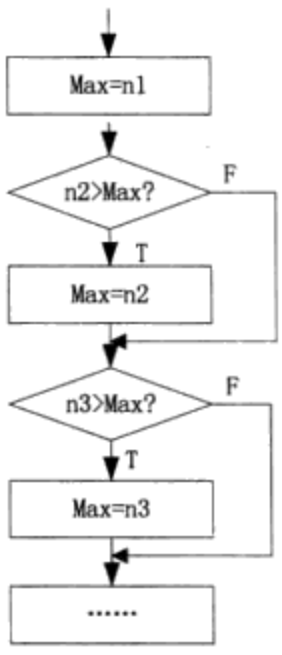


图 5.1 查找最大值算法流程图

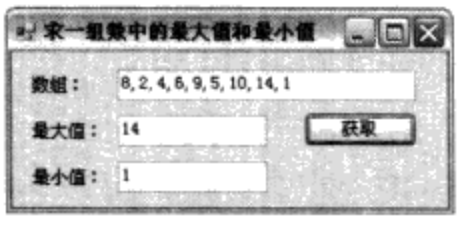


图 5.2 求一组数中的最大值和最小值

程序代码如下：

代码 05-01 光盘位置：光盘\mingrisoft\sl\05\01

```
private void button1_Click(object sender, EventArgs e)
{
    string[] Array_str = textBox1.Text.Split(',');                    //获取要进行遍历的一组数
    int var_Max = Convert.ToInt32(Array_str[0]);                      //记录数组中的第一个数
    int var_Min = Convert.ToInt32(Array_str[0]);                      //记录数组中的第一个数
    for (int i = 1; i < Array_str.Length; i++)                        //对数组进行遍历
    {
        //获取最大的数值
        var_Max = var_Max >= Convert.ToInt32(Array_str[i]) ? var_Max : Convert.ToInt32(Array_str[i]);
        //获取最小的数值
        var_Min = var_Min <= Convert.ToInt32(Array_str[i]) ? var_Min : Convert.ToInt32(Array_str[i]);
    }
    textBox2.Text = var_Max.ToString();                               //显示最大值
    textBox3.Text = var_Min.ToString();                               //显示最小值
}
```

5.2.2 计算最大公约数和最小公倍数

计算两个自然数的最大公约数和最小公倍数是初等的数学问题，在程序中能够通过算法求出两个数的最大公约数和最小公倍数。下面介绍实现计算两个数的最大公约数和最小公倍数的算法。

（1）已知两个自然数 m 和 n，使得 $m>n$。

（2）m 除以 n 得余数 r。

（3）判断，若 $r=0$，则 n 是最大公约数，算法结束；否则执行（4）。

（4）将 n 值赋给 m，r 值赋给 n，重复执行（2）。

最大公约数算法流程图如图 5.3 所示。

最小公倍数为两个自然数的乘积除以这两个数的最大公约数，即：最小公倍数 = 两个数之积/最大公约数。

示例 05-02 计算最大公约数和最小公倍数

本实例实现，计算在文本框中输入的两个自然数的最大公约数和最小公倍数，并显示在窗体上，程序运行效果如图 5.4 所示。

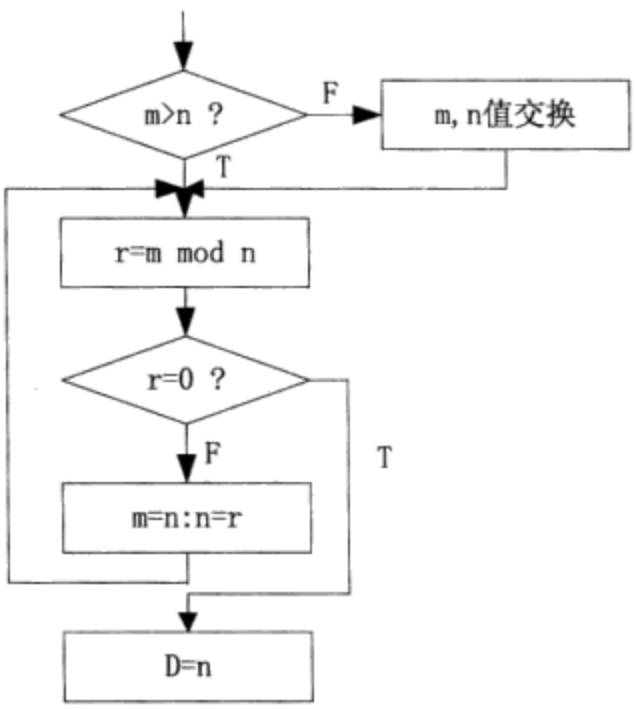


图 5.3 最大公约数算法流程图

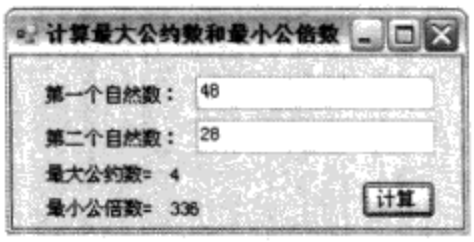


图 5.4 计算两个自然数的最大公约数和最小公倍数

程序代码如下：

代码 05-02 光盘位置：光盘\mingrisoft\sl\05\02

```
private void button1_Click(object sender, EventArgs e)
{
    int m = Convert.ToInt32(textBox1.Text);                    //获取自然数
    int n = Convert.ToInt32(textBox2.Text);                    //获取自然数
    int mn = m * n;                                            //计算两个自然的乘积
    int t = 0;                                                 //临时变量
    if (m < n)                                                 //如果除数小于被除数
    {                                                          //掉换两个数的位置
        t = m;
        m = n;
        n = t;
    }
    int div = 0;
    Math.DivRem(m, n, out div);                                //求 m 除以 n 的余数
    while (div != 0)                                           //如果余数不为 0
    {                                                          //用上以次的被除数除以余数
```

```
            m = n;
            n=div;
            Math.DivRem(m, n, out div);
        }
        label5.Text = n.ToString();                          //获取最大公约数
        label6.Text = Convert.ToString(mn / n);              //获取最小公倍数
    }
```

5.2.3 累加和累乘

所谓累加就是将一些数值进行累计相加的运算，而累乘则是将一些数值进行累计相乘的运算。

图 5.5 累加与累乘的算法

示例 05-03 累加与累乘的算法

本例通过使用循环结构语句，计算 0～100 累加与累乘的结果。程序运行效果如图 5.5 所示。

程序代码如下：

代码 05-03 光盘位置：光盘\mingrisoft\sl\05\03

计算累加的代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    int a = 0;                                               //记录累加值
    for (int i = 1; i <= 100; i++)                           //遍历 1～100 中的数
    {
        a += i;                                              //累加
    }
    textBox1.Text = a.ToString();                            //显示累加值
}
```

计算累乘的代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    Double a = 1;                                            //记录累乘值
    for (int i = 1; i <= 100; i++)                           //遍历 1～100 中的数
    {
        a *= i;                                              //累乘
    }
    textBox2.Text = a.ToString();                            //显示累乘值
}
```

注意：由于累乘结果较大，已超出 Int64 和 long 的范围，因此将变量 a 定义为 Double 数据类型。

5.2.4 求 100 以内素数

数学中的素数是指只能被 1 和它本身整除的整数。下面介绍查找素数的方法。

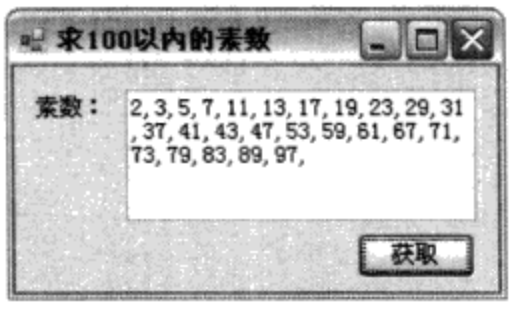


图 5.6 100 以内的素数

根据素数的定义来判别一个数是否为素数。其基本思路为：对整数 n，依次使用 2、3、…、n-2、n-1 去除，如果没有一个能整除，则判定 n 为素数。也就是说，只要有一个能整除，则 n 不是素数。

示例 05-04 查找 100 以内的所有素数

本例通过使用循环结构语句实现素数算法，程序运行以后，单击【获取】按钮将 100 以内所有素数显示在窗体上，如图 5.6 所示。

程序代码如下：

代码 05-04　光盘位置：光盘\mingrisoft\sl\05\04

```
private void button1_Click(object sender, EventArgs e)
{
    bool var_b = false;                                  //定义一个标识
    for (int i = 2; i <= 100; i++)                       //遍历 100 以内的数
    {
        var_b = false;                                   //设置标识
        for (int j = i - 1; j >= 2; j--)                 //遍历 2 到当前数之间的数
        {
            if (i / j == i / (float)j)                   //如果当前数与小于它的数相除没有余数
            {
                var_b = true;                            //设置标识，不为素数
                break;                                   //退出当前循环
            }
        }
        if (!var_b)                                      //如果是素数
            textBox1.Text += i.ToString() + ",";         //显示素数
    }
}
```

5.2.5　进制转换

这里介绍的是将一个十进制整数转换成二或十六进制的数值，进制转换的算法思想是：将十进制正整数 n 不断除进制数 r 取余数，直到商为零，将得到的值反序连接，即最后得到的余数在最高位。

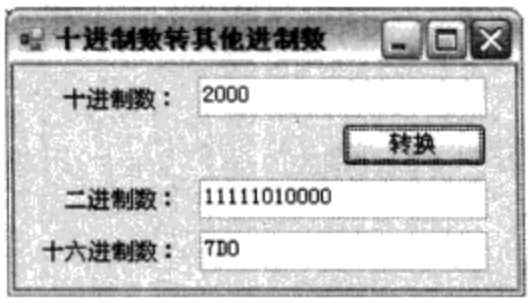


图 5.7　进制转换程序运行界面

示例 05-05　十进制数转其他进制数

本例实现将十进制转换成二或十六进制的字符串。程序运行界面如图 5.7 所示。

程序代码如下：

代码 05-05　光盘位置：光盘\mingrisoft\sl\05\05

自定义方法 Conversion 将十进制转换成二进制或十六进制。参数 d 表示十进制的值，参数 g 表示要转换的进制。代码如下：

```
public string Conversion(int d, int g)
{
    string b = "";
    if (d < g)                                           //如果当前值小于 2
    {
        if (g == 2)
            b = d.ToString();                            //记录该值
        if (g==16)
            b = Letter(d);
    }
    else
    {
        int c;
        int s = 0;                                       //记当位数
        int n = d;                                       //记录当前值
        while (n >= g)                                   //获取转换后的位数
        {
            s++;                                         //位数累加
            n = n / g;
        }
        string[] m = new string[s];                      //根据位数定义一个数组
        int i = 0;
        do
        {                                                //计算转换后的值
            c = d / g;
            if (g == 2)
```

```
                m[i++] = Convert.ToString(d % g);
            if (g==16)
                m[i++] = Letter(d % 16);          //判断是否大于 10，如果大于 10，则转换为 A~F 的格式
            d = c;
        } while (c >= g);
        b = d.ToString();                                    //获取转换后第一位的值
        for (int j = m.Length - 1; j >= 0; j--)              //获取转换后的二进制数
        {
            b += m[j];
        }
    }
    return b;
}
```

自定义方法 Letter 用于判断转换后的数字是否在 10～15 之间，如果是则将其转换成 A～F。代码如下：

```
public string Letter(int d)
{
    string x = "";                                           //设置返回值
    switch (d)                                               //根据值进行相应的转换
    {
        case 10: x = "A"; break;
        case 11: x = "B"; break;
        case 12: x = "C"; break;
        case 13: x = "D"; break;
        case 14: x = "E"; break;
        case 15: x = "F"; break;
        default: x = d.ToString(); break;
    }
    return x;
}
```

5.2.6 杨辉三角

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。

示例 05-06 杨辉三角算法

本例通过使用循环语句实现在窗体上显示一个杨辉三角，显示的杨辉三角行数由用户设定。程序运行效果如图 5.8 所示。

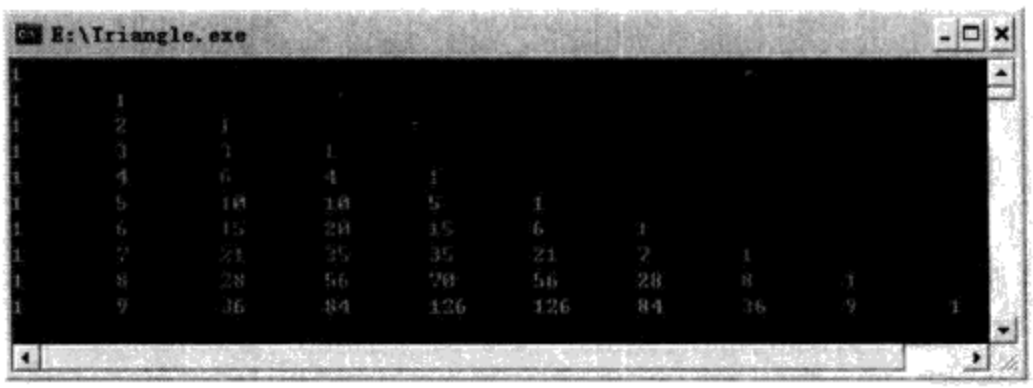


图 5.8 杨辉三角算法运行演示

程序代码如下：

代码 05-06 光盘位置：光盘\mingrisoft\sl\05\06

```
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();
}
```

5.3 排序算法

排序是将一组数按照递增或是递减的顺序排列。常用的算法有选择法、冒泡法、插入法和合并排序等。

5.3.1 选择排序

选择排序法指每次选择所要排序的数组中最大值（由大到小排序，由小到大排序则选择最小值）的数组元素，将这个数组元素的值与前面的数组元素的值互换。

示例 05-07 选择排序

本实例使用选择排序法实现对数列的排序，程序运行结果如图 5.9 所示。

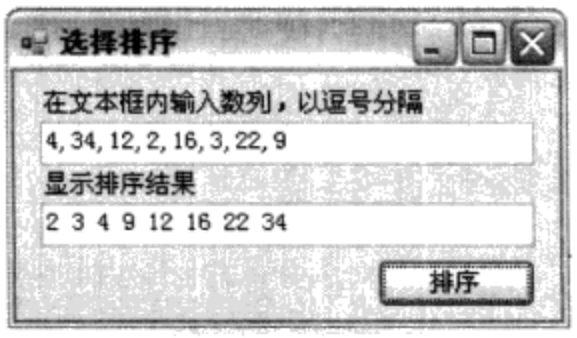


图 5.9 选择排序

程序代码如下：

代码 05-07 光盘位置：光盘\mingrisoft\sl\05\08

在【排序】按钮的单击事件中，以逗号为分隔符将文本框中的数列存入指定的数组中，然后对该数组进行排序，并显示排序后的结果。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string[] Array_Str = textBox1.Text.Split(',');          //以逗号为分隔符将文本框中的数列存入指定的数组中
    Sort(Array_Str);                                        //调用自定义方法
    for (int i = 0; i < Array_Str.Length; i++)              //遍历排序后的数组
        textBox2.Text += Array_Str[i] + " ";                //显示排序后的结果
}
```

自定义方法 Sort 以选择排序对数组中的数据进行排序。代码如下：

```
public void Sort(string[] list)
{
    int min = 0;
    for (int i = 0; i < list.Length - 1; i++)               //遍历数组中的数据，不包含最后一个
```

```
        {
            min = i;                                    //读取当前数据
            for (int j = i + 1; j < list.Length; j++)   //遍历当前数据以后的数据
            {
                if (Convert.ToInt32(list[j]) < Convert.ToInt32(list[min]))      //如果当前是最小值
                    min = j;                            //将最小值放在相应位置
            }
            int t = Convert.ToInt32(list[min]);
            list[min] = list[i];                        //交换数据
            list[i] = t.ToString();
        }
    }
```

5.3.2 冒泡排序

冒泡排序的过程很简单。首先将一个记录的关键字和第二个关键字进行比较，若为逆序，则将两个记录交换，然后比较第 2 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，排序完成。

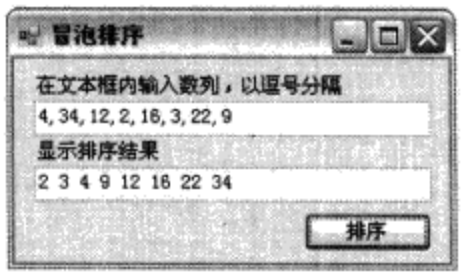


图 5.10 冒泡排序

示例 05-08 冒泡排序

本实例使用冒泡排序法实现对一个数列进行排序，并将排序后的结果显示出来。程序运行结果如图 5.10 所示。

程序代码如下：

代码 05-08 光盘位置：光盘\mingrisoft\sl\05\09

```
public void Sort(string[] list)
{
    int i, j;                                           //定义变量
    string temp = "";
    bool done = false;
    j = 1;
    while ((j < list.Length) && (!done))                //判断长度
    {
        done = true;
        for (i = 0; i < list.Length - j; i++)           //遍历数组中的数值
        {
            if (Convert.ToInt32(list[i]) > Convert.ToInt32(list[i + 1]))    //如果前一个值大于后一个值
            {
                done = false;
                temp = list[i];
                list[i] = list[i + 1];                  //交换数据
                list[i + 1] = temp;
            }
        }
        j++;
    }
}
```

5.3.3 插入排序

插入排序是将一个记录插入到有序数列中，使得到的新的数列仍然有序。插入排序算法思想是：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。

示例 05-9 插入排序

本实例使用插入排序法实现对一个数列进行排序，并将排序后结果显示出来。程序运行结果如图 5.11 所示。

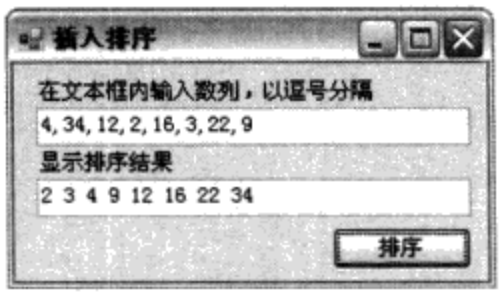


图 5.11　插入排序

程序代码如下：

代码 05-9　光盘位置：光盘\mingrisoft\sl\05\10

```
public void Sort(string[] list)
{
    for (int i = 1; i < list.Length; i++)                    //遍历当前数组，不包含第一个和最后一个
    {
        int t = Convert.ToInt32(list[i]);                    //获取当前值
        int j = i;                                           //记录当前值的标记
        while ((j > 0) && (Convert.ToInt32(list[j - 1]) > t))  //插入法
        {
            list[j] = list[j - 1];                           //交换顺序
            --j;
        }
        list[j] = t.ToString();
    }
}
```

5.3.4　希尔排序

希尔排序又称“缩小增量排序”，也是一种插入排序类的方法，但在时间效率上较前面所述的几种排序方法有较大的改进。在待排序记录按关键字“基本有序”时，直接插入排序效率就可以大大提高。而且记录较少时效率也较高。希尔排序正是从这两点分析出发对直接插入排序进行改进得到的一种方法。

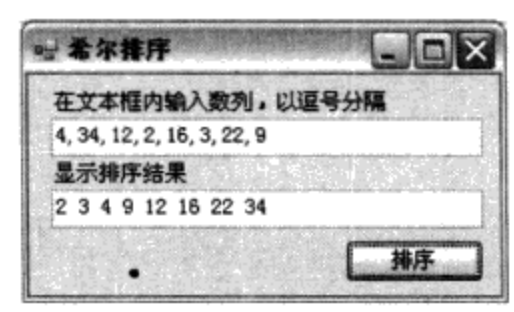


图 5.12　希尔排序

示例 05-10　希尔排序

本实例在程序运行时给定一个无序的数列，使用希尔排序法将数列进行排序，将排序后的数列显示在窗体上。程序运行效果如图 5.12 所示。

代码 05-10　光盘位置：光盘\mingrisoft\sl\05\11

```
public void Sort(string[] list)
{
    int inc;
    for (inc = 1; inc <= list.Length / 9; inc = 3 * inc + 1) ;      //遍历当前数组
    for (; inc > 0; inc /= 3)                                       //遍历当前值
    {
        for (int i = inc + 1; i <= list.Length; i += inc)
        {
            int t = Convert.ToInt32(list[i - 1]);                   //获取值
            int j = i;
            while ((j > inc) && (Convert.ToInt32(list[j - inc - 1]) > t))   //希尔排序法
            {
                list[j - 1] = list[j - inc - 1];                    //交换数据
                j -= inc;
            }
            list[j - 1] = t.ToString();
        }
    }
}
```

5.4 查 找 算 法

查找是在线性表中（在此为数组中）根据给定的某个值确定一个值等于给定值的记录或数据元素。查找一般使用顺序查找和折半查找法。

5.4.1 顺序查找

顺序查找是根据查找的值与数组中的元素逐一比较，若找到满足条件的记录，则查找成功，否则，查找失败。

示例 05-11 顺序查找

本示例用顺序查找法在数组中查找指定的数字。运行结果如图 5.13 所示。

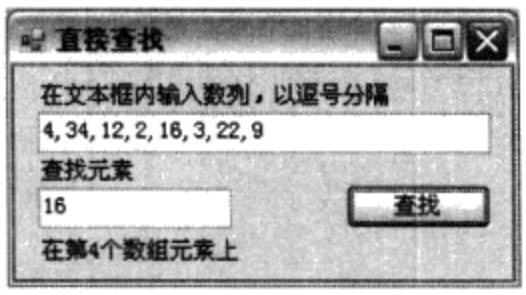


图 5.13 顺序查找运行效果

程序代码如下：

代码 05-11 光盘位置：光盘\mingrisoft\sl\05\12

在【查找】按钮的单击事件中，以逗号为分隔符将文本框中的数列存入指定的数组中，然后根据要查找的数字进行顺序查找，并显示查找后的结果。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string[] Array_Str = textBox1.Text.Split(',');                  //以逗号为分隔符将文本框中的数列存入指定的数组中
    int t=Locate(Array_Str,Convert.ToInt32(textBox2.Text));         //调用自定义方法
    if (t >= 0)                                                     //如果标志位大于等于 0 表示查找成功
        label3.Text = "在第" + t.ToString() + "个数组元素上";
    else                                                            //查找失败
        label3.Text = "没有查找到";
}
```

自定义方法 Locate 用于对数组进行顺序查找，并将找到数字的标志位返回。代码如下：

```
public int Locate(string[] list, int n)
{
    for (int i = 0; i < list.Length; i++)                           //遍历数组
        if (Convert.ToInt32(list[i]) == n)                          //查找数组中的指定元素
            return i;                                               //返回查找到的元素
    return -1;
}
```

5.4.2 二分法查找

二分查找又称折半查找，它是一种效率较高的查找方法。二分查找的基本思路：首先将结点按关键字排序，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。

示例 05-12 二分法查找

本示例是用二分查找方法在一个数组中查找已知数字的近似值。程序运行结果如图 5.14 所示。

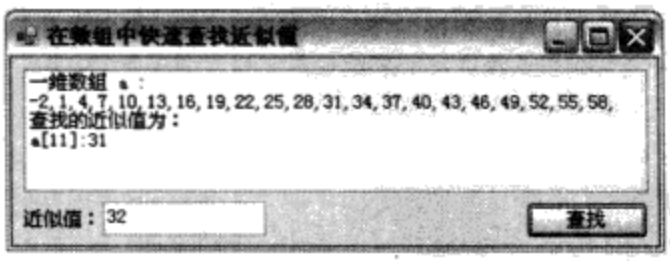


图 5.14 在数组中快速查找近似值

程序代码如下：

代码 05-12 光盘位置：光盘\mingrisoft\sl\05\13

```
public string Locate(int[] XX, int N, int X)//N 数值的个数，M 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))                    //判断左右那个更接近
        tem_n = jl + 1;
    return "a[" + tem_n.ToString() + "]:" + XX[tem_n].ToString();   //返回近似值
}
```

5.5 字符处理应用

5.5.1 简单的加密解密算法

简单的加密的思想是：将要加密的每个字母的 ASCII 码加上或减去一个序数，例如加上序数 5，这时“A”→“F”，“a”→“f,”“B”→“G”，“Y”→“D”，“Z”→“E”。

示例 05-13 加密和解密

本实例使用简单的加密和解密算法实现对字符串的加密和解密。运行程序后在字符串文本框中输入要加密的字符串，单击加密按钮，即可在文本框内显示加密后的字符串。程序运行效果如图 5.15 所示。

图 5.15 加密和解密

实现字符串加密的算法代码如下：

代码 05-13 光盘位置：光盘\mingrisoft\sl\05\14

自定义方法 Encrypt 用于对指定的字符串进行加密。代码如下：

```
private string Encrypt(string s)
{
    Encoding ascii = Encoding.ASCII;                            //实例化 Encoding 类
    string EncryptString;                                       //定义字符串
    EncryptString = "";
    for (int i = 0; i < s.Length; i++)                          //遍历要加密的字符串
    {
        int j;
        byte[] b = new byte[1];                                 //定义一个 byte 数组
        j = Convert.ToInt32(ascii.GetBytes(s[i].ToString())[0]);   //获取当字符的 ASCII 码
        j = j + 5;                                              //加密
        b[0] = Convert.ToByte(j);                               //转换为 8 位无符号的整数
        EncryptString = EncryptString + ascii.GetString(b);     //以字符串的形式显示
    }
    return EncryptString;
}
```

说明：字符串的解密其实就是加密算法的逆向操作，也就是对每个字符的 ASCII 码进行减 5 操作。

5.5.2 字符串处理

示例 05-14 统计文本框中英文单词的个数

本示例实现统计文本框中英文单词的个数，运行程序后，在文本框中输入带有英文单词的文本，单击“确定”按钮，即可显示输入的英文单词个数。运行效果如图 5.16 所示。

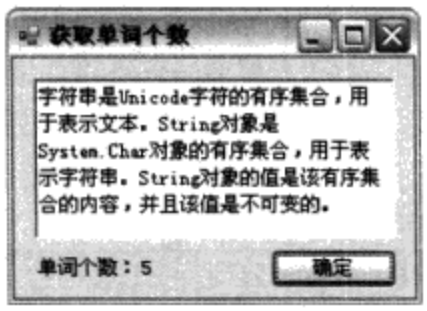


图 5.16 统计英文单词个数

程序代码如下：

代码 05-14 光盘位置：光盘\mingrisoft\sl\05\15

```
public int WordC(string Str)
{
    int n = 0;                                          //记录单词个数
    bool b = false;                                     //判断是否为英文单词
    for (int i = 1; i < Str.Length; i++)                //遍历文本
    {
        char ch = Str[i - 1];                           //获取指向当前文本中字符的前一个字符
        if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))   //判断该字符是否为英文
        {
            b = true;                                   //是英文
        }
        else
        {
            b = false;                                  //不是英文
        }
        if (b)                                          //如果前一个字符是英文
        {
            ch = Str[i];                                //获取当前字符
            if (!((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')))//如果当前字符不是英文
            {
                ++n;                                    //单词个数加 1
            }
        }
    }
    return n;
}
```

5.6 穷 举 法

穷举法又称枚举法，其算法思想是：一一列举所有可能的情况，并判断哪种情况是符合要求的解，对于那些不能用解析法求解的问题，使用此方法往往能奏效。下面使用实例来说明穷举法的算法实现过程。

示例 05-15 穷举法实现百钱百鸡

使用穷举法的典型例子如百钱百鸡，即现在有 100 元，要求买 100 只鸡，其中小鸡每只 5 角，公鸡每只 2 元，母鸡每只 3 元。列出所有满足百钱百鸡的条件，程序运行效果如图 5.17 所示。

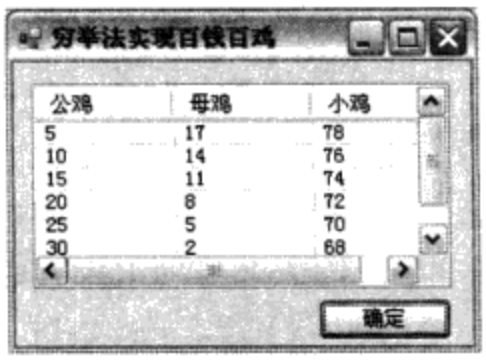


图 5.17　穷举法实现百钱百鸡

代码如下：

代码 05-15　光盘位置：光盘\mingrisoft\sl\05\16

```
private void button1_Click(object sender, EventArgs e)
{
    int a = 0, b = 0, c = 0, p = 0;                          //定义变量
    for (b = 1; b <= 50; b++)                                //公鸡的百元中的遍历
    {
        for (a = 1; a <= 33; a++)                            //母鸡在百元中的遍历
        {
            c = 100 - a - b;                                 //获取小鸡的个数
            if (((b * 2 + a * 3 + c * 0.5) == 100))          //如果公鸡、母鸡和小鸡的总钱数加起来为 100
            {
                ListViewItem OptionItem = new ListViewItem(b.ToString());   //在列中添加公鸡的个数
                OptionItem.SubItems.Add(a.ToString());       //在列中添加母鸡的个数
                OptionItem.SubItems.Add(c.ToString());       //在列中添加小鸡的个数
                listView1.Items.Add(OptionItem);             //向 listView 控件中追加新添加的各列
            }
        }
    }
}
```

5.7　迭　代　法

迭代法也称为递推法，对于一个问题求解，可先给定一个初值，通过某一算法（公式）求得一个新值，再以新值作为初值，按照同样的算法求得另一个新值，这样经过有限次运算，即可求得其解。

示例 05-16　迭代法求某数平方根

用迭代法求得某数的平方根基本算法思想是：已知求平方根 $\sqrt{a}$ 的迭代公式为：

$$x1=\frac{1}{2}(x0+\frac{a}{x0})$$

先给定一个初值 $x0=a/2$(估计值)，根据迭代公式得到一个新的值 $x1$，这个新值 $x1$ 比初值 $x0$ 更接近要求的值；再以新值作为初值，重新按原来的方法求新值 $x1$，重复这一过程直到 $|x1-x0|<\varepsilon$（给定的精度）。此时得到的 $x1$ 即可作为问题的解。

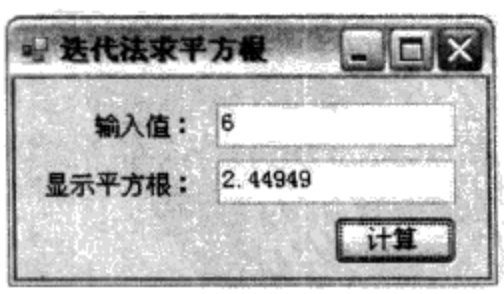


图 5.18　迭代法求平方根

程序运行效果如图 5.18 所示。

实现迭代算法求某数平方根的程序代码如下：

代码 05-16　光盘位置：光盘\mingrisoft\sl\05\17

```
public float Fsqr(float x)
{
    float x0, x1;
```

```
    x0 = x / 2;                                    //给定初值
    x1 = 0.5F * (x0 + x / x0);                     //使用迭代公式求新值
    do
    {
        x0 = x1;                                   //将新值赋给初值
        x1 = 0.5F * (x0 + x / x0);                 //求新值
    } while (Math.Abs(x1 - x0) > 0.00001);         //如果比较结果小于给定精度
    return x1;                                     //返回平方根
}
```

第 6 章

数组结构

- 单向链表
- 双向链表
- 堆栈
- 队列

6.1 单向链表

6.1.1 单向链表的定义

链表是一种重要的数据结构，该结构由结点组成。每个结点包含两部分数据，第一部分是结点本身的数据，第二部分是指向下一个结点的指针。对于单向链表，链表中存在两个特殊的结点，分别为“头结点”和“尾结点”。头结点本身没有数据，只存储下一个结点的指针，尾结点只存储数据。单向链表结构如图 6.1 所示。

图 6.1 单向链表结构

把单链表结点看作一个类，其类名为 ListNode，单链表结点类的实现如下：

```
public class ListNode                                   //结点类
{
    public ListNode(int NewValue)                       //自定义方法，获取结点的值
    {
        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 用于储存结点本身的数据，Next 用于存储下结点的指针，Previous 用于存储上一个结点的指针。

6.1.2 单链表的操作

链表的主要操作是对结点进行查找、插入和删除。下面对其操作过程进行详细讲解。

1．查找结点

在进行结点查找时，通常根据结点的数据查找结点。要实现结点的查找，首先需要解决的问题就是遍历链表中的所有结点。可以从链表头结点开始，利用循环的方式向下查找，如果当前结点指向的下一个结点不为空，则将下一个结点设置为当前结点。

2．插入结点

插入结点需要考虑两种情况。第一种情况是在链表尾结点处插入结点，这种情况比较简单，只要将当前尾结点的 FNextNode（指向下一个结点的指针）指向新插入的结点就可以了。第二种情况是在链表的中间位置插入结点，首先需要确定在哪个结点后插入新的结点，这就应该由用户来确定了（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结点），然后需要记录当前结点的下一个结点（避免发生“脱节”），可以定义一个临时的结点进行记录。将当前结点的下一个结点指向新添加的结点，再将新添加结点的下一个结点指向临时结点，整个插入结点的流程如图 6.2 所示。

3．删除结点

删除结点与插入结点类似，也需要考虑两种情况。第一种情况是删除尾结点，这种情况操作相对比较简单，首先找到尾结点之前的一个结点，将该结点指向的下一个结点设置为空，然后释放尾结点资源。第二种情况是删除链表中间部分的结点。首先需要找到当前结点的上一个结点，并定义一个临时结点 PriorNode 进行记录，然后再定义一个临时结点 TempNode 记录当前结点的下一个结

点，最后将 PriorNode 的下一个结点指向 TempNode，并释放当前结点，整个流程如图 6.3 所示。

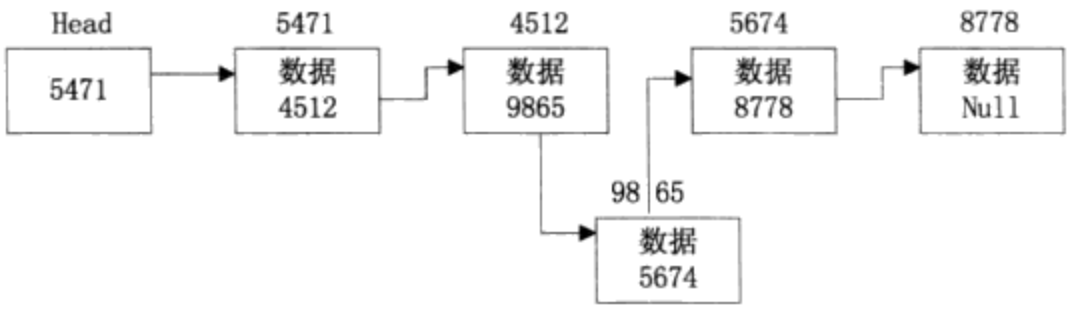


图 6.2 插入结点流程图

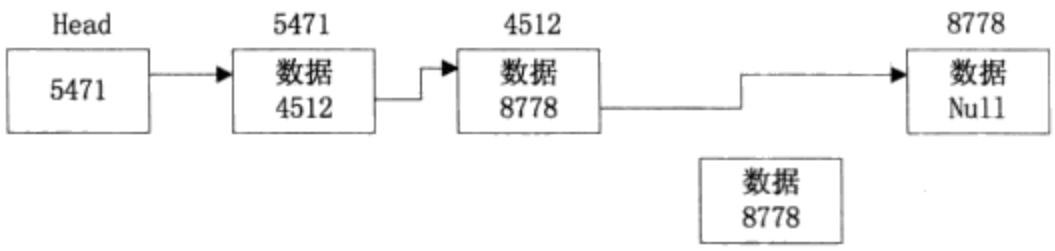


图 6.3 删除结点流程图

说明：对单链表的操作除了查找、插入和删除外，还有结点的上下移动，判断结点是否为空，结点是否在链表首或链表尾等。

6.1.3 单链表的制作

示例 06-01 单链表的制作

本示例将通过类库制作一个单链表，并实现链表的查询、插入、删除等操作。

程序代码如下：

代码 06-01 光盘位置：光盘\mingrisoft\sl\06\01

在类库中创建一个类 Clist，并在该类的构造函数中对定义的指针变量进行初始化。代码如下：

```
public class Clist
{
    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
```

自定义方法 Append 用于在当前结点的尾部添加数据。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;                                 //设置当前结点为头指针
            Tail = NewNode;                                 //设置当前结点为尾指针
        }
        else
        {
            Tail.Next = NewNode;                            //将当前结点指向尾指针后
            NewNode.Previous = Tail;                        //设置尾指是当前结点的上一个结点
            Tail = NewNode;                                 //将当前结点设置为尾指针
        }
```

```
        Current = NewNode;
        ListCountValue += 1;                    //链表数据个数加一
    }
```

自定义方法 Delete 用于删除当前结点中的数据。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;   //若删除指定的数据
            Current = Current.Previous;
            ListCountValue -= 1;
            return;
        }
    }
```

自定义方法 MoveNext 用于将指针移动到当前结点的下一个结点。代码如下：

```
    public void MoveNext()
    {
        if (!IsEof()) Current = Current.Next;
    }
```

自定义方法 MovePrevious 用于将指针移动到当前结点的前一个结点。代码如下：

```
    public void MovePrevious()
    {
        if (!IsBof()) Current = Current.Previous;
    }
```

自定义方法 MoveFrist 用于将指针移动到指定的结点上。代码如下：

```
    public void MoveFrist()
    {
        Current = Head;
    }
```

自定义方法 MoveLast 用于将指针移动到最后一个结点上。代码如下：

```
    public void MoveLast()
    {
        Current = Tail;
    }
```

自定义方法 IsNull 用于判断是否为空链表。代码如下：

```
    public bool IsNull()
    {
        if (ListCountValue == 0)
            return true;
        return false;
    }
```

自定义方法 IsEof 用于判断是否达到链表尾。代码如下：

```
    public bool IsEof()
    {
        if (Current == Tail)
            return true;
        return false;
    }
```

自定义方法 IsBof 用于判断是否达到链表首。代码如下：

```
public bool IsBof()
{
    if (Current == Head)
        return true;
    return false;
}
```

自定义方法 GetCurrentValue 用于获取当前结点的值。代码如下：

```
public int GetCurrentValue()
{
    return Current.Value;
}
```

自定义属性 ListCount 用于获取当前链表的数据个数。代码如下：

```
public int ListCount
{
    get
    {
        return ListCountValue;
    }
}
```

自定义方法 Clear 用于清空当前链表。代码如下：

```
public void Clear()
{
    MoveFrist();
    while (!IsNull())
    {
        Delete();                                   //若不为空链表,从尾部删除
    }
}
```

自定义方法 Insert 用于在指定结点的前面插入一个结点，如果是空链表，则在尾部添加一个结点。代码如下：

```
public void Insert(int DataValue)
{
    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;                        //将新结点的指针指向头部结点
        Head.Previous = NewNode;                    //设置头指针的上一个结点
        Head = NewNode;                             //将插入结点设置为头结点
        Current = Head;
        ListCountValue += 1;                        //节点值加 1
        return;
    }
    //中间插入
    NewNode.Next = Current;                         //将新结点的指针指向当前结点
    NewNode.Previous = Current.Previous;            //设置当前结点的上一个结点
    Current.Previous.Next = NewNode;                //将当前结点的上一个结点的指针指向插入结点
    Current.Previous = NewNode;                     //插入结点
    Current = NewNode;
    ListCountValue += 1;                            //结点数加 1
}
```

自定义方法 InsertAscending 按升序在指定结点前插入数据。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
        }
    }
```

自定义方法 InsertUnAscending 按降序在指定结点前插入数据。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
        }
    }
}
```

6.2 双向链表

6.2.1 双向链表的定义

前面介绍的单链表允许从一个结点直接访问它的后继结点。但是，要找某个结点的直接前

驱结点，只能从表头引用开始遍历各结点，如果某个结点的 Next 等于该结点，那么，这个结点就是该结点的直接前驱结点。

为了克服单链表这种单向性的缺点，可以使用双向链表。在双向链表的结点中有两个指针域，一个保存直接前驱结点的地址，叫 prev，另一个直接后继结点的地址，叫 next，这样的链表就是双向链表（Doubly Linked List）。双向链表的结点结构示意图如图 6.4 所示。

图 6.4　双向链表的结点结构示意图

6.2.2　双向链表的操作

1．插入操作

由于双向链表有两个引用结点，所以，在双向链表中插入和删除结点比单链表要复杂。双向链表中结点的插入分为在结点之前插入和在结点之后插入两种，插入操作要对 4 个引用进行操作。下面以在结点之后插入结点为例来说明在双向链表中插入结点的过程。

设 p 是指向双向链表中的某一结点，即 p 存储的是该结点的地址，现将一个结点 L 插入到结点 p 的后面，插入过程如图 6.5 所示（以 p 的直接后继结点存在为例）。

操作如下：

① p.Next.Prev = L;

② L.Prev = p;

③ L.Next = p.Next;

④ p.Next = L。

在向结点后插入结点可以有多种方式，但操作③必须放到操作④的前面完成，否则 p 就无法找到后继结点了。

2．删除操作

本节主要讲解如何在双向链表结点后删除结点。设 p 是指向双向链表中的某一结点，即 p 存储的是该结点的地址，现将结点 p 从双向链表中删除，删除过程如图 6.6 所示。

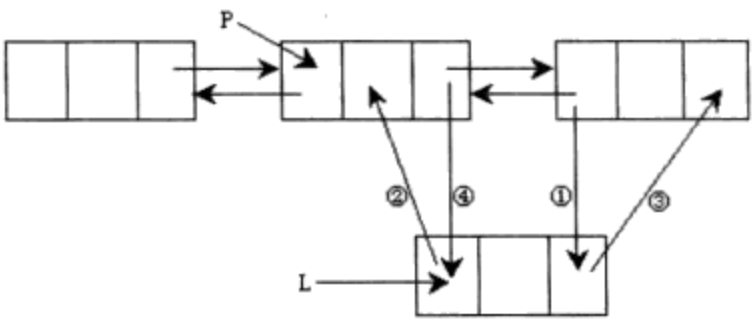


图 6.5　双向链表结点插入示意图

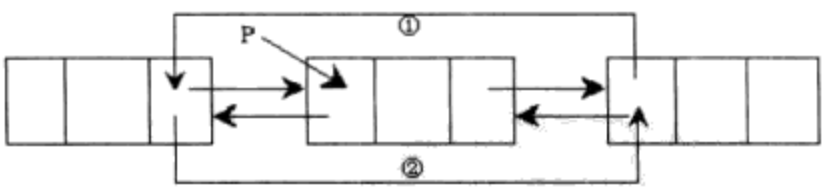


图 6.6　双向链表结点插入示意图

操作如下：

① p.Next = P.Next.Next；

② p.Next.Prev = p.Prev。

双向链表的其他操作与单链表基本相同，在这里不做讲解。

6.2.3　双向链表的制作

示例 06-02　双向链表的制作

本示例将通过类库制作一个双向链表，并实现链表的查询、插入、删除等操作。

程序代码如下：

代码 06-02　光盘位置：光盘\mingrisoft\sl\06\02

在类库中创建一个类 Objects，并在该类的构造函数中对设置的变量进行初始化。代码如下：

```
public class Objects
{
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;
        }
    }
}
```

在 ListNode 类中定义链表中的前一个结点、后一个结点以及结点值。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;
            }
        }
    }
```

在 Clists 类中定义链表的头指针、尾指针和当前指针等。代码如下：

```
    public class Clists//类
    {
        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
```

自定义方法 Append 用于在链表尾部添加数据。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
    }
```

自定义方法 Delete 用于删除指定的结点。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;               //当前结点的指针指向上一个结点
            Current = Current.Previous;                         //删除当前结点
            ListCountValue -= 1;                                //结点数减 1
            return;
        }
    }
```

自定义方法 MoveNext 用于向后移动一个结点。代码如下：

```
    public void MoveNext()
    {
        if (!IsEof()) Current = Current.Next;
    }
```

自定义方法 MovePrevious 用于向前移动一个结点。代码如下：

```
    public void MovePrevious()
    {
        if (!IsBof()) Current = Current.Previous;
    }
```

自定义方法 MoveFrist 用于移动到首结点。代码如下：

```
    public void MoveFrist()
    {
        Current = Head;
    }
```

自定义方法 MoveLast 用于移动到尾结点。代码如下：

```
    public void MoveLast()
    {
        Current = Tail;
    }
```

自定义方法 IsNull 用于判断当前链表是否为空链表。代码如下：

```
    public bool IsNull()
    {
        if (ListCountValue == 0)                                //如果为空链表
          return true;                                          //返回 true
        else
          return false;                                         //返回 false
    }
```

自定义方法 IsEof 用于判断指针是否指向结点尾。代码如下：

```
    public bool IsEof()
    {
        if (Current == Tail)                                    //如果指向结点尾
          return true;
        else
```

```
        return false;
    }
```

自定义方法 IsBof 用于判断指针是否指向首结点。代码如下：

```
    public bool IsBof()
    {
        if (Current == Head)                                    //如果指向首结点
          return true;
        else
          return false;
    }
```

自定义方法 Objects 用于获取当前结点的值。代码如下：

```
    public Objects GetCurrentValue()
    {
        return Current.goods;
    }
```

自定义属性 ListCount 用于获取当前的数据个数。代码如下：

```
    public int ListCount
    {
        get
        {
            return ListCountValue;
        }
    }
```

自定义方法 Clear 用于清空当前链表。代码如下：

```
    public void Clear()
    {
        MoveFrist();                                            //移除首结点的指针
        while (!IsNull())
        {
            Delete();                                           //若不为空链表，从尾部删除
        }
    }
```

自定义方法 Insert 用于在指定结点的前面插入一个结点，如果是空链表，则在尾部添加一个结点。代码如下：

```
    public void Insert(Objects DataValue)
    {
        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;
    }
```

自定义方法 InsertAscending 按升序在指定结点前插入数据。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
        }
    }
```

自定义方法 InsertUnAscending 按降序在指定结点前插入数据。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
        }
    }
}
```

6.3 堆　　栈

6.3.1 堆栈的定义及操作思路

栈是一种重要的数据结构。它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（LIFO）或先进后出（FILO）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。

栈通常记为：S = (a1,a2,…,an)，S 是英文单词 stack 的缩写。a1 为栈底元素，an 为栈顶元素。这 *n* 个数据元素按照 a1,a2,…,an 的顺序依次入栈，而出栈的次序相反，an 第一个出栈，a1 最后一个出栈。所以，栈的操作是按照后进先出的原则进行的。栈的操作示意图如图 6.7 所示。

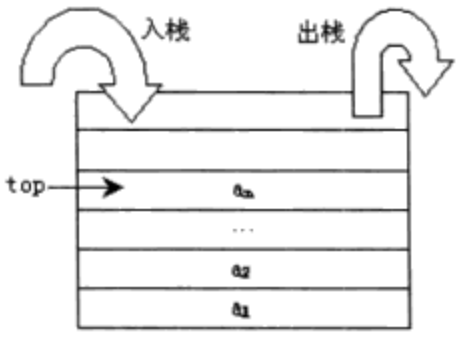


图 6.7　栈的操作示意图

由于栈只能在栈顶进行操作，所以栈不能在栈的任意一个元素处插入或删除元素。因此，栈的操作是线性表操作的一个子集。栈的操作主要包括在栈顶插入元素和删除元素、取栈顶元素和判断栈是否为空等。

栈的操作好比是一个弹夹，将子弹一个接一个地放入弹夹中（元素的入栈操作），然后将子弹一发发地打出去（元素的出栈操作），当弹夹为空（堆栈为空）时停止发射。

6.3.2　堆栈的制作

示例 06-03　堆栈的制作

本示例将通过类库制作一个堆栈，并实现堆栈的入栈、出栈等操作。

程序代码如下：

代码 06-03　光盘位置：光盘\mingrisoft\sl\06\03

在类库中创建一个类 CStack，并在该类中设置堆栈的相关操作。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;                                         //空的时候为 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>
/// 向上移动
/// </summary>
public void MoveDown()
{
    m_List.MovePrevious();
}
/// <summary>
/// 取得当前的值
/// </summary>
public int GetCurrentValue()
{
    return m_List.GetCurrentValue();
}
/// <summary>
/// 删除取得的当前结点
/// </summary>
public void Delete()
{
    m_List.Delete();
}
/// <summary>
```

```
        /// 清空堆栈
        /// </summary>
        public void Clear()
        {
            m_List.Clear();
        }
    }
```

注意：因为在算法中引用了链表的操作，所以要添加对示例 06-01 所生成类库的引用。

6.4 队　　列

6.4.1 队列的定义及操作思路

队列（Queue）也是链表的扩展，它是一种特殊的链表，如堆栈一样。它和堆栈的不同在于，堆栈采用的是先进后出原则。队列是插入操作限定在表的尾部而其他操作限定在表的头部进行的线性表。把进行插入操作的表尾称为队尾（Rear），把进行其他操作的头部称为队头（Front）。当队列中没有数据元素时称为空队列(Empty Queue)。

队列通常记为：Q= (a1,a2,…,an)，Q 是英文单词 queue 的缩写。a1 为队头元素，an 为队尾元素。这 n 个元素是按照 a1,a2,…,an 的次序依次入队的，出队的次序与入队相同，a1 第一个出队，an 最后一个出队。所以，队列的操作是按照先进先出（FIFO）或后进后出（LILO）的原则进行的。队列的操作示意图如图 6.8 所示。

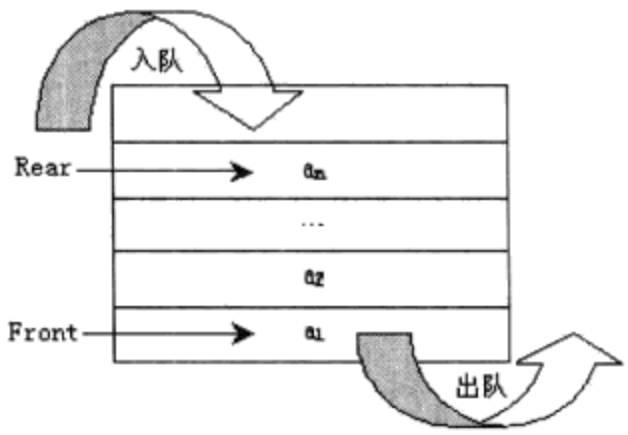


图 6.8　队列的操作示意图

在日常生活中也到处可见，比如，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。

队列的操作是线性表操作的一个子集。其操作主要包括在队尾插入元素、在队头删除元素、取队头元素和判断队列是否为空等。

与栈一样，队列的运算是定义在逻辑结构层次上的，而运算的具体实现是建立在物理存储结构层次上的。因此，把队列的操作作为逻辑结构的一部分，每个操作的具体实现只有在确定了队列的存储结构之后才能完成。队列的基本运算不是它的全部运算，而是一些常用的基本运算。

6.4.2 队列的制作

示例 06-04　队列的制作

本示例将通过类库制作一个队列，并实现队列的入队、出队等操作。

程序代码如下：

代码 06-04　光盘位置：光盘\mingrisoft\sl\06\04

在类库中创建一个类 CQueue，并在该类中设置队列的相关操作。代码如下：

```
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
            }
        }
    }
```

第 7 章

程序调试与错误处理

- 错误类型
- 程序调试
- 错误处理

在程序编写过程中，难免会遇到一些问题。为了解决这些问题，开发人员需要对应用程序进行调试，查出问题的根源。本章将主要介绍如何对程序进行调试以及错误处理。

7.1 错误类型

在调试过程中可能会遇到各种类型的错误，这些错误可以分为语法错误和语义错误两种。

语法错误是一种程序错误，它会影响编译器的工作，是一种最简单的错误。如果不按照编程语言的规则编写代码，就会产生语法错误。在 Visual Studio 2008 中，当遇到语法错误时，错误消息将显示在“错误列表”窗口中。这些消息告诉程开发人员语法错误的位置（行、列和文件），并给出错误的简要说明，如图 7.1 所示。

语义错误是一种比较棘手的错误，它通常在程序运行过程中出现。语义错误会导致程序非正常终止。前面章节介绍过的异常处理就属于语义错误，如图 7.2 所示。

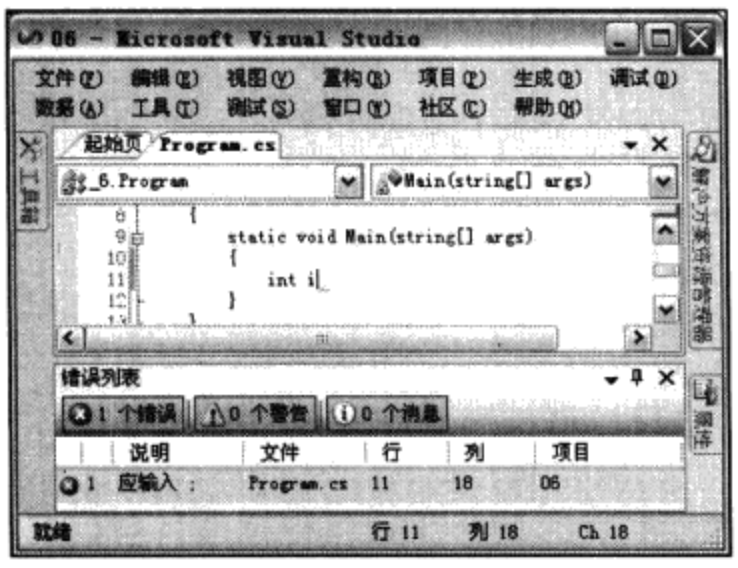


图 7.1 语法错误

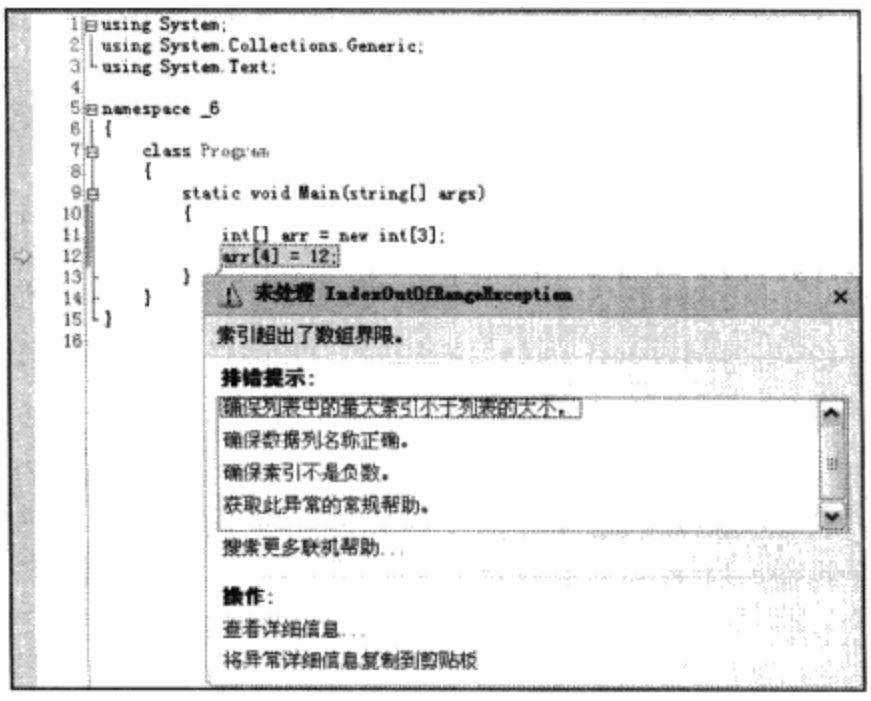


图 7.2 语义错误

如果程序的语法是完全正确的，但在运行过程中检测到数组的索引大于定义数组的界限，那么将会出现语义错误。

7.2 程序调试

如果希望应用程序能够成功运行并能产生正确的结果，必须消除应用程序中的语法错误和语义错误，那么就必须进行调试。

在 Visual Studio.NET 环境中可应用 Visual Studio 调试器进行调试，该调试器提供了功能强大的命令来控制应用程序的执行。下面介绍几种对应用程序进行调试的方法。

7.2.1 断点

断点是一个信号，它通知调试器应在某点上中断应用程序并暂停执行。发生中断时，程序和调试器处于中断模式。进入中断模式并不会终止或结束程序的执行，所有元素（例如函数、变量和对象）都保留在内存中，执行可以在任何时候继续。

通过调试器界面可以请求断点，由调试器加入断点。利用“断点”窗口可以处理复杂的断点。

插入断点有 3 种方式：在要设置断点的行旁边的灰色空白中单击；右键单击设置断点的代码行；选择“断点”/“插入断点”命令，如图 7.3 所示；单击要设置断点的代码行，选择菜单中的“调试”/“切换断点（G）”命令，如图 7.4 所示。

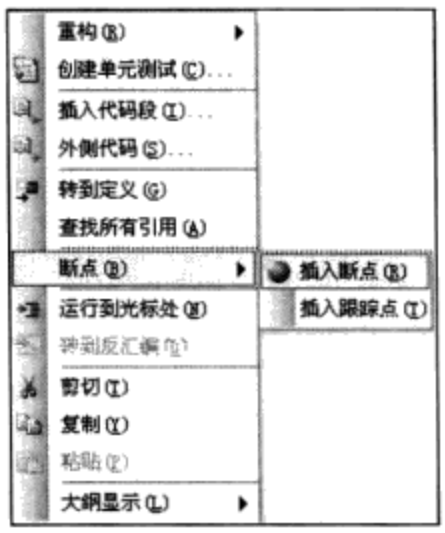


图 7.3　插入断点

图 7.4　菜单栏插入断点

插入断点后，就会在设置断点的行旁边灰色空白处出现一个红色圆点，并且该行代码也高亮显示，如图 7.5 所示。

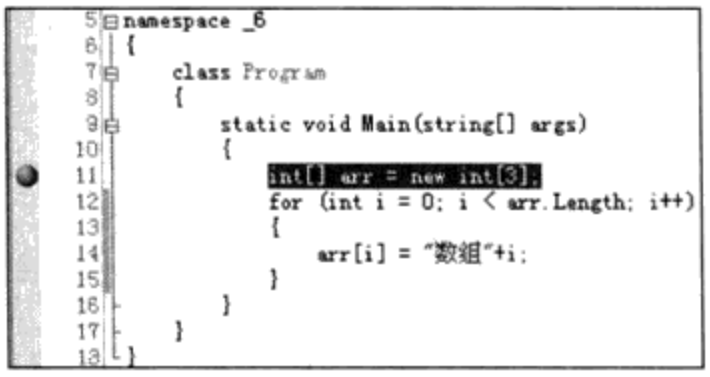


图 7.5　插入断点后效果图

删除断点有 4 种方式：单击断点行旁边的灰色空白处的红色圆点；右击断点行旁边的灰色空白处的红色圆点，选择“删除断点”，如图 7.6 所示；右键单击设置断点的代码行，选择“断点”/“删除断点”命令，如图 7.7 所示；单击要设置断点的代码行，选择菜单中的“调试”/“切换断点（G）”命令。

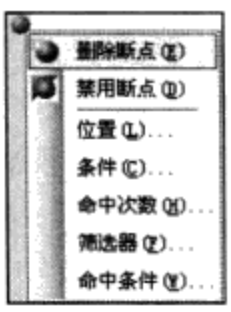


图 7.6　删除断点

图 7.7　菜单删除断点

7.2.2　开始执行

可以通过在“调试”菜单中选择“启动调试”、“逐语句”或“逐过程”来执行程序并调试，也可以通过右键单击可执行代码中的某行，然后从快捷菜单中选择“运行到光标处”执行程序并调试。

如果选择“启动调试”，则应用程序启动并一直运行到断点，如图 7.8 所示。可以在任何时刻中断执行，以检查值、修改变量或检查程序状态。

如果选择了“逐语句”或“逐过程”，应用程序启动并执行，然后在第一行中断，如图 7.9

所示。

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace _6
6  {
7      class Program
8      {
9          static void Main(string[] args)
10         {
11             string[] arr = new string[3];
12             for (int i = 0; i < arr.Length; i++)
13             {
14                 arr[i] = "数组"+i;
15             }
16         }
17     }
18 }
19
```

图 7.8 “启动调试”运行结果图

如果选择“运行到光标处”，则应用程序启动并一直运行到断点或光标位置。如果光标在断点前，则程序运行到光标处，如图 7.10 所示；如果光标在断点后则运行到断点。可以在代码窗口中设置光标的位置。某些情况下，不出现中断，这意味着执行始终未到达设置光标处的代码。

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace _6
6  {
7      class Program
8      {
9          static void Main(string[] args)
10         {
11             string[] arr = new string[3];
12             for (int i = 0; i < arr.Length; i++)
13             {
14                 arr[i] = "数组"+i;
15             }
16         }
17     }
18 }
19
```

图 7.9 “逐语句”或“逐过程”调试运行结果图

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace _6
6  {
7    class Program
8    {
9      static void Main(string[] args)
10     {
11       string[] arr = new string[3];
12       for (int i = 0;i<arr.Length;i++)
13       {
14         arr[i] = "数组"+i;
15       }
16     }
17   }
18 }
```

图 7.10 光标在断点前运行结果图

7.2.3 中断执行

当应用程序执行到达一个断点或发生异常时，调试器就会中断程序的执行。用户也可以通过在“调试”菜单上单击“全部中断”命令手动中断执行。这时调试器将停止所有在调试器下运行的程序的执行，但程序并不退出，而且可以随时恢复执行。调试器和应用程序此时处于中断模式。

7.2.4 停止执行

停止调试意味着终止当前正在调试的程序并结束调试会话。与中断执行不同，中断执行意味着暂停正在调试的进程的执行，但调试会话仍处于活动状态。

可以通过选择菜单中的“调试”/“停止调试”命令或单击“调试”工具栏中的■按钮来结束运行和调试，也可以退出正在调试的应用程序，调试将自动停止。

7.2.5 单步执行

单步执行是最常见的调试过程之一，即每次执行一行代码。“调试”菜单中提供了 3 个逐句执行代码的命令，即“逐语句”、“逐过程”和“跳出”。

“逐语句”和“逐过程”的差异仅在于它们处理函数调用的方式不同。这两个命令都指示调试器执行下一行的代码。如果某一行包含函数调用，“逐语句”仅执行调用本身，然后在函数内的第一个代码行处停止；而“逐过程”执行整个函数，然后在函数外的第一行处停止。如果要查看函数调用的内容，则使用“逐语句”。若要避免单步执行函数，那么最好使用“逐过程”。

当位于函数调用的内部并想返回到调用函数时，可以使用“跳出”。“跳出”将一直执行代码，直到函数返回，然后在调用函数中的返回点处中断。

7.2.6 运行到指定位置

如果在调试过程中想执行到代码中的某一点，然后中断，可以在要中断的位置设置断点，也可以在“调试”菜单中选择“启动”或“继续”。该位置可以在源窗口或“反汇编”窗口中设置。

在代码窗口中右键单击某行，并从快捷菜单中选择“运行到光标处”，如图 7.11 所示，执行程序将在光标所在行中断。

在“反汇编”窗口中右键单击某行，并从快捷菜单中选择“运行到光标处”，程序也将在光标所在行中断。如果没有显示“反汇编”窗口，那么从“调试”菜单中选择“窗口”/“反汇编”。“反汇编”窗口只能在中断模式下才能查看。

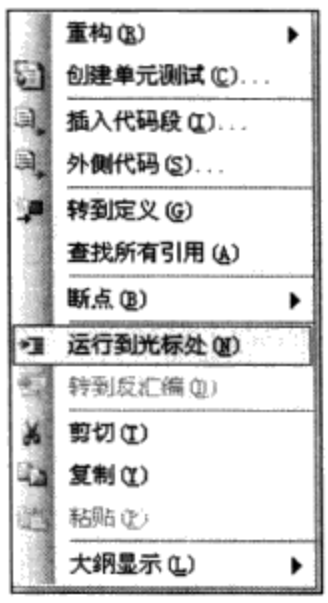


图 7.11 运行到光标处

7.3 错 误 处 理

在 Visual Studio 中测试文件系统时，ASP.NET Development Server 将自动运行。某些情况下，使用 ASP.NET Development Server 会产生错误。本节介绍 Web 服务器可能产生的错误，并提供相应的解决办法。

7.3.1 Web 服务器配置不正确

此错误显示如下：

```
The web server is not configured correctly. See help for common configuration errors. Running the web page outside of
the debugger may provide further information.
```

可能引起该错误的原因如下。

- 尝试调试一个已复制到不同的计算机上、经过手动重命名或移动过的.NET Web 应用程序。
- 没有足够的 IIS 连接。

- DEBUG 谓词没有与.aspx 关联。
- 在 IIS 中该网站没有配置为应用程序。
- 在尝试调试 Web 应用程序时，DEBUG 谓词没有与 ISAPI 扩展名关联。

7.3.2 IIS 管理服务没有响应

当 IIS 管理服务没有响应时，会发生“安全检查失败，因为 IIS 管理服务没有响应”的错误，这通常表示 IIS 的安装有问题。

解决此错误的方法如下：

使用“管理工具”中的“服务”工具验证该服务是否正在运行，然后按照如下方法进行操作。

- 使用控制面板中的“添加/删除程序”重新安装 IIS。
- 使用控制面板中的“添加/删除程序”从计算机中删除 IIS 并重新安装 IIS。

说明：执行这两个步骤中的任一步骤后，重新启动计算机。

7.3.3 未安装 ASP.NET

当用户尝试调试的计算机上未正确安装 ASP.NET 时，会发生“未安装 ASP.NET”的错误。此错误可能意味着从未安装过 ASP.NET，或者先安装了 ASP.NET，后安装 IIS。

解决此错误的方法如下：

选择“开始”菜单中的“运行”命令，打开“运行”窗口，在“运行”窗口输入下列命令卸载 IIS：

```
\WINNT\Microsoft.NET\Framework\version\aspnet_regiis -i
```

其中，version 表示安装在用户计算机上的.NET Framework 的版本号（例如，v1.0.370）。

说明：对于 Windows Server 2003，可以使用控制面板中的“添加/删除程序”来安装 ASP.NET。

第 2 篇

Windows 窗体设计篇

C#

第 8 章

Windows 窗体介绍

- 创建 Windows 窗体
- 创建继承窗体
- 创建 MDI 窗体

Windows 窗体（即 WinForms）是 Microsoft .NET Framework 的一部分，开发人员可以在 Visual Studio 2008 开发环境中通过新建项目来创建一个 Windows 应用程序。本章将主要介绍 Windows 窗体的相关内容。

8.1 设置 Windows 窗体

Windows 窗体是程序界面设计的基础，也是应用程序的一个重要组成部分。程序运行时，每一个窗体对应一个可视化的窗口。

Windows 窗体提供了定义窗体外观的属性、定义行为的方法和定义与用户交互的事件。

通过设置窗体属性和编写代码来响应窗体事件，可以开发满足应用程序需求的窗体。Windows 窗体是从 Form 类派生而来的，而 Form 类是从 Control 类派生而来的，这种框架决定了其可以继承现有的窗体来添加功能或修改现有行为。当为应用程序添加一个 Windows 窗体时，可以选择从.NET Framework 提供的 Form 类继承，也可以选择从先前创建的 Windows 窗体继承。

Windows 窗体除了具有自己的属性、方法和事件之外，还可以作为其他控件的容器，用户可以将其他控件放置在 Windows 窗体上。

8.1.1 创建窗体

Windows 窗体必须依赖于 Windows 应用程序，因此，要创建 Windows 窗体，首先应该新建一个 Windows 应用程序。在 Visual Studio 2008 开发环境中，创建 Windows 应用程序和添加新窗体的步骤如下。

（1）依次选择“开始”/“所有程序”/“Microsoft Visual Studio 2008”/“Microsoft Visual Studio 2008”选项，打开“Visual Studio 2008”开发环境，在“文件”菜单中，选择“新建”/“项目”选项，如图 8.1 所示。

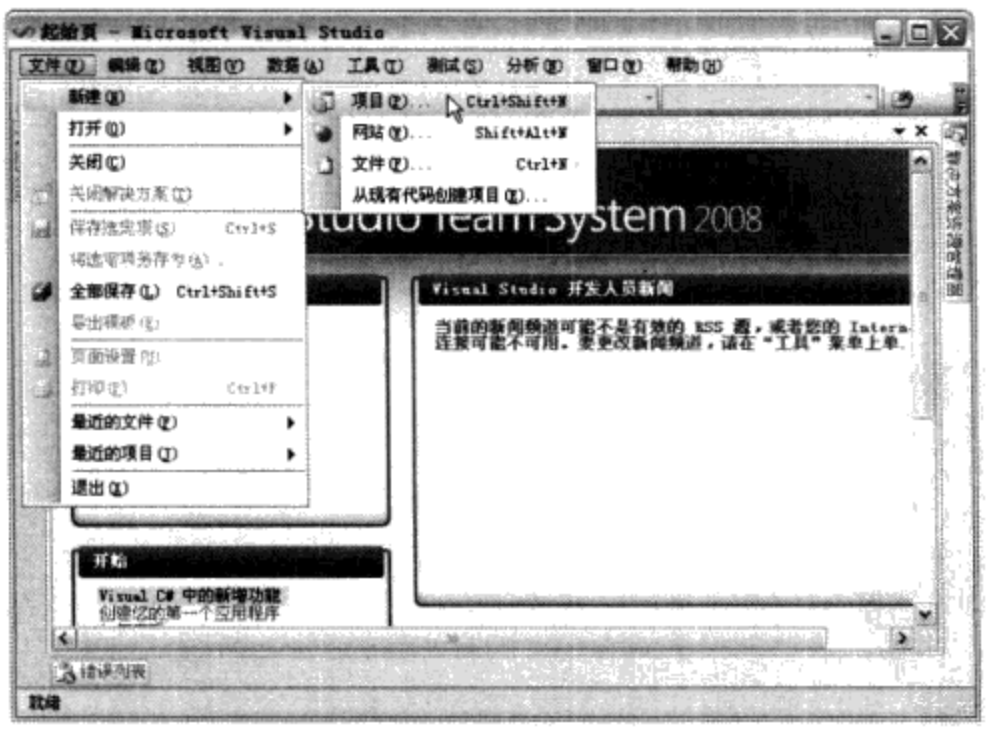


图 8.1 选择“新建”/“项目”选项

（2）弹出如图 8.2 所示的“新建项目”对话框，在该对话框中，用户可以选择项目要使用的语言、Visual Studio 已安装的模板等，在这里，选择“Windows 窗体应用程序”。

（3）单击【确定】按钮，即可新建一个 Windows 应用程序。如果想在当前工程中创建新的 Windows 窗体，可以在“解决方案资源管理器”中选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建项”或在菜单栏中选择“添加”/“Windows 窗体”选项，如图 8.3 所示。

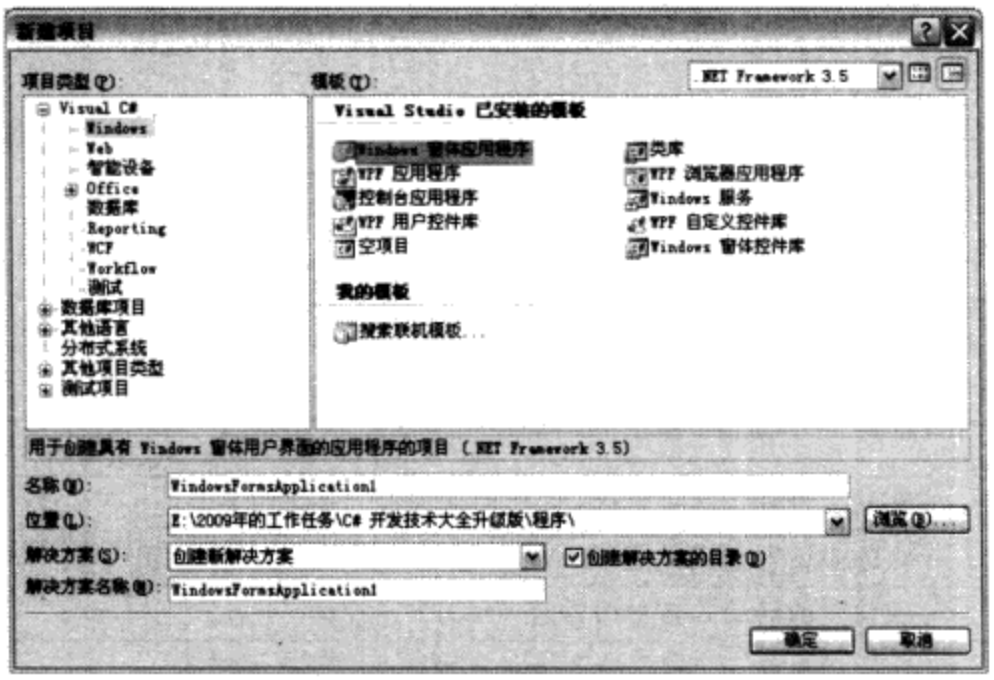


图 8.2 “新建项目”对话框

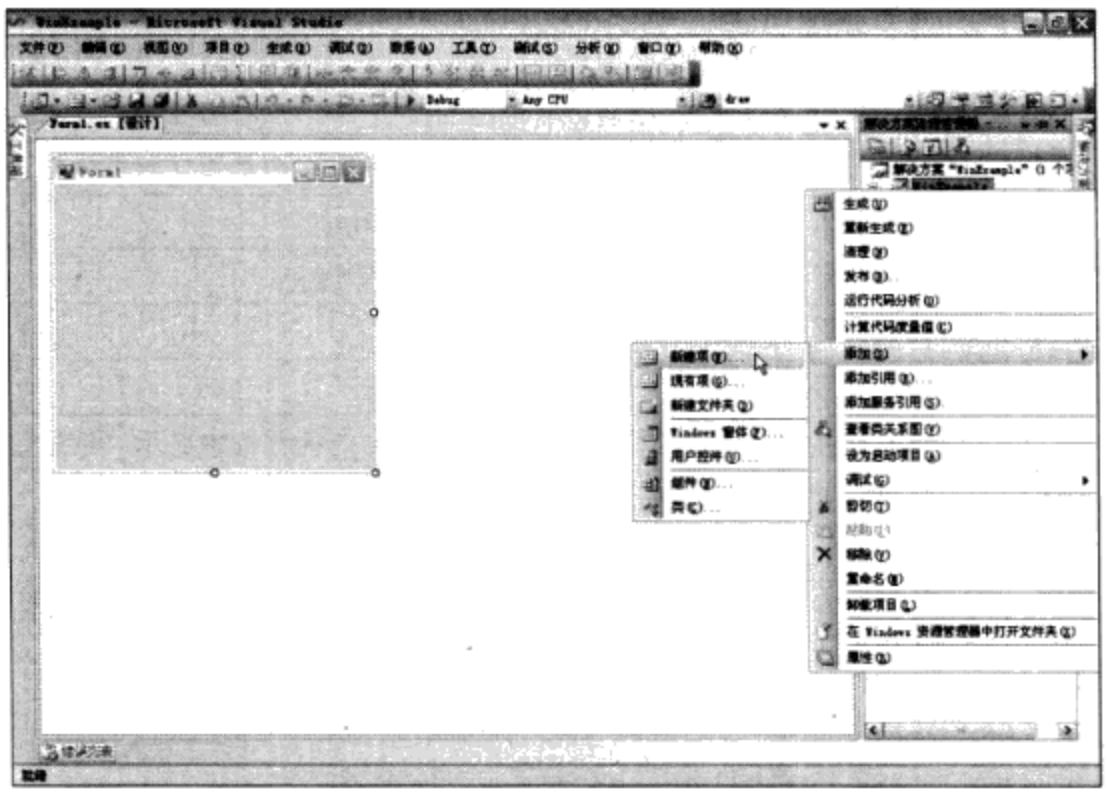


图 8.3 选择“添加”/“新建项”选项

(4) 弹出如图 8.4 所示的“添加新项”对话框，在该对话框中，选择“Windows 窗体”，并在“名称”文本框中为添加的 Windows 窗体命名，然后单击【添加】按钮，即可在当前项目中添加一个新的 Windows 窗体。

图 8.4 “添加新项”对话框

8.1.2 窗体的属性、方法和事件

Windows 窗体界面的外观是应用程序的重要组成部分，用户可以通过属性来设置 Windows 窗体的外观，也可以通过方法、事件对窗体进行相应的操作。

1．属性

Windows 窗体的常用属性及说明如表 8.1 所示。

表 8.1　　Windows 窗体的常用属性及说明

属　性	说　明
Name	获取或设置控件的名称
AcceptButton	获取或设置当用户按<Enter>键时所单击的窗体上的按钮
ActiveMdiChild	获取当前活动的多文档界面（MDI）子窗体
AutoScroll	获取或设置一个值，该值指示窗体是否实现自动滚动
AutoScrollMinSize	获取或设置自动滚动的最小尺寸
BackColor	获取或设置控件的背景色
BackgroudImage	获取或设置在控件中显示的背景图像
CancelButton	获取或设置当用户按<Esc>键时单击的按钮
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
ContextMenuStrip	获取或设置与此控件关联的 ContextMenuStrip
Cursor	获取或设置当鼠标指针位于控件上时显示的光标
Controls	获取包含在控件内的控件的集合
FormBorderStyle	获取或设置窗体的边框样式
HelpButton	获取或设置一个值，该值指示是否应在窗体的标题框中显示“帮助”按钮
Icon	获取或设置窗体的图标
IsMdiChild	获取一个值，该值指示窗体是否为多文档界面（MDI）子窗体
IsMdiContainer	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
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
MinimumSize	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
Opacity	获取或设置窗体的不透明度级别
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
StartPosition	获取或设置运行时窗体的起始位置
TopMost	获取或设置一个值，指示该窗体是否应显示为最顶层窗体
Text	获取或设置窗体显示的标题
TransparencyKey	获取或设置将表示窗体透明区域的颜色
WindowState	获取或设置窗体的窗口状态

下面对重要属性进行详细介绍。

（1）Name 属性，用于获取或设置控件的名称，在后台编写代码时会用到该名称。

语法：

```
public string Name { get; set; }
```

属性值：控件的名称。

例如，在制作应用程序时，为了便于窗体的查找或使用，一般都按指定的名称命名窗体，如果在创建时没有对窗体进行命名，可以在窗体创建后通过属性窗口中的 Name 属性对窗体进行重命名。

（2）AutoScroll 属性，用于获取或设置一个值，该值指示窗体是否实现自动滚动。

语法：

```
public override bool AutoScroll { get; set; }
```

属性值：若要在窗体上启用自动滚动，该属性设置为 true；否则，该属性设置为 false。默认为 false。

AutoScroll 属性的应用有两种方式：一是直接将该属性设置为 true，当窗体中的控件超出窗体的工作区域时，会自动出现滚动条；二是与 AutoScrollMinSize 属性一起使用，当窗体的工作区域小于 AutoScrollMinSize 属性所设置的区域时，会自动出现滚动条。

例如，当程序运行时，在空窗体（窗体宽度为 300，窗体高度为 300）中显示滚动条。代码如下：

```
public Frm_Main()                                          //窗体的重载
{
    InitializeComponent();
    AutoScroll = true;                                     //实现自动滚动
    AutoScrollMinSize = new Size(300, 300);                //设置滚动区域的最小值
}
```

运行结果如图 8.5 所示。

图 8.5　在空窗体中显示滚动条

说明：在显示窗体滚动条时，不一定非要在窗体显示时显示滚动条，也可以根据需要在适当的时候显示滚动条，例如，在窗体中绘制图像时，如果图像的大小大于窗体大小时显示滚动条。

（3）StartPosition 属性，用于获取或设置运行时窗体的起始位置。

语法：

```
public FormStartPosition StartPosition { get; set; }
```

属性值：FormStartPosition，表示窗体的起始位置。FormStartPosition 枚举成员及说明如表 8.2 所示。

表 8.2　　FormStartPosition 枚举成员及说明

成　员	说　明
Manual	窗体的位置由 Location 属性确定
CenterScreen	窗体在当前显示窗口中居中，其尺寸在窗体大小中指定
WindowsDefaultLocation	窗体定位在 Windows 默认位置，其尺寸在窗体大小中指定
WindowsDefaultBounds	窗体定位在 Windows 默认位置，其边界也由 Windows 默认决定
CenterParent	窗体在其父窗体中居中

例如，当窗体显示时，使窗体居中显示。代码如下：

```
public Frm_Main()                                          //窗体的重载
{
    InitializeComponent();
    this.StartPosition = FormStartPosition.CenterScreen;   //使窗体居中显示
}
```

说明：以上加粗的代码也可以放在窗体的 Load（加载）事件中，实现窗体在显示时的居中效果。

（4）FormBorderStyle 属性，用于获取或设置在窗体标题栏中显示的图标。

语法：

```
public FormBorderStyle FormBorderStyle { get; set; }
```

属性值：FormBorderStyle 枚举，表示要为窗体显示的边框样式，默认为 FormBorderStyle.Sizable。FormBorderStyle 枚举成员及说明如表 8.3 所示。

表 8.3　　FormBorderStyle 枚举成员及说明

成　员	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
SizableToolWindow	可调整大小的工具窗口边框

例如，设置窗体为无边框窗体。代码如下：

```
this.FormBorderStyle = FormBorderStyle.None;
```

说明：如果想在程序运行时对窗体进行无边框样式的设置，可以将以上代码写在窗体的重载事件中，也可以将其写在 Load 或 Show 事件中。在这里要说明的是，将窗体设为无边框窗体时，不能设置窗体的大小、不能移动窗体，也不能关闭当前窗体，如果想关闭当前窗体，必须通过控件执行窗体的 Close 方法。

（5）WindowState 属性，用于获取或设置窗体的窗口状态，也就是窗体的最大化、最小化或还原状态。

语法：

```
public FormWindowState WindowState { get; set; }
```

属性值：FormWindowState 枚举，表示窗体的窗口状态，默认为 FormWindowState.Normal。FormWindowState 枚举成员及说明如表 8.4 所示。

表 8.4　　FormWindowState 枚举成员及说明

成　员	说　明
Normal	默认大小的窗口
Minimized	最小化的窗口
Maximized	最大化的窗口

例如，通过按钮设置窗体为最大化。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Maximized;
}
```

（6）Controls 属性，获取 ControlCollection 对象，该对象表示 UI 层次结构中指定服务器控件的子控件。

语法：

```
[BrowsableAttribute(false)]
public Control.ControlCollection Controls { get; }
```

属性值：一个 Control.ControlCollection，它表示控件内包含的控件的集合。

例如，获取当前窗体中按键类型为 Button 的所有子控件名称。代码如下：

```
foreach (Control c in this.Controls)                    //遍历所有子控件
{
    if (c.GetType().Name == "Button")                   //如果当前控件为 Button 类型
```

```
        {
            MessageBox.Show((Button)c).Text;                    //显示 Button 控件的 Text 属性值
        }
    }
```

（7）TransparencyKey 属性，用于获取或设置将表示窗体透明区域的颜色。

语法：

```
public Color TransparencyKey { get; set; }
```

属性值：Color，表示要在窗体上透明显示的颜色。

如果直接将 TransparencyKey 属性设置为 Color.Transparent，将以白色作为透明色，如果将窗体的工作区域设为透明，将 BackColor 和 TransparencyKey 属性设为同一种颜色。

说明：在用图片填充窗体时，将图片透明部分的颜色设置为与 TransparencyKey 属性相同的颜色。

注意：如果当前工程中没有“属性窗口”，可以在窗体的工作区域上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令项，弹出“属性窗口”。也可以从“视图”菜单中选择“属性窗口”将其打开。

2．方法

Windows 窗体的常用方法及说明如表 8.5 所示。

表 8.5　　Windows 窗体的常用方法及说明

方　法	说　明
Activate	激活窗体并给予它焦点
ActivateMdiChild	激活窗体的 MDI 子级
AddOwnedForm	向此窗体添加附属窗体
CenterToScreen	使窗体在当前屏幕上居中
Close	关闭窗体
CreateGraphics	为控件创建 Graphics
Hide	对用户隐藏控件
Invalidate	使控件的整个图面无效并导致重绘控件
Refresh	强制控件使其工作区无效并立即重绘自己和任何子控件
Show	显示窗体
ShowDialog	将窗体显示为模式对话框
ScaleCore	执行窗体缩放操作
Update	使控件重绘其工作区内的无效区域

下面对重要方法进行详细介绍。

（1）Close 方法，用于关闭窗体。

语法：

```
public void Close()
```

属性值：控件的名称。

例如，用指定按钮关闭当前窗体。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.Close();
}
```

（2）CreateGraphics 方法，用于为控件创建 Graphics。

语法：

```
public Graphics CreateGraphics()
```

属性值：控件的 Graphics。

例如，通过按钮在窗体的工作区域中绘制矩形。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = this.CreateGraphics();                                    //创建窗体的 Graphics
    g.DrawRectangle(new Pen(new SolidBrush(Color.Red), 1), 0, 0, 20, 20);  //在窗体上绘制矩形
}
```

（3）Invalidate 方法，使控件的整个图面无效并导致重绘控件。

语法：

```
public void Invalidate()
```

Invalidate 方法并不强制同步绘制；若要强制同步绘制，请在调用 Invalidate 方法之后调用 Update 方法。

有两种重新绘制窗体及其内容的方法：

① 可以将 Invalidate 方法的重载之一与 Update 方法一起使用。

② 可以调用 Refresh 方法，此方法强制控件重新绘制其自身及其所有子级。这等效于调用 Invalidate 方法后，再调用 Update 方法。

Invalidate 方法控制绘制或重新绘制的内容。Update 方法控制发生绘制或重新绘制的时间。如果将 Invalidate 和 Update 方法一起使用，而不是调用 Refresh，则重新绘制的内容取决于使用的 Invalidate 的重载。Update 方法仅仅是强制立即绘制控件，而 Invalidate 方法则控制当调用 Update 方法时所绘制的内容。

交叉链接：关于窗体的 Show 和 ShowDialog 方法的介绍可参见本书 8.1.3 小节。

3．事件

Windows 窗体的常用事件及说明如表 8.6 所示。

表 8.6　　Windows 窗体的常用事件及说明

事　件	说　明
Activated	当使用代码激活或用户激活窗体时发生
Load	在第一次显示窗体前发生
Paint	在重绘控件时发生
Resize	在调整控件大小时发生
Shown	只要窗体是首次显示就发生
Closed	在关闭窗体时发生
Closing	在关闭窗体时发生
KeyDown	在控件有焦点的情况下按下键时发生
KeyPress	在控件有焦点的情况下按下键时发生
KeyUp	在控件有焦点的情况下释放键时发生
MouseDown	当鼠标指针位于控件上并按下鼠标键时发生
MouseMove	在鼠标指针移到控件上时发生
MouseUp	在鼠标指针在控件上并释放鼠标键时发生
Scroll	用户或代码滚动工作区时发生

下面对重要事件进行详细介绍。

（1）Load 事件，在第一次显示窗体前发生。

语法：

```
public event EventHandler Load
```

可以使用此事件执行一些任务，如分配窗体使用的资源，在当前窗体显示前调用其他窗体，连接数据库等。

例如，在开发工程时，都要制作一个登录窗体，而登录窗体的调用一般都是在主窗体的加

载事件中完成的。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    Form2 Frm_logon = new Form2();                                  //实例化 Form2 窗体
    Frm_logon.StartPosition = FormStartPosition.CenterScreen;       //设置窗体局中
    Frm_logon.ShowDialog();                                         //显示当前窗体
    Frm_logon.Dispose();                                            //释放当前窗体
}
```

这里要说明的是，调用登录窗体也可以用 Shown 事件完成，只是用 Shown 事件调用登录窗体，是在主窗体显示后调用的，而 Load 事件是在主窗体显示前调用的。在一般情况下，Shown 事件用于执行连接数据库的操作。

（2）Paint 事件，在重绘控件时发生。

语法：

```
public event PaintEventHandler Paint
```

该事件将 PaintEventArgs 的实例传递给用来处理 Paint 事件的方法。Paint 事件主要用于在控件上进行绘制。

示例 08-01　用鼠标在窗体上绘制矩形

通过 MouseDown、MouseUp 和 Paint 事件在窗体上用鼠标绘制一个指定大小的矩形。程序代码如下：

代码 08-01　光盘位置：光盘\mingrisoft\sl\08\08_01

```
int PaintTop = 0;                                                   //记录矩形的 x 坐标值
int PaintLeft = 0;                                                  //记录矩形的 y 坐标值
int PaintWidth = 0;                                                 //记录矩形的宽度
int PaintHeight = 0;                                                //记录矩形的高度
bool isProtract = false;                                            //是否绘制矩形
private void Form1_MouseDown(object sender, MouseEventArgs e)       //鼠标按下事件
{
    PaintTop = e.X;                                                 //记录矩形的左上角坐标
    PaintLeft = e.Y;
    isProtract = false;                                             //不绘制
}
private void Form1_MouseUp(object sender, MouseEventArgs e)         //鼠标松开事件
{
    PaintWidth = e.X - PaintTop;                                    //记录矩形的大小
    PaintHeight = e.Y - PaintLeft;
    if (PaintWidth < 0)                                             //如果向上绘制矩形
    {
        PaintWidth = Math.Abs(PaintWidth);                          //获取矩形的宽度
        PaintTop -= PaintWidth;                                     //设置矩形的左上角的 x 坐标
    }
    if (PaintHeight < 0)                                            //如果向左绘制矩形
    {
        PaintHeight = Math.Abs(PaintHeight);                        //获取矩形的高度
        PaintLeft -= PaintHeight;                                   //设置矩形的左上角的 y 坐标
    }
    isProtract = true;                                              //绘制
    this.Invalidate();                                              //控件重绘
}
private void Form1_Paint(object sender, PaintEventArgs e)           //在窗体上绘制
{
    if (isProtract)                                                 //如果绘制
    {
        e.Graphics.DrawRectangle(new Pen(new SolidBrush(Color.Red), 1), new Rectangle(PaintTop, PaintLeft,
PaintWidth, PaintHeight));                                          //绘制矩形
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 8.6 所示。

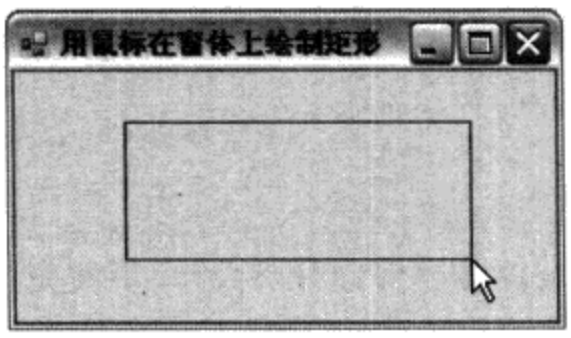


图 8.6　用鼠标在窗体上绘制矩形

（3）Scroll 事件，用户移动滚动框时发生。

语法：

```
public event ScrollEventHandler Scroll
```

当用户移动滚动框时触发 Scroll 事件，可以在该事件中使用 ScrollOrientation 属性来确定 Scroll 事件的滚动条方向。

ScrollOrientation 属性获取引发 Scroll 事件的滚动条方向。其属性值为 ScrollOrientation 枚举，该枚举成员及说明如表 8.7 所示。

表 8.7　ScrollOrientation 枚举成员及说明

成　员	说　明
HorizontalScroll	水平滚动条
VerticalScroll	垂直滚动条

示例 08-02　自动生成滚动条

通过 AutoScroll 和 AutoScrollMinSize 在窗体上自动生成滚动条，并在 Scroll 事件中获取水平滚动条和垂直滚动条的值，使窗体上绘制的文字随滚动条的移动而移动。程序代码如下：

代码 08-02　光盘位置：光盘\mingrisoft\sl\08\08_02

```
bool isPaint = false;                                                    //是否绘制
float x = 0F;                                                            //记录左上角的 x 坐标
float y = 0F;                                                            //记录左上角的 y 坐标
private void button1_Click(object sender, EventArgs e)                   //button1 按钮的单击事件
{
    isPaint = true;                                                      //绘制
    AutoScroll = true;                                                   //窗体出滚动条
    AutoScrollMinSize = new Size(300, 300);                              //设置自动滚动的最小尺寸
    button1.Left = 220;                                                  //设置按钮的位置
    button1.Top = 250;
    x = 10 - 300;                                          //设置出现滚动条后绘制文字的起始位置
    y = 10 - 300;
    this.Invalidate();                                                   //控件重绘
}
private void Form1_Paint(object sender, PaintEventArgs e)                //窗体的绘制事件
{
    if (isPaint)
    {
        e.Graphics.DrawString("图", new Font("宋体", 100), new SolidBrush(Color.Black), new PointF(10 - x, 10 –
y));                                                                     //绘制文字
    }
}
private void Form1_Scroll(object sender, ScrollEventArgs e)              //移动滚动框事件
{
    if (e.ScrollOrientation == ScrollOrientation.HorizontalScroll)       //如果当前滚动的是水平滚动条
    {
```

```
        x = e.NewValue;                                          //获取水平滚动条的值
    }
    if (e.ScrollOrientation == ScrollOrientation.VerticalScroll)  //如果当前滚动的是垂直滚动条
    {
        y = e.NewValue;                                          //获取垂直滚动条的值
    }
    this.Invalidate();                                           //控件重绘
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 8.7 和图 8.8 所示。

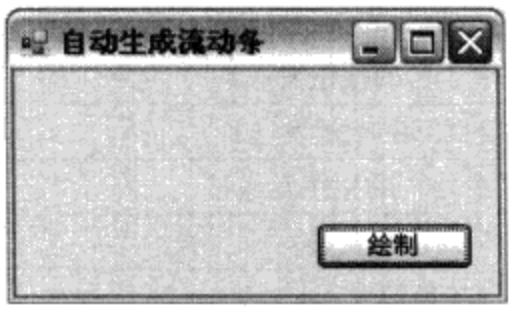


图 8.7　单击“绘制”按钮前

图 8.8　单击“绘制”按钮后

8.1.3　调用窗体

根据 Windows 窗体的显示状态，可将其分为模式窗体和无模式窗体。模式窗体和无模式窗体只有在显示的时候才存在差别。当使用 ShowDialog 方法显示窗体时，被显示的窗体称为模式窗体；当使用 Show 方法显示窗体时，被显示的窗体称为无模式窗体。如果多个窗体以无模式状态显示，则单击任何一个窗体，它将立即成为当前窗体并且显示于屏幕的最前面；否则，如果显示的是模式窗体，则该窗体为当前窗体，同时，其他窗体不可用，只有当模式窗体关闭时，其他窗体才能恢复到原来的可用状态。

1．模式窗体和无模式窗体的调用

（1）模式窗体的调用。

模式窗体的调用是通过 ShowDialog 方法实现的。代码如下：

```
Form2 Frm_2 = new Form2();
Frm_2.ShowDialog();                                          //调用模式窗体
Frm_2.Dispose();                                             //释放当前窗体的资源
```

在调用窗体时，也可以在调用窗体前对窗体的样式进行设置。

例如，在调用窗体前设置窗体的位置、标题栏文本等。代码如下：

```
Form2 Frm_2 = new Form2();
Frm_2.StartPosition = FormStartPosition.CenterScreen;        //设置窗体居中
Frm_2.Text = "调用模式窗体";                                  //设置窗体的标题栏文本
Frm_2.ShowDialog();
Frm_2.Dispose();
```

注意：在关闭调用窗体后，应该用 Dispose 方法释放调用窗体的所有资源，以节省内存。

（2）非模式窗体的调用。

模式窗体的调用是通过 ShowDialog 方法实现的。代码如下：

```
Form2 Frm_2 = new Form2();
Frm_2.Show();                                                //调用非模式窗体
Frm_2.Dispose();                                             //释放当前窗体的资源
```

在调用非模式窗体前，也可以对窗体进行相应的设置。

2．设置和应用窗体的返回值

在关闭窗体前可以根据窗体的操作情况设置窗体的 DialogResult 属性，然后通过 ShowDialog 方法的返回值执行相应的操作。

DialogResult 属性用于获取或设置一个值，该值在单击按钮时返回到父窗体。

语法：

```
public virtual DialogResult DialogResult { get; set; }
```

返回值：DialogResult 值之一。默认值为 None。DialogResult 枚举成员及说明如表 8.8 所示。

表 8.8　　DialogResult 枚举成员及说明

成　员	说　明
None	对话框的返回值是 Nothing，这表明有模式对话框继续运行
OK	对话框的返回值是 OK（通常从标签为“确定”的按钮发送）
Cancel	对话框的返回值是 Cancel（通常从标签为“取消”的按钮发送）
Abort	对话框的返回值是 Abort（通常从标签为“中止”的按钮发送）
Retry	对话框的返回值是 Retry（通常从标签为“重试”的按钮发送）
Ignore	对话框的返回值是 Ignore（通常从标签为“忽略”的按钮发送）
Yes	对话框的返回值是 Yes（通常从标签为“是”的按钮发送）
No	对话框的返回值是 No（通常从标签为“否”的按钮发送）

示例 08-03　窗体返回值的应用

本示例通过窗体的 DialogResult 属性设置窗体的返回值，并用调用窗体的 ShowDialog 方法的返回值判断窗体操作情况。程序代码如下：

代码 08-03　光盘位置：光盘\mingrisoft\sl\08\08_03

用主窗体的加载事件调用子窗体。代码如下：

```
private void Frm_Main_Load(object sender, EventArgs e)
{
    FormSon Frm_Son = new FormSon();                          //实例化子窗体
    Frm_Son.StartPosition = FormStartPosition.CenterScreen;   //使子窗体居中显示
    Frm_Son.Text = "子窗体";                                  //设置子窗体的标题栏文本
    DialogResult dial = Frm_Son.ShowDialog();                 //显示子窗体并获取窗体的返回值
    if (dial == DialogResult.OK)                              //如果返回的是 OK
    {
        MessageBox.Show("单击了【确定】按钮");                 //在子窗体中单击的【确定】按钮
    }
    else if (dial == DialogResult.No)                         //如果返回的是 No
    {
        MessageBox.Show("单击了【取消】按钮");                 //在子窗体中单击的【取消】按钮
    }
    else if (dial == DialogResult.Cancel)                     //如果返回的是 Cancel
    {
        MessageBox.Show("直接关闭了子窗体");                   //用标题栏右上角的"关闭"按钮直接关闭了子窗体
    }
}
```

设置在子窗体中单击【确定】按钮时窗体的返回值。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.OK;                      //设置窗体的返回值
    this.Close();                                             //关闭窗体
}
```

设置在子窗体中单击【取消】按钮时窗体的返回值。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.No;                      //设置窗体的返回值
    this.Close();                                             //关闭窗体
}
```

说明：在关闭模式窗体时，其窗体的默认返回值为 Cancel。

注意：非模式窗体 Show 没有返回值。

用户在制作应用程序时窗体返回值的应用是十分广泛的，比如在添加和更改窗体中对数据进行修改，主窗体可以根据窗体返回值对主窗体中相应数据进行刷新。

8.1.4 为窗体添加控件

控件是包含在窗体中的对象，如 Button 按钮、TextBox 文本框等都是控件。

为窗体添加控件有两种方法，第一种方法是先向窗体中添加若干个控件，然后再分别调整它们的大小和位置；第二种方法是在向窗体中添加控件时，同时调整其大小和位置。

1．第一种方法

(1) 在“视图”菜单中，选择“工具箱”选项，打开“工具箱”窗口。

(2)“工具箱”窗口中，双击要添加的控件，即可在窗体的左上角放置一个具有默认大小的控件。按这种方法，在窗体中添加多个控件。

(3) 在窗体中，选中控件，将其拖放到适当的位置，以重新调整选中控件的位置。

(4) 在窗体中，选中控件，拖动控件的 8 个尺寸柄之一，可以调整选中控件的大小。

为窗体添加控件的方式如图 8.9 所示。

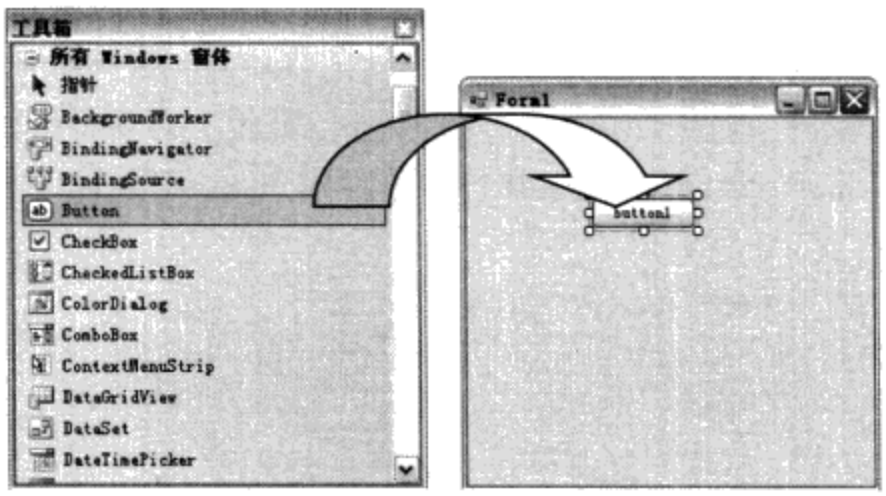


图 8.9 为窗体添加控件

2．第二种方法

(1) 在“视图”菜单中，选择“工具箱”选项，打开“工具箱”窗口。

(2)“工具箱”窗口中，选中想要添加的控件，在窗体上面移动鼠标指针，指针变为十字形。

(3) 将十字形光标放在窗体中适当的位置，然后按住鼠标左键，拖动到适当的大小后，释放鼠标，即可将选择的控件按指定大小添加到窗体中的指定位置。

8.2 创建继承窗体

在特殊情况下，程序可能需要一个与以前程序中类似的窗体，或者用户可能希望创建一个基窗体，这时可以使用 Windows 继承窗体。通过继承基窗体来创建 Windows 窗体是利用已有工作成果的一种高效方法，本节将详细介绍如何创建继承窗体。

8.2.1 继承窗体概述

继承窗体，是指根据现有窗体的结构创建一个与其一样的新窗体，这种从现有窗体继承的过程称为可视化继承。可视化继承给开发人员带来了许多好处，例如，如果当前项目中所需的窗体与一个已经设计好的窗体类似的话，就可以继承先前的窗体。

继承窗体不仅可以从基窗体上继承控件和属性，还可以继承代码，这意味着可以创建公共代码来增强代码的重用性。使用继承的优点是：当代码不适用于某种特定情况时，可以重写代码。但要注意的是，要想在继承窗体中重写基窗体的代码，则要重写的代码段在基窗体中必须

被定义为 public 或 protected 类型。另外，如果是想在继承窗体中编辑基窗体的控件，则该控件的 Modifiers 属性不能设置为 private。

8.2.2 创建继承窗体

开发人员如果要创建继承窗体，首先要制作一个基本窗体，其中含有用户所需窗体的基本设置与布局。然后，在指定的项目中创建一个继承窗体，该继承窗体保留了基本窗体的原始设置，也可以对基本窗体中控件（窗体）的属性、事件进行重新设置。

1．制作基本窗体

制作基本窗体与制作 Windows 普通窗体的步骤基本相同，只是为了便于在继承窗体中修改基本窗体中控件（窗体）的属性和事件，要将控件的 Modifiers 属性设置为 public 或 protected。下面介绍基本窗体的设置步骤。

（1）新建一个 Windows 应用程序，将其命名为 InheritStencil。

（2）在“解决方案资源管理器”的 InheritStencil 项目中将 Form1 窗体名改为 FrmEmployeeInfoAdd，以便于设置多个继承窗体。

（3）在窗体中添加相应的控件，如果想在继承窗体中对控件进行设置，需要将控件的 Modifiers 属性设置为 public 或 protected。

（4）如果想在继承窗体中对控件或窗体的事件进行重载，必需将事件中的相关代码封装成方法，其自定义方法必需用 virtual 关键字修饰。下面以 button1 控件的单击事件为例，对事件的设置进行说明。

定义一个自定义方法 button1Click。代码如下：

```
public virtual void button1Click()
{
}
```

说明：也可以设置一个具有返回值或参数的自定义方法。

在 button1 控件的单击事件中调用自定义方法 button1Click。代码如下：

```
public void button1_Click(object sender, EventArgs e)
{
    button1Click();
}
```

（5）由于在创建继承窗体时选择的文件是 DLL 文件，所以，要将基本窗体所在的工程生成为 DLL 文件。首先在“解决方案资源管理器”的 InheritStencil 项目中单击鼠标右键，在快捷菜单中选择“属性”，即可在“输出类型”中选择“类库”，最后在“生成”菜单中选择“生成解决方案”选项或按<F6>键，即可在当前应用程序的可执行文件路径下生成 InheritStencil.dll 文件。

2．创建继承窗体

在程序中创建继承窗体的方法有两种，一是通过窗体设计器实现，二是通过编程方式实现，下面具体介绍这两种方法。

（1）通过窗体设计器实现。

通过窗体设计器创建继承窗体的步骤如下。

① 新建一个 Windows 应用程序，将其命名为 WinInherit。

② 在“解决方案资源管理器”中的 WinInherit 项目上单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建项”选项，如图 8.10 所示。

③ 弹出如图 8.11 所示的“添加新项”对话框，在该对话框中，选择“继承的窗体”，并在“名称”文本框中给要添加的继承窗体命名。

④ 单击【添加】按钮，弹出如图 8.12 所示的“继承选择器”对话框。在该对话框中，选择要继承的窗体名称，或者单击【浏览】按钮，选择要继承的组件（InheritStencil 工程所生成

的 InheritStencil.dll 文件），然后单击【确定】按钮，即可在现有项目中添加一个 Windows 继承窗体。

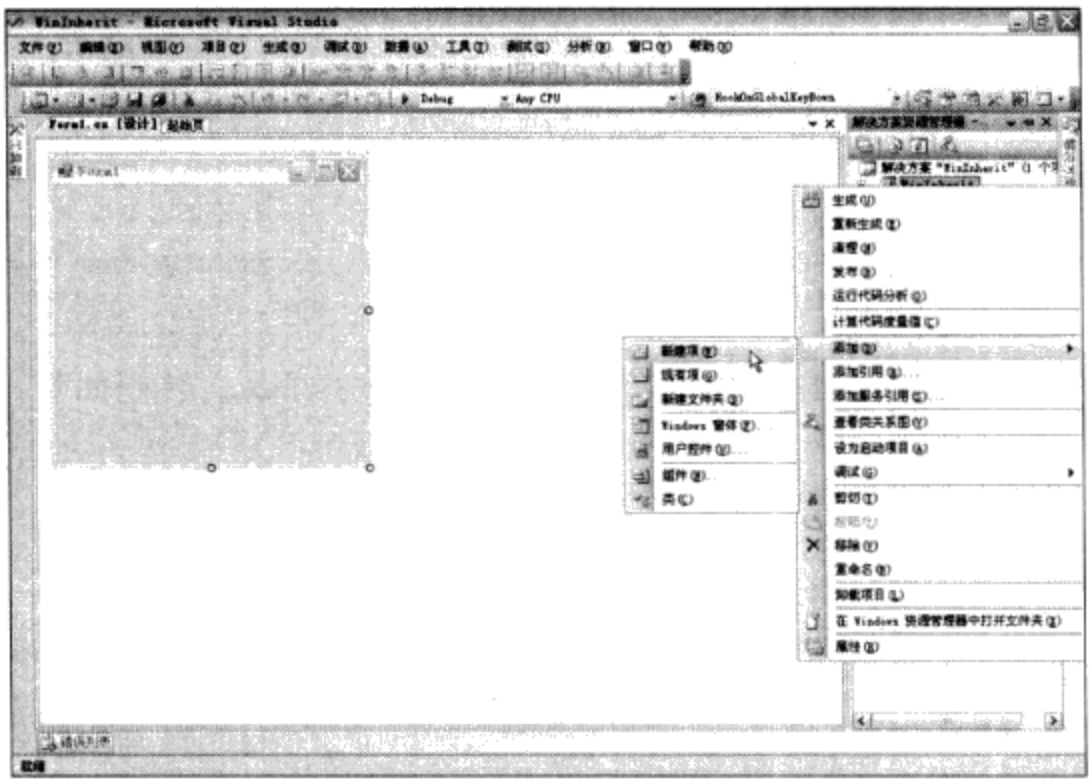


图 8.10　选择“添加”/“新建项”选项

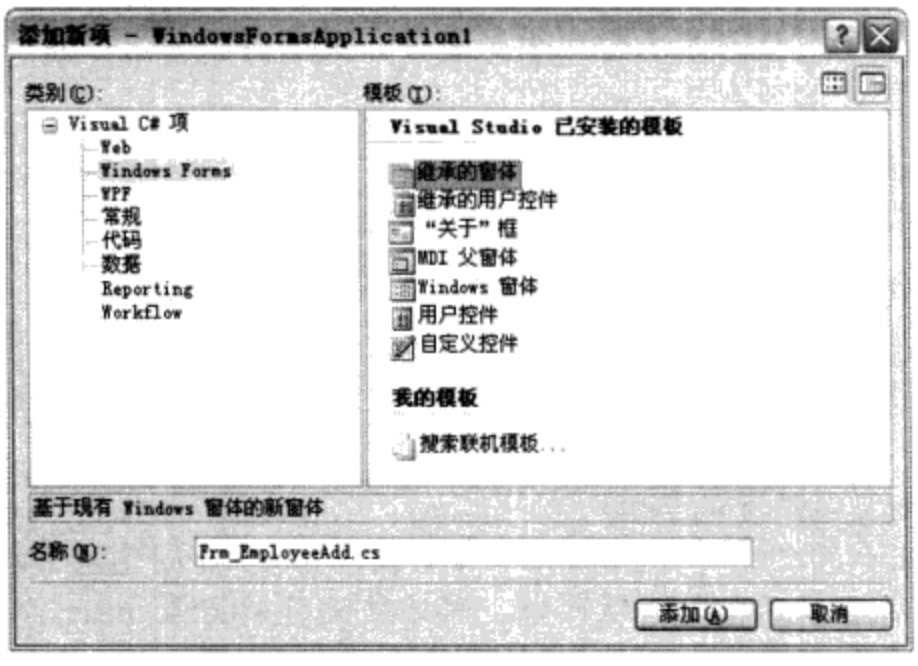


图 8.11　“添加新项”对话框

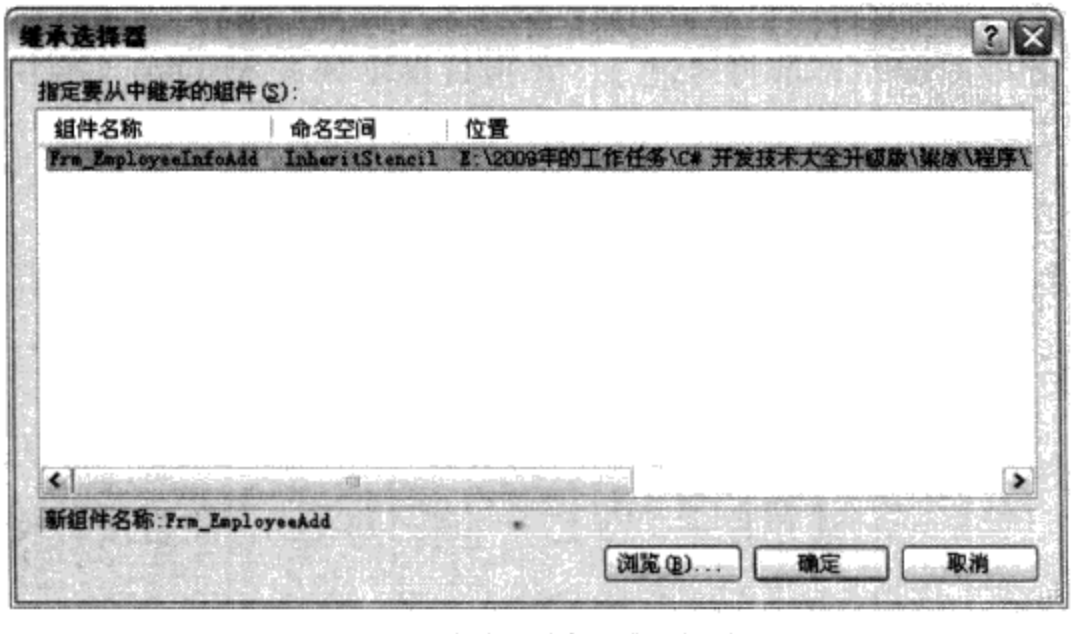


图 8.12　“继承选择器”对话框

⑤ 如果想要在继承窗体中修改控件（窗体）的相关事件，必须将要修改的事件进行重载，然后用 override 关键字修饰基本窗体中相应事件所调用的自定义方法。下面以 button1 控件的单击事件为例，对其进行详细说明。

在继承窗体的窗体构造函数中重载 button1 控件的单击事件。代码如下：

```
public Frm_EmployeeAdd()
{
    InitializeComponent();
    //实例化 InheritStencil.Frm_EmployeeInfoAdd 窗本
    InheritStencil.Frm_EmployeeInfoAdd F_SuccessiveAdd = new InheritStencil.Frm_EmployeeInfoAdd();
    F_SuccessiveAdd. Button1.Click += Frm_EmployeeAdd_button1_Click;            //事件重载
}
```

用 override 关键字重载基本窗体中的自定义方法 button1Click。代码如下：

```
public override void button1Click()
{
    MessageBox.Show("添加成功");
}
```

在重载的单击事件中调用自定义方法 button1Click。代码如下：

```
private void Frm_EmployeeAdd_button1_Click(object sender, EventArgs e)
{
    base. button1Click();                                                  //设用重载方法
}
```

注意：在继承窗体中修改基本窗体的事件时，必须在继承窗体中对相应的事件进行重载。如果在继承窗体中用“属性窗口”生成相应的事件，然后在事件中调用相应的重载方法，运行程序后执行事件，将先执行基本窗体中的事件，然后再执行继承窗体中的事件。

（2）通过编程方式实现。

通过编程方式创建继承窗体的步骤如下。

①新建一个 Windows 应用程序。

②再在该 Windows 应用程序中添加一个窗体，命名为 Form2。

③双击 Form2 窗体，进入该窗体的代码视图中，在其类定义中，添加对所继承窗体 Form1 的引用，该引用的写法包含 Form1 窗体的命名空间，后面跟一个句点及 Form1 窗体的名称。具体代码如下：

```
public partial class Form2 : namespace.Form1
```

注意：如果是在同一个项目中创建继承窗体，可以不写“namespace”命名空间；如果继承的窗体与要创建的窗体不在同一项目中，则必须使用“namespace”命名空间引用。

8.2.3 继承窗体的应用

在开发应用程序时可以将一些常用的窗体设置为可继承的窗体，这样可以节省窗体设置和常用代码编写所用的时间。本节将通过一个示例说明如何在 Windows 应用程序中使用继承窗体。

示例 08-04 将员工信息的添加窗体设为继承窗体

将员工信息的添加窗体设置为继承窗体，可以在其他工程序继承该窗体，以便于快速完成添加窗体的制作。程序代码如下：

代码 08-04 光盘位置：光盘\mingrisoft\sl\08\08_04

（1）制作可继承的基本窗体，开发步骤如下。

① 新建一个 Windows 应用程序，命名为 InheritStencil，将窗体名改为 FrmEmployee InfoAdd.cs。

② 根据员工的基本信息设置 FrmEmployeeInfoAdd 窗体，如图 8.13 所示。

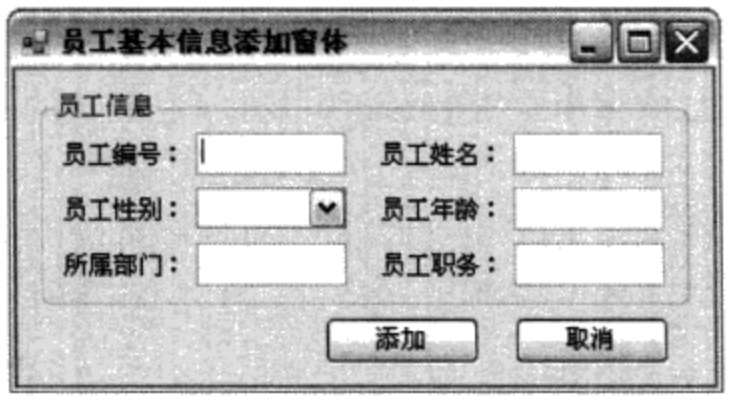


图 8.13　添加窗体的基本窗体

注意：在向窗体中添加控件时，如果想在继承窗体中对控件进行编辑，必须将 Modifiers 属性设置为 public，如果该属性的值为 private，则不可以在继承窗体中对控件进行编辑。

③程序主要代码。

在 FrmEmployeeInfoAdd 窗体中定义 virtual 类型的无返回值方法 TextBAgeKeyPress ，该方法用于限制在文本框中输入的字符，也就是只能在文本框中输入不超过两位的数字字符。参数 sender 和 e 继承于 KeyPress 事件参数，主要用于获取事件的基本数据。代码如下：

```
public virtual void TextBAgeKeyPress(object sender, KeyPressEventArgs e)
{
    if (!((e.KeyChar >= '0' && e.KeyChar <= '9') || e.KeyChar == '\b'))//如果键值不是数字键或左删除键
    {
        e.Handled = true;                                   //不处理当前事件
    }
    else
    {
        if (((TextBox)sender).Text.Length >= 2)             //如果文本框中的字符串长度大于等于 2
            if (e.KeyChar != '\b')                          //如果当前键值是左删除键
                e.Handled = true;                           //不处理当前事件
         //如果在文本框中选定了文本
        if (((TextBox)sender).SelectedText.Length <= 2 && ((TextBox)sender).SelectedText.Length >= 1)
            e.Handled = false;                              //处理当前事件
    }
}
```

在 FrmEmployeeInfoAdd 窗体中定义 virtual 类型的有返回值的 GetTextBoxNoNull 方法，该方法主要用于遍历指定容器中的控件是否有空文本。如果有则返回 false，否则返回 true。参数 Con 表示指定容器的对象集合。代码如下：

```
public virtual bool GetTextBoxNoNull(Control.ControlCollection Con)
{
    bool tem_bool = true;                                   //判断是否有空文本
    foreach (Control C in Con)                              //遍历指定容器中的控件
    {
        if (C.GetType().Name == "TextBox")                  //如果当前控件是 TextBox
        {
            if (((TextBox)C).Text.Length == 0)              //如果当前控件为空
            {
                tem_bool = false;                           //有空文本
                break;                                      //退出该方法
            }
        }
        if (C.GetType().Name == "ComboBox")                 //如果当前控件是 ComboBox
        {
            if (((ComboBox)C).Text.Length == 0)             //如果当前控件为空
            {
                tem_bool = false;                           //有空文本
                break;                                      //退出该方法
            }
        }
```

```
        }
        return tem_bool;                                    //返回布尔型值
    }
```

在 FrmEmployeeInfoAdd 窗体中定义 virtual 类型的无返回值的 ButAddClick 方法，该方法主要用于调用自定义方法 GetTextBoxNoNull，并根据返回值进行相应的操作。代码如下：

```
public virtual void ButAddClick()
{
    if (GetTextBoxNoNull(groupBox1.Controls))                //如果容器中的控件没有空文本
    {
        this.DialogResult = DialogResult.OK;                 //设置当前窗体的返回值
    }
    else
    {
        MessageBox.Show("请将员工的基本信息填写完整。");      //提示信息
    }
}
```

TextB_Age（“年龄”文本框）控件的 KeyPress 事件主要用于调用 TextBAgeKeyPress 方法限制文本框只能输入数字字符。代码如下：

```
public void TextB_Age_KeyPress(object sender, KeyPressEventArgs e)
{
    TextBAgeKeyPress(sender, e);                             //调用自定义方法
}
```

But_Add（“添加”按钮）控件的 Click 事件主要用于调用 ButAddClick 方法判断添加的员工信息是否填写完整。代码如下：

```
public void But_Add_Click(object sender, EventArgs e)
{
    ButAddClick();                                           //调用自定义方法
}
```

But_Cancel（“取消”按钮）控件的 Click 事件主要用于退出当前窗体，并设置该窗体的返回值。代码如下：

```
public void But_Cancel_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.No;                     //设置当前窗体的返回值
    this.Close();                                            //关闭当前窗体
}
```

最后在“解决方案资源管理器”的 InheritStencil 项目中单击鼠标右键，在快捷菜单中选择“属性”，在“输出类型”中选择“类库”，然生按<F6>键以 DLL 文件的形式生成解决方案。

（2）创建继承窗体，开发步骤如下。

① 新建一个 Windows 应用程序，命名为 InheritStencil，将窗体名改为 FrmEmployeeInfoAdd.cs。

② 按照 8.2.2 小节中“通过窗体设计器实现”中的 2~4 步骤创建继承窗体。

③ 在 FrmEmployeeInfoAdd 窗体中添加 Button 控件，在该控件的单击事件中调用继承窗体。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Frm_EmployeeAdd F_EmployeeAdd = new Frm_EmployeeAdd();   //实例化 Frm_EmployeeAdd 窗体
    if (F_EmployeeAdd.ShowDialog() == DialogResult.OK)       //显示窗体
    {
        //如果添加成功，可以在这里刷新窗体中的内容
    }
    F_EmployeeAdd.Dispose();                                 //释放所有资源
}
```

④ 在继承窗体中，“添加”按钮的实现过程并不符合当前工程的要求，必须将“添加”按钮的单击事件进行重载，重新对该事件进行设置。修改代码如下：

在继承窗体的构造函数中，重新加载 But_Add 控件的 Click 事件。代码如下：

```
public Frm_EmployeeAdd()
{
    InitializeComponent();
    //实例化 InheritStencil.Frm_EmployeeInfoAdd 窗体，也就是基本窗体
    InheritStencil.Frm_EmployeeInfoAdd F_SuccessiveAdd = new InheritStencil.Frm_EmployeeInfoAdd();
    F_SuccessiveAdd.But_Add.Click += Frm_EmployeeAdd_But_Add;      //事件重载
}
```

自定义重载事件 Frm_EmployeeAdd_But_Add，主要用于重新执行 But_Add 控件的 Click 事件。代码如下：

```
private void Frm_EmployeeAdd_But_Add(object sender, EventArgs e)        //自定义事件
{
    base.ButAddClick();
}
```

自定义方法 ButAddClick 是基本窗体中 ButAddClick 方法的重载方法，主要用于替换基本窗体中 ButAddClick 方法的操作。代码如下：

```
public override void ButAddClick()
{
    if (GetTextBoxNoNull(groupBox1.Controls))                    //如果容器中的控件没有空文本
    {
        MessageBox.Show("添加成功");
        this.DialogResult = DialogResult.OK;                     //设置当前窗体的返回值
    }
    else
    {
        MessageBox.Show("添加失败");                             //提示信息
    }
}
```

按<Ctrl+F5>键运行程序，运行结果如图 8.14 所示。

图 8.14　继承窗体的应用

8.3　创建 MDI 窗体

8.3.1　MDI 窗体概述

MDI（Multiple-Document Interface 多文档界面）窗体由一个父窗体和若干个子窗体组成。MDI 窗体允许同时显示多个子窗体，它常包括含有子菜单的“窗口”菜单项，该菜单项用于在窗口之间进行切换。

1．多文档界面

多文档界面可以同时支持多个文档，它是一种在应用程序中能够同时处理两个或者更多个子窗体的界面形式。图 8.15 所示的企业客户资源管理系统就是一个多文档窗体界面。

2．父窗体和子窗体

MDI 窗体由父窗体和子窗体两部分组成，子窗体被包含在父窗体中，当父窗体最小化时，所

有的子窗体也被最小化，只有父窗体的图标显示在任务栏中。

图 8.15　多文档窗体界面

用户可以通过将窗体的 IsMdiContainer 属性设置为 true 来指定父窗体，而在一个窗体显示之前，正确地设置其 MdiParent 属性即可将其指定为子窗体。

一个父窗体中可以同时显示多个子窗体，需要对显示在父窗体中的多个子窗体进行排列。用户可以通过使用带有 MdiLayout 枚举的 LayoutMdi 方法来排列多文档界面父窗体中的子窗体。

调用 Form 类的 LayoutMdi 方法的语法如下：

```
public void LayoutMdi ( MdiLayout value)
```

value：MdiLayout 枚举值之一，用来定义 MDI 子窗体的布局。MdiLayout 枚举成员及说明如表 8.9 所示。

表 8.9　　MdiLayout 枚举成员

成　员	说　明
ArrangeIcons	所有 MDI 子图标均排列在 MDI 父窗体的工作区内
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

例如，在主窗体中将所子窗体层叠在 MDI 父窗体的工作区内。代码如下：

```
LayoutMdi ( MdiLayout.Cascade);
```

8.3.2　创建 MDI 窗体

创建 MDI 窗体时，有 3 个主要步骤，分别为：创建 MDI 父窗体、创建 MDI 子窗体和从父窗体调用子窗体。

1．创建 MDI 父窗体

创建 MDI 父窗体的步骤如下。

（1）依次选择“开始”/“所有程序”/“Microsoft Visual Studio 2008”/“Microsoft Visual Studio 2008”选项，打开“Visual Studio 2008”开发环境，在“文件”菜单中，选择“新建”/“项目”选项。

（2）在弹出的“新建项目”对话框中，选择新建“Windows 应用程序”选项，然后单击【确定】按钮，新建一个 Windows 应用程序。

（3）选中 Windows 应用程序的默认窗体 Form1，在“属性窗口”中，将其 IsMdiContainer 属性设置为 true，将该窗体指定为子窗口的多文档界面容器，如图 8.16 所示。

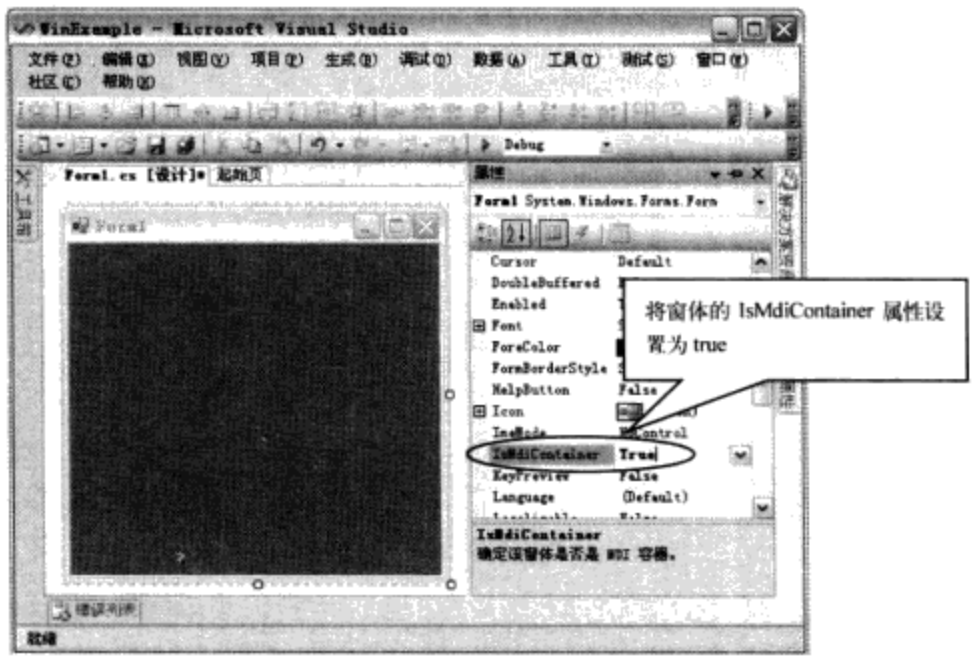


图 8.16　设置窗体的 IsMdiContainer 属性

（4）在 Form1 窗体中添加一个 MenuStrip 控件，用来从父窗体调用子窗体。设置 MenuStrip 控件的一个顶级菜单项和一个子菜单项，如图 8.17 所示。MDI 父窗体的设计即可完成。

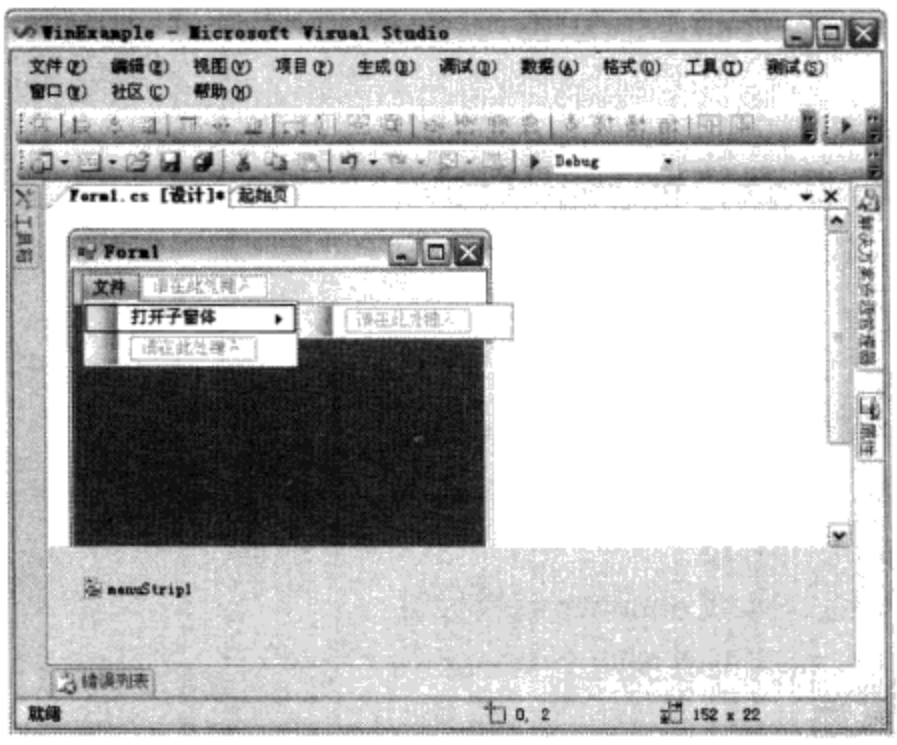


图 8.17　MenuStrip 控件菜单项设置

2．创建 MDI 子窗体

MDI 子窗体对于多文档界面应用程序来说是至关重要的，因为用户是通过子窗体来与应用程序交互的。创建 MDI 子窗体的方法如下。

（1）在“解决方案资源管理器”中选中含有父窗体的 Windows 应用程序，单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建项”或“添加”/“Windows 窗体”选项，弹出如图 8.18 所示的“添加新项”对话框。

（2）在“添加新项”对话框中，选择“Windows 窗体”，并在“名称”文本框中给要添加的 Windows 窗体命名，然后单击【添加】按钮，在当前应用程序中添加一个新的 Windows 窗体。

3．从父窗体调用子窗体

这里以前两节中创建的 MDI 父窗体和 MDI 子窗体为例，说明如何从父窗体调用子窗体。选择 MDI 父窗体中 MenuStrip 控件的“打开子窗体”菜单项，触发其 Click 事件，并在该事件

下添加如下代码：

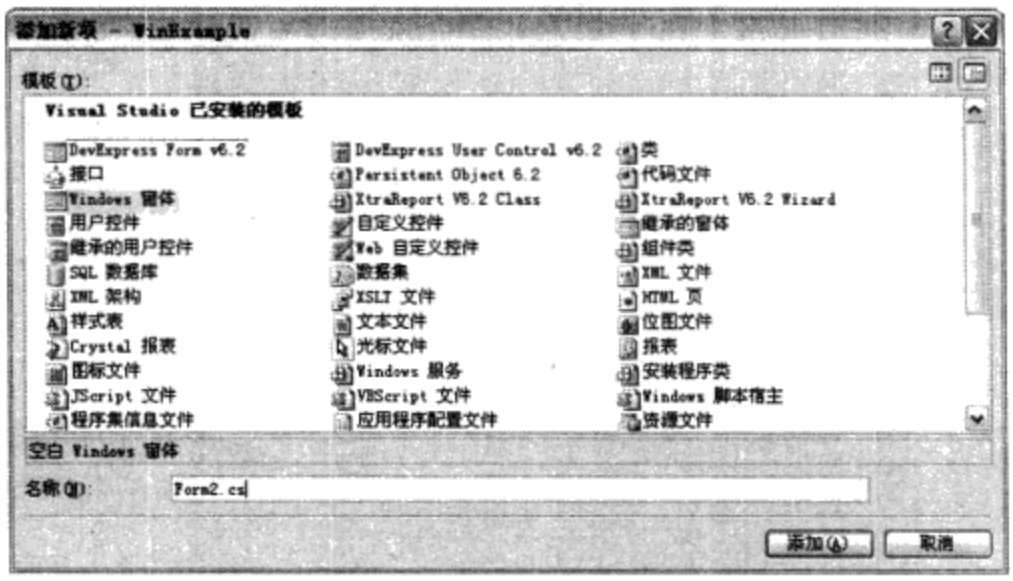


图 8.18 “添加新项”对话框

```
private void 打开子窗体 ToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 frmchild = new Form2();                    //实例化 Form2 窗体
    frmchild.MdiParent = this;                       //设置父窗体
    frmchild.Show();                                 //显示子窗体
}
```

完成上述操作后，运行 Windows 应用程序，当用户单击“打开子窗体”菜单项时，程序就会创建一个新的多文档界面子窗体。

8.3.3 创建 MDI 应用程序

本节通过一个实例介绍创建 MDI 应用程序的具体过程及设置窗体的布局。实例运行结果如图 8.19 所示。

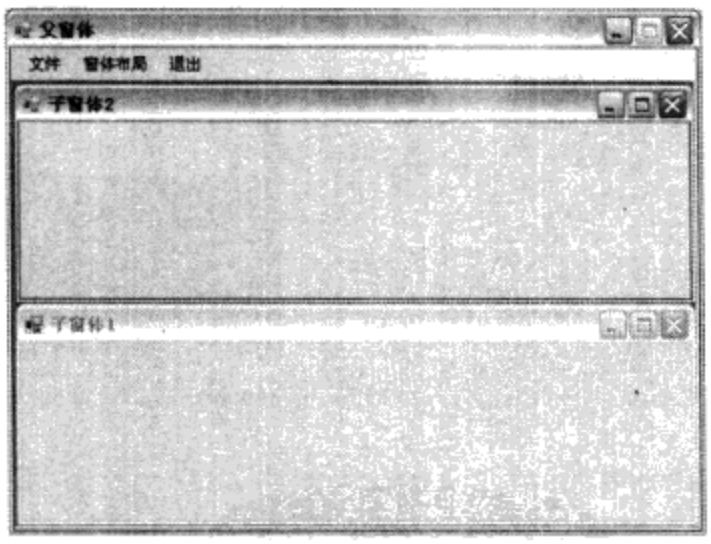


图 8.19 MDI 应用程序运行结果

程序开发步骤如下。

实例位置：光盘\mingrisoft\dxyy\08\08_01

（1）新建一个 Windows 应用程序，命名为 MDIApple，默认窗体为 Form1.cs。

（2）将窗体 Form1 的 IsMdiContainer 属性设置为 true，作为 MDI 父窗体，然后再添加两个 Windows 窗体，作为 MDI 子窗体。

（3）在 Form1 窗体中，添加一个 MenuStrip 控件，作为该父窗体的菜单项。

（4）程序主要代码如下。

在 MDI 父窗体 Form1 中，声明 MDI 子窗体 Form2 和 Form3 的两个实例对象，并给它们赋值为 null，用来判断子窗体是第一次打开，还是已经加载过而被释放掉了。声明 MDI 子窗体 Form2 和 Form3 对象的代码如下：

```
Form2 frmChild1 = null;                              //实例化 Form2 窗体
Form3 frmChild2 = null;                              //实例化 Form3 窗体
```

单击“打开子窗体 1”菜单项时，程序首先判断“子窗体 1”是否打开，如果没有，则实例化一个新的对象，并以当前窗体为父窗体进行显示，否则，判断“子窗体 1”的对象是否释放。如果释放，则重新实例化一个对象；如果没有释放，则获得“子窗体 1”对象的焦点。“打开子窗体 1”菜单项的 Click 事件代码如下：

```
private void 打开子窗体 1ToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
  if (frmChild1 == null)                          //如果当前窗体没有打开
  {
    frmChild1 = new Form2();                      //实例化 Form2
    frmChild1.MdiParent = this;                   //将当前窗体设置为父窗体
    frmChild1.Show();                             //显示窗体
  }
  else
  {
    if (frmChild1.IsDisposed)                     //如果窗体被释放
    {
      frmChild1 = new Form2();                    //实例化 Form2
    }
    frmChild1.MdiParent = this;                   //将当前窗体设置为父窗体
    frmChild1.Show();                             //显示窗体
    frmChild1.Focus();                            //设置输入焦点
  }
}
```

说明："打开子窗体 2"菜单项的 Click 事件与"打开子窗体 1"菜单项的 Click 事件代码实现原理类似，其详细代码请参见本书附带光盘。

单击"水平排列"菜单项时，如果父窗体中有多个子窗体，程序会将这些窗体按照一定的顺序水平排列在父窗体中。"水平排列"菜单项的 Click 事件代码如下：

```
private void 水平排列 ToolStripMenuItem_Click(object sender, EventArgs e)
{
  LayoutMdi(MdiLayout.TileHorizontal);
}
```

说明："垂直排列"菜单项的 Click 事件与"水平排列"菜单项的 Click 事件代码实现原理类似，其详细代码请参见本书附带光盘。

第 9 章

Windows 控件与组件

- 控件与组件概述
- 文本类型控件
- 动作类型控件
- 列表类型控件
- 值类型控件
- 图形显示类型控件
- 日期类型控件
- 其他类型控件
- 容器控件
- 组件

在 Windows 应用程序中，控件是用户界面的基本单元。在容器所定义的区域中，控件直接与用户交互。Windows 应用程序支持多种控件，根据控件提供的功能不同，可将控件划分为不同的类型。本章将对 Windows 应用程序中的控件和组件进行详细介绍。

9.1 控件与组件概述

9.1.1 控件概述

控件直接或间接派生于 System.Windows.Forms.Control 基类，它负责绘制容器中的内容，容器可以是一个窗体，也可以是另一个控件。Visual Studio 2008 开发环境中提供了许多个控件，默认情况下，可以从“工具箱”窗口中获得这些控件。Visual Studio 2008 开发环境的控件可划分为以下 7 种类型。

- 文本类控件。在这类控件中，用户可以输入新的文本，也可以编辑现有文本，如 TextBox 控件等。
- 动作类控件。使用此类控件，用户可以与应用程序进行交互，这类控件可以帮助用户在应用程序中执行特定任务，如 Button 控件等。
- 列表类控件。此类控件为用户提供了值列表，用户可以从控件的值列表中选择值，也可以在值列表中输入值，如 ComboBox 控件等。
- 值类控件。使用这类控件，用户可以设置要显示的任何一个值，如 CheckBox 控件等。
- 图形显示类控件。此类控件用于显示如位图、GIF、JPEG、图元文件、图标等，它一次只能显示一张图片，如 PictureBox 控件等。
- 日期类控件。此类控件允许用户选取一个或多个 DateTime 值，如 DateTimePicker 控件等。
- 容器控件。此类控件可以容纳和布置其他控件，如 GroupBox 控件等。

9.1.2 组件概述

在.NET Framework 中，组件是指实现 System.ComponentModel.IComponent 接口的一个类，是一种非可视化的类，是从实现 IComponent 的类中直接或间接派生的类。在编写程序时，“组件”这个术语通常用于可重复使用并且可以和其他对象进行交互的对象。

在.NET Framework 中，组件的一个重要特性就是可设计性，这意味着作为组件的类可用于快速应用程序开发工具（RAD）（如 Visual Studio）中。用户可将组件添加到 Visual Studio 的工具箱中，也可将组件拖放到某个窗体上，还可以在设计图面上对组件进行操作。对 IComponent 类型的基本设计时功能支持已经内置于.NET Framework 中，组件开发人员无需进行额外的工作，就可利用基本设计时的功能。

组件可以放置在一个容器中。当组件被放置后，它通过其站点与容器进行交互并能够通过站点从其容器查询并得到服务。为了确保容器拆卸后资源被释放，容器必须实现 IDisposable 接口。在其 Dispose 方法的实现中，容器必须释放它保留的所有资源，并调用它包含的每个组件的 Dispose 方法。

组件可以是可远程控制的或不可远程控制的。可远程控制的组件由引用或值进行封送。封送涉及跨边界（如应用程序域、进程甚至是计算机）发送对象。当对象由引用进行封送时，会创建一个代理对该对象进行远程调用。当对象由值进行封送时，跨相关边界发送该对象的一个序列化的副本。

9.2 文本类型控件

文本类型控件可以用于输入新的文本，也可以用于编辑现有文本。要获取用户的某些输入内容

时，需要使用这些控件，如 TextBox 控件，有些文本控件需要通过编程的方式修改文本，如 Label 控件、LinkLabel 控件、Statubar 控件和 RichTextBox 控件等。下面对文本类型控件进行详细介绍。

9.2.1 Label 控件

1．功能

Label 控件即标签控件，是 Windows 应用程序中最基本的控件之一。Label 控件用来显示用户不能直接改变的文本信息。可以在“属性”窗口中设置控件的文本信息，也可以通过编写代码来设置要显示的文本信息。在向窗体添加 Label 控件时，系统会自动创建 Label 控件的一个对象。Label 控件如图 9.1 所示。

向窗体添加 Label 控件的步骤如下。

（1）在“工具箱”窗口中选择要添加的 Label 控件。

（2）将鼠标指针移动到窗体上，当鼠标指针变为十字形时，按住鼠标左键同时拖曳鼠标，当达到所需要的大小时释放鼠标，此时控件将被添加到窗体中，如图 9.2 所示。其他控件的添加方法与 Label 控件类似，不再详细介绍。

图 9.1　Label 控件

图 9.2　在窗体中添加控件

2．属性

Label 控件常用属性及说明如表 9.1 所示。

表 9.1　Label 控件常用属性及说明

属　性	说　明
Image	此属性用于指定将在标签上显示的图像
Text	此属性是与文件相关联的文本
Name	指示代码中用来标识该对象的名称
AutoSize	如果此属性设为 true，则表示启用了根据字号自动调整大小，请注意，这只对文本不换行的标签控件有效
Enabled	指示是否已启用该控件
Visible	确定该控件是可见的还是隐藏的

下面对 Label 控件中比较重要的属性进行详细介绍。

（1）Text 属性。该属性用来设置要在该控件上显示的文本。

语法：

```
public override string Text { get; set; }
```

例如，设置 Label 控件的 Text 属性为“世博会欢迎您！”，代码如下：

```
this.Label.Text=“世博会欢迎您！”;
```

示例 09-01　使用 Text 属性设置控件的显示文本

本示例实现的功能是：当程序运行时，在 Label 控件中显示“欢迎您使用明日科技图书，

谢谢合作！”文本信息，在窗体的右下端显示关闭本窗体的剩余时间。代码如下：

代码 09-01　光盘位置：光盘\mingrisoft\sl\09\01

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "欢迎您使用明日科技图书，谢谢合作！";  //在 Label 控件上显示提示性信息
    timer1.Start();                                      //启动计时器
}
public static int i = 6;                                 //实例化一个静态变量
private void timer1_Tick(object sender, EventArgs e)
{
    if (i > 0)                                           //当 i 的值大于 0 时
    {
        i--;                                             //递减 i 的值
        label2.Text = i.ToString() + "s";                //设置关闭本窗体的时间
    }
    if (i == 0)                                          //当 i 的值等于 0 时
    {
        timer1.Stop();                                   //停止计时器
        Application.Exit();                              //退出应用程序
    }
}
```

运行本示例，效果如图 9.3 所示。

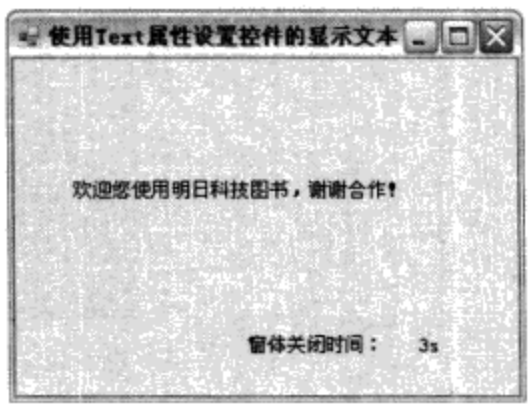


图 9.3　使用 Text 属性设置控件的显示文本

（2）Visible 属性。该属性用于设置控件是否可见。

语法：

```
public bool Visible { get; set; }
```

属性值：如果显示该控件，则为 true，否则为 false。一般默认为 true。

例如，设置 Label1 控件是可见状态，代码如下：

```
label1.Visible = True;
```

3．方法

（1）Show 方法。该方法将控件的 Visible 属性设为 true 并显示控件。

语法：

```
public void Show ()
```

例如，设置 label1 控件为显示状态，代码如下：

```
this.label1.Show();
```

（2）Hide 方法。该方法用于隐藏控件。

语法：

```
public void Hide ()
```

例如，设置 label1 控件为隐藏状态，代码如下：

```
this.label1.Hide();
```

4．事件

Label 控件常用事件及说明如表 9.2 所示。

表 9.2 **Label 控件常用事件及说明**

事　件	说　明
Keypress	当标签有焦点且用户按下某个键时发生
Keyup	当标签有焦点且用户释放某个键时发生
Click	当单击标签时发生的事件
Paint	在重绘或更新控件时发生

下面详细介绍 Click 事件，该事件在单击控件时触发。

语法：

```
public event EventHandler Click
```

示例 09-02　Label 控件的 Click 事件应用

本示例实现的是：当程序运行时，单击“隐藏控件”标签，标签控件隐藏；单击“显示控件”标签，标签控件显示。代码如下：

代码 09-02　光盘位置：光盘\mingrisoft\sl\09\02

```
private void label2_Click(object sender, EventArgs e)
{
    Application.Exit();                    //退出应用程序
}
private void label4_Click(object sender, EventArgs e)
{
    if (this.label3.Visible == true)       //当 Label3 控件的可见属性为真时
    {
        this.label3.Hide();                //隐藏 Labe3 控件
        this.label4.Text = "显示控件";     //设置 Label4 控件的 Text 属性为“显示控件”
        return;                            //返回
    }
    if (this.label3.Visible == false)      //当 Label3 控件的可见属性为假时
    {
        this.label3.Visible = true;        //显示 Label3 控件
        this.label4.Text = "隐藏控件";     //设置 Label4 控件的 Text 属性为“隐藏控件”
        return;                            //返回
    }
}
private void Form1_Load(object sender, EventArgs e)
{
    this.label1.Text = "欢迎你";           //显示提示性信息
}
```

运行本示例，效果如图 9.4 所示。

图 9.4　Label 控件的 Click 事件应用

9.2.2 LinkLabel 控件

1．功能

LinkLabel 控件使用户能够向 Windows 窗体应用程序中添加 Web 样式的链接。一切可以使用 Label 控件的地方都可以使用 LinkLabel 控件。另外，用户可以将文本的一部分设置为指向某个对象或网页的链接。图 9.5 所示为 LinkLabel 控件，该控件支持 Label 控件中的所有属性、方法和事件。

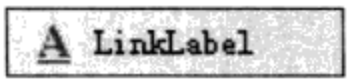


图 9.5　LinkLabel 控件

2．属性

LinkLabel 控件常用属性及说明如表 9.3 所示。

表 9.3　　LinkLabel 控件常用属性及说明

属　性	说　明
ActiveLinkColor	用于指定表示用户单击超级链接时显示的颜色
DisabledLinkColor	用于指定表示超级链接被禁止用时的颜色
LinkArea	用于表示必须显示为超级链接的文本
LinkBehaviour	用于表示链接行为
LinkColor	用于表示超级链接初始的颜色
VisitedLinkcolor	用于表示超级链接以前访问过的颜色
Dock	用于表示超级链接在容器内的布局

LinkArea 属性用于表示必须显示为超级链接的文本。

语法：

```
public LinkArea LinkArea { get; set; }
```

说明：该属性为可读、可写属性。

示例 09-03　通过 LinkLabel 控件的 LinkArea 属性设置链接文字的区域

本示例中将“窗体”两个字设为超级链接文本，代码如下：

代码 09-03　光盘位置：光盘\mingrisoft\sl\09\03

```
private void Form1_Load(object sender, EventArgs e)
{
    linkLabel2.LinkArea = new LinkArea(5, 7);          //设置文本中被视为链接的范围
}
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    MessageBox.Show("连接成功！", "信息提示");     //信息提示
}
private void linkLabel2_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    MessageBox.Show("连接成功！", "信息提示");     //信息提示
}
```

运行本示例，效果如图 9.6 所示。

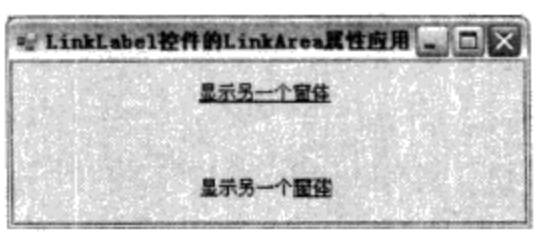


图 9.6　LinkLabel 控件的 LinkArea 属性应用

3．方法

（1）Focus 方法。该方法表示控件设置输入焦点。

语法：

```
public bool Focus()
```

返回值：如果输入焦点请求成功，则为 true；否则为 false。

例如，设置 linkLabel1 控件为当前窗体的焦点，代码如下：

```
this.linkLabel1.Focus();
```

（2）FindForm 方法。该方法表示检索控件所在的窗体。

语法：

```
public Form FindForm ()
```

返回值：控件所在的窗体。

例如，检索 linkLabel1 控件所在的窗体，代码如下：

```
this.linkLabel1.FindForm();
```

4．事件

LinkLabel 控件常用事件为 LinkClicked 事件，该事件在单击链接时触发。

语法：

```
public event LinkLabelLinkClickedEventHandler LinkClicked
```

说明：通常，当用户单击控件中的链接时，处理 LinkClicked 事件以执行任务。将 LinkLabelLinkClickedEventArgs 类的实例传递给 LinkClicked 事件的处理程序，该实例包含与所单击的链接关联的 LinkLabel.Link 对象。可以使用在 LinkLabel.Link 类的 LinkData 属性中指定的信息确定单击了哪个链接或单击该链接后要执行的任务类型。

示例 09-04　使用 LinkLabel 控件链接到另一个窗体

本示例使用 LinkLabel 控件链接到另一个窗体。代码如下：

代码 09-04　光盘位置：光盘\mingrisoft\sl\09\04

```
private void Form1_Load(object sender, EventArgs e)
{
    linkLabel1.LinkArea = new LinkArea(0, 2);    //设置链接文字的个数
}
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    Form2 WindowsTwo = new Form2();              //实例化一个 Form2 窗体
    WindowsTwo.Show();                           //显示实例化的 Form2 窗体
    this.Hide();                                 //隐藏当前窗体
    this.ShowInTaskbar = false;                  //设置一个值指示不在任务栏中显示当前窗体
}
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    Application.Exit();                          //退出应用程序
}
private void Form2_Load(object sender, EventArgs e)
{
    linkLabel1.LinkArea = new LinkArea(0, 2);    //设置连接文字的个数
}
private void linkLabel1_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    Form1 WindowsOne = new Form1();              //实例化一个 Form1 对象
    WindowsOne.Show();                           //显示实例化的对象 WindowsOne
    this.Hide();                                 //关闭当前窗体
    this.ShowInTaskbar = false;                  //设置一个值指示不在任务栏中显示当前窗体
}
private void Form2_FormClosing(object sender, FormClosingEventArgs e)
{
    Application.Exit();                          //退出应用程序
}
```

运行本示例，效果如图 9.7 和图 9.8 所示。

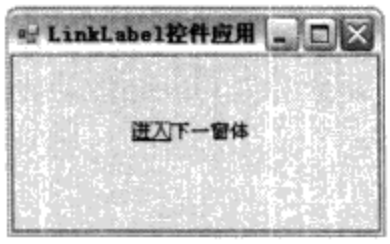


图 9.7　使用 LinkLabel 控件链接到另一个窗体

图 9.8　使用 LinkLabel 控件链接到另一个窗体

9.2.3　TextBox 控件

1．功能

TextBox 控件用于获取用户输入或显示的文本，它通常用于可编辑文本，有时也可使其成为只读控件。另外，TextBox 控件还可以显示密码或多行。图 9.9 所示为 TextBox 控件。

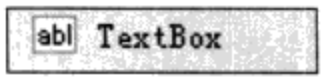


图 9.9　TextBox 控件

2．属性

TextBox 控件常用属性及说明如表 9.4 所示。

表 9.4　　TextBox 控件常用属性及说明

属　性	描　述
AcceptsReturn	此属性用于指示在多行编辑控件中是否允许输入回车符
Maxlength	此属性指定可以编辑控件中输入的最大字符数
Mulitiline	此属性用于指定是否可以在文本框中输入多行文本
PasswordChar	此属性表示所显示的是字符而不是实际输入的文本，当接收用户输入的机密数据和敏感数据时，通常需要设置此属性
ReadOnly	此属性值用于确定文本框中文本是否是只读
TextAlign	此属性用于指示文本框中文本以何种方式对齐
Scrollbars	此属性用于指定是否在多行 TextBox 控件中显示滚动条

下面详细介绍 PasswordChar 属性。其语法格式如下：

```
public char PasswordChar { get; set; }
```

属性值：用于屏蔽在单行 TextBox 控件中输入的字符。由用户自定义，如#、@、$、%、&和*等。如果不想让控件在字符键入时将其屏蔽，请将此属性值设置为 0（字符值）。默认值等于 0。

例如，设置 textBox1 控件为密码输入框，代码如下：

```
this.textBox1.PasswordChar='●';
```

示例 09-05　将 TextBox 控件变为密码输入框

本示例通过设置 TextBox 控件的 PasswordChar 属性，使用户输入的密码显示为“●”。代码如下：

代码 09-05　光盘位置：光盘\mingrisoft\sl\09\05

```
private void Unlogin_Click(object sender, EventArgs e)
{
    Application.Exit();                          //退出应用程序
}
private void Form1_Load(object sender, EventArgs e)
{
    UserPwd.PasswordChar = '●';                  //设置 TextBox 控件的 PasswordChar 属性
    UserName.Focus();                            //设置用户名文本框获得焦点
}
private void login_Click(object sender, EventArgs e)
{
    this.Hide();                                 //隐藏当前窗体
```

```
        MessageBox.Show("登录成功！", "信息提示");      //弹出信息提示
        Application.Exit();                              //退出应用程序
    }
```

运行本示例，效果如图 9.10 所示。

图 9.10　PasswordChar 属性的应用

3．方法

TextBox 控件常用方法及说明如表 9.5 所示。

表 9.5　　TextBox 控件常用方法及说明

方　法	说　明
AppendText	用于在文本框中当前显示的文本末尾添加指定的文本
Clear	用于清空当前文本框中显示的文本
Copy	用于将文本框中的选定文本复制到剪贴板
Cut	删除文本框中的选定文本并将其放置在剪贴板
Paste	用于将剪贴板中文本替换文本框中选定的文本
Focus	为控件设置输入焦点
Dispose	使服务器控件得以在从内存中释放之前执行最后的清理操作

下面详细介绍 Clear 方法，该方法用于清空文本框中当前显示的文本。

语法：

```
public void Clear ()
```

注意：可以使用此方法清除控件的内容，而不是给 Text 属性分配一个空字符串。

例如，清空 textBox1 文本框中的内容，代码如下：

```
this.textBox1.Clear()
```

4．事件

（1）MultilineChanged 事件。当 Multilin 属性的值更改时，触发此事件。

语法：

```
public event EventHandler MultilineChanged
```

（2）TextChanged 事件。当 TextBox 控件为默认控件时，文本框中的文本更改，触发此事件。

语法：

```
public event EventHandler TextChanged
```

示例 09-06　Clear 方法和 TextChanged 事件的联合应用

运行本示例，当文本框中的内容发生变化时，弹出信息提示。当单击“清空”按钮时，无论文本框中有任何内容都执行清空操作。代码如下：

代码 09-06　光盘位置：光盘\mingrisoft\sl\09\06

```
private void Cancel_Click(object sender, EventArgs e)
{
```

```
        textBox1.Clear();            //清空 TextBox 文本框
    }

    private void Affirm_Click(object sender, EventArgs e)
    {
        Application.Exit();          //退出应用程序
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        MessageBox.Show("文本框内容发生变化", "提示信息"); //弹出提示信息
    }
```

运行本示例，效果如图 9.11 所示。

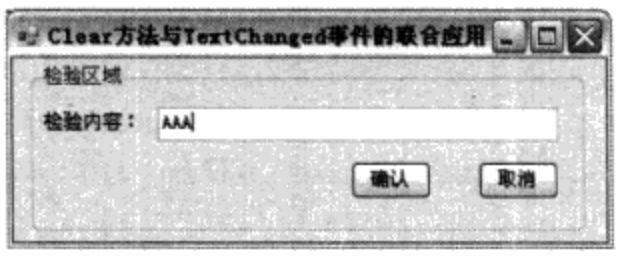


图 9.11　Clear 方法和 TextChanged 事件的联合应用

9.2.4　RichTextBox 控件

1．功能

RichTextBox 控件用于显示、输入和操作带有格式的文本。RichTextBox 控件除了能够执行 TextBox 控件的所有功能之外，它还可以显示字体、颜色、超级链接，从文件加载文本或者图像，撤销、重复编辑操作以及查找指定的字符。RichTextBox 控件通常用于提供文本操作和显示功能。与 TextBox 控件一样，RichTextBox 控件也可以显示滚动条；与 TextBox 控件不同的是，默认情况下，该控件将同时显示水平滚动条和垂直滚动条，并具有更多的滚动条设置。RichTextBox 控件如图 9.12 所示。

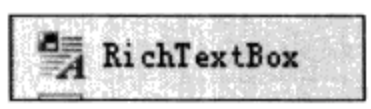


图 9.12　RichTextBox 控件

2．属性

RichTextBox 控件常用属性及说明如表 9.6 所示。

表 9.6　　RichTextBox 控件常用属性及说明

属　性	描　述
DetectUrls	此属性用于指定键入 URL 时 RTF 文本框执行的操作。当此属性设为 Fasle 时，RTF 文本框将 URL 视为普通的文本；当设为 true 时，RTF 文本框将 URL 自动转换为 URL 格式
RedoActionName	此属性表示当调用 Redo 方法时，可再次在 RichTextBox 控件上执行的操作
SelecteText	此属性表示在当前 RichTextBox 控件中选定的文本
SelectionColor	此属性表示当前选中的文本和当前插入点的文本颜色
ZoomFactor	此属性表示 RichTextBox 控件中内容当前缩放的级别或百分比
Anchor	获取或设置控件绑定到的容器的边缘并确定控件如何随其父级一起调整大小
BackColor	获取或设置控件的背景色
BulletIndent	获取或设置对文本应用项目符号样式时，RichTextBox 控件中使用的缩进
ContextMenu	获取或设置与控件关联的快捷菜单
ReadOnly	获取或设置一个值，该值指示文本框中的文本是否为只读
Text	获取或设置多格式文本框中的当前文本
WordWrap	指示多行文本框控件在必要时是否自动换行到下一行的开始

下面对 RichTextBox 控件中比较重要的属性进行详细介绍。

（1）SelectionColor 属性。该属性表示 RichTextBox 控件中选定部分的颜色。

语法：

```
public Color SelectionColor { get; set; }
```

属性值：表示应用到当前选定文本或在插入点后输入的文本的颜色。

例如，设置 richTextBox1 控件中选定部分颜色为红色，代码如下：

```
this.richTextBox1.SelectionColor=Color.Red;
```

（2）SelectedText 属性。该属性表示 RichTextBox 控件中的选定文本。

语法：

```
public override string SelectedText { get; set; }
```

属性值：表示控件中选定文本的字符串。

例如，获取 richTextBox1 控件中的选定文本显示在 label1 控件中，代码如下：

```
this.label1.Text=this.richTextBox1.SelectedText;
```

示例 09-07　选定 RichTextBox 控件中的内容在 Label 控件中显示

本示例将 RichTextBox 文本框内的文字设置为红色，当存在选定文本时，将选定的文本显示在 Label 标签控件中。代码如下：

代码 09-07　光盘位置：光盘\mingrisoft\sl\09\07

```
private void display_Click(object sender, EventArgs e)
{
    if (richTextBox1.SelectedText == "")                    //当选定内容为空时
    {
        MessageBox.Show("请选定文本!", "信息提示");          //弹出信息提示
    }
    else                                                    //当选定内容不为空时
    {
        this.label1.Text = richTextBox1.SelectedText;       //设置 Label1 控件的 Text 属性
    }
}
private void Form1_Load(object sender, EventArgs e)
{
    richTextBox1.SelectionFont = new Font("Tahoma", 12, FontStyle.Bold); //设置当前选定文本或插入点的字体
    //设置当前选定文本或插入点的文本颜色
    richTextBox1.SelectionColor = System.Drawing.Color.Red;
}
```

运行本示例，效果如图 9.13 所示。

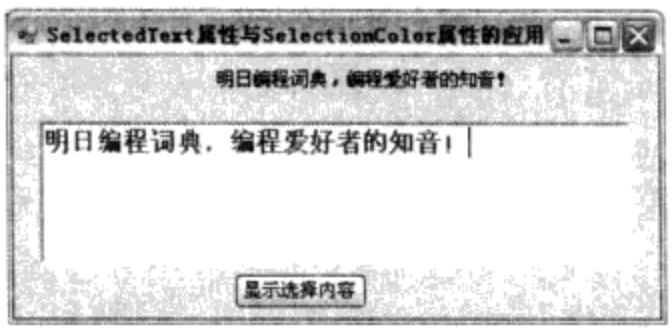


图 9.13　SelectedText 属性与 SelectionColor 属性的应用

3．方法

RichTextBox 控件常用方法及说明如表 9.7 所示。

表 9.7　RichTextBox 控件常用方法及说明

方　法	描　述
Find	此方法用于在 RichTextBox 控件中搜索指定文本
GetCharFromPosition	此方法用于检索 RTF 文本框中指定位置的字符。如果该位置没有字符，则返回指定位置最近的字符
LoadFile	此方法用于指定文件的内容加载或检索到 RichTextBox 控件中

续表

方　法	描　述
Redo	此方法用于再次执行已撤销或取消的操作
Equals	确定指定的 Object 是否等于当前的 Object
SelectAll	选定文本框中的所有文本
SendToBack	将控件发送到 Z 顺序的后面
WndProc	（重写 TextBoxBase.WndProc(Message)。）
SaveFile	此方法用于将 RTF 文本框的内容保存到指定文件

下面对 RichTextBox 控件中比较重要的方法进行详细介绍。

（1）Find 方法。该方法用于在 RichTextBox 控件中搜索指定文本。

语法：

```
public int Find (string str)
```

参数说明如下。

str：要在控件中定位的文本。

返回值：在控件中找到搜索文本的位置。如果未找到搜索字符串或者在 str 参数中指定了空搜索字符串，则为-1。

例如，在 richTextBox1 控件的文本中查找是否有字符串“青龙偃月刀”，代码如下：

```
this.richTextBox1.Find("青龙偃月刀");
```

（2）LoadFile 方法。该方法表示将特定类型的文件加载到 RichTextBox 控件中。

语法：

```
public void LoadFile(string path,RichTextBoxStreamType fileType)
```

参数说明如下。

path：表示要加载到控件中的文件的名称和位置。

fileType：表示 RichTextBoxStreamType 值之一。

RichTextBoxStreamType 的可取值及说明如下。

- RichText：表示 RTF 格式流。
- PlainText：表示用空格代替对象链接与嵌入（OLE）对象的纯文本流。
- RichNoOleObjs：表示用空格代替 OLE 对象的丰富文本格式（RTF 格式）流。该值只在用于 RichTextBox 控件的 SaveFile 方法时有效。
- TextTextOleObjs：表示具有 OLE 对象的文本表示形式的纯文本流。该值只在用于 RichTextBox 控件的 SaveFile 方法时有效。
- UnicodePlainText：表示包含用空格代替对象链接与嵌入（OLE）对象的文本流。该文本采用 Unicode 编码。

例如，将一个文本文件打开到 RichTextBox 控件中。代码如下：

```
public void LoadMyFile()
{
    OpenFileDialog openFile1 = new OpenFileDialog(); //实例化一个打开文件对话框
    openFile1.DefaultExt = "*.rtf";                  //设置将要打开文件的默认扩展名
    openFile1.Filter = "RTF Files|*.rtf";            //设置将要打开文件的过滤参数
    //当单击“打开”按钮且文件路径存在时
    if (openFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK && openFile1.FileName.Length > 0)
    {
        //向 RichTextBox 控件中加载文件
        richTextBox1.LoadFile(openFile1.FileName, RichTextBoxStreamType.PlainText);
    }
}
```

（3）SaveFile 方法。该方法表示将 RichTextBox 的内容保存到特定类型的文件中。

语法：

```
public void SaveFile(string path,RichTextBoxStreamType fileType)
```

参数说明如下。

path：表示要保存的文件的名称和位置。

fileType：表示 RichTextBoxStreamType 值之一。详细说明见 LoadFile 方法的参数说明。

例如，将 RichTextBox 控件中的内容保存到 ASCII 文本文件中。代码如下：

```
public void SaveMyFile()
{
    SaveFileDialog saveFile1 = new SaveFileDialog();   //实例化一个保存文件对话框
    saveFile1.DefaultExt = "*.rtf";                    //设置文件的默认保存类型
    saveFile1.Filter = "RTF Files|*.rtf";              //设置保存文件的过滤参数
    //当单击“保存”按钮且保存路径存在时
    if (saveFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK && saveFile1.FileName.Length > 0)
    {
        //在指定的路径下保存 RichTextBox 控件中的内容
        richTextBox1.SaveFile(saveFile1.FileName, RichTextBoxStreamType.PlainText);
    }
}
```

示例 09-08　在 RichTextBox 控件中实现关键字描红

输入要进行查找的关键字，单击“描红”按钮，如果在 RichTextBox 中存在该关键字，那么就描红该关键字。代码如下：

代码 09-08　光盘位置：光盘\mingrisoft\sl\09\08

```
private void KeyWordsPlotRed_Load(object sender,EventArgs e)
{
    richTextBox1.ReadOnly = true;          //设置文本为只读格式
    string filePath = "tomorrow.TXT";      //设置打开文件的路径
    if(File.Exists(filePath))              //当存在该文件时
    {
        FetchFile(filePath);               //在 RichTextBox 控件中显示指定路径下的文件
        unfold.Enabled = false;            //设置“打开”按钮为不可用状态
    }
    plotRed.Enabled = false;               //设置“描红”按钮为不可用状态
    keyWord.Focus();                       //使焦点位于文本框中
}
private void FetchFile(string path)
{
    //实现一个 TextReader，使其以一种特定的编码从字节流中读取字符。
    StreamReader UTF_8Reader = new StreamReader(path,Encoding.Default);
    richTextBox1.Text = UTF_8Reader.ReadToEnd();//在 RichTextBox 控件中显示从字节流中读取的字符
}
private void unfold_Click(object sender,EventArgs e)
{
    //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。
    OpenFileDialog UTF_8File = new OpenFileDialog();
    UTF_8File.Filter = "TXT 文件（*.TXT）|*.TXT";          //设置打开对话框的打开过滤参数
    if(UTF_8File.ShowDialog() == DialogResult.OK)          //当在打开对话框中点击“打开”按钮时
    {
        //从指定的位置加载 TxT 文件
        richTextBox1.LoadFile(UTF_8File.FileName,RichTextBoxStreamType.PlainText);
        //弹出读取成功的信息提示
        MessageBox.Show("读取成功！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
        unfold.Enabled = false;                            //设置“打开”按钮为不可用状态
    }
}
private int flag = 0;                                      //定义一个 int 型的标识符
private void plotRed_Click(object sender,EventArgs e)
{
```

```
        if((flag = richTextBox1.Text.IndexOf(keyWord.Text,flag)) == -1)  //当文件中不存在要搜索的关键字时
        {
            //弹出对应的信息提示
            MessageBox.Show("没有要查找的结果","提示信息",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
            keyWord.Clear();                                          //清空文本框中的内容
            flag = 0;                                                 //重新为 flag 赋值
        }
        else                                                          //当在文件中存在对应的关键字时
        {
            richTextBox1.Select(flag,keyWord.Text.Length);            //在 RichTextBox 控件中搜索关键字
            flag = flag + keyWord.Text.Length;                        //递增标识查询关键字的初始长度
            richTextBox1.SelectionColor = Color.Red;                  //设定关键字为红色
        }
    }
    private void keyWord_TextChanged(object sender,EventArgs e)
    {
        if(keyWord.Text == "" || keyWord.Text == null)//当文本框中不存在内容或者内容为空时
        {
            plotRed.Enabled = false;                   //设定“描红”按钮为不可用状态
        }
        else                                           //当文本框中存在内容时
        {
            plotRed.Enabled = true;                    //设定“描红”按钮为可用状态
        }
    }
```

运行本示例，效果如图 9.14 所示。

图 9.14 在 RichTextBox 控件中实现关键字描红

4．事件

RichTextBox 控件的常用事件如下。

（1）LinkClicked 事件。该事件在单击 RichTextBox 控件中的超级链接时被触发。

语法：

```
public event LinkClickedEventHandler LinkClicked
```

（2）SelectionChanged 事件。当在 RichTextBox 中选择文本或修改文本时，触发该事件。

语法：

```
public event EventHandler SelectionChanged
```

9.3 动作类型控件

使用动作类控件，用户可以与应用程序进行交互，这些控件可以帮助用户在应用程序中执行特定任务。动作类控件包括 Button 控件、ToolStrip 控件、MenuStrip 控件和 ConTextMenuStrip 控件等。下面将以 Button 控件为代表详细讲解动作类控件。

1．Button 控件功能

Button 控件允许用户通过单击来执行各种操作。当 Button 控件被单击时，调用其 Click 事件执行指定的操作。图 9.15 所示为 Button 控件。

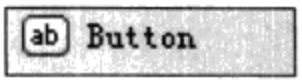


图 9.15　Button 控件

2．Button 控件属性

Button 控件常用属性及说明如表 9.8 所示。

表 9.8　　Button 控件常用属性及说明

属　性	说　明
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
Dialogresult	此属性代表单击窗体中的按钮时返回到窗体的值
Enabled	获取或设置一个值，该值指示是否启用该控件
FlatAppearance	获取用于指示选中状态和鼠标状态的边框外观和颜色
FlatStyle	确定当用户将鼠标移动到控件上并单击时该控件的外观
Image	获取或设置显示在按钮控件上的图像
Imagelist	获取控件上显示的图像
Name	获取或设置控件的名称
TabStop	获取或设置一个值，该值指示用户能否使用 Tab 键将焦点放到该控件上
Text	获取或设置在 Button 控件中显示的文本标题
TextAlign	此属性的值确定 Button 控件上的文本对齐方式

下面对 Button 控件中比较重要的属性进行详细介绍。

（1）Dialogresult 属性。该属性表示单击窗体中的按钮时返回到窗体的值。

语法：

```
public virtual DialogResult DialogResult { get; set; }
```

属性值：DialogResult 值之一。默认值为 None。

DialogResult 的可取值及说明如下。

- None：从对话框返回了 Nothing。这表明有模式对话框继续运行。
- OK：对话框的返回值是 OK（通常从标签为“确定”的按钮发送）。
- Cancel：对话框的返回值是 Cancel（通常从标签为“取消”的按钮发送）。
- Abort：对话框的返回值是 Abort（通常从标签为“中止”的按钮发送）。
- Retry：对话框的返回值是 Retry（通常从标签为“重试”的按钮发送）。
- Ignore：对话框的返回值是 Ignore（通常从标签为“忽略”的按钮发送）。
- Yes：对话框的返回值是 Yes（通常从标签为“是”的按钮发送）。
- No：对话框的返回值是 No（通常从标签为“否”的按钮发送）。

例如，当单击 button1 按钮时返回确认，代码如下：

```
this.button1.DialogResult = DialogResult.OK;
```

（2）FlatAppearance 属性。该属性表示获取用于指示选中状态和鼠标状态的边框外观和颜色。

语法：

```
public FlatButtonAppearance FlatAppearance { get; }
```

属性值：FlatButtonAppearance 值之一。

（3）Image 属性。该属性表示获取或设置显示在按钮控件上的图像。

语法：

```
public Image Image { get; set; }
```

属性值：按钮控件上显示的 Image。默认值为 null。

例如，当应用程序中存在一个“1.bmp”图片时，将它设置为 button1 控件上的图片，代码

如下：

```
this.button1.Image =(Image)Properties.Resources._1;
```

3．Button 控件方法

Button 控件常用方法及说明如表 9.9 所示。

表 9.9　　Button 控件常用方法及说明

方　法	说　明
Dispose	该方法表示释放 Button 控件所占用的系统资源
Equals	确定指定的对象是否等于当前的对象
FindForm	检索控件所在的窗体
Focus	为控件设置输入焦点
GetNextControl	按照子控件的 Tab 键顺序向前或向后检索下一个控件
Hide	对用户隐藏控件
NotifyDefault	通知 Button 是否是默认按钮，以便它能够相应地调整自己的外观
Show	对用户显示控件
PerformClick	引发 Button 控件的 Click 事件

下面对 Button 控件中比较重要的方法进行详细介绍。

（1）Focus 方法。该方法表示为控件设置输入焦点。

语法：

```
public bool Focus()
```

返回值：如果输入焦点请求成功，则为 true；否则为 false。

例如，设置 button1 控件为当前窗体的焦点，代码如下：

```
this.button1.Focus();
```

（2）Hide 方法。该方法表示对用户隐藏控件。

语法：

```
public void Hide()
```

例如，对用户隐藏 button1 控件，代码如下：

```
this.button1.Hide();
```

（3）Show 方法。该方法表示对用户显示控件。

语法：

```
public void Show()
```

例如，对用户显示 button1 控件，代码如下：

```
this.button1.Show();
```

4．Button 控件事件

Click 事件：该事件主要用于 Button 控件的默认事件和主要事件，单击按钮时触发该事件。

示例 09-09　在 Click 事件中清空文本框中的内容并退出应用程序

本示例中，当程序运行时，单击“清空”按钮，清空文本框中内容。单击“退出”按钮，退出应用程序。代码如下：

代码 09-09　光盘位置：光盘\mingrisoft\sl\09\09

```
private void button1_Click(object sender, EventArgs e)
{
    UserName.Clear();     //清空用户名文本框中的内容
    UserPwd.Clear();      //清空密码文本框中的内容
}
private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();   //退出应用程序
}
```

运行本示例，效果如图 9.16 所示。

图 9.16　Click 事件的应用

9.4　列表类型控件

列表类控件为用户提供了值列表，用户可以从控件的值列表中选择值，也可以在其值列表中输入值。列表类型控件主要包括 ComboBox 控件、ListBox 控件、CheckedListBox 控件、ListView 控件、TreeView 控件、NumericUpDown 控件和 DomainUpDown 控件等，本节将对重要的列表控件进行详细介绍。

9.4.1　ListBox 控件

1．功能

ListBox 控件可显示较长的选项列表，用户可从中选择一项或多项。如果项总数超出可以显示的项数，则自动向 ListBox 控件添加滚动条。ListBox 控件列表中的每个元素称为项。图 9.17 所示为 ListBox 控件。

图 9.17　ListBox 控件

2．属性

ListBox 控件常用属性及说明如表 9.10 所示。

表 9.10　　ListBox 控件常用属性及说明

属　性	说　明
AllowDrop	获取或设置一个值，该值指示控件是否可以接受用户拖放到它上面的数据
SelectionMode	获取或设置在 ListBox 中选择项所用的方法
Items	此属性用于查看列表框中的项
MultiColumn	属性设置为 true 时列表框以多列形式显示项，并且会出现一个水平滚动条。属性设置为 False 时，列表框以单列形式显示项，并且会出现一个垂直滚动条
ScrollAlwaysVisible	属性设置为 true 时，无论项数多少都将显示滚动条
SelectionMode	此属性确定一次可以选择多少列表项
SelectedIndex	此属性代表当前选中项的索引号，如果未选定任何项，则 SelectedIndex 值为-1。如果选定了列表中的第一项，则 SelectedIndex 值为 0。当选定多项时，SelectedIndex 值反映列表中最先出现的选定项
Count	此属性反映列表的项数，由于 SelectedIndex 是从 0 开始的，所以 Count 属性的值通常比 SelectedIndex 的最大可能值大 1
SelectedItem	此属性表示当前选中的项
SelectedItems	此属性表示当前选中的所有项
SelectedValue	此属性表示当前选中项的值
sorted	此属性的值确定是否对列表框中的项进行排序
Text	此属性表示当前选中的项的文本

下面对 ListBox 控件中比较重要的属性进行详细介绍。

（1）Items 属性。该属性用于查看列表框中的项。

语法：

```
public ObjectCollection Items { get; }
```

属性值：ListBox.ObjectCollection 表示 ListBox 中的项。

例如，获取 listBox1 控件中所有项的个数，代码如下：

```
int I=this.listBox1.Items.Count;
```

说明：

① 该属性使用户可以获取对当前存储在 ListBox 中的项列表的引用。通过此引用，可以在集合中添加项、移除项和获得项的计数。

② 可以使用 DataSource 属性来操控 ListBox 的项。如果使用 DataSource 属性向 ListBox 添加项，则可以使用 Items 属性查看 ListBox 中的项，但不能使用 ListBox.ObjectCollection 的方法向该列表添加项或从中移除项。

（2）SelectedItem 属性。该属性表示当前选中的项。

语法：

```
public Object SelectedItem { get; set; }
```

属性值：表示控件中当前选定内容的对象。

例如，使用消息框弹出当前 listBox1 控件中的选定项，代码如下：

```
MessageBox.Show( this.listBox1.SelectedItem.ToString());
```

说明：对于标准 ListBox 控件，可以使用此属性确定在 ListBox 控件中选定了哪个项。如果 ListBox 控件的 SelectionMode 属性设置为 SelectionMode.MultiSimple 或 SelectionMode.MultiExtended（它指示多重选择 ListBox），并在该列表中选定了多个项，则此属性可返回任何选定的项。

示例 09-10　在 ListBox 控件中实现选择项功能

本示例主要使用 Items 属性向 ListBox1 控件中添加项，然后使用 SelectedItem 属性将 ListBox1 控件中的选中项添加到 ListBox2 控件中。代码如下：

代码 09-10　光盘位置：光盘\mingrisoft\sl\09\10

```
private void OptionFunction_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();             //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
}
private void allLeft_Click(object sender,EventArgs e)
{
    for(int i = 0; i < listBox2.SelectedItems.Count; )          //循环遍历 listBox2 中的所有选定项
    {
        listBox1.Items.Add(listBox2.SelectedItems[i]);          //向 listBox1 中添加 listBox2 中选定的项
        listBox2.Items.Remove(listBox2.SelectedItems[i]);       //移除 listBox2 中的选定项
    }
    DecideTrueOrFalse();             //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
}
private void left_Click(object sender,EventArgs e)
{
    DecideTrueOrFalse();             //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
```

```
        object SettleOnItem = listBox2.SelectedItem;  //保存 listBox2 中的选定项
        if(listBox1.Items.Contains(SettleOnItem))      //当 listBox1 中已存在该项时
        {
            MessageBox.Show(SettleOnItem.ToString() + "已存在！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.
Warning);                                              //弹出该项已存在的信息
        }
        else                                           //当 listBox1 中不存在该项时
        {
            listBox2.Items.Remove(SettleOnItem);  //从 listBox2 中移除该项
            listBox1.Items.Add(SettleOnItem);      //向 listBox1 中添加该项
        }
    }
    private void listBox1_SelectedIndexChanged(object sender,EventArgs e)
    {
        if(listBox1.SelectedItem == null)              //当 listBox1 中的选定项为空时
        {
            allRight.Enabled = false;                  //设定全部右移的按钮为不可用状态
            allLeft.Enabled = false;                   //设定全部左移的按钮为不可用状态
            right.Enabled = false;                     //设定右移的按钮为不可用状态
            left.Enabled = false;                      //设定左移的按钮为不可用状态
        }
        else if(listBox1.SelectedItems.Count == 1)     //当 listBox1 中的选定项为 1 时
        {
            allRight.Enabled = false;                  //设定全部右移的按钮为不可用状态
            allLeft.Enabled = false;                   //设定全部左移的按钮为不可用状态
            right.Enabled = true;                      //设定右移的按钮为可用状态
            left.Enabled = false;                      //设定左移的按钮为不可用状态
        }
        else if(listBox1.SelectedItems.Count > 1)      //当 listBox1 中的选定项大于 1 时
        {
            right.Enabled = false;                     //设定右移的按钮为可用状态
            left.Enabled = false;                      //设定左移的按钮为不可用状态
            allLeft.Enabled = false;                   //设定全部左移的按钮为不可用状态
            allRight.Enabled = true;                   //设定全部右移的按钮为可用状态
        }
    }
    private void right_Click(object sender,EventArgs e)
    {
        //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
        DecideTrueOrFalse();
        object SettleOnItem = listBox1.SelectedItem;                //保存 listBox1 中的选定项
        if(listBox2.Items.Contains(SettleOnItem))                   //当 listBox2 中已存在该项时
        {
            MessageBox.Show(SettleOnItem.ToString() + "已存在！","提示信息",MessageBoxButtons.OK,MessageBoxIcon.
Warning);                                                           //弹出该项已存在的信息
        }
        else                                                        //当 listBox2 中不存在该项时
        {
            listBox1.Items.Remove(SettleOnItem);                    //从 listBox1 中移除该项
            listBox2.Items.Add(SettleOnItem);                       //向 listBox2 中添加该项
        }
    }
    private void allRight_Click(object sender,EventArgs e)
    {
        for(int i = 0; i < listBox1.SelectedItems.Count; )          //循环遍历 listBox1 中选定的各项
        {
            listBox2.Items.Add(listBox1.SelectedItems[i]);          //向 listBox2 中添加 listBox1 中选定的各项
            listBox1.Items.Remove(listBox1.SelectedItems[i]);       //从 listBox1 中移除 listBox1 中选定的项
        }
        DecideTrueOrFalse();                           //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
    }
    private void DecideTrueOrFalse()
    {
```

```
        if(listBox1.SelectedItem == null)          //当 listBox1 中不存在选择项时，设定所有按钮为不可用状态
        {
            allRight.Enabled = false;              //设定全部右移按钮为不可用状态
            allLeft.Enabled = false;               //设定全部左移按钮为不可用状态
            right.Enabled = false;                 //设定右移按钮为不可用状态
            left.Enabled = false;                  //设定左移按钮为不可用状态
        }
    }
    private void listBox2_SelectedIndexChanged(object sender,EventArgs e)
    {
        if(listBox2.SelectedItem == null)          //当 listBox2 中的选择项为空时
        {
            allRight.Enabled = false;              //设定全部右移的按钮为不可用状态
            allLeft.Enabled = false;               //设定全部左移的按钮为不可用状态
            right.Enabled = false;                 //设定右移按钮为不可用状态
            left.Enabled = false;                  //设定左移按钮为不可用状态
        }
        else if(listBox2.SelectedItems.Count == 1)  //当 listBox2 中的选择项为 1 时
        {
            allRight.Enabled = false;              //设定全部右移按钮为不可用状态
            allLeft.Enabled = false;               //设定全部左移按钮为不可用状态
            right.Enabled = false;                 //设定右移按钮为不可用状态
            left.Enabled = true;                   //设定左移按钮为可用状态
        }
        else if(listBox2.SelectedItems.Count > 1)   //当 listBox2 中的选定项大于 1 时
        {
            right.Enabled = false;                 //设定右移按钮为不可用状态
            left.Enabled = false;                  //设定左移按钮为不可用状态
            allLeft.Enabled = true;                //设定全部左移按钮为可用状态
            allRight.Enabled = false;              //设定全部右移按钮为不可用状态
        }
    }
```

运行本示例，效果如图 9.18 所示。

图 9.18　在 ListBox 控件中实现选择项功能

3．方法

ListBox 控件常用方法及说明如表 9.11 所示。

表 9.11　　ListBox 控件常用方法及说明

方　法	说　明
FindString	从列表中检索以指定字符串开始的第一项
GetSelected	如果选中了指定项，此方法返回 true；如果没有选中指定项，则返回 false
SetSelected	选择或取消选定的项。可将一个 Boolean 值 true 传递至此方法，选中该项
GetItemText	返回指定项的文本表示形式

下面详细介绍 FindString 方法，该方法用于从列表中检索以指定字符串开始的第一项。

语法：

```
public int FindString (string s)
public int FindString (string s,int startIndex)
```

参数说明如下。

s：要搜索的文本。

StartIndex：从第 0 项开始的索引，该项在要搜索的第一个项之前。设置为-1 表示从控件的开始处进行搜索。

返回值：找到的第一个项从 0 开始的索引；如果找不到匹配项，则返回 ListBox. NoMatches。

例如，从 listBox1 控件的第 0 项开始查找“福尔摩斯”字符串，代码如下：

```
this.listBox1.FindString("福尔摩斯",0);
```

示例 09-11　使用 FindString 方法查找指定字符串

本示例通过使用 FindString 方法查找输入字符串在 ListBox 控件中的位置，并通过消息提示框显示。代码如下：

代码 09-11　光盘位置：光盘\mingrisoft\sl\09\11

```
private void Form1_Load(object sender, EventArgs e)
{
    Content.MaxLength = 1;          //设置字符串文本框的最大长度
    listBox1.Items.Add("吉");//向 ListBox 控件中添加“吉”字符串
    listBox1.Items.Add("林");//向 ListBox 控件中添加“林”字符串
    listBox1.Items.Add("省");//向 ListBox 控件中添加“省”字符串
    listBox1.Items.Add("明");//向 ListBox 控件中添加“明”字符串
    listBox1.Items.Add("日");//向 ListBox 控件中添加“日”字符串
    listBox1.Items.Add("科");//向 ListBox 控件中添加“科”字符串
    listBox1.Items.Add("技");//向 ListBox 控件中添加“技”字符串
    listBox1.Items.Add("有");//向 ListBox 控件中添加“有”字符串
    listBox1.Items.Add("限");//向 ListBox 控件中添加“限”字符串
    listBox1.Items.Add("公");//向 ListBox 控件中添加“公”字符串
    listBox1.Items.Add("司");//向 ListBox 控件中添加“司”字符串
}
private void Search_Click(object sender, EventArgs e)
{
    if (Content.Text != null && Content.Text != "")      //当字符串文本框的内容不为空且存在内容时
    {
        MessageBox.Show(Content.Text + "字符串位于第" + (Convert.ToInt32(this.listBox1.FindString(Content.Text,
0)) + 1) + "位", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);//弹出查找字符串所在的位置
    }
    else                                                 //当字符串文本框的内容为空或者不存在时
    {
        MessageBox.Show("请检查输入字符串内容及格式是否正确，谢谢合作！", "信息提示");//弹出信息提示
    }
}
```

运行本示例，效果如图 9.19 所示。

图 9.19　使用 FindString 方法查找指定字符串

4．事件

（1）SelectedIndexChanged 事件。当 SelectedIndex 属性值发生更改时，触发此事件。

语法：

```
public event EventHandler SelectedIndexChanged
```

（2）SelectedValueChanged 事件。当 SelectedValue 属性值发生更改时，触发此事件。

语法：

```
public event EventHandler SelectedValueChanged
```

9.4.2 ComboBox 控件

1．功能

ComboBox 控件用于在下拉组合框中显示数据，它结合了 TextBox 控件和 ListBox 控件的功能。使用此控件时，可以在组合框中输入文本项，也可以从列表中选择项。图 9.20 所示为 ComboBox 控件。

图 9.20　ComboBox 控件

2．属性

ComboBox 控件常用属性及说明如表 9.12 所示。

表 9.12　ComboBox 控件常用属性及说明

属　性	说　明
DropDownStyle	此属性表示 ComboBox 的样式，如 Simple、DropdownList 和 Dropdown
Focused	此属性表示组合框是否获得焦点
MaxDropdownItems	此属性表示下拉列表中项的最多数
Items	获取一个对象，该对象表示该 ComboBox 中所包含项的集合
SelectedIndex	获取或设置指定当前选定项的索引
SelectedItem	获取或设置 ComboBox 中当前选定的项
Sorted	获取或设置指示是否对组合框中的项进行了排序的值

下面对 ComboBox 控件中比较重要的属性进行详细介绍。

（1）DropDownStyle 属性。该属性表示获取或设置指定组合框样式的值。

语法：

```
public ComboBoxStyle DropDownStyle { get; set; }
```

属性值：ComboBoxStyle 值之一，默认为 DropDown。

ComboBoxStyle 的可取值及说明如下。

- DropDown：表示文本部分可编辑，用户必须单击箭头按钮来显示列表部分，这是默认样式。
- DropDownList：用户不能直接编辑文本部分，必须单击箭头按钮来显示列表部分。
- Simple：文本部分可编辑，但列表部分不可见。

例如，设置 comboBox1 控件的 DropDownStyle 属性，代码如下：

```
this.comboBox1.DropDownStyle=ComboBoxStyle.DropDownList;
```

（2）Items 属性。该属性表示获取一个对象，该对象表示该 ComboBox 中所包含项的集合。

语法：

```
public ComboBox.ObjectCollection Items { get; }
```

属性值：表示 ComboBox 控件中项的集合。

例如，获取 comboBox1 控件中所有项的个数，代码如下：

```
int I=this.comboBox1.Items.Count;
```

（3）SelectedIndex 属性。该属性表示获取或设置指定当前选定项的索引。

语法：

```
public override int SelectedIndex { get; set; }
```

属性值：当前选定项从零开始的索引。如果未选定任何项，则返回值为-1。

例如，设置 comboBox1 控件中默认选中第一项，代码如下：

```
this.comboBox1.SelectedIndex=0;
```

示例 09-12　ComboBox 控件美化样式

本示例将通过对 ComboBox 控件的相关属性进行设置，美化 ComboBox 控件。代码如下：

代码 09-12　光盘位置：光盘\mingrisoft\sl\09\12

```
private void PrettifyListedBelow_Load(object sender,EventArgs e)
{
    beautyComboBox.Items.Add("白菜");//向 ComboBox 中添加“白菜”字段
    beautyComboBox.Items.Add("萝卜");//向 ComboBox 中添加“萝卜”字段
    beautyComboBox.Items.Add("土豆");//向 ComboBox 中添加“土豆”字段
    beautyComboBox.Items.Add("洋葱");//向 ComboBox 中添加“洋葱”字段
    beautyComboBox.Items.Add("南瓜");//向 ComboBox 中添加“南瓜”字段
    beautyComboBox.SelectedIndex = 0;//设置 ComboBox 控件默认选中第一项
}

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();             //定义一个字体类对象
    Font Style = typeFace.Font;                         //定义一个用于定义特定的文本格式类对象
    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);
            e.Graphics.DrawString(temp,Style,new SolidBrush(Color.Black),rComboBox.Left + imageSize.Width,
rComboBox.Top);                                     //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.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);
            e.Graphics.DrawString(temp,Style,new SolidBrush(Color.Black),rComboBox.Left + imageSize.Width,
rComboBox.Top);                                     //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.DrawFocusRectangle();                     //在指定的边界范围内绘制聚焦框
        }
    }
}
```

运行本示例，效果如图 9.21 所示。

3．方法

（1）Select 方法。该方法从 ComboBox 中选取指定的项。

语法：

```
public void Select(int start,int length)
```

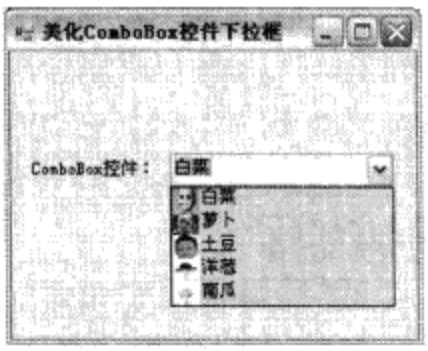


图 9.21　ComboBox 控件美化样式

参数说明如下。

start：表示文本框中当前选定文本的第一个字符的位置。

length：表示要选择的字符数。

例如，选定 comboBox1 控件中的前两个字符，代码如下：

```
this.comboBox1.Select(0, 2);
```

（2）SelectAll 方法。该方法用来选择 ComboBox 控件可编辑部分的所有文本。

语法：

```
public void SelectAll ()
```

例如，选中 comboBox1 控件中可编辑部分的所有文本，代码如下：

```
this.comboBox1.SelectAll();
```

示例 09-13　选定 ComboBox 控件中的内容

本示例通过使用 SelectAll 方法选择 ComboBox 控件中可编辑区域的所有内容，通过 Select 方法选择 ComboBox 控件中可编辑区域的指定部分。代码如下：

代码 09-13　光盘位置：光盘\mingrisoft\sl\09\13

```
private void Form1_Load(object sender, EventArgs e)
{
    string[] Content = { "《Microsoft SQL Server 2000 标准教程》","《XML 基础教程》","《UML Rose 建模工具》","《Visual C# 2005 程序设计与应用简明教程》","《Visual Basic 程序设计》","《Asp.NET 2.0 简明教程》","《数据结构（C++版）》","《数据库原理》","《大学英语》","《高等数学》","《HTML 从入门到精通》","《C 语言程序设计》","《软件测试》" };          //实例化一个字符串数组
    for (int i = 0; i < Content.Length ; i++)          //循环遍历数组中的每一项内容
    {
        this.comboBox1.Items.Add(Content[i]);   //向 ComboBox 控件中添加内容
    }
    this.comboBox1.SelectedIndex = 0;          //设置 ComboBox 控件默认选中第 0 项
}
private void ChoiceAll_Click(object sender, EventArgs e)
{
    this.comboBox1.SelectAll();                //选择 ComboBox 控件中可编辑的所有部分
}
private void Choice_Click(object sender, EventArgs e)
{
    this.comboBox1.Select(0, 6);               //选择 ComboBox 控件中可编辑的自定义部分
}
```

运行本示例，效果如图 9.22 所示。

图 9.22　SelectAll 方法与 Select 方法的应用

4．事件

ComboBox 控件常用事件及说明如表 9.13 所示。

表 9.13　　ComboBox 控件常用事件及说明

属　性	说　明
DropDown	当显示 ComboBox 的下拉部分时发生
DropDownClosed	在 ComboBox 的下拉部分不再可见时发生
SelectedIndexChanged	在 SelectedIndex 属性更改后发生
SelectedValueChanged	当 SelectedValue 属性更改时发生

示例 09-14　在文本框中显示 ComboBox 控件的事件操作提示

本示例使用 ComboBox 控件的 DropDown 事件、DropDownClosed 事件、SelectedIndexChanged 事件实现对 ComboBox 控件下拉菜单的操作进行提示。当下拉菜单显示时，在 ComboBox 控件的上面显示为“开启”；当下拉菜单关闭时，在 ComboBox 控件的上面显示为“关闭”；当 ComboBox 控件中的选择项发生改变时，弹出消息框提示选择的内容。代码如下：

代码 09-14　光盘位置：光盘\mingrisoft\sl\09\14

```
private void Form1_Load(object sender, EventArgs e)
{
    this.richTextBox1.ReadOnly = true;                          //设置 RichTextBox 控件的可读属性为真
    //实例化一个字符串数组
    string[] Content = { "高尔夫球", "乒乓球", "羽毛球", "曲棍球", "排球", "篮球", "足球", "水球" };
    for (int i = 0; i < Content.Length; i++)                    //循环遍历数组中的每一项内容
    {
        this.comboBox1.Items.Add(Content[i]);                   //向 ComboBox 控件中添加内容
    }
    this.comboBox1.SelectedIndex = 0;                           //设置 ComboBox 控件的默认索引
}
private void comboBox1_DropDown(object sender, EventArgs e)
{
    this.richTextBox1.BackColor = Color.BurlyWood;              //设置 RichTextBox 控件的背景色
    this.richTextBox1.ForeColor = Color.BlueViolet;             //设置 RichTextBox 控件中的文字颜色
    this.richTextBox1.Text = "开　　启";                        //设置 RichTextBox 控件中的文字内容
    this.richTextBox1.SelectAll();                              //选定当前 ComboBox 控件中的所有内容
    this.richTextBox1.SelectionAlignment = HorizontalAlignment.Center;//设置 RichTextBox 控件中的内容居中显示
}
private void comboBox1_DropDownClosed(object sender, EventArgs e)
{
    this.richTextBox1.BackColor = Color.BurlyWood;              //设置 RichTextBox 控件的背景色
    this.richTextBox1.ForeColor = Color.BlueViolet;             //设置 RichTextBox 控件中的文字颜色
    this.richTextBox1.Text = "关　　闭";                        //设置 RichTextBox 控件中的文字内容
    this.richTextBox1.SelectAll();                              //选定当前 ComboBox 控件中的所有内容
    this.richTextBox1.SelectionAlignment = HorizontalAlignment.Center;//设置 RichTextBox 控件中的内容居中显示
}
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.richTextBox1.SelectAll();                              //选定当前 ComboBox 控件中的所有内容
    MessageBox.Show("选项的索引发生变化\n 你现在选中的是：" + this.comboBox1.SelectedItem.ToString(), "信
息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);    //弹出对应的信息提示
}
```

运行本示例，效果如图 9.23 所示。

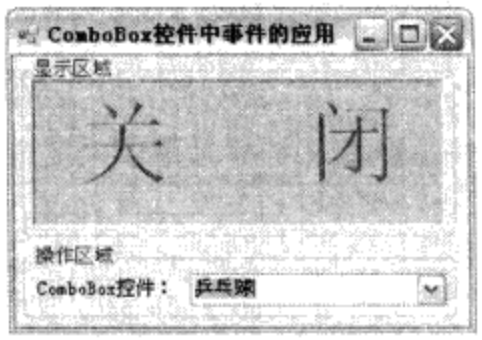


图 9.23　在文本框中显示 ComboBox 控件的事件操作提示

9.4.3　ListView 控件

1．功能

ListView 控件可以显示带图标的项列表，用户可使用该控件创建类似 Windows 资源管理器的用户界面。ListView 控件具有 4 种视图模式：

（1）仅文本，这是默认视图，此视图下，只显示列表项的文本；

（2）带有小图标的文本，此视图下，小图标随列表项的文本同时显示；

（3）带有大图标的文本，此视图下，大图标随列表项的文本同时显示；

（4）报表视图，此视图下，列表项显示在多个列中。图 9.24 所示为 ListView 控件。

图 9.24　ListView 控件

2．属性

ListView 控件常用属性及说明如表 9.14 所示。

表 9.14　　ListView 控件常用属性及说明

属　性	说　明
FullRowSelect	此属性用于指定在 ListView 控件中单击某项时要执行的操作过程。单击某项时，可以指定是只选择该项，还是应选该项所在的整行
View	此属性指定将创建列表视图的类型。视图类型主要包括：大图标、小图标、列表和详细信息
Alignment	获取或设置控件中项的对齐方式
AllowDrop	获取或设置一个值，该值指示控件是否可以接受用户拖放到它上面的数据
CheckBoxes	获取或设置一个值，该值指示控件中各项的旁边是否显示复选框
Columns	获取控件中显示的所有列标题的集合
HideSelection	获取或设置一个值，该值指示当控件没有焦点时，该控件中选定的项是否保持突出显示
Items	获取包含控件中所有项的集合
LabelEdit	获取或设置一个值，该值指示用户是否可以编辑控件中项的标签
MultiSelect	获取或设置一个值，该值指示是否可以选择多个项
SelectedIndices	获取控件中选定项的索引
SelectedItems	获取在控件中选定的项
Sorting	对项进行排序的方式
Multiselect	此属性设为 true 时，表示在控件中一次可以选择多个项
GridLines	获取或设置一个值，该值指示在包含控件中项及其子项的行和列之间是否显示网格线

下面对 ListView 控件中比较重要的属性进行详细介绍。

（1）View 属性。该属性表示用于获取或设置项在控件中的显示方式。

语法：

```
public View View { get; set; }
```

属性值：View 值之一。默认为 LargeIcon。

例如，设置 listView1 控件的视图方式为列表视图，代码如下：

```
this.listView1.View=View.Details;
```

View 的属性值及说明如表 9.15 所示。

表 9.15　View 的属性值及说明

属 性 值	说　明
Details	每个项显示在不同的行上，并带有关于列中所排列的各项的进一步信息。最左边的列包含一个小图标和标签，后面的列包含应用程序指定的子项。列显示一个标头，它可以显示列的标题。用户可以在运行时调整各列的大小
LargeIcon	每个项都显示为一个最大化图标，在它的下面带一个标签
SmallIcon	每个项都显示为一个小图标，在它的右边带一个标签
List	每个项都显示为一个小图标，在它的右边带一个标签。各项排列在列中，没有列标头
Tile	每个项都显示为一个完整大小的图标，在它的右边带项标签和子项信息。显示的子项信息由应用程序指定。此视图仅在下面的平台上受支持：Windows XP 和 Windows Server 2003 系列。在之前的操作系统上，此值被忽略，并且 ListView 控件在 LargeIcon 视图中显示

（2）FullRowSelect 属性。该属性用于指定是只选择某一项，还是选择某一项所在的整行。

语法：

```
public bool FullRowSelect { get; set; }
```

属性值：如果单击某项会选择该项及其所有子项，则为 true；如果单击某项仅选择项本身，则为 False。默认为 False。

说明：除非将 ListView 控件的 View 属性设置为 Details，否则 FullRowSelect 属性无效。在 ListView 显示带有许多子项的项时，通常使用 FullRowSelect 属性，并且，在由于控件内容的水平滚动而无法看到项文本时，能够查看选定项是非常重要的。

例如，设置 listView1 控件中当单击某一行时整行选中，代码如下：

```
this.listView1.FullRowSelect=True;
```

（3）GridLines 属性。该属性表示指定在包含控件中的项及其子项的行和列之间是否显示网格线。

语法：

```
public bool GridLines { get; set; }
```

属性值：如果在项及其子项的周围绘制网格线，则为 true，否则为 false。默认为 false。

说明：除非将 ListView 控件的 View 属性设置为 Details，否则 GridLines 属性无效。

例如，设置 listView1 控件中显示网格线，代码如下：

```
this.listView1.GridLines=True;
```

（4）HideSelection 属性。该属性表示获取或设置一个值，该值指示当控件没有焦点时，该控件中选定的项是否保持突出显示。

语法：

```
public bool HideSelection { get; set; }
```

属性值：如果当控件没有焦点时，控件中的选定项仍保持高亮显示，则为 true；否则，该值为 false。

例如，设置 listView1 控件中无论处于任何状态下，选定项都保持高亮显示。代码如下：

```
this.listView1.HideSelection=True;
```

（5）LabelEdit 属性。该属性表示获取或设置一个值，该值指示用户是否可以编辑控件中项的标签。

语法：

```
public bool LabelEdit { get; set; }
```

属性值：当用户可以编辑控件中项的文本时，该值为 true；否则，该值为 false。

例如，设置 listView1 控件中可以编辑控件中项的标签，代码如下：

```
this.listView1.LabelEdit=True;
```

示例 09-15　ListView 控件属性的综合应用

本示例主要介绍 ListView 控件的 View 属性、FullrowSelect 属性、LabelEdit 属性、HideSelection

属性以及 GridLines 属性的使用方法。代码如下：

代码 09-15　光盘位置：光盘\mingrisoft\sl\09\15

```
public static bool flag = false;            //定义一个全局变量标识
private void OptionBrightFlash_Load(object sender,EventArgs e)
{
    listView1.LabelEdit = true;         //设置是否可以在 listView1 中进行编辑文本内容
    listView1.GridLines = true;         //设置是否在 listView1 控件中显示网格线
    listView1.Dock = DockStyle.Fill; //设置 listView1 控件在其父容器中的停靠方式
    listView1.Columns.Add("文件名称",120,HorizontalAlignment.Left);     //在 listView1 中添加“文件名称”列
    listView1.Columns.Add("文件属性",210,HorizontalAlignment.Left);     //在 listView1 中添加“文件属性”列
    listView1.Columns.Add("创建时间",200,HorizontalAlignment.Left);     //在 listView1 中添加“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);          //实例化一个 listView 控件中选择项的实例
        OptionItem.SubItems.Add(file.Attributes.ToString());            //在 listView 控件中添加文件属性列
        OptionItem.SubItems.Add(file.CreationTime.ToString());          //在 listView 控件中文件创建时间列
        listView1.Items.Add(OptionItem);                                //向 listView 控件中追加新添加的各列
    }
    listView1.HideSelection = true;                         //设置控件的高亮显示属性为 true
}
private void listView1_MouseClick(object sender,MouseEventArgs e)
{
    listView1.SelectedItems[0].ForeColor = Color.Red;   //设置当前选择项为红色
}
private void 取消选择 ToolStripMenuItem_Click(object sender,EventArgs e)
{
    if(listView1.SelectedItems.Count != 0)              //当 listView1 控件中的选择项不为 0 时
    {
        for(int i = 0; i < listView1.SelectedItems.Count; i++)          //循环遍历控件中的每一个选择项
        {
            if(listView1.SelectedItems[i].ForeColor == Color.Red)       //当选择项为红颜色时
            {
                listView1.SelectedItems[i].ForeColor = Color.Black;     //设置选择项为黑颜色
            }
        }
    }
}
```

运行本示例，效果如图 9.25 所示。

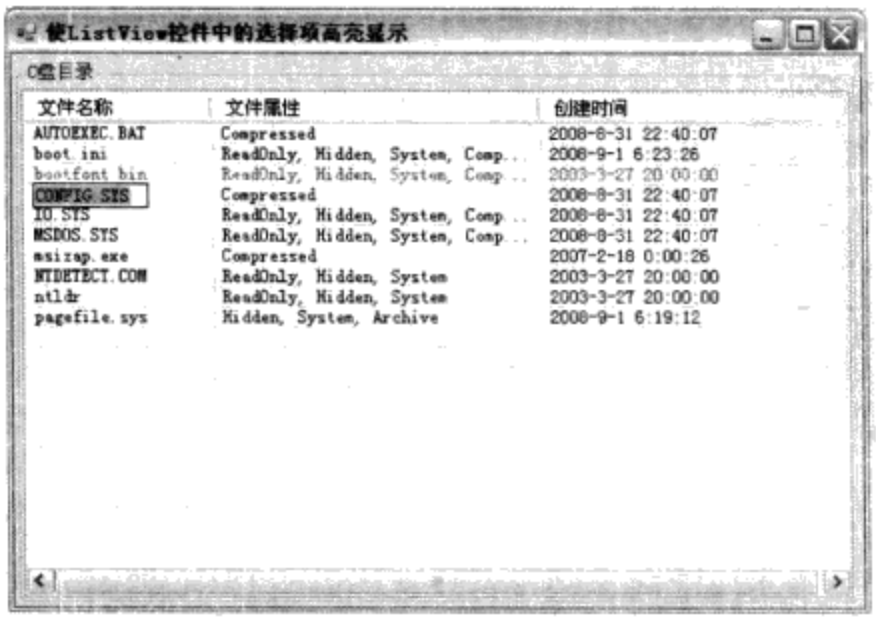


图 9.25　ListView 控件属性的综合应用

3．方法

（1）Clear 方法。该方法表示从控件中移除所有项和列。

语法：

```
public void Clear()
```

例如，从 listView1 控件中移除所有的项和列，代码如下：

```
this.listView1.Clear();
```

（2）Sort 方法。该方法表示对列表视图的项进行排序。

语法：

```
public void Sort()
```

例如，在 listView1 控件中进行升序排列，代码如下：

```
this.listView1.Sorting = SortOrder.Ascending;
this.listView1.Sort();
```

示例 09-16　ListView 控件方法的综合应用

本示例主要介绍 ListView 控件的 Clear 方法和 Sort 方法。代码如下：

代码 09-16　光盘位置：光盘\mingrisoft\sl\09\16

```
public static bool flag = true;            //定义一个全局变量标识
private void ClearUp_Click(object sender, EventArgs e)
{
    this.listView1.Clear();                //清空 ListView 控件中的列和行
}
private void SortAll_Click(object sender, EventArgs e)
{
    if (flag == true)                      //当 flag 为 true 时
    {
        this.listView1.Sorting = SortOrder.Descending;  //设置 ListView 控件为降序排列
        this.listView1.Sort();             //对列表视图的项进行排序
        flag = false;                      //设置 flag 为 false
    }
    else                                   //当 flag 为 False 时
    {
        this.listView1.Sorting = SortOrder.Ascending;   //设置 ListView 控件为升序排列
        this.listView1.Sort();             //对列表视图的项进行排序
        flag = true;                       //设置 flag 为 true
    }
}
private void Form1_Load(object sender, EventArgs e)
{
    this.listView1.View = View.List;  //设置 ListView 控件中项的显示方式
    this.listView1.Items.Add("1");    //向 ListView 控件中添加内容“1”
    this.listView1.Items.Add("2");    //向 ListView 控件中添加内容“2”
    this.listView1.Items.Add("3");    //向 ListView 控件中添加内容“3”
    this.listView1.Items.Add("4");    //向 ListView 控件中添加内容“4”
    this.listView1.Items.Add("5");    //向 ListView 控件中添加内容“5”
    this.listView1.Items.Add("6");    //向 ListView 控件中添加内容“6”
}
```

运行本示例，效果如图 9.26 所示。

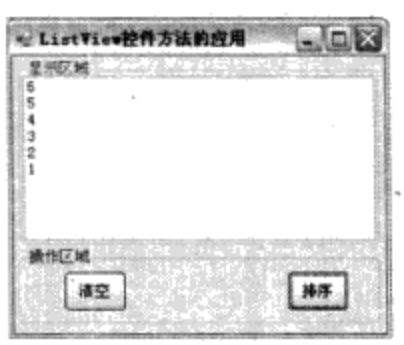


图 9.26　ListView 控件方法的综合应用

4．事件

（1）ItemCheck 事件。该事件在选中 ListView 控件项时触发。

语法：

```
public event ItemCheckEventHandler ItemCheck
```

（2）Click 事件。该事件在单击 ListView 控件列时触发。

语法：

```
public event EventHandler Click
```

示例 09-17　在文本框中显示列表中被单击项的名称

本示例实现效果为，当程序运行时，单击 ListView 控件中的某一项时，将选中项中的姓名信息显示在文本框中。代码如下：

代码 09-17　光盘位置：光盘\mingrisoft\sl\09\17

```
private void Form1_Load(object sender, EventArgs e)
{
    this.listView1.View = View.Details;            //设置 ListView 控件的视图方式
    this.listView1.FullRowSelect = true;           //设置 ListView 控件的选择全行属性为真
    this.listView1.Columns.Add("用户编号", 68);//向 ListView 控件中添加“用户编号”列
    this.listView1.Columns.Add("用户姓名", 68);//向 ListView 控件中添加“用户姓名”列
    this.listView1.Columns.Add("用户性别", 68);//向 ListView 控件中添加“用户性别”列
    this.listView1.Columns.Add("用户电话", 68);//向 ListView 控件中添加“用户电话”列
    this.listView1.Columns.Add("用户地址", 68);//向 ListView 控件中添加“用户地址”列
    //实例化一个数据库连接对象
    SqlConnection ConnNine = new SqlConnection("server=.;database=db_09;integrated security=sspi");
    ConnNine.Open();                                          //打开数据库连接
    //实例化一个执行 SQL 语句的对象
    SqlCommand CommNine = new SqlCommand("select * from tb_Users", ConnNine);
    SqlDataReader ReaderNine = CommNine.ExecuteReader();//实例化一个数据读取器对象
    while (ReaderNine.Read())                                 //读取数据表中的每一行记录
    {
        //实例化一个 ListViewItem 类的实例并添加数据库中第 1 列的内容
        ListViewItem ItemNine = new ListViewItem(ReaderNine.GetValue(0).ToString());
        //向 ListViewItem 中添加并添加数据库中第 2 列的内容
        ItemNine.SubItems.Add(ReaderNine.GetValue(1).ToString());
        //向 ListViewItem 中添加并添加数据库中第 3 列的内容
        ItemNine.SubItems.Add(ReaderNine.GetValue(2).ToString());
        //向 ListViewItem 中添加并添加数据库中第 4 列的内容
        ItemNine.SubItems.Add(ReaderNine.GetValue(3).ToString());
        //向 ListViewItem 中添加并添加数据库中第 5 列的内容
        ItemNine.SubItems.Add(ReaderNine.GetValue(4).ToString());
        this.listView1.Items.Add(ItemNine);                   //向 ListView 控件中添加项
    }
    ReaderNine.Close();   //关闭数据读取器
    ConnNine.Close();     //关闭数据库连接
}
private void listView1_Click(object sender, EventArgs e)
{
    this.Content.Text = this.listView1.SelectedItems[0].SubItems[1].Text;    //在文本框中显示选择的内容
}
```

运行本示例，效果如图 9.27 所示。

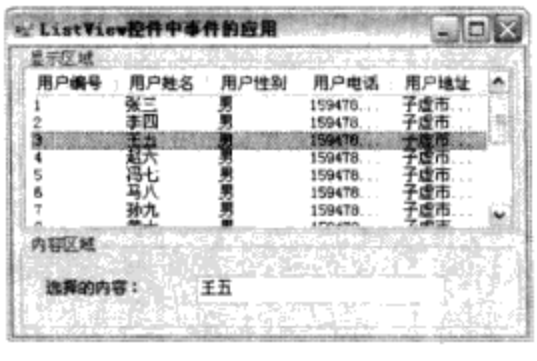


图 9.27　在文本框中显示列表中被单击项的名称

9.4.4 TreeView 控件

1．功能

TreeView 控件可为用户显示节点层次结构，和在 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。树视图中的各个节点都可以包含其他节点。用户可以按展开或折叠的方式显示父节点或包含子节点的节点。另外，通过将树视图的 CheckBoxes 属性设置为 true，可以在节点旁边显示复选框，用户可以通过将节点的 Checked 属性设置为 true 或 false 来选中或清除节点。图 9.28 所示为 TreeView 控件。

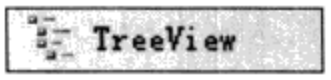


图 9.28　TreeView 控件

2．属性

TreeView 控件常用属性及说明如表 9.16 所示。

表 9.16　　TreeView 控件常用属性及说明

属　性	说　明
AllowDrop	获取或设置一个值，该值指示控件是否可以接受用户拖放到它上面的数据
CheckBoxes	获取或设置一个值，用以指示是否在树视图控件中的树节点旁显示复选框
HideSelection	获取或设置一个值，用以指示选定的树节点是否即使在树视图已失去焦点时仍会保持突出显示
ImageIndex	此属性显示节点图像的索引
ImageList	表示可显示节点处的图像列表
LabelEdit	获取或设置一个值，用以指示是否可以编辑树节点的标签文本
Nodes	此属性可设置 TreeView 控件中的所有节点
Scrollable	获取或设置一个值，用以指示树视图控件是否在需要时显示滚动条
SelectedNode	此属性表示 TreeView 控件中当前选中的节点
ShowLines	此属性指定树视图的同级节点之间以及树节点和根节点之间是否有线
ShowPlusminus	指示是否在父节点旁边显示加减按钮

下面对 TreeView 控件中比较重要的属性进行详细介绍。

（1）CheckBoxes 属性。该属性表示获取或设置一个值，用以指示是否在树视图控件中的树节点旁显示复选框。

语法：

```
public bool CheckBoxes { get; set; }
```

属性值：若在树节点旁边显示复选框，则该值为 true；否则，该值为 false。默认值为 false。

例如，设置 treeView1 控件中的树节点旁边显示复选框，代码如下：

```
this.treeView1.CheckBoxes=True;
```

（2）LabelEdit 属性。该属性表示获取或设置一个值，用以指示是否可以编辑树节点的标签文本。

语法：

```
public bool LabelEdit { get; set; }
```

属性值：若可以直接编辑树节点的标签文本，则该值为 true；否则，该值为 false。默认值为 false。

例如，设置 treeView1 控件中不能编辑树节点，代码如下：

```
this.treeView1.LabelEdit=False;
```

（3）Nodes 属性。该属性用于设置 TreeView 控件中的所有节点。

语法：

```
public TreeNodeCollection Nodes { get; }
```

例如，向 treeView 控件中添加内容“拉芳”，代码如下：

```
this.treeView1.Nodes.Add("拉芳");
```

（4）ShowLines 属性。该属性表示树视图的同级节点之间以及树节点和根节点之间是否有线。

语法：

```
public bool ShowLines { get; set; }
```

属性值：当在同级节点之间或者树节点之间有线存在时，该值为 true；否则，该值为 fasle。默认值为 true。

例如，设置 treeView1 控件中的同级节点之间以及树节点和根节点中有线，代码如下：

```
this.treeView1.ShowLines = true;
```

（5）ShowPlusminus 属性。该属性表示是否在父节点旁边显示加减号。

语法：

```
public bool ShowPlusMinus { get; set; }
```

属性值：当在父节点旁边显示加号或者减号时，该值为 true；否则，该值为 false。默认值为 true。

例如，设置在 treeView1 控件中的父级节点旁边显示加减号，代码如下：

```
this.treeView1.ShowPlusminus=true;
```

示例 09-18　向 TreeView 控件中添加指定节点

本示例使用 Nodes 属性向 TreeView 控件中添加节点。代码如下：

代码 09-18　光盘位置：光盘\mingrisoft\sl\09\18

```
TreeNode StuNode;                                   //定义一个 TreeNode 对象
private void Form1_Load(object sender, EventArgs e)
{
    treeView1.LabelEdit = true;                     //设置 TreeView 控件的 LabelEdit 属性为 true
    treeView1.ShowLines = true;                     //设置 TreeView 控件的 ShowLines 属性为 true
    treeView1.ShowPlusMinus = true;                 //设置 TreeView 控件的 ShowPlusMinus 属性为 true
    treeView1.CheckBoxes = true;                    //设置 TreeView 控件的 CheckBoxes 属性为 true
    StuNode = new TreeNode("学生姓名");             //实例化一个 StuNode 对象
    treeView1.SelectedNode = StuNode;               //设置当前在树视图控件中选定的树节点
    treeView1.Nodes.Add(StuNode);                   //向 TreeView 控件中添加节点
}
private void Add_Click(object sender, EventArgs e)
{
    treeView1.SelectedNode.Nodes.Add(this.StuName.Text);   //向 TreeView 控件中添加新节点
    treeView1.ExpandAll();                          //展开所有的树节点
    this.StuName.Clear();                           //清空文本框中的内容
    this.StuName.Focus();                           //设置文本框中的焦点
}
```

运行本示例，效果如图 9.29 所示。

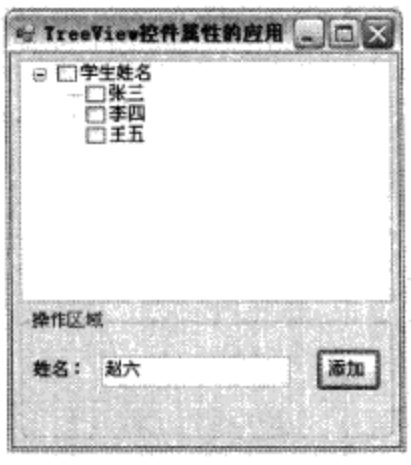


图 9.29　向 TreeView 控件中添加指定节点

3．方法

TreeView 控件常用方法及说明如表 9.17 所示。

表 9.17　　TreeView 控件常用方法及说明

方　法	说　明
CollapseAll	此方法折叠所有树节点。当树节点折叠时，只显示父节点
ExpandAll	此方法展开所有树节点，包括子节点
GetNodeCount	此方法用于获取 Treeview 控件中的节点总数

下面详细介绍 ExpandAll 方法，该方法用于展开所有树节点，包括子节点。

语法：

```
public void ExpandAll ()
```

说明：ExpandAll 方法展开 TreeView 控件中的所有 TreeNode 对象（包括所有子树节点）。

例如，设置展开 treeView1 控件中的所有项，代码如下：

```
this.treeView1.ExpandAll();
```

示例 09-19　单击 TreeView 控件从数据库中获取信息

本示例中，当程序运行时，单击“展开”按钮，调用 ExpandAll 方法展开 TreeView 控件的所有节点。双击控件中的任意节点，程序会自动添加节点下的对应信息。代码如下：

代码 09-19　光盘位置：光盘\mingrisoft\sl\09\19

```
//实例化一个数据库连接对象
SqlConnection con = new SqlConnection("server=(local);integrated security=sspi;database=db_09");
static TreeNode newNode1;                                   //实例化一个静态 TreeNode 对象
private void Form1_Load(object sender, EventArgs e)
{
    newNode1 = this.treeView1.Nodes.Add("部门信息");        //向 TreeView 控件中添加内容
    con.Open();                                             //打开数据库连接
    //实例化一个 SqlDataAdapter 对象
    SqlDataAdapter SelectDepAdapter = new SqlDataAdapter("select * from tb_Department", con);
    DataSet dataSet = new DataSet();                        //实例化一个数据集
    SelectDepAdapter.Fill(dataSet, "Department");           //向数据集中填充数据
    for (int i = 0; i < dataSet.Tables["Department"].Rows.Count; i++)//循环遍历表中的每一行数据
    {
        //向 TreeNode 控件中的根节点中添加数据
        newNode1.Nodes.Add(dataSet.Tables["Department"].Rows[i][1].ToString());
    }
    con.Close();                                            //关闭数据库连接
}
private void treeView1_NodeMouseDoubleClick(object sender, TreeNodeMouseClickEventArgs e)
{
    for (int i = 0; i < newNode1.Nodes.Count; i++)          //循环遍历根节点下的每一个子节点
    {
        string SelectUserString = "Select User_Name,User_Sex,User_Age,User_Phone,User_Address from tb_UserInfo " +"
where tb_UserInfo.Dep_Id = ( Select Dep_Id from tb_Department where Dep_Name='" + newNode1.Nodes[i].Text + "')";
                                                            //实例化一个查询字符串
        //实例化一个 SqlDataAdapter 对象
        SqlDataAdapter SelectUserInfoAdapter = new SqlDataAdapter(SelectUserString, con);
        DataSet UserSet = new DataSet();                    //实例化一个数据集对象
        UserSet.Clear();                                    //清空数据集中的内容
        SelectUserInfoAdapter.Fill(UserSet, "UserInfo");    //向数据集中填充内容
        for (int j = 0; j < UserSet.Tables["UserInfo"].Rows.Count; j++)//循环遍历表中的每一行数据
        {
            //当前节点下的数据记录小于当前表中的数据记录时
            if (newNode1.Nodes[i].Nodes.Count != UserSet.Tables["UserInfo"].Rows.Count)
            {
                //向当前节点下添加内容
                newNode1.Nodes[i].Nodes.Add(UserSet.Tables["UserInfo"].Rows[j][0].ToString());
            }
        }
    }
```

```
}
private void Exit_Click(object sender, EventArgs e)
{
    Application.Exit();            //退出应用程序
}
private void Expand_Click(object sender, EventArgs e)
{
    this.treeView1.ExpandAll();//展开所有树节点
}
```

运行本示例，效果如图 9.30 所示。

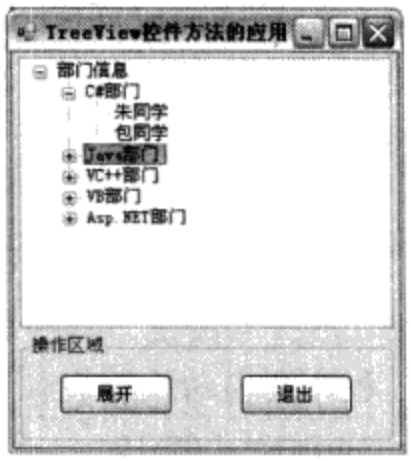


图 9.30　单击 TreeView 控件从数据库中获取信息

4．事件

TreeView 控件常用事件及说明如表 9.18 所示。

表 9.18　　TreeView 控件常用事件及说明

事　件	说　明
AfterCheck	选中显示在树节点旁边的复选框，会触发此事件
AfterCollapse	当树节点折叠时，会触发此事件
AfterExpand	当树节点展开时，会触发此事件
AfterLabelEdit	在编辑树节点标签文本后发生
AfterSelect	当选中树节点时，会触发此事件
BeforeCheck	在选中树节点复选框前发生
BeforeCollapse	在折叠树节点前发生
BeforeExpand	在展开树节点前发生
BeforeLabelEdit	在编辑树节点标签前发生
BeforeSelect	在选定树节点前发生
Click	在单击控件时发生

下面对 TreeView 控件中比较重要的事件进行详细介绍。

（1）AfterCheck 事件。该事件在选中显示在树节点旁边的复选框时发生。

语法：

```
public event TreeViewEventHandler AfterCheck
```

（2）AfterCollapse 事件。该事件在树节点折叠后发生。

语法：

```
public event TreeViewEventHandler AfterCollapse
```

（3）AfterExpand 事件。该事件在树节点展开时发生。

语法：

```
public event TreeViewEventHandler AfterExpand
```

（4）AfterLabelEdit 事件。该事件在编辑树节点标签文本后发生。

语法：

```
public event TreeViewEventHandler AfterLabelEdit
```

（5）AfterSelect 事件。该事件在选中树节点后发生。

语法：

```
public event TreeViewEventHandler AfterSelect
```

（6）BeforeCheck 事件。该事件在选中复选框前发生。

语法：

```
public event TreeViewEventHandler BeforeCheck
```

（7）BeforeCollapse 事件。该事件在树节点折叠前发生。

语法：

```
public event TreeViewEventHandler BeforeCollapse
```

（8）BeforeExpand 事件。该事件在树节点展开前发生。

语法：

```
public event TreeViewEventHandler BeforeExpand
```

（9）BeforeLabelEdit 事件。该事件在编辑树节点标签文本前发生。

语法：

```
public event TreeViewEventHandler BeforeLabelEdit
```

（10）BeforeSelect 事件。该事件在选定树节点前发生。

语法：

```
public event TreeViewEventHandler BeforeSelect
```

（11）Click 事件。该事件在单击控件时发生。

语法：

```
public event EventHandler Click
```

示例 09-20　以信息框的形式显示 TreeView 事件的操作提示

本示例中，进行任何操作都会有对应的信息提示出现。代码如下：

代码 09-20　光盘位置：光盘\mingrisoft\sl\09\20

```
private void Form1_Load(object sender, EventArgs e)
{
    this.treeView1.LabelEdit = true;                    //设置 TreeView 控件可以对树节点的文本进行编辑
    this.treeView1.CheckBoxes = true;                   //设置 TreeView 控件中可以在树节点的旁边显示复选框
    TreeNode ClockNode = new TreeNode("世界名表");//实例化一个树节点
    this.treeView1.Nodes.Add(ClockNode);                //向 TreeView 控件中添加根节点
    this.treeView1.SelectedNode = ClockNode;                    //设置 TreeView 控件中的当前选择项
    string[] Content = { "欧米茄", "浪琴表", "伯爵表", "雷达表", "天梭表", "宝珀", "卡地亚 ", "夏奈尔", "百达翡丽
腕表", "豪华表 ", "劳力士", "MAUBOUSSIN", "爱彼表 ", "萧邦表" };     //实例化一个字符串数组
    for (int i = 0; i < Content.Length; i++)                    //循环遍历数组中的每一项内容
    {
        this.treeView1.SelectedNode.Nodes.Add(Content[i]);      //向 TreeView 控件中添加内容
    }
}
private void treeView1_AfterCheck(object sender, TreeViewEventArgs e)
{
    if (this.treeView1.SelectedNode.Checked == true)            //当 TreeView 控件中的当前项被选定时
    {
        //弹出选择复选框成功的信息提示
        MessageBox.Show("选择复选框成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    else//当 TreeView 控件中的当前项未被选定时
    {
        //弹出取消选择复选框成功提示
        MessageBox.Show("取消选择成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
}
private void treeView1_AfterCollapse(object sender, TreeViewEventArgs e)
{
    //弹出折叠成功的信息提示
```

```
        MessageBox.Show("折叠节点成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_AfterExpand(object sender, TreeViewEventArgs e)
    {
        //弹出展开节点成功的信息提示
        MessageBox.Show("展开节点成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_AfterLabelEdit(object sender, NodeLabelEditEventArgs e)
    {
        //弹出编辑标签内容成功的提示
        MessageBox.Show("编辑标签内容成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
    {
        //弹出节点选择成功的信息提示
        MessageBox.Show("节点选择成功！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_BeforeCheck(object sender, TreeViewCancelEventArgs e)
    {
        //弹出开始选择复选框的提示
        MessageBox.Show("开始选择复选框！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_BeforeCollapse(object sender, TreeViewCancelEventArgs e)
    {
        //弹出开始折叠节点的信息提示
        MessageBox.Show("开始折叠节点！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_BeforeExpand(object sender, TreeViewCancelEventArgs e)
    {
        //弹出开始展开节点的信息提示
        MessageBox.Show("开始展开节点！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_BeforeLabelEdit(object sender, NodeLabelEditEventArgs e)
    {
        //弹出编辑节点文本标签前的信息提示
        MessageBox.Show("编辑节点文本标签前！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_BeforeSelect(object sender, TreeViewCancelEventArgs e)
    {
        //弹出开始选择节点的信息提示
        MessageBox.Show("开始选择节点！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    private void treeView1_Click(object sender, EventArgs e)
    {
        //弹出单击 TreeView 控件的信息提示
        MessageBox.Show("你单击了 TreeView 控件！", "信息提示",
MessageBoxButtons.OK,MessageBoxIcon.Asterisk);
    }
```

运行本示例，效果如图 9.31 所示。

图 9.31　以信息框的形式显示 TreeView 事件的操作提示

9.4.5 NumericUpDown 控件

1．功能

NumericUpDown 控件看起来像是一个文本框与一对箭头的组合，用户可以单击箭头来调整数值。NumericUpDown 控件显示并设置选择列表中的单个数值，用户可以通过单击向上和向下按钮、按向上键和向下键或键入一个数字来增大和减小数字。单击向上键时，值沿最大值方向增加；单击向下键时，值沿最小值方向减少。图 9.32 所示为 NumericUpDown 控件。

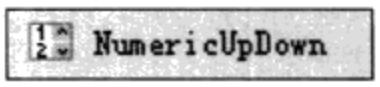


图 9.32　NumericUpDown 控件

2．属性

NumericUpDown 控件常用属性及说明如表 9.19 所示。

表 9.19　　NumericUpDown 控件常用属性及说明

属　性	说　明
AllowDrop	指示控件可以接受用户拖到它上面的数据
Cursor	指针移动过该控件时显示的光标
DecimalOkaces	指定 NumericUpDown 控件内数据的小数位数
Increment	指定 NumericUpDown 控件递增递减的值
Maximum	指定 NumericUpDown 控件可增加到的最大值
Minimum	指定 NumericUpDown 控件可减小到的最小值
Value	指定 NumericUpDown 控件当前的值

下面对 NumericUpDown 控件中比较重要的属性进行详细介绍。

（1）Maximum 属性。该属性表示 NumericUpDown 控件可以增加到的最大值。

语法：

```
public decimal Maximum { get; set; }
```

属性值：数字显示框的最大值。默认值为 100。

例如，设置 numericUpDown1 控件可以增加到的最大值为 100，代码如下：

```
this.numericUpDown1.Maximum=100;
```

（2）Minimum 属性。该属性表示 NumericUpDown 控件可以减小到的最小值。

语法：

```
public decimal Minimum { get; set; }
```

属性值：数字显示框的最小允许值。默认值为 0。

例如，设置 numericUpDown1 控件可以减小到的最小值为 0，代码如下：

```
this.numericUpDown1.Minimum=0;
```

（3）Value 属性。该属性表示 NumericUpDown 控件的当前值。

语法：

```
public decimal Value { get; set; }
```

属性值：NumericUpDown 控件的数值。

例如，设置 numericUpDown1 控件的当前值为 88，代码如下：

```
this.numericUpDown1.Value=88;
```

（4）Increment 属性。该属性表示 NumericUpDown 控件值的变化幅度。

语法：

```
public decimal Increment { get; set; }
```

属性值：单击数字显示框上的向上或向下按钮时，Value 属性要递增或递减的值。默认值为 1。

例如，设置 numericUpDown1 控件的变化幅度为 10，代码如下：

```
this.numericUpDown1.Increment=10;
```

示例 09-21　设置 NumericUpDown 控件的默认值

本示例主要介绍如何在 Windows 窗体中设置 NumericUpDown 控件的属性。代码如下：

代码 09-21　光盘位置：光盘\mingrisoft\sl\09\21

```
private void Form1_Load(object sender, EventArgs e)
{
    this.numericUpDown1.Maximum = 20;        //设置 NumericUpDown 控件的最大值
    this.numericUpDown1.Minimum = 1;         //设置 NumericUpDown 控件的最小值
    this.numericUpDown1.Increment = 2;       //设置 NumericUpDown 控件的递增或递减变量
    this.numericUpDown1.Value = 4;           //设置 NumericUpDown 控件中当前显示的值
}
```

运行本示例，效果如图 9.33 所示。

图 9.33　设置 NumericUpDown 控件的默认值

3．方法

（1）DownButton 方法。该方法表示将 NumericUpDown 控件的值按照 Increment 属性指定数值递减。

语法：

```
public override void DownButton ()
```

说明：调用 DownButton 方法时，无论是在代码中调用还是通过单击向下按钮调用，都对新值进行验证，并以适当的格式用新值对控件进行更新。

例如，使 numericUpDown1 控件按指定的变化幅度递减，代码如下：

```
this.numericUpDown1.DownButton();
```

（2）UpButton 方法。该方法表示将 NumericUpDown 控件的值按照 Increment 属性指定数值递增。

语法：

```
public override void UpButton ()
```

例如，使 numericUpDown1 控件按指定的变化幅度递增，代码如下：

```
this.numericUpDown1.UpButton();
```

示例 09-22　通过按钮改变 NumericUpDown 控件中的值

本示例主要介绍 UpButton 方法和 DownButton 方法的使用。代码如下：

代码 09-22　光盘位置：光盘\mingrisoft\sl\09\22

```
private void Form1_Load(object sender, EventArgs e)
{
    this.numericUpDown1.Maximum = 9999;      //设置 NumericUpDown 控件的最大值
    this.numericUpDown1.Minimum = 1;         //设置 NumericUpDown 控件的最小值
    this.numericUpDown1.Increment = 100;     //设置 NumericUpDown 控件的递增或递减变量
    this.numericUpDown1.Value = 1;           //设置 NumericUpDown 控件中当前显示的值
}
private void Add_Click(object sender, EventArgs e)
{
    this.numericUpDown1.UpButton();          //递增 NumericUpDown 控件中的值
}
private void DeCrease_Click(object sender, EventArgs e)
{
```

```
    this.numericUpDown1.DownButton();        //递减 NumericUpDown 控件中的值
}
```

运行本示例，效果如图 9.34 所示。

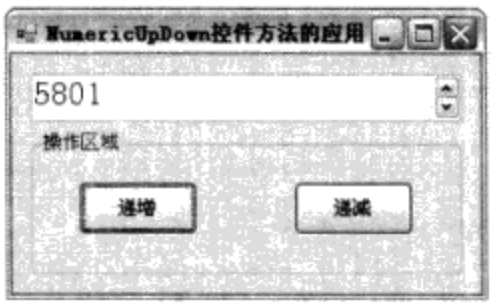


图 9.34　通过按钮改变 NumericUpDown 控件中的值

4．事件

在该控件的事件当中，ValueChanged 事件是比较重要的一个。

该事件在属性值被修改时发生。

语法：

```
public event EventHandler ValueChanged
```

示例 09-23　在信息框中显示 NumericUpDown 控件改变后的值

本示例实现的效果为：当 NumericUpDown 控件的值发生改变时，将改变的值显示在弹出的消息框中。代码如下：

代码 09-23　光盘位置：光盘\mingrisoft\sl\09\23

```
private void Form1_Load(object sender, EventArgs e)
{
    this.numericUpDown1.Maximum = 1024;    //设置 NumericUpDown 控件的最大值
    this.numericUpDown1.Minimum = 1;       //设置 NumericUpDown 控件的最小值
    this.numericUpDown1.Increment = 1;     //设置 NumericUpDown 控件的改变幅度
    this.numericUpDown1.Value = 1;         //设置 NumericUpDown 控件的当前值
}
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    MessageBox.Show(this.numericUpDown1.Value.ToString(), "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                 //弹出信息提示
}
```

运行本示例，效果如图 9.35 所示。

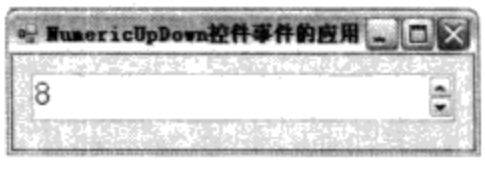


图 9.35　在信息框中显示 NumericUpDown 控件改变后的值

9.5　值类型控件

值类型控件使用户可以设置所显示的任何一个值，该类控件主要包括 CheckBox 控件、CheckedListBox 控件和 RadioButton 控件等。下面对主要的值类型控件进行详细介绍。

9.5.1　CheckBox 控件

1．功能

CheckBox 控件指定某个特定条件是处于打开状态还是处于关闭状态，它常用于为用户提供

"是/否"或"真/假"选项。图 9.36 所示为 CheckBox 控件。

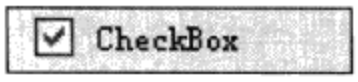


图 9.36　CheckBox 控件

2．属性

CheckBox 控件常用属性及说明如表 9.20 所示。

表 9.20　CheckBox 控件常用属性及说明

属　性	说　明
Appearance	此属性用于指定 CheckBox 控件的外观。可设为 Button 或 Normal
AutoCheck	此属性使单击复选框时自动更改状态
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
Checked	此属性用于指定复选框是否处于选中状态，如果处于选中状态，则设为 true；否则设为 false
CheckState	获取或设置 CheckBox 的状态
Enabled	获取或设置一个值，该值指示控件是否可以对用户交互做出响应
FlatStyle	确定当用户鼠标移到控件上并单击时，控件的外观
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	指定复选框是否允许使用 3 种复选状态，而不是两种，如果设为 true，则复选框的 3 种状态为"已选中"、"未选中"和"中间"

下面对 CheckBox 控件中比较重要的属性进行详细介绍。

（1）Checked 属性。该属性用于指定复选框是否处于选中状态。

语法：

```
public bool Checked { get; set; }
```

属性值：如果 CheckBox 处于选定状态，则为 true；否则为 false。默认值为 false。

例如，设置 checkBox1 控件为选中状态，代码如下：

```
this.checkBox1.Checked=true;
```

（2）FlatStyle 属性。该属性表示获取或设置按钮控件的平面样式外观。

语法：

```
public FlatStyle FlatStyle { get; set; }
```

属性值：FlatStyle 值之一。

FlatStyle 的可选值及说明如下。

- Flat：表示该控件以平面的形式显示。
- Popup：表示该控件以平面显示，直到鼠标指针移动到该控件为止，此时该控件外观为三维。
- Standard：表示该控件的外观为三维的。默认情况下为 Standard。
- System：表示该控件的外观是由用户的操作系统决定的。

注意：如果将 ThreeState 属性设置为 true，则 Checked 属性将为 Checked 或 Indeterminate，CheckState 返回 true。

例如，设置 checkBox1 控件的外观以平面形式显示。代码如下：

```
this.checkBox1.FlatStyle=FlatStyle.Popup;
```

示例 09-24　判断 CheckBox 控件的选中状态以及改变其外观

本示例使用 Checked 属性判断复选框是否选中，并给出提示信息。当单击"修改外观"按钮时，修改 CheckBox 控件的外观。代码如下：

代码 09-24　光盘位置：光盘\mingrisoft\sl\09\24

```
public static int Taste = 0,Interest=0;   //实例化两个 int 型的静态变量
private void Exit_Click(object sender, EventArgs e)
```

```
    {
        Application.Exit();                    //退出应用程序
    }
    private void Submit_Click(object sender, EventArgs e)
    {
        if (Taste >= 1)                        //当爱好中的选定项不小于 1 时
        {
            if (Interest >= 1)                 //当兴趣中的选定项不小于 1 时
            {
                MessageBox.Show("信息填写成功，谢谢合作！", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                     //弹出信息提示
            }
            else                               //当兴趣中的选定项小于 1 时
            {
                MessageBox.Show("兴趣中至少选择一项", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                     //弹出信息提示
            }
        }
        else                                   //当爱好中的选定项小于 1 时
        {
            MessageBox.Show("爱好中至少选择一项", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                     //弹出信息提示
        }
    }
    private void checkBox_CheckedChanged(object sender, EventArgs e)
    {
        foreach (Control control in this.groupBox2.Controls) //循环遍历 groupBox2 中的控件
        {
            //当当前的控件类型为 CheckBox 类型时
            if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
            {
                if (((CheckBox)control).Checked)        //当当前控件为 Checked 时
                {
                    Taste++;                            //累加 Taste 的值
                }
            }
        }
        foreach (Control control in this.groupBox3.Controls) //循环遍历 groupBox3 中的控件
        {
            //当当前的控件类型为 CheckBox 类型时
            if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
            {
                if (((CheckBox)control).Checked)        //当当前控件为 Checked 时
                {
                    Interest++;                         //累加 Interest 的值
                }
            }
        }
    }
    private void Amend_Click(object sender, EventArgs e)
    {
        foreach (Control control in this.groupBox2.Controls) //循环遍历 groupBox2 中的控件
        {
            //当当前的控件类型为 CheckBox 类型时
            if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
            {
                ((CheckBox)control).FlatStyle = FlatStyle.Popup;        //设置按钮控件的平面样式外观
            }
        }
        foreach (Control control in this.groupBox3.Controls)            //循环遍历 groupBox3 中的控件
        {
            //当当前的控件类型为 CheckBox 类型时
            if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
            {
```

```
                ((CheckBox)control).FlatStyle = FlatStyle.Popup;                    //设置按钮控件的平面样式外观
            }
        }
    }
```

运行本示例，效果如图 9.37 所示。

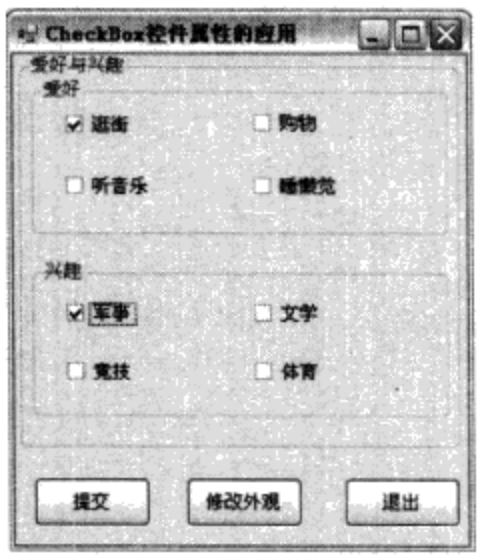


图 9.37　判断 CheckBox 控件的选中状态以及改变其外观

3．方法

下面详细讲解 CheckBox 控件中比较重要的方法。

（1）Focus 方法。该方法为控件输入焦点。

语法：

```
public bool Focus()
```

例如，设置 checkBox1 控件为当前窗体的输入焦点。代码如下：

```
this.checkBox1.Focus();
```

（2）Hide 方法。该方法表示向用户隐藏控件。

语法：

```
public void Hide()
```

例如，设置 checkBox1 控件为向用户隐藏。代码如下：

```
this.checkBox1.Hide();
```

（3）Show 方法。该方法表示向用户显示控件。

语法：

```
public void Show()
```

例如，设置 checkBox1 控件为向用户显示。代码如下：

```
this.checkBox1.Show();
```

4．事件

下面对 CheckBox 控件中比较重要的事件进行详细讲解。

（1）CheckedChanged 事件。该事件在 Checked 属性值被修改时发生。

语法：

```
public event EventHandler CheckedChanged
```

（2）Click 事件。该事件在单击控件时发生。

语法：

```
public event EventHandler Click
```

示例 09-25　在 CheckBox 控件的事件中更改选中状态

本示例通过使用 CheckBox 控件的事件实现全选和取消。代码如下：

代码 09-25　光盘位置：光盘\mingrisoft\sl\09\25

```
private void CheckAll_CheckedChanged(object sender, EventArgs e)
{
```

```
        if (CheckAll.Checked ==true)                          //当全选复选框处于选中状态时
        {
            foreach (Control control in this.groupBox1.Controls)   //循环遍历 groupBox1 中的控件
            {
                //当当前的控件类型为 CheckBox 类型时
                if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
                {
                    ((CheckBox)control).Checked = true;  //设置 groupBox1 控件中的 CheckBox 控件为选中状态
                }
            }
        }
    }
    private void CheckAll_Click(object sender, EventArgs e)
    {
        this.CheckAll.Enabled = false;                        //设置全选复选框为不可用状态
        this.CheckNone.Enabled = true;                        //设置取消复选框为可用状态
        this.CheckNone.Checked = false;                       //设置取消复选框为未选中状态
    }
    private void CheckNone_CheckedChanged(object sender, EventArgs e)
    {
        if (CheckNone.Checked == true)                        //当取消复选框为选中状态时
        {
            foreach (Control control in this.groupBox1.Controls)   //循环遍历 groupBox1 中的控件
            {
                //当当前的控件类型为 CheckBox 类型时
                if (control.GetType().ToString() == "System.Windows.Forms.CheckBox")
                {
                    ((CheckBox)control).Checked = false; //设置 groupBox1 控件中的 CheckBox 控件为未选中状态
                }
            }
        }
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        this.CheckNone.Enabled = false;                       //设置取消复选框为不可用状态
    }
    private void CheckNone_Click(object sender, EventArgs e)
    {
        this.CheckNone.Enabled = false;                       //设置取消复选框为不可用状态
        this.CheckAll.Enabled = true;                         //设置全选复选框为可用状态
        this.CheckAll.Checked = false;                        //设置全选复选框为未选中状态
    }
```

运行本示例，效果如图 9.38 所示。

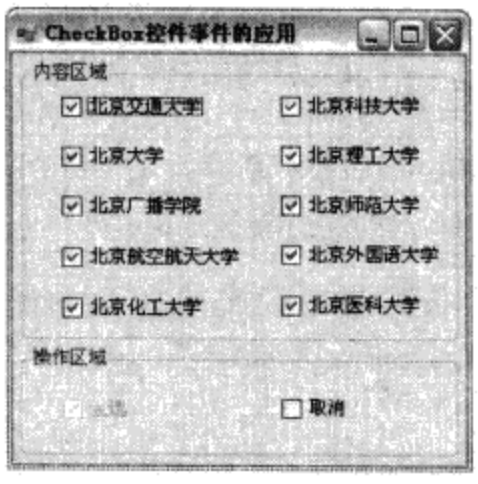


图 9.38　在 CheckBox 控件的事件中更改选中状态

9.5.2　RadioButton 控件

RadioButton 控件，即单选按钮，它为用户提供由两个或多个互斥选项组成的选项集。虽然单选按钮和复选框的功能看似类似，实际上却存在很大差异。当用户选择某单选按钮时，同一

组中的其他单选按钮不能同时选定；而在复选框中可以。图 9.39 所示为 RadioButton 控件。

图 9.39　RadioButton 控件

交叉链接：RadioButton 控件的属性、方法和事件与 CheckBox 控件类似，这里不再详细说明，可参见 9.5.1 小节。

9.5.3　CheckedListBox 控件

1．功能

CheckedListBox 控件扩展了 ListBox 控件的功能，它几乎能完成列表框可以完成的所有任务，并且还可以在列表中的项旁边显示复选标记。CheckedListBox 控件与 ListBox 控件的主要区别在于复选列表框只能有一项选中或未选中任何项。图 9.40 所示为 CheckedListBox 控件。

图 9.40　CheckedListBox 控件

注意：选定的项在窗体上突出显示，与已选中的项不同。

2．属性

CheckedListBox 控件常用属性及说明如表 9.21 所示。

表 9.21　　CheckedListBox 控件常用属性及说明

属　性	说　明
CheckedIndices	此属性是指 CheckedListBox 控件中所有选中项索引的集合
CheckedItems	此属性是指 CheckedListBox 控件中所有选中项的集合
CheckOnClick	获取或设置一个值，该值指示当选定项时是否应切换复选框
HorizontalScrollbar	获取或设置一个值，该值指示是否在控件中显示水平滚动条
Items	向控件添加项
MaximumSize	获取或设置大小，该大小是 GetPreferredSize 可以指定的上限。GetPreferredSize 表示可以容纳控件的矩形区域的大小
MinimumSize	获取或设置大小，该大小是 GetPreferredSize 可以指定的下限
MultiColumn	获取或设置一个值，该值指示 ListBox 是否支持多列
ScrollAlwaysVisible	获取或设置一个值，该值指示是否任何时候都显示垂直滚动条
SelectedIndex	获取或设置 ListBox 中当前选定项的从零开始的索引
SelectedIndices	获取一个集合，该集合包含 ListBox 中所有当前选定项的从零开始的索引
SelectedItem	获取或设置 ListBox 中的当前选定项
SelectedItems	获取包含 ListBox 中当前选定项的集合
SelectedValue	获取或设置由 ValueMember 属性指定的成员属性的值
SelectionMode	获取或设置指定选择模式的值
Sorted	获取或设置一个值，该值指示 ListBox 中的项是否按字母顺序排序
ThreeDCheckBoxes	获取或设置一个值，该值指示复选框是否有 Flat 或 Normal

下面对 CheckedListBox 控件中比较重要的属性进行详细介绍。

（1）Items 属性。该属性表示向控件中添加项。

语法：

```
public ObjectCollection Items { get; }
```

属性值：代表 CheckedListBox 中项的 CheckedListBox.ObjectCollection 集合。

例如，向 checkedListBox1 控件中添加一项“德克萨斯州”，默认情况下为选中状态。代码如下：

```
this.checkedListBox1.Items.Add("德克萨斯州",true);
```

（2）CheckedItems 属性。该属性表示 CheckedListBox 控件中所有选中项的集合。

语法：

```
public CheckedItemCollection CheckedItems { get; }
```

属性值：CheckedListBox 的 CheckedListBox.CheckedItemCollection 集合。

（3）MultiColumn 属性。该属性表示获取或设置一个值，该值指示是否多列显示 CheckedListBox 控件。

语法：

```
public bool MultiColumn { get; set; }
```

属性值：当 CheckedListBox 控件允许多列显示时，该值为 true；否则，该值为 false。

例如，设置 checkedListBox1 控件为多列显示内容。代码如下：

```
this.checkedListBox1.MultiColumn=true;
```

（4）SelectionMode 属性。该属性表示获取或设置选择模式的值。

语法：

```
public override SelectionMode SelectionMode { get; set; }
```

属性值：SelectionMode 值之一。

SelectionMode 的可取值及说明如下。

- None：表示无法选择项。
- One：表示允许选择一项。
- MultiSimple：表示可以选择多项。
- MultiExtened：表示可以选择多项，并且用户可使用 Shift 键、Ctrl 键和箭头键来进行选择。

例如，设置 checkedListBox1 控件为多项选择模式。代码如下：

```
this.checkedListBox1.SelectionMode=SelectionMode.MultiExtened;
```

（5）Sorted 属性。该属性表示获取或设置一个值，该值指示 ListBox 中的项是否按字母排序。

语法：

```
public bool Sorted { get; set; }
```

属性值：如果排序控件中的项，则为 true；否则为 false。默认为 false。

例如，设置 checkedListBox1 控件中的内容按字母排序，代码如下：

```
this.checkedListBox1.Sorted=true;
```

示例 09-26　向 CheckedListBox 控件中添加 CheckBox 控件

本示例使用 CheckedListBox 控件添加多个 CheckBox 控件。代码如下：

代码 09-26　光盘位置：光盘\mingrisoft\sl\09\26

```
private void Form1_Load(object sender, EventArgs e)
{
    this.checkedListBox1.MultiColumn = true;                    //设置 CheckedListBox 控件支持多列选择
    this.checkedListBox1.ScrollAlwaysVisible = true;            //设置 CheckedListBox 控件的进度条总是显示
    this.checkedListBox1.SelectionMode = SelectionMode.One;     //设置 CheckedListBox 控件为单选模式
    this.checkedListBox1.Sorted = true;                         //设置 CheckedListBox 控件的排序模式
    this.checkedListBox1.ThreeDCheckBoxes = true;//设置 CheckedListBox 控件中的复选框具有 Flat 和 Normal
    string[] Content = { "密歇根州", "明尼苏达州", "密西西比州" ,"密苏里州","蒙大拿州","内华达州","新罕布什
尔州"+,"新泽西州","新墨西哥州","纽约州","北卡罗来州","北达科他州","俄亥俄州","俄克拉荷马州","俄勒冈州","宾夕法
尼亚州","罗得州","南卡罗来州","南达科他州","田纳西州"};      //实例化一个字符串数组
    for (int i = 0; i < Content.Length ; i++)                   //循环遍历数组中的每一个元素
    {
        this.checkedListBox1.Items.Add(Content[i], false);      //向 CheckedListBox 控件中添加数组中的元素
    }
}
```

运行本示例，效果如图 9.41 所示。

3．方法

CheckedListBox 控件常用方法及说明如表 9.22 所示。

图 9.41　向 CheckedListBox 控件中添加 CheckBox 控件

表 9.22　　CheckedListBox 控件常用方法及说明

方　法	说　明
ClearSelected	取消选择 ListBox 中的所有项
GetItemChecked	返回指示指定项是否选中的值
GetItemCheckState	返回指示当前项的复选状态的值
GetItemText	返回指定项的文本表示形式
GetSelected	返回一个值，该值指示是否选定了指定的项
SetItemChecked	将指定索引处的项的 CheckState 设置为 Checked
SetItemCheckState	设置指定索引处项的复选状态
SetItemsCore	清除 ListBox 的内容，并向控件中添加指定项
SetSelected	选择或清除对 ListBox 中指定项的选定
SetTopLevel	将控件设置为顶级控件
Sort	对 ListBox 中的项排序

下面对 CheckedListBox 控件中比较重要的方法进行详细介绍。

（1）ClearSelected 方法。该方法表示取消选择 CheckedListBox 控件中的所有项。

语法：

```
public void ClearSelected()
```

例如，取消 checkedListBox1 控件中的所有选择项。代码如下：

```
this.checkedListBox1.ClearSelected();
```

（2）GetItemChecked 方法。该方法表示判断指定项是否为选中状态。

语法：

```
public bool GetItemChecked(int index)
```

参数说明如下。

index：表示项的索引。

返回值：若当前项处于选中状态，则返回值为 true；否则，该值为 false。

例如，判断第 1 项是否处于选中状态。代码如下：

```
this.checkedListBox1.GetItemChecked(1);
```

（3）GetItemText 方法。该方法表示返回指定项的文本表示形式。

语法：

```
public string GetItemText(Object item)
```

参数说明如下。

item：表示从中要显示内容的对象。

返回值：表示项的文本表示形式。

例如，返回项“喜马拉雅山”的文本表示形式。代码如下：

```
this.checkedListBox1.GetItemText("喜马拉雅山");
```

（4）GetSelected 方法。该方法表示返回一个值，指示是否选定了指定的项。

语法：

```
public bool GetSelected(int index)
```

参数说明如下。

Index：项从零开始的索引，根据该索引确定该项是否被选定。

返回值：若当前项处于选中状态，则返回值为 true；否则，该值为 false。

例如，判断第 8 项是否处于选定状态。代码如下：

```
this.checkedListBox1.GetSelected(8);
```

（5）SetItemCheckState 方法。该方法表示设置当前的复选状态。

语法：

```
public void SetItemCheckState (int index,CheckState value)
```

参数说明如下。

index：要为其设置状态的项的索引。

value：CheckState 值之一。

例如，设置 checkedListBox1 控件中第 17 项的复选状态为选中状态。代码如下：

```
this.checkedListBox1.SetItemCheckState(16, CheckState.Checked);
```

（6）SetSelected 方法。该方法表示选择或清除 ListBox 控件中对指定项的选择。

语法：

```
public void SetSelected(int index,bool value)
```

参数说明如下。

index：表示 ListBox 中要选择或清除对其选定的项的从零开始的索引。

value：如果为 true，则选择指定的项；否则为 false。

例如，取消对 checkedListBox1 控件中第 9 项的选择。代码如下：

```
this.checkedListBox1.SetSelected(8,false);
```

示例 09-27　CheckedListBox 控件方法的应用

本示例中，当程序运行时，单击“判断”按钮来判断当前项是否被选定，单击“取消”按钮，取消对当前项的选定，单击“返回”按钮返回当前选定项的文本内容，单击“全部取消”按钮取消 CheckedListBox 控件中的所有选定项。Form_Load 事件中的代码与示例 09-26 中的相同，其他控件对应的代码如下：

代码 09-27　光盘位置：光盘\mingrisoft\sl\09\27

```
private void Judge_Click(object sender, EventArgs e)
{
    if (this.checkedListBox1.SelectedIndex==-1)          //当没有选择项存在时
    {
        //弹出对应的信息提示
        MessageBox.Show("不存在选定的项", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    else                                                 //当有选择项存在时
    {
        if (this.checkedListBox1.GetSelected(this.checkedListBox1.SelectedIndex))      //当当前项被选定时
        {
            //弹出选定的信息提示
            MessageBox.Show("该项被选定", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }
        else                                             //当没有项目被选定时
        {
            //弹出对应的信息提示
            MessageBox.Show("该项未被选定", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }
    }
}
private void AllCancel_Click(object sender, EventArgs e)
{
    this.checkedListBox1.ClearSelected();                //取消所有选定的项
```

```
    }
    private void Return_Click(object sender, EventArgs e)
    {
        if (this.checkedListBox1.GetItemText(this.checkedListBox1.SelectedItem) == null || this.checkedListBox1.GetItemText(this.
checkedListBox1.SelectedItem) == "")                                    //当选定项为空或者不存在时
        {
            MessageBox.Show("选定的内容为空或者不存在", "信息提示");          //弹出相应的信息提示
        }
        else                                                            //当选定的内容存在时
        {
            MessageBox.Show("当前选择的内容为：\n" + this.checkedListBox1.GetItemText(this.checkedListBox1.
SelectedItem), "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);   //弹出信息提示
        }
    }
    private void Cancel_Click(object sender, EventArgs e)
    {
        this.checkedListBox1.SetSelected(this.checkedListBox1.SelectedIndex, false); //取消对当前项的选定
    }
```

运行本示例，效果如图 9.42 所示。

图 9.42　CheckedListBox 控件方法的应用

4．ItemCheck 事件

该事件当前复选框状态更改时发生。

语法：

```
public event ItemCheckEventHandler ItemCheck
```

示例 09-28　在消息框中显示选定项的内容

本示例中，当程序运行时，单击控件中的项，弹出对话框提示用户所选择项的内容。Form1_Load 事件中的代码与代码 09-26 中的一样，这里只书写其他部分的代码。代码如下：

代码 09-28　光盘位置：光盘\mingrisoft\sl\09\28

```
private void checkedListBox1_ItemCheck(object sender, ItemCheckEventArgs e)
{
    MessageBox.Show(this.checkedListBox1.SelectedItem.ToString(),"信息提示");//弹出选定项的内容
}
```

运行本示例，效果如图 9.43 所示。

图 9.43　在消息框中显示选定项的内容

9.6 图形显示类型控件

图形显示类控件用于显于位图、GIF、JPEG、图元文件和图标等，它们一次只能显示一张图片，如 PictureBox 控件和 NotifyIcon 控件等。下面将对主要的图形显示类控件进行详细介绍。

9.6.1 PictureBox 控件

1．功能

PictureBox 控件用于显示位图、GIF、JPEG、图元文件和图标格式的图形。图 9.44 所示为 PictureBox 控件。

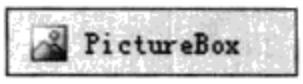


图 9.44 PictureBox 控件

2．属性

PictureBox 控件常用的属性及说明如表 9.23 所示。

表 9.23 PictureBox 控件常用属性及说明

属　性	说　明
ErrorImage	加载图片失败时显示的图片
Image	用于指定图片框中的图片
ImageLocation	获取或设置要在 PictureBox 中显示的图像的路径或 URL
SizeMode	控件 PictureBox 将如何处理图像位置和大小
Text	获取或设置 PictureBox 的文本

下面对 PictureBox 控件中比较重要的属性进行详细介绍。

（1）ErrorImage 属性。该属性表示获取或设置加载失败时 PictureBox 控件中显示的图片。

语法：

```
public Image ErrorImage { get; set; }
```

属性值：表示在图像加载过程中发生错误或图像加载取消时要显示的 Image。

例如，设置 pictureBox1 控件加载失败时显示的图片为“1.jpg”。代码如下：

```
this.pictureBox1.ErrorImage=(Image)Properties.Resources._1;
```

（2）Image 属性。此属性用于指定图片框中的图片。

语法：

```
public Image Image { get; set; }
```

属性值：要显示的 Image。

例如，设置 pictureBox1 控件图片框中的图片为“1.bmp”。代码如下：

```
this.pictureBox1.Image=(Image)Properties.Resources._1;
```

（3）SizeMode 属性。该属性用于设置 PictureBox 控件将如何处理图像位置和大小。

语法：

```
public PictureBoxSizeModeSizeMode { get; set; }
```

属性值：PictureBoxSizeMode 值之一，默认为 Normal。

例如，设置 PictureBox1 控件中图片显示模式为自动调整以适应 pictureBox1 控件的大小。代码如下：

```
this.pictureBox1.SizeMode=SizeMode.StretchImage;
```

PictureBoxSizeMode：一个返回值，用来处理图像位置和大小。PictureBoxSizeMode 的属性值及说明如表 9.24 所示。

表 9.24 PictureBoxSizeMode 的属性值及说明

属 性 值	说　明
AutoSize	调整 PictureBox 大小，使其等于所包含的图像大小，则图像将居中显示。如果图像比 PictureBox 大，则图片将居于 PictureBox 中心，而外边缘将被剪裁

续表

属 性 值	说 明
CenterImage	如果 PictureBox 比图像大，则图像将居中显示
Normal	图像被置于 PictureBox 的左上角。如果图像比包含它的 PictureBox 大，则该图像将被剪裁掉
StretchImage	PictureBox 中的图像被拉伸或收缩，以适合 PictureBox 的大小
Zoom	图像大小按其原有的大小比例被增加或减小

示例 09-29 在 PictureBox 控件中显示指定的图像

本示例中，当程序运行时，通过 PictureBox 控件的 Image 属性设置要在控件中显示的图片。代码如下：

代码 09-29 光盘位置：光盘\mingrisoft\sl\09\29

```
private void Form1_Load(object sender, EventArgs e)
{
    this.pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage; //设置图片在 PictureBox 控件中的显示方式
    //设置当加载图片过程中出现错误导致加载失败时显示的图片
    this.pictureBox1.ErrorImage = (Image)Properties.Resources.Image_1;
    this.pictureBox1.Image = (Image)Properties.Resources.Image_2;  //设置 PictureBox 控件中显示的图片
}
```

运行本示例，效果如图 9.45 所示。

图 9.45 在 PictureBox 控件中显示指定的图像

3．CancelAsync 方法

该方法主要用于取消异步图像加载。

语法：

```
public void CancelAsync()
```

例如，在 pictureBox1 控件中设置图像的加载方式为异步图像加载。代码如下：

```
this.pictureBox1.CancelAsync();
```

4．事件

PictureBox 控件常用事件及说明如表 9.25 所示。

表 9.25 **PictureBox 控件常用事件及说明**

事 件	说 明
Click	当单击图片时，发生的事件
DoubleClick	在双击控件时发生
MouseEnter	在鼠标指针进入控件时发生
MouseLeave	在鼠标指针离开控件时发生
SizeModeChanged	当 SizeMode 属性修改时，会发生此事件。SizeMode 属性可通过编程方式修改
TextChanged	当 Text 属性被修改时，发生的事件

下面对 PictureBox 控件中比较重要的事件进行详细介绍。

（1）Click 事件。该事件在单击图片时发生。

语法：

```
public event EventHandler Click
```

（2）DoubleClick 事件。该事件在双击控件时发生。

语法：

```
public event EventHandler DoubleClick
```

（3）MouseEnter 事件。该事件在鼠标进入该控件时发生。

语法：

```
public event EventHandler MouseEnter
```

（4）MouseLeave 事件。该事件在鼠标离开控件时发生。

语法：

```
public event EventHandler MouseLeave
```

示例 09-30　单击 PictureBox 控件显示当前系统时间

本示例中，当程序运行时，单击时钟图片，在窗体下方显示当前时间。代码如下：

代码 09-30　光盘位置：光盘\mingrisoft\sl\09\30

```
private void Form1_Load(object sender, EventArgs e)
{
    this.timer1.Interval = 1000;                                    //设置 timer1 控件的 Tick 事件发生的频率
    this.pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage; //设置图片在 PictureBox 控件中的显示方式
    //设置当加载图片过程中出现错误导致加载失败时显示的图片
    this.pictureBox1.ErrorImage = (Image)Properties.Resources.Image_1;
    this.pictureBox1.Image = (Image)Properties.Resources.Image_2;  //设置 PictureBox 控件中显示的图片
}
private void pictureBox1_Click(object sender, EventArgs e)
{
    this.timer1.Start();                                            //启动计时器
}
private void timer1_Tick(object sender, EventArgs e)
{
    this.label1.Text = "当前系统时间为：" + DateTime.Now.ToString();    //在 Label 控件中显示系统时间
}
private void pictureBox1_DoubleClick(object sender, EventArgs e)
{
    Application.Exit();                                             //退出应用程序
}
private void pictureBox1_MouseEnter(object sender, EventArgs e)
{
    this.pictureBox1.Image = (Image)Properties.Resources.Image_3;   //更换 PictureBox 控件中显示的图片
}
private void pictureBox1_MouseLeave(object sender, EventArgs e)
{
    this.pictureBox1.Image = (Image)Properties.Resources.Image_2;   //更换 PictureBox 控件中显示的图片
}
```

运行本示例，效果如图 9.46 所示。

图 9.46　单击 PictureBox 控件显示当前系统时间

9.6.2　NotifyIcon 控件

1．功能

NotifyIcon 控件通常用于显示在后台运行的进程的图标，这些进程大部分时间不显示用户

界面。每个 NotifyIcon 控件都在状态栏区域显示一个图标。如果有 3 个后台进程，并希望为每个后台进程各显示一个图标，则需要添加 3 个 NotifyIcon 控件。图 9.47 所示为 NotifyIcon 控件。

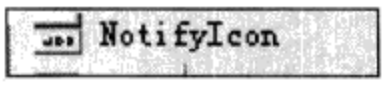


图 9.47　NotifyIcon 控件

2．属性

NotifyIcon 控件常用属性及说明如表 9.26 所示。

表 9.26　　NotifyIcon 控件常用属性及说明

属　　性	说　　明
BalloonTipIcon	获取或设置要显示在与 NotifyIcon 关联的气球提示上的图标
BalloonTipText	获取或设置要显示在与 NotifyIcon 关联的气球提示上的文本
BalloonTipTitle	获取或设置显示在 NotifyIcon 上的气球提示的标题
Icon	获取或设置当前图标
Text	此属性设置当鼠标悬停在图标上时显示的文字
Visible	获取或设置一个值，指示图标在任务栏的通知区域中是否可见

下面对 NotifyIcon 控件中比较重要的属性进行详细介绍。

（1）BalloonTipIcon 属性。该属性用来获取或设置要显示在与 NotifyIcon 关联的气球提示上的图标。

语法：

```
public ToolTipIcon BalloonTipIcon { get; set; }
```

属性值：表示要显示在与 NotifyIcon 关联的气球提示上的 ToolTipIcon。

例如，设置 notifyIcon1 控件气球提示中的图标为信息图标。代码如下：

```
this.notifyIcon1.BalloonTipIcon = ToolTipIcon.Info;
```

（2）BalloonTipText 属性。该属性用来获取或设置要显示在与 NotifyIcon 关联的气球提示上的文本。

语法：

```
public string BalloonTipText { get; set; }
```

属性值：表示要显示在与 NotifyIcon 关联的气球提示上的文本。

例如，设置 notifyIcon1 控件中气球提示中的文本为“明日编程词典”。代码如下：

```
this.notifyIcon1.BalloonTipText="明日编程词典";
```

（3）BalloonTipTitle 属性。该属性用来获取或设置显示在 NotifyIcon 上的气球提示的标题。

语法：

```
public string BalloonTipTitle { get; set; }
```

属性值：显示为气球提示的标题的文本。

例如，设置 notifyIcon1 控件中的气球提示标题为“友情提示”。代码如下：

```
this.notifyIcon1.BalloonTipTitle="友情提示";
```

（4）Icon 属性。该属性用来获取或设置当前图标。

语法：

```
public Icon Icon { get; set; }
```

属性值：由 NotifyIcon 组件显示的 Icon。默认值为 null。

例如，设置 notifyIcon1 控件的图标为“1.bmp”。代码如下：

```
this.notifyIcon1.Icon=Properties.Resources._1;
```

（5）Text 属性。该属性表示当鼠标悬停在图标上时显示的文本。

语法：

```
public string Text { get; set; }
```

属性值：当鼠标指针停留在通知区域图标上时显示的工具提示文本。

例如，设置当鼠标悬停在 notifyIcon1 控件上时显示的文本为“欢迎使用 NotifyIcon 控件！”，

代码如下：

```
this.notifyIcon1.Text="欢迎使用 NotifyIcon 控件！";
```

（6）Visible 属性。该属性表示获取或设置一个值，指示图标在任务栏的通知区域中是否可见。

语法：

```
public bool Visible { get; set; }
```

属性值：如果图标在通知区域中可见，则为 true；否则为 false。默认值为 false。

例如，设置在任务栏中显示 notifyIcon1 控件的图标。代码如下：

```
this.notifyIcon.Visible=true;
```

示例 09-31　NotifyIcon 控件属性的应用

本示例中，当程序运行时，在状态栏中显示一个小人图标，在图标的旁边会出现一个气泡提示，该气泡提示的显示时间为 30ms。当鼠标放在图标上时，会显示“该控件为 NotifyIcon 控件”的信息提示。代码如下：

代码 09-31　光盘位置：光盘\mingrisoft\sl\09\31

```
private void Form1_Load(object sender, EventArgs e)
{
    this.notifyIcon1.Text = "该控件为 NotifyIcon 控件";          //设置当鼠标放在图标上时显示的文本
    this.notifyIcon1.Icon = Properties.Resources._1;             //设置 NotifyIcon 控件显示的图标
    this.notifyIcon1.BalloonTipIcon = ToolTipIcon.Info;          //设置 NotifyIcon 控件气泡提示窗口中的图标
    this.notifyIcon1.BalloonTipTitle = "友情提示";               //设置 NotifyIcon 控件气泡提示窗口中的文字标题
    //设置 NotifyIcon 控件气泡提示窗口中的文本内容
    this.notifyIcon1.BalloonTipText = "欢迎使用 NotifyIcon 控件";
    this.notifyIcon1.ShowBalloonTip(30);                         //设置显示气泡的时间，单位为毫秒
}
```

运行本示例，效果如图 9.48 和图 9.49 所示。

图 9.48　NotifyIcon 控件属性的应用

图 9.49　NotifyIcon 控件属性的应用

3．ShowBalloonTip 方法

该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。

语法：

```
public void ShowBalloonTip (int timeout,string tipTitle,string tipText,ToolTipIcon tipIcon)
```

参数说明如下。

timeout：气球提示显示的时间长度（以 ms 为单位）。

tipTitle：要在气球提示上显示的标题。

tipText：要在气球提示上显示的文本。

tipIcon：ToolTipIcon 值之一。

说明：有关 ShowBalloonTip 方法的使用，请参考可实例 09-31。

4．事件

NotifyIcon 控件常用事件及说明如表 9.27 所示。

表 9.27　NotifyIcon 控件常用事件及说明

事　件	说　明
BalloonTipClicked	在单击气球提示时发生
BalloonTipClosed	在用户关闭气球提示时发生

续表

事　件	说　明
BalloonTipShown	当气球提示显示在屏幕上时发生
Click	当用户单击通知区域中的图标时发生
DoubleClick	当用户双击任务栏的通知区域中的图标时发生

下面对 NotifyIcon 控件中比较重要的事件进行详细介绍。

（1）BalloonTipClicked 事件。该事件在单击气泡提示时发生。

语法：

```
public event EventHandler BalloonTipClicked
```

（2）BalloonTipClosed 事件。该事件在用户关闭气泡提示时发生。

语法：

```
public event EventHandler BalloonTipClosed
```

（3）BalloonTipShown 事件。该事件在气泡提示显示在屏幕上时发生。

语法：

```
public event EventHandler BalloonTipShown
```

（4）Click 事件。该事件在单击通知区域的图标时发生。

语法：

```
public event EventHandler Click
```

（5）DoubleClick 事件。该事件在用户双击任务栏的通知区域中的图标时发生。

语法：

```
public event EventHandler DoubleClick
```

示例 09-32　在文本框中显示 NotifyIcon 控件的事件提示

本示例中，当程序运行时，通过对 NotifyIcon 控件的操作，从而引发一系列事件。当事件发生时，在相对应的文本框中显示提示性信息。Form_Load 中的代码与示例 09-31 中的代码一致，这里不再给出。其他代码如下：

代码 09-32　光盘位置：光盘\mingrisoft\sl\09\32

```
private void notifyIcon1_BalloonTipClicked(object sender, EventArgs e)
{
    this.Clicked.ForeColor = Color.Violet;                    //设置文本框的文字颜色
    this.Clicked.BackColor = Color.Yellow;                    //设置文本框的背景颜色
    this.Clicked.ReadOnly = true;                             //设置文本框为只读的
    this.Clicked.Text = "单击气泡提示事件已经发生…";          //设置文本框中显示的文字
}
private void notifyIcon1_BalloonTipClosed(object sender, EventArgs e)
{
    this.CloseInfo.ForeColor = Color.Pink;                    //设置文本框的文字颜色
    this.CloseInfo.BackColor = Color.Snow;                    //设置文本框的背景颜色
    this.CloseInfo.ReadOnly = true;                           //设置文本框为只读的
    this.CloseInfo.Text = "气泡关闭事件已经发生…";            //设置文本框中显示的文字
}
private void notifyIcon1_BalloonTipShown(object sender, EventArgs e)
{
    this.ShownInfo.ForeColor = Color.SkyBlue;                 //设置文本框的文字颜色
    this.ShownInfo.BackColor = Color.Brown;                   //设置文本框的背景颜色
    this.ShownInfo.ReadOnly = true;                           //设置文本框为只读的
    this.ShownInfo.Text = "托盘图标已显示…";                  //设置文本框中显示的文字
}
private void notifyIcon1_Click(object sender, EventArgs e)
{
    this.ClickInfo.ForeColor = Color.Orchid;                  //设置文本框的文字颜色
    this.ClickInfo.BackColor = Color.MediumSpringGreen;       //设置文本框的背景颜色
```

```
        this.ClickInfo.ReadOnly = true;                          //设置文本框为只读的
        this.ClickInfo.Text = "托盘图标单击事件已经发生…";          //设置文本框中显示的文字
    }
    private void notifyIcon1_DoubleClick(object sender, EventArgs e)
    {
        this.DoubleClickInfo.ForeColor = Color.Linen;            //设置文本框的文字颜色
        this.DoubleClickInfo.BackColor = Color.LightSteelBlue;   //设置文本框的背景颜色
        this.DoubleClickInfo.ReadOnly = true;                    //设置文本框为只读的
        this.DoubleClickInfo.Text = "托盘图标上双击事件已经发生…";//设置文本框中显示的文字
    }
```

运行本示例，效果如图 9.50 所示。

图 9.50　在文本框中显示 NotifyIcon 控件的事件提示

9.7　日期类型控件

允许用户选取一个或多个 DateTime 类型的实例控件，该类控件主要包括 DateTimePicker 和 MonthCalender 控件。下面分别对其进行详细介绍。

9.7.1　DateTimePicker 控件

1．功能

用户可以从日期或时间列表中选择单个项，用来表示日期。该控件显示为两部分：一部分为下拉列表（以文本形式表示的日期），另一部分为网格（在单击列表旁边的向下箭头时显示）。图 9.51 所示为 DateTimePicker 控件。

图 9.51　DateTimePicker 控件

2．属性

DateTimePicker 控件常用属性及说明如表 9.28 所示。

表 9.28　　DateTimePicker 控件常用属性及说明

属　　性	说　　明
CalendarFont	获取或设置应用于日历的字体样式
CalendarForeColor	获取或设置日历的前景色
CalendarMonthBackground	获取或设置日历中月份的背景色
CalendarTitleBackColor	获取或设置日历标题的背景色
CalendarTitleForeColor	获取或设置日历标题的前景色
CalendarTrailingForeColor	获取或设置日历结尾日期的前景色
Checked	获取或设置一个值，该值指示是否已用有效日期/时间值设置了 Value 属性且显示的值可以更新
CustomFormat	获取或设置自定义日期/时间格式字符串
MaxDate	获取或设置可在控件中选择的最大日期和时间

续表

属　性	说　明
MaximumDateTime	获取 DateTimePicker 控件允许的最大日期值
MinDate	获取或设置可在控件中选择的最小日期和时间
MinimumDateTime	获取 DateTimePicker 控件允许的最小日期值
ShowCheckBox	获取或设置一个值，该值指示在选定日期的左侧是否显示一个复选框
ShowUpDown	获取或设置一个值，该值指示是否使用数值调节钮控件（也称为 up-down 控件）调整日期/时间值
Value	获取或设置分配给控件的日期/时间值。

下面对 DateTimePicker 控件中比较重要的属性进行详细的介绍。

（1）Checked 属性。该属性表示获取或设置一个值，该值指示是否已用有效日期/时间设置了 Value 属性且值可以更新。

语法：

```
public bool Checked { get; set; }
```

属性值：如果 Value 属性已用有效的 DateTime 值设置并且能更新显示的值，则为 true；否则为 false。默认为 true。

例如，设置 dateTimePicker1 控件已用有效日期/时间设置了 Value 属性且值可以更新。代码如下：

```
this.dateTimePicker1.Checked=true;
```

（2）ShowCheckBox 属性。该属性表示获取或设置一个值，该值指示在选定日期的左侧是否显示一个复选框。

语法：

```
public bool ShowCheckBox { get; set; }
```

属性值：如果在选定日期的左边显示复选框，则为 true；否则为 false。默认为 false。

例如，设置在 dateTimePicker1 控件的左侧不显示复选框。代码如下：

```
this.dateTimePicker1.ShowCheckBox=false;
```

（3）ShowUpDown 属性。该属性表示获取或设置一个值，该值指示是否使用数值调节钮控件（也称为 up-down 控件）调整日期/时间值。

语法：

```
public bool ShowUpDown { get; set; }
```

属性值：如果数值调节钮控件用于调整日期/时间值，则为 true；否则为 false。默认为 false。

例如，设置在 dateTimePicker1 控件中使用数值调节钮控件。代码如下：

```
this.dateTimePicker1.ShowUpDown=true;
```

（4）Value 属性。该属性是指定当前控件的日期或时间值。

语法：

```
public DateTime Value { get; set; }
```

属性值：分配给控件的 DateTime 值。

例如，设置 dateTimePicker1 控件中的值为当前系统时间。代码如下：

```
this.dateTimePicker1.Value=DateTime.Now;
```

说明：如果 Value 属性未在代码中或被用户更改，它将设置为当前的系统日期和时间（DateTime.Now）。

示例 09-33　获取 DateTimePicker 控件的属性在文本框中显示

本示例通过使用 DateTimePicker 的属性，把控件的当前应用显示在文本框中。代码如下：

代码 09-33　光盘位置：光盘\mingrisoft\sl\09\33

```
private void Form1_Load(object sender, EventArgs e)
{
    //获取 DateTimePicker 控件的日历前景色并转换为 RGB 值显示
```

```
    this.ForeColorInfo.Text =this.dateTimePicker1.CalendarForeColor.ToArgb().ToString();
    //获取 DateTimePicker 控件的月历背景色并转换为 RGB 值显示
    this.BackGroudInfo.Text = this.dateTimePicker1.CalendarMonthBackground.ToArgb().ToString();
    //获取 DateTimePicker 控件中可以显示的最大日期及时间
    this.MaxDateInfo.Text = this.dateTimePicker1.MaxDate.ToString();
    //获取 DateTimePicker 控件中可以显示的最小日期及时间
    this.MinDateInfo.Text = this.dateTimePicker1.MinDate.ToString();
    this.FontInfo.Text = this.dateTimePicker1.CalendarFont.Name;//获取 DateTimePicker 控件中显示的字体的内容
    this.dateTimePicker1.ShowCheckBox = true;                    //设置在选定日期的左侧显示一个复选框
    this.dateTimePicker1.ShowUpDown = true;                      //设置使用数值调节钮
}
```

运行本示例，效果如图 9.52 所示。

图 9.52　获取 DateTimePicker 控件的属性在文本框中显示

3．ToString 方法

该方法用于返回表示当前 DateTimePicker 控件的字符串。

语法：

```
public override string ToString ()
```

返回值：返回表示当前 DateTimePicker 的字符串，该字符串包括该控件的类型和 Value 属性。

示例 09-34　在 Label 控件中显示 DateTimePicker 控件的信息

本示例通过使用 ToString 方法，把与 DateTimePicker 控件有关的信息显示在 Label 控件中。代码如下：

代码 09-34　光盘位置：光盘\mingrisoft\sl\09\34

```
private void Form1_Load(object sender, EventArgs e)
{
    this.label1.Text = this.dateTimePicker1.ToString();    //显示与 DateTimerPicker 有关的信息
}
textBox1.Text = dateTimePicker1.Value.ToString();
```

运行本示例，效果如图 9.53 所示。

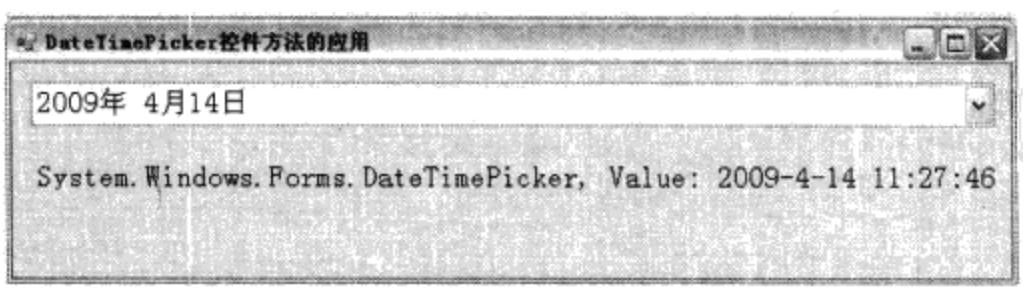


图 9.53　在 Label 控件中显示 DateTimePicker 控件的信息

4．事件

（1）CloseUp 事件。该事件在下拉日历被关闭并消失时发生。

语法：

```
public event EventHandler CloseUp
```

（2）DropDown 事件。该事件在显示下拉日历时发生。

语法：

```
public event EventHandler DropDown
```

（3）ValueChanged 事件。该事件在控件的 Value 值被更改时发生。

语法：

```
public event EventHandler ValueChanged
```

示例 09-35　在文本框中显示 DateTimePicker 控件的事件提示

本示例介绍对 DateTimePicker 控件的应用，当日期的值改变时，在对应的文本框中显示改变后的值。当一些事件发生时，在文本框中给出对应的提示。代码如下：

代码 09-35　光盘位置：光盘\mingrisoft\sl\09\35

```
private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
{
    this.ValueChangedInfo.Text = this.dateTimePicker1.Value.ToString();    //显示 DateTimePicker 控件改变后的值
}
private void dateTimePicker1_DropDown(object sender, EventArgs e)
{
    this.DropDownInfo.Text = "日历下拉开始…";                              //设置文本框中显示的文本
    this.DropDownInfo.ForeColor = Color.Red;                               //设置文本框的文字颜色
    this.DropDownInfo.BackColor = Color.Black;                             //设置文本框的背景颜色
}
private void dateTimePicker1_CloseUp(object sender, EventArgs e)
{
    this.CloseUpInfo.Text = "日历下拉关闭…";                               //设置文本框中显示的文本
    this.CloseUpInfo.ForeColor = Color.Red;                                //设置文本框的文字颜色
    this.CloseUpInfo.BackColor = Color.Black;                              //设置文本框的背景颜色
}
```

运行本示例，效果如图 9.54 所示。

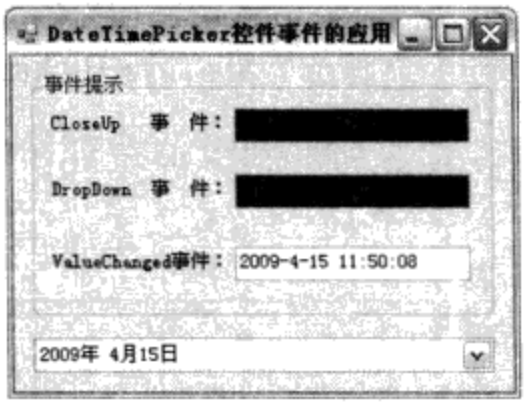


图 9.54　在文本框中显示 DateTimePicker 控件的事件提示

9.7.2　MonthCalendar 控件

1．功能

MonthCalendar 控件为用户查看和设置日期信息提供了一个直观的图形界面。该控件以网格形式显示日历，网格包含月份的编号日期，这些日期排列在周一到周日下的 7 个列中，并且突出显示选定的日期范围。可以单击月份标题任何一侧的箭头按钮来选择不同的月份，与 DateTimePicker 控件不同的是，开发人员可以使用该控件选择多个日期。图 9.55 所示为 MonthCalendar 控件。

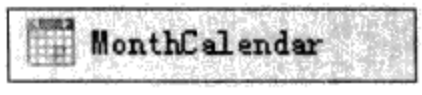


图 9.55　MonthCalendar 控件

2．属性

MonthCalendar 控件常用属性及说明如表 9.29 所示。

表 9.29 **MonthCalendar 控件常用属性及说明**

属　性	说　明
CalendarDimensions	获取或设置所显示月份的列数和行数
FirstDayOfWeek	根据月历中的显示获取或设置一周中的第一天
MaxDate	获取或设置允许的最大日期
MinDate	获取或设置允许的最小日期
ScrollChange	获取或设置月历控件的滚动率
SelectionEnd	获取或设置选定日期范围的结束日期
SelectionRange	为月历控件获取或设置选定的日期范围
SelectionStart	获取或设置所选日期范围的开始日期
ShowToday	获取或设置一个值，该值指示控件底端是否显示 TodayDate 属性表示的日期
ShowTodayCircle	获取或设置一个值，该值指示是否用圆圈或正方形标识今天日期
ShowWeekNumbers	获取或设置一个值，该值指示月历控件是否在每行日期的左侧显示周数（1-52）
SingleMonthSize	获取显示一个日历月所需的最小大小
TodayDate	获取或设置由 MonthCalendar 用作今天的日期的值
TodayDateSet	获取指示是否已显式设置 TodayDate 属性的值

下面对 MonthCalendar 控件中比较重要的属性进行详细介绍。

（1）CalendarDimensions 属性。该属性表示获取或设置所显示月份的列数和行数。

语法：

```
public Size CalendarDimensions { get; set; }
```

属性值：Size，带有用于显示日历的列数和行数。

例如，设置 monthCalendar1 控件显示为 4 行 3 列。代码如下：

```
this.monthCalendar1.CalendarDimensions=new Size(3,4);
```

（2）FirstDayOfWeek 属性。该属性表示根据月历中的显示获取或设置一周中的第一天。

语法：

```
public Day FirstDayOfWeek { get; set; }
```

属性值：Day 值之一。默认为 Day.Default。

例如，设置 monthCalendar1 控件中一周的第一天为星期一。代码如下：

```
this.monthCalendar1.FirstDayOfWeek=Day.Monday;
```

（3）MaxDate 属性。该属性表示获取或设置允许的最大日期。

语法：

```
public DateTime MaxDate { get; set; }
```

属性值：DateTime，表示允许的最大日期，默认值为 12/31/9998。

例如，获取 monthCalendar1 控件中允许显示的最大日期并显示在 textBox1 控件中。代码如下：

```
this.textBox1.Text=this.monthCalendar1.MaxDate.ToString();
```

（4）MinDate 属性。该属性表示获取或设置允许的最小日期。

语法：

```
public DateTime MinDate { get; set; }
```

属性值：DateTime，表示允许的最小日期，默认值为 01/01/1753。

例如，设置 monthCalendar1 控件中允许显示的最小日期。代码如下：

```
this.monthCalendar1.MinDate=new System.DateTime(2009,1,1,0,0,0,0);
```

（5）ShowToday 属性。该属性表示获取或设置一个值，该值指示是否在 MonthCalendar 控件的底端显示 TodayDate 指定的日期。

语法：

```
public bool ShowToday { get; set; }
```

属性值：如果显示的是今天的日期，为 true；否则为 false。默认值为 tue。

例如，设置 monthCalendar1 控件在底端显示今天的日期。代码如下：

```
this.monthCalendar1.ShowToday=true;
```

（6）ShowTodayCircle 属性。该属性用来表示是否在今天的日期上加圆圈或者长方形。

语法：

```
public bool ShowTodayCircle { get; set; }
```

属性值：如果在今天的日期上加圆圈，则为 true；否则为 false。默认值为 true。

例如，设置在当前日期的外面加上长方形。代码如下：

```
this.monthCalendar1.ShowTodayCircle=true;
```

（7）TodayDate 属性。该属性表示获取或设置由 MonthCalendar 用作今天的日期的值。

语法：

```
public DateTime TodayDate { get; set; }
```

属性值：DateTime，表示今天的日期。默认值是当前系统日期。

例如，获取 monthCalendar1 控件中当前系统日期，并显示在 textBox1 控件中。代码如下：

```
this.textBox1.Text=this.monthCalendar1.TodayDate.ToString();
```

示例 09-36　MonthCalendar 控件属性的应用

本示例中，当程序运行时，单击“标红”按钮，在 MonthCalendar 控件中显示当前日期并在当前日期外侧加一个方框；单击“显示”按钮，在文本框中显示当前 MonthCalendar 控件中所能显示的最大日期和最小日期。代码如下：

代码 09-36　光盘位置：光盘\mingrisoft\sl\09\36

```
private void Form1_Load(object sender, EventArgs e)
{
    //设置在 MonthCalendar 控件的左侧显示当前所在的周是该年的第几周
    this.monthCalendar1.ShowWeekNumbers = true;
    this.monthCalendar1.ShowTodayCircle = false;        //设置在当前日期中不显示方框
}
private void SignRed_Click(object sender, EventArgs e)
{
    this.monthCalendar1.ShowTodayCircle = true;         //设置在当前日期中显示方框
}
private void Display_Click(object sender, EventArgs e)
{
    this.monthCalendar1.ShowToday = true;               //设置在 MonthCalendar 控件的底端显示当天日期
    this.MaxDate.Text = this.monthCalendar1.MaxDate.ToString();    //在文本框中显示最大日期
    this.MinDate.Text = this.monthCalendar1.MinDate.ToString();    //在文本框中显示最小日期
}
```

运行本示例，效果如图 9.56 所示。

图 9.56　MonthCalendar 控件属性的应用

3．方法

MonthCalendar 控件常用方法及说明如表 9.30 所示。

表 9.30　　MonthCalendar 控件常用方法及说明

方　法	说　明
AddAnnuallyBoldedDate	在月历中，在一年的基础上添加一个以粗体显示的日期
AddBoldedDate	在月历中添加以粗体显示的日期
AddMonthlyBoldedDate	在月历中，在每月的基础上添加一个以粗体显示的日期
RemoveAllAnnuallyBoldedDates	移除一年中以粗体显示的所有日期
RemoveAllBoldedDates	移除所有非周期性粗体格式的日期
RemoveAllMonthlyBoldedDates	移除每月中以粗体显示的所有日期
RemoveAnnuallyBoldedDate	从一年中用粗体显示的日期的列表中移除指定日期
RemoveBoldedDate	从非周期性粗体格式日期的列表中移除指定日期
RemoveMonthlyBoldedDate	从每月中粗体格式日期的列表中移除指定日期
UpdateBoldedDates	重绘粗体格式的日期以反映在粗体格式日期的列表中设置的日期

下面对 MonthCalendar 控件中比较重要的方法进行详细说明

（1）AddBoldedDate 方法。该方法用于在月历中添加以粗体显示的日期。

语法：

```
public void AddBoldedDate (DateTime date)
```

参数说明如下。

date：要以粗体显示的日期。

例如，在 monthCalendar1 控件中加粗当前系统日期。代码如下：

```
this.monthCalendar1.AddBoldedDate(System.DateTime.Now);
```

（2）RemoveAllAnnuallyBoldedDates 方法。该方法用于移除一年中以粗体显示的所有日期。

语法：

```
public void RemoveAllAnnuallyBoldedDates()
```

例如，在 monthCalendar1 控件中移除一年中所有以粗体显示的日期。代码如下：

```
this.monthCalendar1.RemoveAllAnnuallyBoldedDates();
```

（3）RemoveAllMonthlyBoldedDates 方法。该方法用于移除每月中以粗体显示的所有日期。

语法：

```
public void RemoveAllMonthlyBoldedDates()
```

例如，在 monthCalendar1 控件中移除每月中以粗体显示的日期。代码如下：

```
this.monthCalendar1.RemoveAllMonthlyBoldedDates();
```

（4）RemoveMonthlyBoldedDate 方法。该方法用于从每月中粗体格式日期的列表中移除指定日期。

语法：

```
public void RemoveMonthlyBoldedDate(DateTime date)
```

参数说明如下。

date：要从日期列表中移除的日期。

例如，从 monthCalendar1 控件中的粗体格式日期列表中移除指定日期。代码如下：

```
this.monthCalendar1.RemoveMonthlyBoldedDate(new DateTime(2008,8,8));
```

（5）UpdateBoldedDates 方法。该方法用于重绘粗体格式的日期以反映在粗体格式日期的列表中设置的日期。

语法：

```
protected void UpdateBoldedDates ()
```

例如，更新 monthCalendar1 控件中的字体样式。代码如下：

```
this.monthCalendar1.UpdateBoldedDates();
```

示例 09-37　在 MonthCalendar 控件中加粗显示指定日期

本示例中，当程序运行时，通过 DateTimePicker 控件选择需要加粗的日期，单击“加粗”按钮实现加粗。当需要使某一日期恢复正常时，定位到该日期，单击“正常”按钮可以使该日期恢复正常。代码如下：

代码 09-37　光盘位置：光盘\mingrisoft\sl\09\37

```
private void AddBold_Click(object sender, EventArgs e)
{
    this.monthCalendar1.AddBoldedDate(this.dateTimePicker1.Value);  //在月历中添加以粗体显示的日期
    this.monthCalendar1.UpdateBoldedDates();                        //更新所做的样式变化
}
private void Form1_Load(object sender, EventArgs e)
{
    this.dateTimePicker1.Format = DateTimePickerFormat.Custom;      //设置 dateTimePicker 的日期时间格式
    this.dateTimePicker1.CustomFormat = "yyyy-MM-dd";               //设置用户自定义的日期时间格式
    this.dateTimePicker1.ShowUpDown = true;                         //设置dateTimePicker控件显示数值调节按钮
}
private void Normal_Click(object sender, EventArgs e)
{
    this.monthCalendar1.RemoveBoldedDate(this.dateTimePicker1.Value);//从非周期性粗体格式中移除指定的日期
    this.monthCalendar1.UpdateBoldedDates();                        //更新所做的样式变化
}
```

运行本示例，效果如图 9.57 所示。

图 9.57　在 MonthCalendar 控件中加粗显示指定日期

4．DateChanged 事件

该事件在 MonthCalendar 中的所选日期更改时发生。

语法：

```
public event DateRangeEventHandler DateChanged
```

示例 09-38　在文本框中显示选择的日期

本示例中，当程序运行时，选定 MonthCalendar 控件中的日期，对应的内容会在文本框中显示。代码如下：

代码 09-38　光盘位置：光盘\mingrisoft\sl\09\38

```
private void monthCalendar1_DateChanged(object sender, DateRangeEventArgs e)
{
    this.groupBox1.Visible = true;                                  //设置 groupBox1 控件为可见状态
    this.Content.Text = this.monthCalendar1.SelectionStart.ToString();//在文本框中显示选择的时间
}
private void Form1_Load(object sender, EventArgs e)
{
    this.groupBox1.Visible = false;                                 //设置 groupBox1 控件为不可见状态
```

```
        this.Content.ReadOnly = true;                    //设置文本框为只读文本框
    }
```

运行本示例，效果如图 9.58 所示。

图 9.58　在文本框中显示选择的日期

9.8　其他类型控件

Windows 应用程序所支持的某些控件不能划分在上述任何类别中，它们可用于特定功能，如 ToolTip 控件、WebBrower 控件和 ProgressBar 控件等，本节将对这类控件进行详细介绍。

9.8.1　ToolTip 控件

1．功能

该控件是一个长方形的弹出窗口，在用户将指针悬停在一个控件上时，该窗口显示有关该控件用途的简短说明。图 9.59 所示为 ToolTip 控件。

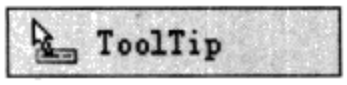


图 9.59　ToolTip 控件

2．属性

ToolTip 控件常用属性及说明如表 9.31 所示。

表 9.31　　ToolTip 控件常用属性及说明

属　性	说　明
Active	此属性获取或设置一个值，指示工具提示当前是否活动
AutomaticDelay	获取或设置工具提示的自动延迟
AutoPopDelay	获取或设置当指针在具有指定工具提示文本的控件内保持静止时，工具提示保持可见的时间期限（以 ms 为单位）
InitialDelay	此属性获取或设置工具提示显示之前经过多长时间，才会显示 ToolTip 控件
IsBalloon	获取或设置一个指示工具提示是否应使用气球状窗口的值
OwnerDraw	获取或设置一个值，该值指示工具提示是由操作系统绘制还是由您提供的代码绘制
ReshowDelay	获取或设置指针从一个控件移到另一控件时，必须经过多长时间才会出现后面的工具提示窗口
ShowAlways	此属性获取或设置一个值，该值指示是否显示工具提示窗口，甚至是在其父控件不活动的时候
ToolTipIcon	获取或设置一个值，该值定义要在工具提示文本旁显示的图标的类型
ToolTipTitle	获取或设置工具提示窗口的标题
UseFading	获取或设置一个值，该值确定在显示工具提示时是否应使用淡入淡出效果
UseAnimation	获取或设置一个值，该值确定在显示工具提示时是否应使用动画效果

下面对 ToolTip 控件中比较重要的属性进行详细介绍。

（1）AutoPopDelay 属性。该属性表示获取或设置当指针在具有指定工具提示文本的控件内保持静止时，工具提示保持可见的时间期限（以 ms 为单位）。

语法：

```
public int AutoPopDelay { get; set; }
```

属性值：当指针在控件上保持静止时，ToolTip 保持可见的时间期限（以毫秒为单位）。默认值为 5 000。

例如，当鼠标放置在具有工具提示的控件上时，设置工具提示保持可见的时间为 3s。代码如下：

```
this.toolTip1.AutoPopDelay=3000;
```

（2）IsBalloon 属性。该属性表示获取或设置一个指示工具提示是否应使用气球状窗口。

语法：

```
public bool IsBalloon { get; set; }
```

属性值：如果应使用气球状窗口，则为 true；如果应使用标准矩形窗口，则为 false。默认为 false。

例如，设置控件或窗体的工具提示窗口为气球状窗口。代码如下：

```
this.toolTip1.IsBalloon=true;
```

（3）ShowAlways 属性。该属性表示获取或设置一个值，该值指示是否显示工具提示窗口，甚至是在其父控件不活动的时候。

语法：

```
public bool ShowAlways { get; set; }
```

属性值：如果始终显示工具提示，则为 true；否则为 false。默认为 false。

例如，如果窗体或控件的工具提示窗口总是显示，那么将影响正常操作的进行，可以通过对该属性的设置让工具提示窗口不总处于显示状态。代码如下：

```
this.toolTip1.ShowAlways=false;
```

（4）ToolTipIcon 属性。该属性表示获取或设置一个值，该值定义要在工具提示文本旁显示的图标的类型。

语法：

```
public ToolTipIcon ToolTipIcon { get; set; }
```

属性值：System.Windows.Forms.ToolTipIcon 枚举值之一。

ToolTipIcon 的可取值及说明如下。

- None：表示不是标准图表。
- Info：表示信息图标。
- Warning：表示警告图标。
- Error：表示错误图标。

例如，当用户操作错误时，可以设置工具提示文本旁的图标为错误图标。代码如下：

```
this.toolTip1.ToolTipIcon=ToolTipIcon.Error;
```

（5）ToolTipTitle 属性。该属性表示获取或设置工具提示窗口的标题。

语法：

```
public string ToolTipTitle { get; set; }
```

属性值：包含窗口标题的字符串。

例如，当需要对用户即将进行的操作进行提示时，可以对工具提示窗口的标题做如下设置。代码如下：

```
this.toolTip1.ToolTipTitle="友情提示";
```

（6）UseFading 属性。该属性表示获取或设置一个值，该值确定在显示工具提示时是否应使用淡入淡出效果。

语法：

```
public bool UseFading { get; set; }
```

属性值：如果使用淡入淡出效果，则为 true；否则为 false。默认为 true。

例如，为了使用户得到更好的使用体验，可以设置工具提示为淡入淡出的效果。代码如下：

```
this.toolTip1.UseFading=true;
```

（7）UseAnimation 属性。该属性表示获取或设置一个值，该值确定在显示工具提示时，是否应使用动画效果。

语法：

```
public bool UseAnimation { get; set; }
```

属性值：如果窗口使用动画效果，为 true；否则为 false。默认值为 true。

例如，设置工具提示窗口的出现效果为动画效果。代码如下：

```
this.toolTip1.UseAnimation=true;
```

3．方法

ToolTip 控件常用方法及说明如表 9.32 所示。

表 9.32　ToolTip 控件常用方法及说明

方　法	说　明
Dispose	此方法释放由工具提示占用的资源
GetToolTip	检索与指定控件关联的工具提示文本
RemoveAll	移除当前与工具提示组件关联的所有工具提示文本
SetToolTip	使工具提示文本与指定的控件相关联
Show	设置与工具提示关联的文本，然后显示该工具提示
StopTimer	停止用于隐藏所显示工具提示的计时器

下面对 ToolTip 控件中比较重要的方法进行详细介绍。

（1）GetToolTip 方法。该方法检索与指定控件关联的工具提示文本。

语法：

```
public string GetToolTip(Control control)
```

参数说明如下。

control：要为其检索 ToolTip 文本的 Control。

例如，检索 button1 控件上的工具提示内容显示在 textBox1 控件中。代码如下：

```
this.textBox1.Text=this.toolTip1.GetToolTip(button1);
```

（2）RemoveAll 方法。该方法用来移除当前与工具提示组件关联的所有工具提示文本。

语法：

```
public void RemoveAll()
```

例如，当需要对工具提示内容进行重新设置时，需要移除以前进行的设置。代码如下：

```
this.toolTip1.RemoveAll();
```

（3）SetToolTip 方法。该方法使工具提示文本与指定的控件相关联。

语法：

```
public void SetToolTip (Control control,string caption)
```

参数说明如下。

control：将工具提示文本与其关联的控件。

caption：指针位于控件上方时要显示的工具提示文本。

例如，为 button1 控件设置工具提示文本。代码如下：

```
this.toolTip1.SetToolTip(this.button1, "单击此按钮，可以进行登录操作");
```

示例 09-39　使用 ToolTip 控件显示各个控件的信息提示

本示例通过对 ToolTip 控件属性和方法的应用，实现当鼠标在任意控件之上时显示相应的提示信息。代码如下：

代码 09-39　光盘位置：光盘\mingrisoft\sl\09\39

```
private void Form1_Load(object sender, EventArgs e)
{
    this.toolTip1.AutomaticDelay = 100;          //设置工具提示的自动延迟
    this.toolTip1.AutoPopDelay = 3000;           //设置工具提示保持的可见时间
```

```
        this.toolTip1.IsBalloon = true;                    //设置工具提示为气泡提示的形式
        this.toolTip1.ReshowDelay = 1000;                  //设置从一个控件移动到另一个控件所用的时间
        this.toolTip1.ShowAlways = false;                  //设置工具提示窗口在需要时显示
        this.toolTip1.ToolTipIcon = ToolTipIcon.Info;      //设置工具提示窗口的图标
        this.toolTip1.ToolTipTitle = "友情提示";           //设置工具提示窗口的标题
        this.toolTip1.UseAnimation = true;                 //设置工具提示窗口的出现形式
        this.toolTip1.SetToolTip(Login, "执行登录操作");                    //设置“登录”按钮的工具提示
        this.toolTip1.SetToolTip(Cancel, "执行取消操作");                   //设置“取消”按钮的工具提示
        this.toolTip1.SetToolTip(UserName, "请在文本框中输入用户名");       //设置“用户名”文本框的工具提示
        this.toolTip1.SetToolTip(UserPwd, "请在文本框中输入密码");          //设置“密码”文本框的工具提示
    }
```

运行本示例，效果如图 9.60 所示。

图 9.60　使用 ToolTip 控件显示各个控件的信息提示

9.8.2　ProgressBar 控件

1．功能

ProgressBar 控件通过在水平条中显示相应数目的矩形来指示操作的进度。操作完成时，进度栏被填满。进度栏通常用于帮助用户了解等待一定长时间的操作（如加载大文件等）完成所需的时间。图 9.61 所示为 ProgressBar 控件。

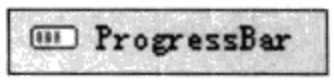


图 9.61　ProgressBar 控件

2．属性

ProgressBar 控件常用属性及说明如表 9.33 所示。

表 9.33　　ProgressBar 控件常用的属性及说明

属　性	说　明
Maximum	此属性设置进度栏可以显示的最大值
Minimum	此属性设置进度栏可以显示的最小值
Step	此属性可以用于指定 Value 属性递增的值
Style	此属性获取或设置在进度栏上指示进度应使用的方式
Value	此属性表示操作过程中已完成的进度

下面对 ProgressBar 控件中比较重要的属性进行详细介绍。

（1）Step 属性。该属性用于指定 Value 属性递增的值。

语法：

```
public int Step {get;set;}
```

属性值：每次调用 PerformStep 方法增加进度栏所根据的数量，默认值为 10。

例如，设置 progressBar1 控件的递增值为 10。代码如下：

```
this.progressBar1.Step=10;
```

说明：可以使用 Step 属性指定操作中完成的每个任务更改进度栏的值的数量。例如，如果要复制一组文件，则可能需要将 Step 属性的值设置为 1，并将 Maximum 属性的值设置为要复制的文件总数。在复制每个文件时，可以调用 PerformStep 方法，按 Step 属性的值增加进度栏。

（2）Minimum 属性。该属性表示设置进度栏可以显示的最小值。

语法：

```
public int Minimum { get; set; }
```

属性值：范围的最小值，默认值为 0。

例如，可以设置 progressBar1 控件的下限为 10。代码如下：

```
this.proressBar1.Minimum=10;
```

（3）Maximum 属性。该属性表示设置进度栏可以显示的最大值。

语法：

```
public int Maximum {get;set;}
```

属性值：范围的最大值，默认值为 100。

例如，可以设置 progressBar1 控件的上限为 100。代码如下：

```
this.progressBar1.Maximum=100;
```

（4）Value 属性。该属性表示获取或设置进度栏的当前位置。

语法：

```
public int Value { get; set; }
```

属性值：进度栏范围内的位置。默认值为 0。

例如，可以设置 progressBar1 控件的当前进度。代码如下：

```
this.progressBar1.Value=50;
```

（5）Style 属性。该属性表示获取或设置在进度栏上指示进度应使用的方式。

语法：

```
public ProgressBarStyle Style { get; set; }
```

属性值：ProgressBarStyle 值之一。默认值为 Blocks。

ProgressBarSyle 的可取值及说明如下。

- Blocks：表示通过在 ProgressBar 中增加分段块的数量来指示进度。
- Continuous：表示通过在 ProgressBar 中增加平滑连续的栏的大小来指示进度。
- Marquee：表示通过以字幕方式在 ProgressBar 中连续滚动一个块来指示进度。

例如，可以将进度栏设置为类似操作系统启动时的样式。代码如下：

```
this.progressBar1.Style=ProgressBarStyle.Marquee;
```

3．方法

（1）Increment 方法。该方法将进度栏的当前位置移动到指定的增量值。

语法：

```
public void Increment(int value)
```

参数说明如下。

value：增加进度栏的当前位置所根据的数量。

例如，可以直接将进度栏中的进度移动至指定位置，代码中的 N 为许可范围内的任意数值。代码如下：

```
this.progressBar1.Increment(N);
```

（2）PerformStep 方法。该方法按照 Step 属性中指定的数量增加进度栏的值。用户可以使用 Step 属性指定操作中每个完成的任务更改进度栏的值的数量。

语法：

```
public void PerformStep ()
```

例如，当设置进度栏的变化幅度后就可以使用进度栏。代码如下：

```
this.progressBar1.Step=10;
this.progressBar1.PerformStep();
```

4．Click 事件

该事件在单击控件时发生。

语法：

```
public event EventHandler Click
```

示例 09-40　在 Label 控件中显示当前进度

本示例演示对 ProgressBar 控件属性、方法和事件的应用，当程序运行时，单击“开始”按钮，进度条开始走动，同时在 Label 控件中显示当前的进度占总进度的百分比。代码如下：

代码 09-40　光盘位置：光盘\mingrisoft\sl\09\40

```
public static bool flag=false ;                                  //设置一个全局变量标识
private void Form1_Load(object sender, EventArgs e)
{
    this.progressBar1.Style = ProgressBarStyle.Continuous;       //设置 ProgressBar 控件的样式
    this.progressBar1.Maximum = 100;                             //设置 ProgressBar 控件范围的最大值
    this.progressBar1.Minimum = 0;                               //设置 ProgressBar 控件范围的最小值
    this.progressBar1.Step = 1;                                  //设置 ProgressBar 控件的递增进度
    this.Operate.Text = "开始";                                  //设置 Button 按钮上的文本
}
private void Operate_Click(object sender, EventArgs e)
{
    if (flag == false)                                           //当全局变量标识为 false 时
    {
        this.timer1.Start();                                     //启动计时器
        flag = true;                                             //设置全局变量标识为 true
        this.Operate.Text = "暂停";                              //设置 Button 按钮的文本内容为“暂停”
    }
    else                                                         //当全局变量标识为 true 时
    {
        this.timer1.Stop();                                      //停止计时器
        flag = false;                                            //设置全局变量标识为 false
        this.Operate.Text = "开始";                              //设置 Button 按钮的文本内容为“开始”
    }
}
private void timer1_Tick(object sender, EventArgs e)
{
    this.progressBar1.PerformStep();        //按照 ProgressBar 控件的 Step 属性递增 ProgressBar 控件中的值
    this.label1.Text = "";                  //清空 Label 控件中的文本
    this.label1.Text = "当前进度为：" + this.progressBar1.Value.ToString() + "%";//设置 Label 控件中的内容
}
```

运行本示例，效果如图 9.62 所示。

图 9.62　在 Label 控件中显示当前进度

9.8.3　WebBrowser 控件

1．功能

WebBrowser 控件为 WebBrowser ActiveX 控件提供托管包装。托管包装使用户可以在 Windows 窗体客户端应用程序中显示网页。使用 WebBrowser 控件，可以使应用程序具有 Internet Explorer Web 浏览功能，还可以禁用默认的 Internet Explorer 功能，并将该控件用作简单的 HTML 文档查看器。此外，使用该控件可以将基于 DHTML 的用户界面元素添加到窗体中，同时还可以将这些元素隐藏在 WebBrowser 控件承载的窗体中。通过这种方法，可以将 Web 控件和 Windows 窗体控件无缝地整合到一个应用程序中。图 9.63 所示为 WebBrowser 控件。

图 9.63　WebBrowser 控件

2．属性

WebBrowser 控件常用属性及说明如表 9.34 所示。

表 9.34　　WebBrowser 控件常用属性及说明

属　性	说　明
AllowNavigation	获取或设置一个值，该值指示控件在加载其初始页之后是否可以导航到其他页
AllowWebBrowserDrop	获取或设置一个值，该值指示 WebBrowser 控件是否导航到拖放到它上面的文档
CanGoBack	获取一个值，该值指示导航历史记录中的上一页是否可用，如果可用，GoBack 方法才能成功
CanGoForward	获取一个值，该值指示导航历史记录中的下一页是否可用，如果可用，GoForward 方法才能成功
Document	此属性获取一个对象，用于提供对当前网页的 HTML 文档对象模型（DOM）的托管访问
DocumentStream	获取或设置一个流，该流包含显示在 WebBrowser 控件中的网页内容
DocumentText	此属性获取或设置当前网页的 HTML 内容
DocumentTitle	此属性获取当前网页的标题
DocumentType	获取当前显示在 WebBrowser 控件中的文档的类型
EncryptionLevel	获取一个值，该值指示当前显示在 WebBrowser 控件中的文档所使用的加密方法
IsBusy	获取一个值，该值指示 WebBrowser 控件当前是否正在加载新文档
IsOffline	获取一个值，该值指示 WebBrowser 控件是否处于脱机模式
IsWebBrowserContextMenuEnabled	获取或设置一个值，该值指示 WebBrowser 控件是否启用了快捷菜单
ObjectForScripting	获取或设置一个对象，该对象可由显示在 WebBrowser 控件中的网页所包含的脚本代码访问
ReadyState	获取一个值，该值指示 WebBrowser 控件的当前状态
ScriptErrorsSuppressed	获取或设置一个值，该值指示 WebBrowser 是否显示对话框，如脚本错误消息
ScrollBarsEnabled	获取或设置一个值，该值指示是否在 WebBrowser 控件中显示滚动条
StatusText	获取 WebBrowser 控件的状态文本
Url	此属性获取或设置当前网页的 URL。设置该属性时，会将该控件定位到新的 URL
WebBrowserShortcutsEnabled	获取或设置一个值，该值指示 WebBrowser 控件中是否启用了键盘快捷键

下面对 WebBrowser 控件中比较重要的属性进行详细介绍。

（1）Document 属性。该属性表示获取一个对象，用于提供对当前网页的 HTML 文档对象模型（DOM）的托管访问。

语法：

```
public HtmlDocument Document { get; }
```

属性值：表示当前页的 HtmlDocument，如果未加载任何页，则为 null。

例如，可以判断当前窗体是否已经写入网页，如果为写入，那么弹出信息提示。代码如下：

```
HtmlDocument document=this.webBrowser1.Document;
if(document==null)
{
        MessageBox.Show("当前窗体未加载文档!","信息提示");
}
```

（2）DocumentTitle 属性。该属性表示获取当前网页的标题。

语法：

```
public string DocumentTitle { get; }
```

属性值：当前文档的标题，如果未加载任何文档，则为空字符串("")。

例如，将当前加载的文档标题显示在标签控件中。代码如下：

```
this.label1.Text=this.webBrowser1.DocumentTitle;
```

（3）DocumentText 属性。该属性表示获取或设置显示在 WebBrowser 控件中的页的 HTML 内容。

语法：

```
public string DocumentText { get; set; }
```

属性值：显示的页的 HTML 文本，如果未加载任何文档，则为空字符串("")。

例如，将当前加载的文档内容显示在 RichTextBox 控件中。代码如下：

```
this.richTextBox.Text=this.webBrowser1.DocumentText;
```

（4）DocumentType 属性。该属性表示获取当前显示在 WebBrowser 控件中的文档的类型。

语法：

```
public string DocumentType { get; }
```

属性值：当前文档的类型。

例如，单击窗体中的“获取”按钮得到当前加载的文档类型。代码如下：

```
private void GetDocumentType_Click(object sender, EventArgs e)
{
    MessageBox.Show(this.webBrowser1.DocumentType ,"信息提示");
}
```

（5）ReadyState 属性。该属性表示获取一个值，该值指示 WebBrowser 控件的当前状态。

```
public WebBrowserReadyState ReadyState { get; }
```

属性值：WebBrowserReadyState 值之一。

WebBrowserReadyState 的可选值及说明如下。

- Uninitialized：表示当前未加载任何文档。
- Loading：表示该控件正在加载新文档。
- Loaded：表示该控件已经加载并已初始化新文档，但尚未接收到所有文档数据。
- Interactive：表示该控件已加载足够的文档以允许有限的用户交互，比如单击已显示的超级链接。
- Complete：表示该控件已完成新文档及其所有内容的加载。

例如，当 WebBrowser 控件正处于文档加载状态时给出提示信息。代码如下：

```
if (this.webBrowser1.ReadyState==WebBrowserReadyState.Loading)
{
    MessageBox.Show("请稍候，文档正在加载过程中…","信息提示");
}
```

（6）Url 属性。该属性表示可以获取或设置当前网页的 URL。设置该属性时，会将该控件定位到新的 URL。

语法：

```
public Uri Url { get; set; }
```

属性值：URL，表示当前文档的名称。

例如，单击窗体中的“明日科技”按钮即可导航到明日科技有限公司的网站主页。代码如下：

```
private void MRKJ_Click(object sender, EventArgs e)
{
    this.webBrowser1.Url = new Uri("http://www.mingrisoft.com");
}
```

说明：设置此属性等效于向 Navigate 方法传递指定的 URL 并调用该方法。WebBrowser 控件维护浏览会话期间访问的所有网页的历史记录列表。设置 Url 属性时，WebBrowser 控件导航到指定的 URL 并将该 URL 添加到历史记录列表的末尾。

3．方法

WebBrowser 控件常用方法及说明如表 9.35 所示。

表 9.35 WebBrowser 控件常用方法及说明

方　法	说　明
GoBack	此方法定位到历史记录中的上一页
GoForward	此方法定位到历史记录中的下一页
GoHome	将 WebBrowser 控件导航到当前用户的主页
GoSearch	将 WebBrowser 控件导航到当前用户的默认搜索页
Navigate	此方法定位到指定的 URL
Print	此方法打印当前的网页
Refresh	此方法重新加载当前的网页
ShowPageSetupDialog	打开 Internet Explorer“页面设置”对话框
ShowPrintDialog	打开 Internet Explorer 的“打印”对话框，但不设置页眉或页脚值
ShowPrintPreviewDialog	打开 Internet Explorer 的“打印预览”对话框
ShowPropertiesDialog	打开当前文档的 Internet Explorer“属性”对话框
ShowSaveAsDialog	打开 Internet Explorer 的“保存网页”对话框，如果承载的文档不是 HTML 页，则打开其“保存”对话框
Stop	取消所有挂起的导航并停止所有动态页元素（如背景声音和动画）

下面对 WebBrowser 控件中比较重要的方法进行详细介绍。

（1）GoBack 方法。该方法用于定位到历史记录中的上一页。

语法：

```
public bool GoBack ()
```

返回值：如果导航成功，则为 true；如果导航历史记录中的上一页不可用，则为 false。

说明：WebBrowser 控件维护浏览会话期间访问的所有网页的历史记录列表。可以使用 GoBack 方法实现一个“后退”按钮，该按钮类似于 Internet Explorer 中的“后退”按钮，它允许用户返回到导航历史记录中的上一页。

例如，返回历史记录的上一页。代码如下：

```
this.webBrowser1.GoBack();
```

（2）GoForward 方法。如果导航历史记录中的下一页可用，则该方法将 WebBrowser 控件导航到该页。

语法：

```
public bool GoForward ()
```

返回值：如果导航成功，则为 true；如果导航历史记录中的下一页不可用，则为 false。

说明：WebBrowser 控件维护浏览会话期间访问的所有网页的历史记录列表。可以使用 GoForward 方法实现一个“前进”按钮，该按钮类似于 Internet Explorer 中的“前进”按钮，它允许用户返回到导航历史记录中的下一页。

例如，返回历史记录中的下一页。代码如下：

```
this.webBrowser1.GoForward();
```

（3）GoHome 方法。该方法用来将 WebBrowser 控件导航到当前用户的主页。

语法：

```
public void GoHome()
```

例如，导航到当前网站的主页。代码如下：

```
this.webBrowser1.GoHome();
```

（4）GoSearch 方法。该方法用来将 WebBrowser 控件导航到当前用户的默认搜索页。

语法：

```
public void GoSearch()
```

例如，导航到当前用户默认的搜索页。代码如下：

```
this.webBrowser1.GoSearch();
```

（5）Navigate 方法。该方法定位到指定的 URL。

语法：

```
public void Navigate (string urlString)
```

参数说明如下。

urlString：要加载文档的 URL。

例如，当单击“搜索”按钮时，导航到百度主页。代码如下：

```
private void Search_Click(object sender, EventArgs e)
{
    this.webBrowser1.Navigate("http://www.baidu.com");
}
```

（6）Print 方法。该方法表示此方法打印当前的网页。

语法：

```
public void Print()
```

例如，当当前页面中的文档对用户比较重要时，可以对其进行打印保存，单击“打印”按钮进行打印。代码如下：

```
private void Print_Click(object sender, EventArgs e)
{
    this.webBrowser1.Print();
}
```

（7）Refresh 方法。该方法用于重新加载当前的网页。

语法：

```
public override void Refresh ()
```

例如，当网页中的图片加载不正确时，需要用户对其进行刷新。代码如下：

```
private void Refresh_Click(object sender, EventArgs e)
{
    this.webBrowser1.Refresh();
}
```

（8）Stop 方法。该方法用于取消所有挂起的导航并停止所有动态页元素（如背景声音和动画）。

语法：

```
public void Stop()
```

例如，当网速过慢文档长时间加载未显示时，可以停止所有动态页元素立即显示已加载的部分。代码如下：

```
private void Stop_Click(object sender, EventArgs e)
{
    this.webBrowser1.Stop();
}
```

4．事件

WebBrowser 控件常用事件及说明如表 9.36 所示。

表 9.36　　WebBrowser 控件常用事件及说明

事　件	说　明
DocumentCompleted	在 WebBrowser 控件完成加载文档时发生
DocumentTitleChanged	在 DocumentTitle 属性值更改时发生
EncryptionLevelChanged	在 WebBrowser 控件导航到或离开使用了加密的网站时发生
FileDownload	在 WebBrowser 控件下载文件时发生
LocationChanged	在 Location 属性值更改后发生
Navigated	在 WebBrowser 控件导航到新文档并开始加载该文档时发生
Navigating	在 WebBrowser 控件导航到新文档之前发生
NewWindow	在浏览器新窗口打开之前发生
ProgressChanged	在 WebBrowser 控件已更新有关要导航到的文档的下载进度的信息时发生

下面对 WebBrowser 控件中比较重要的事件进行详细介绍。

（1）DocumentCompleted 事件。该事件在 WebBrowser 控件完成加载文档时发生。

语法：

```
public event WebBrowserDocumentCompletedEventHandler DocumentCompleted
```

（2）DocumentTitleChanged 事件。该事件在 DocumentTitle 属性值更改时发生。

语法：

```
public event EventHandler DocumentTitleChanged
```

（3）FileDownload 事件。该事件在 WebBrowser 控件下载文件时发生。

语法：

```
public event EventHandler FileDownload
```

（4）Navigated 事件。该事件在 WebBrowser 控件完成加载文档时发生。

语法：

```
public event WebBrowserNavigatedEventHandler Navigated
```

（5）Navigating 事件。该事件在 WebBrowser 控件导航到新文档之前发生。

```
public event WebBrowserNavigatingEventHandler Navigating
```

（6）NewWindow 事件。该事件在浏览器新窗口打开之前发生。

语法：

```
public event CancelEventHandler NewWindow
```

（7）ProgressChanged 事件。该事件在 WebBrowser 控件已更新有关要导航到的文档的下载进度的信息时发生。

语法：

```
public event WebBrowserProgressChangedEventHandler ProgressChanged
```

示例 09-41　WebBrowser 控件的应用

运行本示例，在地址文本框中输入可以正常访问的网站，然后单击“转到”按钮即可进入相应的网站。在工具栏中有各个操作的快捷按钮。代码如下：

代码 09-41　光盘位置：光盘\mingrisoft\sl\09\41

```
private void Form1_Load(object sender, EventArgs e)
{
    this.webBrowser1.IsWebBrowserContextMenuEnabled = true;    //在 WebBrowser 控件中启用快捷键
    this.webBrowser1.ScriptErrorsSuppressed = true;            //在 WebBrowser 控件中显示对话框
    this.webBrowser1.ScrollBarsEnabled = true;                 //在 WebBrowser 控件中显示滚动条
    this.Direction.Visible = false;                            //设置下载提示文本框为不可用状态
    this.timer1.Interval = 1000;                               //设置计时器的 Trick 事件间隔
    this.timer1.Start();                                       //启动计时器
}
private void Conversion_Click(object sender, EventArgs e)
{
    if (this.Address.Text != null && this.Address.Text != "")  //当地址栏的内容不为空且存在内容时
    {
        this.webBrowser1.Url = new Uri(this.Address.Text);     //定位到该页
    }
}
private void BackOff_Click(object sender, EventArgs e)
{
    this.webBrowser1.GoBack();                                 //返回上一页面
}
private void GoPrevious_Click(object sender, EventArgs e)
{
    this.webBrowser1.GoForward();                              //返回刚刚进行操作的页面
}
private void Stop_Click(object sender, EventArgs e)
{
    this.webBrowser1.Stop();                                   //取消所有挂起的导航并停止所有动态页元素
```

```
    }
    private void Refurbish_Click(object sender, EventArgs e)
    {
        this.webBrowser1.Refresh();                          //重新加载指定路径下的文档
    }
    private void HomePage_Click(object sender, EventArgs e)
    {
        this.webBrowser1.GoHome();                           //返回网站主页
    }
    private void Search_Click(object sender, EventArgs e)
    {
        this.webBrowser1.GoSearch();                         //将 WebBrowser 导航到用户默认的搜索页
    }
    private void 打印 PToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.webBrowser1.Print();                            //打印当前页面
    }
    private void 打印预览 VToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.webBrowser1.ShowPrintPreviewDialog();           //显示打印预览对话框
    }
    private void 另存为 AToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.webBrowser1.ShowSaveAsDialog();                 //显示另存为对话框
    }
    private void 页面设置 ToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.webBrowser1.ShowPageSetupDialog();              //显示页面设置对话框
    }
    private void 属性 ToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.webBrowser1.ShowPropertiesDialog();             //显示页面属性对话框
    }
    private void webBrowser1_NewWindow(object sender, CancelEventArgs e)
    {
        MessageBox.Show("你打开的网址为:" + this.Address.Text, "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                                   //弹出信息提示
    }
    private void 退出 XToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.timer1.Stop();                                  //停止 Timer 计时器
        Application.Exit();                                  //退出应用程序
    }
    private void webBrowser1_FileDownload(object sender, EventArgs e)
    {
        this.Direction.Visible = true;                       //设置下载提示文本框为可见状态
        this.Direction.ForeColor = Color.Red;                //设置下载提示文本框中的字体为红色
        this.Direction.Text = "正在下载文件…";               //设置下载提示文本框中的文本内容
    }
    private void webBrowser1_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
    {
        if (this.webBrowser1.IsBusy == false)                //当 WebBrowser 控件没有加载文档时
        {
            this.Direction.Visible = false;                  //设置下载提示文本框为不可见状态
        }
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        this.DisplayTime.Text = System.DateTime.Now.ToString(); //在文本框中显示系统时间
    }
```

运行本示例，效果如图 9.64 所示。

图 9.64　WebBrowser 控件的应用

9.8.4　MaskedTextBox 控件

1．功能

MaskedTextBox 控件使用掩码区分正确和不正确的用户输入。MaskedTextBox 控件如图 9.65 所示。

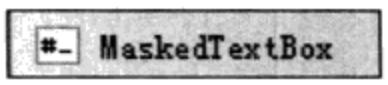


图 9.65　MaskedTextBox 控件

2．属性

MaskedTextBox 控件常用属性及说明如表 9.37 所示。

表 9.37　　MaskedTextBox 控件常用属性及说明

属　性	说　明
AsciiOnly	获取或设置一个值，该值指示 MaskedTextBox 控件是否接受 ASCII 字符集以外的字符
BeepOnError	获取或设置一个值，该值指示掩码文本框控件是否每当用户键入了它拒绝的字符时都发出系统警告声
CausesValidation	此属性获取或设置一个值，该值指示控件是否引发验证事件
HidePromptOnLeave	获取或设置一个值，该值指示当掩码文本框失去焦点时，输入掩码中的提示字符是否隐藏
HideSelection	获取或设置一个值，该值指示当文本框控件没有焦点时，该控件中选定的文本是否保持突出显示
Mask	获取或设置运行时使用的输入掩码
MaskCompleted	获取一个值，该值指示所有必需的输入是否都已输入到输入掩码中
MaskFull	获取一个值，该值指示所有必需和可选的输入是否都已输入到输入掩码中
Modified	获取或设置一个值，该值指示自创建文本框控件或上次设置该控件的内容后，用户修改了该控件
PasswordChar	获取或设置要取代用户输入而显示的字符
PromptChar	此属性获取或设置用于表示 MaskedTextBox 中缺少用户输入的字符
RejectInputOnFirstFailure	此属性获取或设置一个值，该值指示当遇到第一个无效字符时是否应停止对用户输入的分析
ResetOnPrompt	此属性获取或设置一个值，该值确定应如何处理与提示字符匹配的输入字符
SelectedText	获取或设置 MaskedTextBox 控件中当前选择的内容
SelectionLength	获取或设置文本框中选定的字符数
SelectionStart	获取或设置文本框中选定的文本起始点

续表

属　性	说　明
SkipLiterals	获取或设置一个值，该值指示是否允许用户重新输入原义字符
TextAlign	获取或设置掩码文本框控件中文本的对齐方式
TextLength	获取显示的文本的长度
TextMaskFormat	此属性获取或设置一个值，该值决定原义字符和提示字符是否包括在格式化字符串中
UseSystemPasswordChar	此属性获取或设置一个值，该值指示是否应使用操作系统提供的密码字符
UseWaitCursor	获取或设置一个值，该值指示是否将等待光标用于当前控件以及所有子控件
ValidatingType	此属性获取或设置用于验证用户输入的数据的数据类型

下面对 MaskedTextBox 控件中比较重要的属性进行详细介绍。

（1）AsciiOnly 属性。该属性表示获取或设置一个值，该值指示 MaskedTextBox 控件是否接受 ASCII 字符集以外的字符。

语法：

```
public bool AsciiOnly { get; set; }
```

属性值：如果仅接受 ASCII，该值为 true；如果 MaskedTextBox 控件可以接受任意 Unicode 字符，则为 false。默认为 false。

例如，设置在 MaskedTextBox 控件中接受 ASCII 字符集以外的字符。代码如下：

```
this.maskedTextBox1.AsciiOnly=true;
```

（2）BeepOnError 属性。该属性表示获取或设置一个值，该值指示掩码文本框控件是否每当用户键入了它拒绝的字符时都发出系统警告声。

语法：

```
public bool BeepOnError { get; set; }
```

属性值：如果 MaskedTextBox 控件应当在输入无效时发出警告声，该值为 true；否则为 false。默认为 false。

例如，当用户在 MaskedTextBox 控件中键入非法字符时发出提示声。代码如下：

```
this.maskedTextBox1.BeepOnError=true;
```

（3）CausesValidation 属性。该属性表示获取或设置一个值，该值指示控件是否引发验证事件。

语法：

```
public bool CausesValidation { get; set; }
```

属性值：如果控件引起在任何需要在接收焦点时执行验证的控件上执行验证，则为 true；否则为 false。默认为 true。

例如，当用户输入内容后，可以设置对该内容进行验证。代码如下：

```
this.maskedTextBox1.CausesValidation=true;
```

（4）HideSelection 属性。该属性表示获取或设置一个值，该值指示当文本框控件没有焦点时，该控件中选定的文本是否保持突出显示。

语法：

```
public bool HideSelection { get; set; }
```

属性值：如果当文本框控件没有焦点时，选定文本不突出显示，则为 true；如果当文本框控件没有焦点时，选定文本保持突出显示，则为 false。默认值为 true。

例如，当文本框控件中没有焦点时，设置选定文本突出显示。代码如下：

```
this.maskedTextBox1.HideSelection=false;
```

（5）Mask 属性。该属性表示获取或设置运行时使用的输入掩码。

语法：

```
public string Mask { get; set; }
```

属性值：一个字符串，表示当前掩码。默认值是允许任何输入的空字符串。

例如，设置用户输入时的掩码为“★”。代码如下：

```
this.maskedTextBox1.Mask="★";
```

（6）MaskFull 属性。该属性表示获取一个值，该值指示所有必需和可选的输入是否都已输入到掩码中。

语法：

```
public bool MaskFull { get; }
```

属性值：如果所有必需和可选的输入都已输入，该值为 true；否则为 false。

例如，单击“检查”按钮，当所有必需和可选的输入都已输入时弹出相应的信息提示。当有未输入的内容存在时，弹出用户确认的信息。代码如下：

```
private void Check_Click(object sender, EventArgs e)
{
    if (this.maskedTextBox1.MaskFull==true )
    {
        MessageBox.Show("所有输入已输入完毕！","信息提示");
    }
    else
    {
        MessageBox.Show("输入内容不完全，请确认！","信息提示");
    }
}
```

（7）PasswordChar 属性。该属性表示获取或设置要取代用户输入而显示的字符。

语法：

```
public char PasswordChar { get; set; }
```

属性值：用作密码字符的 Char 值。

例如，设置将取代用户输入而显示的字符为“●”。代码如下：

```
this.maskedTextBox1.PasswordChar="●";
```

（8）RejectInputOnFirstFailure 属性。该属性表示获取或设置一个值，该值指示当遇到第一个无效字符时，是否应停止对用户输入的分析。

语法：

```
public bool RejectInputOnFirstFailure { get; set; }
```

属性值：如果输入字符串的处理应当在首次出现分析错误时终止，该值为 true；如果处理应忽略所有错误，则为 false。默认为 false。

例如，设置在 MaskedTextBox 控件中当用户输入第一个无效字符时立即停止对用户输入的分析。代码如下：

```
this.maskedTextBox1.RejectInputOnFirstFailure=true;
```

（9）SelectedText 属性。该属性表示获取或设置 MaskedTextBox 控件中当前选择的内容。

语法：

```
public override string SelectedText { get; set; }
```

属性值：String 类型的当前选定文本。如果当前没有选定任何文本，此属性将解析为一个空字符串。

例如，设置 MaskedTextBox 控件中的默认选择内容为“凯旋司令”。代码如下：

```
this.maskedTextBox1.SelectedText="凯旋司令";
```

（10）ValidatingType 属性。该属性可以获取或设置用于验证用户输入数据的数据类型。

语法：

```
public Type ValidatingType { get; set; }
```

属性值：表示在验证中使用的数据类型的 Type。默认为空引用。

例如，获取用于验证用户输入数据的数据类型显示在标签控件中。代码如下：

```
this.label1.Text=this.maskedTextBox1.ValidatingType.ToString();
```

3．方法

MaskedTextBox 控件常用方法及说明如表 9.38 所示。

表 9.38　　MaskedTextBox 控件常用方法及说明

方　法	说　明
AppendText	向文本框的当前文本追加文本
Clear	从文本框控件中清除所有文本
Copy	将文本框中的当前选定内容复制到“剪贴板”
Cut	将文本框中的当前选定内容移动到“剪贴板”中
DeselectAll	将 SelectionLength 属性的值指定为零，从而不会在控件中选择字符
Focus	为控件设置输入焦点
GetCharFromPosition	检索控件内距离指定位置最近的字符
GetCharIndexFromPosition	检索距离指定位置最近的字符的索引
GetPositionFromCharIndex	检索控件内指定字符索引处的位置
IsInputChar	确定一个字符是否是控件可识别的输入字符
IsInputKey	确定指定的键是输入键还是需要预处理的特殊键
Paste	用“剪贴板”的内容替换文本框中的当前选定内容
SelectAll	选定文本框中的所有文本
ValidateText	将用户输入字符串转换为验证类型的一个实例

下面对 MaskedTextBox 控件中比较重要的方法进行详细的说明。

（1）AppendText 方法。该方法表示向文本框的当前文本中追加内容。

语法：

```
public void AppendText(string text)
```

参数说明如下。

text：表示要向文本框的当前内容追加的文本。

例如，向文本框中的当前内容中追加一个字符串“天龙八部”。代码如下：

```
this.maskedTextBox1.AppendText("天龙八部");
```

（2）DeselectAll 方法。该方法表示将 SelectionLength 属性的值指定为零，从而不会在控件中选择字符。

语法：

```
public void DeselectAll()
```

例如，取消在 MaskedTextBox 控件中选择字符。代码如下：

```
this.maskedTextBox1.DeselectAll();
```

（3）Focus 方法。该方法表示为控件设置输入焦点。

语法：

```
public bool Focus()
```

参数说明如下。

返回值：如果输入焦点请求成功，则为 true；否则为 false。

例如，把当前的 MaskedTextBox 控件设置为当前控件的输入焦点。代码如下：

```
this.maskedTextBox1.Focus();
```

（4）GetCharIndexFromPosition 方法。该方法表示检索距离指定位置最近的字符的索引。

语法：

```
public override int GetCharIndexFromPosition(Point pt)
```

参数说明如下。

pt：要搜索的位置。

返回值：位于指定位置的从零开始的字符索引。

例如，单击“检索”按钮判断和指定位置最近的字符索引，将它显示在标签控件中。代码如下：

```
private void CheckPosition_Click(object sender, EventArgs e)
{
    Point pt=new Point();                          //实例化一个坐标点对象
    pt.X =Convert.ToInt32( this.textBox1.Text); //设置坐标点的 x 坐标
    pt.Y = Convert.ToInt32(this.textBox2.Text); //设置坐标点的 y 坐标
    this.label1.Text = this.maskedTextBox1.GetCharFromPosition(pt).ToString();//在标签控件中显示获取的内容
}
```

（5）SelectAll 方法。该方法表示选定文本框中的所有文本。

```
public void SelectAll()
```

例如，选定文本框中的所有内容。代码如下：

```
this.maskedTextBox1.SelectAll();
```

（6）ValidateText 方法。该方法主要用来将用户输入的字符串转换为验证类别的一个实例。

语法：

```
public Object ValidateText ()
```

返回值：如果转换成功，则为由 ValidatingType 属性指定的类型的 Object；否则，将返回空引用，以指示转换失败。

例如，将用户输入的字符串转换为验证类别的一个实例以信息框的形式显示给用户。代码如下：

```
MessageBox.Show(this.maskedTextBox1.ValidateText().ToString(),"信息提示");
```

4．事件

MaskedTextBox 控件常用事件及说明如表 9.39 所示。

表 9.39　MaskedTextBox 控件常用事件及说明

事　件	说　明
Click	在单击文本框时发生
MaskChanged	在输入掩码更改后发生
MaskInputRejected	当用户的输入或者分配的字符与输入掩码的对应格式元素不匹配时发生
ModifiedChanged	在 Modified 属性的值更改后发生
TextChanged	在 Text 属性值更改时发生

下面对 MaskedTextBox 控件中比较重要的事件进行详细介绍。

（1）MaskChanged 事件。该事件在输入掩码更改后发生。

语法：

```
public event EventHandler MaskChanged
```

（2）MaskInputRejected 事件。该事件在当用户的输入或者分配的字符与输入掩码的对应格式元素不匹配时发生。

语法：

```
public event MaskInputRejectedEventHandler MaskInputRejected
```

说明：MaskInputRejected 是 MaskedTextBox 类的默认事件。

（3）ModifiedChanged 事件。该事件在 Modified 属性的值更改后发生。

语法：

```
public event EventHandler ModifiedChanged
```

（4）TextChanged 事件。该事件在 Text 属性值更改时发生。

语法：

```
public event EventHandler TextChanged
```

示例 09-42　判断输入日期是否合法

本示例通过对 MaskedTextBox 控件的属性、方法以及事件的应用实现对用户输入日期的判断。代码如下：

代码 09-42　光盘位置：光盘\mingrisoft\sl\09\42

```
private void Form1_Load(object sender, EventArgs e)
{
    this.maskedTextBox1.AsciiOnly = true;        //设置 maskedTextBox1 控件允许输入 Ascii 以外的其它字符
    this.maskedTextBox1.BeepOnError = true;   //设置当输入它拒绝的字符时给出系统提示声
    this.maskedTextBox1.CausesValidation = true;                //设置控件在接收焦点后对控件进行验证
    this.maskedTextBox1.HidePromptOnLeave = true;               //当掩码失去焦点时隐藏提示字符
    this.maskedTextBox1.HideSelection = true;                   //设置文本框没有焦点时是否高亮显示
    this.maskedTextBox1.Mask = "00/00/0000";                    //设置 maskedTextBox1 运行时使用的输入掩码
    this.maskedTextBox1.PasswordChar = '●';                     //设置要取代用户输入而显示的字符
    this.maskedTextBox1.PromptChar = '★';                       //设置用户表示缺少用户输入时显示的提示性字符
    this.maskedTextBox1.RejectInputOnFirstFailure = true;       //当遇到一个无效字符时停止对用户输入的分析
    this.maskedTextBox1.TextAlign = HorizontalAlignment.Left;       //设置 maskedTextBox1 控件的文本对齐属性
    //设置在文本框中只显示原义字符
    this.maskedTextBox1.TextMaskFormat = MaskFormat.ExcludePromptAndLiterals;
    this.maskedTextBox1.UseSystemPasswordChar = false;          //设置不使用系统提供的密码
    this.toolTip1.IsBalloon = true;                             //设置气泡提示的类型
    this.toolTip1.ToolTipIcon = ToolTipIcon.Info;               //设置气泡提示的图标
    this.toolTip1.ToolTipTitle = "信息提示";                    //设置气泡提示的标题
    this.toolTip1.UseAnimation = true;                          //设置气泡提示的出现类型
    this.toolTip1.AutoPopDelay = 3000;                          //设置气泡提示的出现时间
}
private void button1_Click(object sender, EventArgs e)
{
    this.maskedTextBox1.Clear();                                //清空文本框中的内容
}
private void maskedTextBox1_MaskInputRejected(object sender, MaskInputRejectedEventArgs e)
{
    //当输入无效字符时给出信息提示
    this.toolTip1.SetToolTip(this.maskedTextBox1, "请输入 XX/XX/XXXX 日期格式");
}
```

运行本示例，效果如图 9.66 所示。

图 9.66　判断输入日期是否合法

9.9　容 器 控 件

容器控件用于容纳和布置其他控件。容器控件主要包括 FlowLayoutPanel 控件、GroupBox 控件、Panel 控件、SplitContainer 控件、TabControl 控件和 TableLayoutPanel 控件等，本节将对比较重要的容器控件作详细介绍。

9.9.1　FlowLayoutPanel 控件

1．功能

FlowLayoutPanel 控件用来在水平或垂直方向排列其内容。开发人员可以将 FlowLayoutPanel 控件的内容从一行换至下一行，或者从一列换至下一列，还可以选择剪裁内容。任何 Windows 窗体控件都可以是 FlowLayoutPanel 控件的子控件，包括 FlowLayoutPanel 的其他实例。利用此

功能，可以在运行时构造适应窗体尺寸大小的复杂布局。图 9.67 所示为 FlowLayoutPanel 控件。

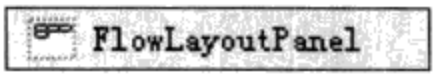


图 9.67 FlowLayoutPanel 控件

2．属性

FlowLayoutPanel 控件常用属性及说明如表 9.40 所示。

表 9.40 FlowLayoutPanel 控件常用属性及说明

属　性	说　明
AutoScroll	获取或设置一个值，该值指示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
AutoScrollMargin	获取或设置自动滚动边距的大小
AutoScrollMinSize	获取或设置自动滚动的最小尺寸
AutoScrollPosition	获取或设置自动滚动定位的位置
AutoSizeMode	指示控件的自动调整大小行为
BackgroundImageLayout	获取或设置在 ImageLayout 枚举中定义的背景图像布局
FlowDirection	获取或设置一个值，该值指示 FlowLayoutPanel 控件的流向
HorizontalScroll	获取与水平滚动条相关联的特性
HScroll	获取或设置一个值，该值指示水平滚动条是否可见
Margin	获取或设置控件之间的空间
VerticalScroll	获取与垂直滚动条相关联的特性
VScroll	获取或设置一个值，该值指示垂直滚动条是否可见
WrapContents	获取或设置一个值，该值指示应当对 FlowLayoutPanel 控件的内容进行换行还是剪裁

下面对 FlowLayOutPanel 控件中比较重要的属性进行详细介绍。

（1）AutoScroll 属性。该属性表示获取或设置一个值，该值指示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。

语法：

```
public virtual bool AutoScroll { get; set; }
```

属性值：如果容器允许自动滚动，则为 true；否则为 false。默认值为 false。

例如，设置容器为允许自动滚动。代码如下：

```
this.flowLayoutPanel1.AutoScroll=true;
```

（2）AutoSizeMode 属性。该属性表示指示控件的自动调整大小行为。

语法：

```
public virtual AutoSizeMode AutoSizeMode { get; set; }
```

属性值：AutoSizeMode 枚举值之一。

AutoSizeMode 的可取值及说明如下。

- GrowAndShrink：表示控件根据它的内容增大或缩小。不能手动调整该控件的大小。
- GrowOnly：表示控件可以根据其内容任意增大，但不会缩小至小于它的 Size 属性值。窗体可以调整大小，但不能缩小到它所包含的任意控件被隐藏。

例如，设置容器的大小可以根据它的内容增大或缩小，同时不能手动调整该控件的大小。代码如下：

```
this.flowLayoutPanel1.AutoSizeMode=AutoSizeMode.GrowAndShrink;
```

（3）BackgroundImageLayout 属性。该属性表示获取或设置在 ImageLayout 枚举中定义的背景图像布局。

语法：

```
public virtual ImageLayout BackgroundImageLayout { get; set; }
```

属性值：ImageLayout 的值之一。

ImageLayout 的可取值及说明如下。

- Center：表示图像在控件的矩形工作区中居中显示。
- None：表示图像沿控件的矩形工作区顶部左对齐。
- Stretch：表示图像沿控件的矩形工作区拉伸。
- Tile：表示图像沿控件的矩形工作区平铺。
- Zoom：表示图像在控件的矩形工作区中放大。

例如，设置容器的背景图像布局为沿着控件的矩形工作区域拉伸。代码如下：

```
this.flowLayoutPanel1.BackgroundImageLayout=ImageLayout.Stretch;
```

（4）FlowDirection 属性。该属性用来指示控件布局的方向。

语法：

```
public FlowDirection FlowDirection { get; set; }
```

属性值：FlowDirection 值之一，指示控件在面板中的连续放置方向。默认值为 LeftToRight。

例如，设置容器中控件的布局方向为自上而下。代码如下：

```
this.flowLayoutPanel1.FlowDirection=FlowDirection.TopDown;
```

说明：FlowDirection 是一个返回值，它用来指定控件布局的方向。当值为 BottomUp 时，元素从设计图面的底部流到顶部；当值为 LeftToRight 时，元素从设计图面的左边缘流到右边缘，默认为此选项；当值为 RightToLeft 时，元素从设计图面的右边缘流到左边缘；当值为 TopDown 时，元素从设计图面的顶部流到底部。

（5）Hscroll 属性。该属性表示获取或设置一个值，该值指示水平滚动条是否可见。

语法：

```
protected bool HScroll { get; set; }
```

属性值：如果水平滚动条是可见的，则为 true；反之，则为 false。

例如，当容器的宽度不够时，可以设置其水平滚动条为可见状态。代码如下：

```
this.flowLayoutPanel1.Hscroll=true;
```

（6）Margin 属性。该属性表示获取或设置控件之间的空间。

语法：

```
public Padding Margin { get; set; }
```

属性值：表示控件之间的间距 Padding。

例如，设置控件间的间距为 10。代码如下：

```
this.flowLayoutPanel1.Margin = new Padding(10,10,10,10);
```

（7）Vscroll 属性。该属性表示获取或设置一个值，该值指示垂直滚动条是否可见。

语法：

```
protected bool VScroll { get; set; }
```

属性值：如果垂直滚动条是可见的，则为 true；反之，则为 false。

例如，当控件的高度不够时，可以设置其垂直滚动条处于可见状态。代码如下：

```
this.flowLayoutPanel1.Vscroll=true;
```

3．方法

FlowLayoutPanel 控件常用方法及说明如表 9.41 所示。

表 9.41　　FlowLayoutPanel 控件常用方法及说明

方　法	说　明
AdjustFormScrollbars	根据当前控件位置和当前所选控件调整容器中的滚动条
GetFlowBreak	返回一个值，该值表示 FlowLayoutPanel 控件的流中断设置
GetScrollState	确定是否设置了指定的标志
ScrollControlIntoView	将指定的子控件滚动到支持自动滚动的控件的视图中
ScrollToControl	计算到指定子控件的滚动偏移量
SetAutoScrollMargin	设置自动滚动边距的大小
SetAutoSizeMode	设置一个值，该值指示当启用控件的 AutoSize 属性时控件的行为方式

续表

方　法	说　明
SetDisplayRectLocation	将显示窗口定位到指定的值
SetFlowBreak	设置一个值，该值表示 FlowLayoutPanel 控件的流中断设置
SetScrollState	设置指定的滚动状态标志

下面对 FlowLayoutPanel 控件中比较重要的方法进行详细介绍。

（1）AdjustFormScrollbars 方法。该方法表示根据当前控件位置和当前所选控件调整容器中的滚动条。

语法：

```
protected virtual void AdjustFormScrollbars(bool displayScrollbars)
```

参数说明如下。

displayScrollbars：如果显示滚动条，则为 true；否则为 false。

例如，当控件的空间不足时，设置滚动条处于显示状态。代码如下：

```
this.flowLayoutPanel1.AdjustFormScrollbars(true);
```

（2）GetFlowBreak 方法。该方法表示返回一个值，该值表示控件的流中断设置。

语法：

```
public bool GetFlowBreak(Control control)
```

参数说明如下。

control：表示子控件。

返回值：如果已设置流中断，则为 true；否则为 false。

例如，获取按钮控件的流中断状态。代码如下：

```
this.flowLayoutPanel1.GetFlowBreak(this.button1);
```

（3）SetFlowBreak 方法。该方法表示设置一个值，该值表示控件的流中断设置。

语法：

```
public void SetFlowBreak(Control control,bool value)
```

参数说明如下。

control：表示子控件。

value：表示要设置的流中断值。

例如，在按钮控件中设置流中断。代码如下：

```
this.flowLayoutPanel1.SetFlowBreak(this.button1,true);
```

4．Layout 事件

该事件是在控件将要对其内容布局进行更改时发生。

语法：

```
public event LayoutEventHandler Layout
```

示例 09-43　通过下拉列表改变控件内部的布局格式

运行本示例，通过在 ComboBox 控件中选择 FlowLayoutPanel 控件的布局方式来改变 FlowLayoutPanel 控件的布局。代码如下：

代码 09-43　光盘位置：光盘\mingrisoft\sl\09\43

```
private void Form1_Load(object sender, EventArgs e)
{
    //设置 flowLayoutPanel1 控件中允许用户滚动到任何放置在其可见边界之外的控件
    this.flowLayoutPanel1.AutoScroll = true;
    //设置 flowLayoutPanel1 控件中自动调整大小方式为依据各自的内容自动调整
    this.flowLayoutPanel1.AutoSizeMode = AutoSizeMode.GrowAndShrink;
    //设置 flowLayoutPanel1 控件中图像的布局方式
    this.flowLayoutPanel1.BackgroundImageLayout = ImageLayout.Stretch;
    this.comboBox1.Items.Add("从左至右");    //向 comboBox1 控件中添加“从左至右”
```

```
        this.comboBox1.Items.Add("从右至左");    //向 comboBox1 控件中添加“从右至左”
        this.comboBox1.Items.Add("从上至下");    //向 comboBox1 控件中添加“从上至下”
        this.comboBox1.Items.Add("从下至上");    //向 comboBox1 控件中添加“从下至上”
        this.comboBox1.SelectedIndex = 0;        //设置 comboBox1 控件默认选中第一项
    }
    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (this.comboBox1.SelectedIndex == 0)      //当 comboBox1 控件的选定项索引为 0 时
        {
            this.flowLayoutPanel1.FlowDirection = FlowDirection.LeftToRight;//设置控件的布局方向为自左至右
        }
        else if (this.comboBox1.SelectedIndex == 1) //当 comboBox1 控件的选定项索引为 1 时
        {
            this.flowLayoutPanel1.FlowDirection = FlowDirection.RightToLeft;//设置控件的布局方向为自右至左
        }
        else if (this.comboBox1.SelectedIndex == 2) //当 comboBox1 控件的选定项索引为 2
        {
            this.flowLayoutPanel1.FlowDirection = FlowDirection.TopDown;//设置控件的布局方向为自上而下
        }
        else if (this.comboBox1.SelectedIndex == 3) //当 comboBox1 控件的选定项索引为 3 时
        {
            this.flowLayoutPanel1.FlowDirection = FlowDirection.BottomUp;//设置控件的布局方向为自下而上
        }
    }
    private void flowLayoutPanel1_Layout(object sender, LayoutEventArgs e)
    {
        //给出布局发生改变的信息
        MessageBox.Show("布局发生改变…", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
```

运行本示例，效果如图 9.68 所示。

图 9.68　通过下拉列表改变控件内部的布局格式

9.9.2　Panel 控件

1．功能

Panel 控件用于为其他控件提供可识别的分组。开发 Windows 应用程序时，通常使用 Panel 控件按功能细分窗体。在窗体设计时，所有控件都可以自由移动，而当移动 Panel 控件时，它包含的所有控件也将随着移动。图 9.69 所示为 Panel 控件。

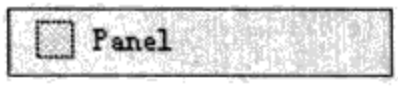


图 9.69　Panel 控件

2．属性

Panel 控件常用属性及说明如表 9.42 所示。

表 9.42　　Panel 控件常用属性及说明

属　性	说　明
AutoScroll	此属性指示当控件内容大于它的可见区域时，是否自动显示滚动条
AutoScrollMargin	获取或设置自动滚动边距的大小
AutoScrollMinSize	获取或设置自动滚动的最小尺寸

续表

属　性	说　明
AutoScrollOffseth	获取或设置一个值，该值指示在 ScrollControlIntoView 中将控件滚动到何处
AutoScrollPosition	获取或设置自动滚动定位的位置
AutoSizeMode	指示控件的自动调整大小行为
BackgroundImageLayout	获取或设置在 ImageLayout 枚举中定义的背景图像布局
BorderStyle	指示控件的边框样式
Controls	获取包含在控件内的控件的集合
HorizontalScroll	获取与水平滚动条相关联的特性
Margin	获取或设置控件之间的空间
VerticalScroll	获取与垂直滚动条相关联的特性

Panel 控件的一些属性在 FlowLayoutPanel 控件中已做过详细说明，这里不再重复讲解。下面详细介绍 Panel 控件的 BorderStyle 属性和 Controls 属性。

（1）BorderStyle 属性。该属性表示指示控件的边框样式。

语法：

```
public BorderStyle BorderStyle { get; set; }
```

属性值：BorderStyle 值之一。默认为 BorderStyle.None。

BorderStyle 的可取值及说明如下。

- None：表示无边框。
- FixedSingle：表示单行边框。
- Fixed3D：表示三维边框。

例如，设置 Panel 控件的边框样式为三维边框。代码如下：

```
this.panel1.BorderStyle=BorderStyle.Fixed3D;
```

（2）Controls 属性。该属性表示获取包含在控件内的控件的集合。

语法：

```
public Control.ControlCollection Controls { get; }
```

属性值：一个 Control.ControlCollection，它表示控件内包含的控件的集合。

例如，获取容器控件中的控件总数。代码如下：

```
int I=this.flowLayoutPanel1.Controls.Count;
```

3．方法

Panel 控件常用方法及说明如表 9.43 所示。

表 9.43　　Panel 控件常用方法及说明

方　法	说　明
AdjustFormScrollbars	根据当前控件位置和当前所选控件调整容器中的滚动条
GetFlowBreak	返回一个值，该值表示 FlowLayoutPanel 控件的流中断设置
GetScrollState	确定是否设置了指定的标志
ScrollControlIntoView	将指定的子控件滚动到支持自动滚动的控件的视图中
ScrollToControl	计算到指定子控件的滚动偏移量
SetAutoScrollMargin	设置自动滚动边距的大小
SetAutoSizeMode	设置一个值，该值指示当启用控件的 AutoSize 属性时控件的行为方式
SetDisplayRectLocation	将显示窗口定位到指定的值
SetScrollState	设置指定的滚动状态标志

Panel 控件常见的方法与 FlowLayoutPanel 控件中的常用方法一致，这里不再进行介绍。

4．事件

下面详细介绍 Panel 控件中比较重要的几个事件。

（1）Layout 事件。该事件在控件将要对其内容布局时发生。

语法：

```
public event LayoutEventHandler Layout
```

（2）Click 事件。该事件在单击 Panel 控件时发生。

语法：

```
public event EventHandler Click
```

（3）DoubleClick 事件。该事件在双击 Panel 控件时发生。

语法：

```
public event EventHandler DoubleClick
```

示例 09-44　Panel 控件的应用

运行本示例，当单击 Panel 控件时，隐藏 Panel 控件中所有的 Button 控件；当双击 Panel 控件时，显示 Panel 控件中的所有 Button 控件。代码如下：

代码 09-44　光盘位置：光盘\mingrisoft\sl\09\44

```
private void Form1_Load(object sender, EventArgs e)
{
    this.panel1.BorderStyle = BorderStyle.Fixed3D;                    //设置 Panel 控件的边框样式
    this.panel1.BackgroundImageLayout = ImageLayout.Stretch;          //设置 Panle 控件中的背景图像布局
    this.panel1.AutoScroll=true ;                                     //设置当控件空间不足时，允许使用滚动条
    this.panel1.Margin = new Padding(10);                             //设置控件间的间距
}
private void panel1_Layout(object sender, LayoutEventArgs e)
{
    //弹出布局改变的提示信息
    MessageBox.Show("正在布局 Panel 控件…", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
}
private void panel1_Click(object sender, EventArgs e)
{
    foreach (Control temp in this.panel1.Controls)                    //循环遍历 panel1 控件中的各个控件
    {
        if (temp.GetType().ToString() == "System.Windows.Forms.Button")//当控件类型为 Button 时
        {
            temp.Visible = false;                                     //隐藏当前控件
        }
    }
}
private void panel1_DoubleClick(object sender, EventArgs e)
{
    foreach (Control temp in this.panel1.Controls)                    //循环遍历 panel1 控件中的各个控件
    {
        if (temp.GetType().ToString() == "System.Windows.Forms.Button")//当控件类型为 Button 时
        {
            temp.Visible = true;                                      //显示当前控件
        }
    }
}
```

运行本示例，效果如图 9.70 所示。

图 9.70　Panel 控件的应用

9.9.3 GroupBox 控件

1．功能

GroupBox 控件用于为其他控件提供可识别的分组。通常，使用 GroupBox 控件细分窗体。当移动单个 GroupBox 控件时，它包含的所有控件也会一起移动。图 9.71 所示为 GroupBox 控件。

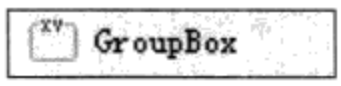


图 9.71　GroupBox 控件

2．属性

GroupBox 控件常用属性及说明如表 9.44 所示。

表 9.44　　GroupBox 控件常用属性及说明

属　性	说　明
AutoScrollOffset	获取或设置一个值，该值指示在 ScrollControlIntoView 中将控件滚动到何处
AutoSize	获取或设置一个值，该值指示 GroupBox 是否根据其内容调整大小
AutoSizeMode	获取或设置启用 AutoSize 属性时 GroupBox 的行为方式
BackColor	获取或设置控件的背景色
BackgroundImage	获取或设置在控件中显示的背景图像
BackgroundImageLayout	获取或设置在 ImageLayout 枚举中定义的背景图像布局
Controls	获取包含在控件内的控件的集合
FlatStyle	获取或设置组框控件的平面样式外观
ForeColor	获取或设置控件的前景色
Tag	获取或设置包含有关控件的数据的对象
Text	获取或设置与此控件关联的文本

下面对 GroupBox 控件中比较重要的属性进行详细介绍。

（1）AutoSize 属性。该属性表示获取或设置一个值，该值指示 GroupBox 是否根据其内容调整大小。

语法：

```
public override bool AutoSize { get; set; }
```

属性值：如果 GroupBox 根据其内容自动调整大小，则为 true；否则为 false。默认为 true。

例如，当 GroupBox 控件的内容不足时，设置其自动显示滚动条。代码如下：

```
this.groupBox1,AutoSize=true;
```

（2）AutoSizeMode 属性。该属性表示获取或设置启用 AutoSize 属性时 GroupBox 的行为方式。

语法：

```
public AutoSizeMode AutoSizeMode { get; set; }
```

属性值：AutoSizeMode 值之一。默认为 GrowOnly。

AutoSizeMode 的可取值及说明如下。

- GrowAndShrink：值表示控件根据它的内容增大或缩小，不能手动调整该控件的大小。
- GrowOnly：值表示控件可以根据其内容任意增大，但不会缩小至小于它的 Size 属性值。窗体可以调整大小，但不能缩小到它所包含的任意控件被隐藏。

例如，设置在 GroupBox 控件中可以依据控件的内容改变控件的大小，但不能手动调整控件大小。代码如下：

```
this.groupBox1.AutoSizeMode=AutoSizeMode.GrowAndShrink;
```

（3）FlatStyle 属性。该属性表示获取或设置组框控件的平面样式外观。

语法：

```
public FlatStyle FlatStyle { get; set; }
```

属性值：FlatStyle 值之一。默认值为 Standard。

FlatSyle 的可取值及说明如下所示。

- Flat：表示该控件以平面显示。
- Popup：表示该控件以平面显示，直到鼠标指针移动到该控件为止，此时该控件外观为三维。
- Standard：表示该控件以平面显示，直到鼠标指针移动到该控件为止，此时该控件外观为三维。
- System：表示该控件的外观是由用户的操作系统决定的。

例如，设置 GroupBox 控件以平面形式显示，直到鼠标指针移动到该控件为止，此时该控件的外观为三维。代码如下：

```
this.groupBox1.FlatStyle=FlatStyle.Popop;
```

（4）Text 属性。该属性表示获取或设置与此控件关联的文本。

语法：

```
public override string Text { get; set; }
```

属性值：与该控件关联的文本。

例如，设置与 GroupBox 控件关联的文本为“操作类型”。代码如下：

```
this.groupBox1.Text="操作类型";
```

3．方法

下面对 GroupBox 控件中比较重要的方法进行详细的讲解。

（1）SetAutoSizeMode 方法。该方法表示设置一个值，该值指示当启用控件的 AutoSize 属性时控件的行为方式。

语法：

```
protected void SetAutoSizeMode(AutoSizeMode mode)
```

参数说明如下。

mode：AutoSizeMode 值之一。

例如，设置当控件启用 AutoSize 属性时控件的行为方式。代码如下：

```
this.groupBox1.SetAutoSizeMode(AutoSizeMode.GrowAndShrink);
```

（2）UpdateStyles 方法。该方法表示强制将分配的样式重新应用到控件。

语法：

```
protected void UpdateStyles()
```

例如，当 GroupBox 控件的样式发生变化时，如果想立刻看到效果，那么可以使用下面的代码实现。代码如下：

```
this.groupBox1.UpdateStyles();
```

4．事件

下面对 GroupBox 控件中比较重要的几个事件进行详细说明。

（1）Enter 事件。该事件在进入控件时发生。

语法：

```
public event EventHandler Enter
```

（2）TextChanged 事件。该事件在 Text 属性值更改时发生。

语法：

```
public event EventHandler TextChanged
```

示例 09-45　GroupBox 控件的应用

本示例中，当程序运行时，单击“焦点”按钮，设置焦点在 groupBox2 控件中，通过对相应事件的设置对 groupBox2 控件进行隐藏。代码如下：

代码 09-45　光盘位置：光盘\mingrisoft\sl\09\45

```
private void Form1_Load(object sender, EventArgs e)
{
    this.groupBox1.AutoSize = true;//设置 groupBox1 控件可以根据控件内的内容大小对自己的大小进行改变
```

```
    this.groupBox1.AutoSizeMode = AutoSizeMode.GrowOnly;                    //设置 groupBox1 控件的行为方式
    this.groupBox1.BackColor = Color.Lavender;                              //设置 groupBox1 控件的背景颜色
    this.groupBox1.BackgroundImage = (Image)Properties.Resources.哼哼;      //设置 groupBox1 控件的背景图片
    this.groupBox1.BackgroundImageLayout = ImageLayout.Stretch;             //设置 groupBox1 控件图片的显示样式
    this.groupBox1.FlatStyle = FlatStyle.Standard;                          //设置 groupBox1 控件的平面样式外观
    this.groupBox2.BackgroundImageLayout = ImageLayout.Stretch;             //设置显示图像的样式
    this.groupBox2.BackgroundImage = (Image)Properties.Resources.幸福;      //设置 groupBox2 控件的背景图像
    this.groupBox2.FlatStyle = FlatStyle.Standard;          //设置 groupBox2 控件的平面样式外观
    this.groupBox1.ForeColor = Color.Red;                   //设置 groupBox1 控件的前景色
    this.groupBox2.ForeColor = Color.Red;                   //设置 groupBox2 控件的前景色
}
private void groupBox2_Enter(object sender, EventArgs e)
{
    this.groupBox1.Text = "容器";    //设置 groupBox1 控件的 Text 属性
}
private void groupBox1_TextChanged(object sender, EventArgs e)
{
    this.groupBox2.Visible = false;    //设置 groupBox2 控件为隐藏状态
}
private void FocusSpot_Click(object sender, EventArgs e)
{
    this.groupBox2.Focus();            //设置 groupBox2 为活动控件
}
```

运行本示例，效果如图 9.72 所示。

图 9.72　GroupBox 控件的应用

9.9.4　SplitContainer 控件

1．功能

SplitContainer 控件被看作是一个复合体，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，指针将相应地呈现为灰度校正状，以显示该拆分条是可移动的。图 9.73 所示为 SplitContainer 控件。

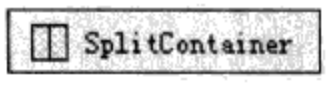


图 9.73　SplitContainer 控件

2．属性

SplitContainer 控件常用属性及说明如表 9.45 所示。

表 9.45　　SplitContainer 控件常用属性及说明

属　　性	说　　明
AutoScaleDimensions	获取或设置控件的设计尺寸
AutoScaleMode	获取或设置控件的自动缩放模式
AutoValidate	获取或设置一个值，该值指示当焦点更改时是否自动验证此容器内的控件
BorderStyle	获取或设置 SplitContainer 的边框样式

续表

属　性	说　明
FixedPanel	获取或设置在调整容器大小时将大小保持不变的 SplitContainer 面板
IsSplitterFixed	获取或设置一个值，用以指示拆分器是固定的还是可移动的
Orientation	获取或设置一个值，该值指示 SplitContainer 面板处于水平方向还是垂直方向
SplitterIncrement	获取或设置一个值，该值表示拆分器移动的增量（以像素为单位）
SplitterWidth	获取或设置拆分器的宽度（以像素为单位）

下面对 SplitContainer 控件中比较重要的属性进行详细介绍。

（1）AutoScaleMode 属性。该属性表示获取或设置控件的自动缩放模式。

语法：

```
public AutoScaleMode AutoScaleMode { get; set; }
```

属性值：表示当前缩放模式的 AutoScaleMode。默认为 None。

AutoScaleMode 的可取值及说明如下。

- None：表示禁用自动缩放。
- Font：表示根据类使用的字体（通常为系统字体）的维度控制缩放。
- Dpi：表示根据显示分辨率控制缩放。常用分辨率为 96DPI 和 120DPI。
- Inherit：表示根据类的父类的缩放模式控制缩放。如果不存在父类，则禁用自动缩放。

例如，设置 SplitContainer 控件禁用自动缩放模式。代码如下：

```
this.splitContainer.AutoScaleMode=AutoScaleMode.None;
```

（2）Orientation 属性。该属性表示获取或设置一个值，该值指示 SplitContainer 面板处于水平方向还是垂直方向。

语法：

```
public Orientation Orientation { get; set; }
```

属性值：Orientation 值之一。默认为 Vertical。当值为 Horizontal 时，水平放置控件或元素；当值为 Verticalt 时，垂直放置控件或元素。

例如，设置控件面板为水平方向。代码如下：

```
this.splitContainer1.Orientation=Orientation.Horizontal;
```

3．方法

（1）SetAutoScrollMargin 方法。该方法表示设置自动滚动边距的大小。

语法：

```
public void SetAutoScrollMargin(int x,int y)
```

参数说明如下。

x：表示边距的宽度。

y：表示边距的高度。

例如，设置自动滚动边距为 50。代码如下：

```
this.splitContainer1.SetAutoScrollMargin(50,50);
```

（2）SetScrollState 方法。该方法表示设置指定的滚动状态标志。

语法：

```
protected void SetScrollState(int bit,bool value)
```

参数说明如下。

bit：表示要设置的滚动状态标志。

value：表示设置标志的值。

4．事件

（1）StyleChanged 事件。该事件在控件样式更改时发生。

语法：

```
public event EventHandler StyleChanged
```

（2）Click 事件。该事件在单击控件时发生。

语法：

```
public event EventHandler Click
```

（3）DoubleClick 事件。该事件在双击控件时发生。

语法：

```
public event EventHandler DoubleClick
```

示例 09-46　通过事件改变 SplitContainer 控件拆分器的面板方向

本示例中，当程序运行时，单击 Panel1 区域，可以使控件水平排列；双击 panel2 区域，可以使控件垂直排列。代码如下：

代码 09-46　光盘位置：光盘\mingrisoft\sl\09\46

```
private void Form1_Load(object sender, EventArgs e)
{
    this.splitContainer1.AutoScaleMode = AutoScaleMode.Font ;//设置 splitContainer1 控件的自动缩放模式
    this.splitContainer1.IsSplitterFixed = false;            //设置可以自动移动 splitContainer1 控件的分隔条
    this.splitContainer1.SplitterIncrement = 2;              //设置拆分器移动的增量
    this.splitContainer1.SplitterWidth = 5;                  //设置拆分器的宽度
}
private void splitContainer1_Panel2_DoubleClick(object sender, EventArgs e)
{
    this.splitContainer1.Orientation = Orientation.Vertical;     //设置拆分器的面板为垂直方向放置
}
private void splitContainer1_Panel1_Click(object sender, EventArgs e)
{
    this.splitContainer1.Orientation = Orientation.Horizontal;   //设置拆分器的面板为水平方向放置
}
```

运行本示例，效果如图 9.74 所示。

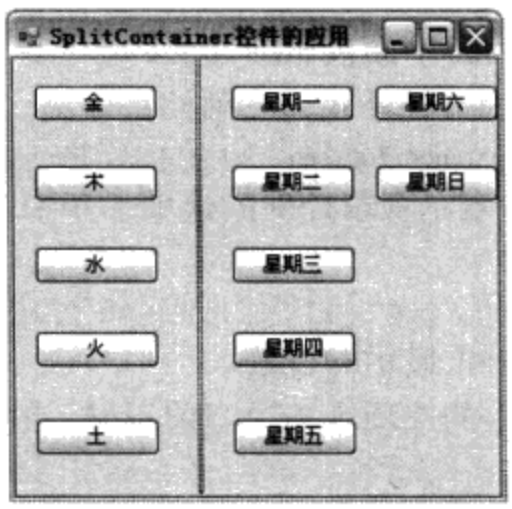


图 9.74　通过事件改变 SplitContainer 控件拆分器的面板方向

9.9.5　TabControl 控件

1．功能

TabControl 控件显示多个选项卡，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。TabControl 控件的选项卡中可包含图片和其他控件。此外，TabControl 控件还可以用来创建一组相关属性的属性页。图 9.75 所示为 TabControl 控件。

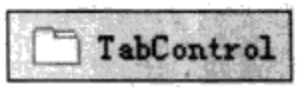


图 9.75　TabControl 控件

2．属性

TabControl 控件常用属性及说明如表 9.46 所示。

表 9.46　　TabControl 控件常用属性及说明

属　性	说　明
Alignment	获取或设置选项卡在其中对齐的控件区域（例如，顶部边缘）
Appearance	获取或设置控件选项卡的可视外观
DrawMode	获取或设置绘制控件的选项卡的方式
HotTrack	获取或设置一个值，该值指示在鼠标移到控件的选项卡时，这些选项卡是否更改外观
ImageList	获取或设置在控件的选项卡上显示的图像
ItemSize	获取或设置控件的选项卡的大小
Multiline	获取或设置一个值，该值指示是否可以显示一行以上的选项卡
RowCount	获取控件的选项卡条中当前正显示的行数
SelectedIndex	获取或设置当前选定的选项卡页的索引
SelectedTab	获取或设置当前选定的选项卡页
ShowToolTips	获取或设置一个值，该值指示当鼠标移到选项卡上时是否显示该选项卡的“工具提示”
SizeMode	获取或设置调整控件的选项卡大小的方式
TabCount	获取选项卡条中选项卡的数目
TabPages	获取该选项卡控件中选项卡页的集合

下面对 TabControl 控件中比较重要的属性进行详细介绍。

（1）Appearance 属性。该属性表示获取或设置控件选项卡的可视外观。

语法：

```
public TabAppearance Appearance { get; set; }
```

属性值：TabAppearance 值之一。默认为 Normal。

TabAppearance 的可取值及说明如下。

- Normal：表示该选项卡具有选项卡的标准外观。
- Buttons：表示选项卡具有三维按钮的外观。
- FlatButtons：表示选项卡具有平面按钮的外观。

例如，设置 TabControl 控件的可视外观为三维按钮外观。代码如下：

```
this.tabControl1.Appearance=TabAppearance.Buttons;
```

（2）RowCount 属性。该属性表示获取控件的选项卡条中当前正显示的行数。

语法：

```
public int RowCount { get; }
```

属性值：表示选项卡条中当前正显示的行数。

例如，获取 TabControl 控件中的选项卡条中当前正显示的行数。代码如下：

```
int I=this.tabControl1.RowCount;
```

（3）SelectedTab 属性。该属性表示获取或设置当前选定的选项卡页。

语法：

```
public TabPage SelectedTab { get; set; }
```

属性值：TabPage，表示选定的选项卡页。如果未选定任何选项卡页，则值为 Null。

例如，设置 TabControl 控件中当前的选项卡页为第 2 页。代码如下：

```
this.tabControl1.SelectedTab=this.tabPage2;
```

（4）SizeMode 属性。该属性表示获取或设置调整控件的选项卡大小的方式。

语法：

```
public TabSizeMode SizeMode { get; set; }
```

属性值：TabSizeMode 值之一。默认为 Normal。

TabSizeMode 的可取值及说明如下。

- Normal：表示调整每个选项卡的宽度以适应在选项卡上显示的内容，并且不调整一行中

的选项卡大小以填充容器控件的整个宽度。

- FillToRight：表示每个选项卡的宽度都要调整，以便选项卡的每行填充容器控件的整个宽度。这仅适用于具有一行以上的选项卡控件。
- Fixed：表示控件中的所有选项卡具有相同的宽度。

例如，设置 TabControl 控件中所有选项卡都具有相同的宽度。代码如下：

```
this.tabControl1.SizeMode=TabSizeMode.Fixed;
```

（5）TabCount 属性。该属性表示获取选项卡条中选项卡的数目。

语法：

```
public int TabCount { get; }
```

属性值：表示选项卡条中选项卡的数目。

例如，获取 TabControl 控件中选项卡条中选项卡的数目。代码如下：

```
int I=this.tabControl1.TabCount;
```

（6）TabPages 属性。该属性表示获取该选项卡控件中选项卡页的集合。

语法：

```
public TabControl.TabPageCollection TabPages { get; }
```

属性值：TabControl.TabPageCollection，它包含该 TabControl 中的 TabPage 对象。

例如，循环遍历 TabControl 控件中的选项卡页，给出其页面标签内容的提示。代码如下：

```
foreach (TabPage item in this.tabControl1.TabPages )
{
    if (item.Text =="")
    {
        MessageBox.Show("该页没有要显示的内容！","信息提示");
    }
    else
    {
        MessageBox.Show("该页的标签为："+item.Text ,"信息提示");
    }
}
```

说明：此集合中的选项卡页的顺序反映了选项卡在控件中出现的顺序。

3．DeselectTab 方法

该方法表示使指定的选项卡后面的选项卡成为当前选项卡。

语法：

```
public void DeselectTab(string tabPageName)
```

参数说明如下。

tabPageName：要取消选择的选项卡的 Name。

例如，设置第 2 页为当前选项卡。代码如下：

```
this.tabControl1.DeselectTab(this.tabPage1);
```

4．事件

（1）Click 事件。该事件在单击选项卡时发生。

语法：

```
public event EventHandler Click
```

（2）Deselected 事件。该事件在当取消选择选项卡时发生。

语法：

```
public event TabControlEventHandler Deselected
```

（3）Deselecting 事件。该事件在取消选择某个选项卡之前发生，使处理程序能够取消选项卡更改。

语法：

```
public event TabControlCancelEventHandler Deselecting
```

（4）Selected 事件。该事件在选择某个选项卡时发生。

语法：

```
public event TabControlEventHandler Selected
```

（5）SelectedIndexChanged 事件。该事件在 SelectedIndex 属性更改后发生。

语法：

```
public event EventHandler SelectedIndexChanged
```

（6）Selecting 事件。该事件在取消选择某个选项卡之前发生，使处理程序能够取消选项卡更改。

语法：

```
public event TabControlCancelEventHandler Selecting
```

示例 09-47　TabControl 控件的应用

本示例中，当程序运行时，单击每一个选项卡按钮，在 DataGridView 控件中显示数据库中与该选项卡标题对应的信息。代码如下：

代码 09-47　光盘位置：光盘\mingrisoft\sl\09\47

```
DataSet dataSet;                                                    //声明一个数据集对象
string ConnString = "Server=.;Database=db_09;integrated security=sspi";   //实例化一个连接数据库字符串
SqlConnection Conn;                                                 //声明一个数据库连接
private void Form1_Load(object sender, EventArgs e)
{
    this.dataGridView1.RowHeadersVisible = false;                   //设置不显示 dataGridView1 控件的行标题
    this.tabControl1.Appearance = TabAppearance.Normal;             //设置 TabControl 控件的外观
    Conn = new SqlConnection(ConnString);                           //实例化一个连接数据库对象
    if (Conn.State == ConnectionState.Closed)                       //当数据库连接处于关闭状态时
    {
        Conn.Open();                                                //打开数据库连接
    }
    dataSet = new DataSet();                                        //实例化一个数据集对象
    string SelectString = "Select * from tb_Department";            //实例化一个查询字符串
    string UserInfoString = "Select * from tb_UserInfo";
    SqlDataAdapter Adapter = new SqlDataAdapter(SelectString, Conn);   //实例化一个 SqlDataAdapter 对象
    SqlDataAdapter UserAdapter = new SqlDataAdapter(UserInfoString, Conn);
    Adapter.Fill(dataSet, "Department");                            //填充数据集
    UserAdapter.Fill(dataSet, "UserInfo");
    Conn.Close();                                                   //关闭数据库连接
    DataTable Table = dataSet.Tables["Department"];                 //实例化一个 DataTable 对象
    this.tabControl1.TabPages.Clear();                              //清空 TabControl 控件中的选项卡页
    for (int i = 0; i < Table.Rows.Count; i++)                      //循环遍历数据表中的每一行数据
    {
        this.tabControl1.TabPages.Add(Table.Rows[i][1].ToString());   //向 TabControl 控件中添加选项卡页
    }
    this.tabControl1.SelectedIndex = 0;     //设置 TabControl 控件中默认选中第 1 项
    this.dataGridView1.Parent = (Control)this.tabControl1.SelectedTab;   //设置 DataGridView 控件的父级容器
    ReadDataBase();                                                 //读取数据库
}
private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.dataGridView1.Parent = (Control)this.tabControl1.SelectedTab;   //设置 DataGridView 控件的父级容器
    ReadDataBase();                                                 //读取数据库
}
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    this.dataSet.Dispose();                         //释放与数据集有关的所有资源
}
public void ReadDataBase()
{
    Conn = new SqlConnection(ConnString);     //实例化一个数据库连接对象
    string SelectUserInfoString = "select User_Id as 编号,User_Name as 姓名,User_Sex as 性别,User_Age as 年
龄,User_Phone as 电话,User_Address as 地址 from tb_UserInfo " +"where tb_UserInfo.Dep_Id=" +
(this.tabControl1.SelectedIndex + 1).ToString();          //实例化一个查询字符串
```

```
    if (Conn.State == ConnectionState.Closed)   //当数据库连接处于关闭状态时
    {
        Conn.Open();                                    //打开数据库连接
    }
    //实例化一个 SqlDataAdapter 对象
    SqlDataAdapter SelectUserInfo = new SqlDataAdapter(SelectUserInfoString, Conn);
    DataSet SelectUserSet = new DataSet();              //实例化一个数据集对象
    SelectUserInfo.Fill(SelectUserSet, "SelectUserInfo");   //填充数据集对象
    //设置 DataGridView 控件的数据源
    this.dataGridView1.DataSource = SelectUserSet.Tables["SelectUserInfo"].DefaultView;
    Conn.Close();                                       //关闭数据库连接
}
```

运行本示例，效果如图 9.76 所示。

图 9.76　TabControl 控件的应用

9.9.6　TableLayoutPanel 控件

1．功能

TableLayoutPanel 控件以网格方式排列内容，因为在设计时和运行时都执行布局，所以当应用程序环境更改时，布局可以动态更改，这使得面板中的控件能够按比例调整大小，以便响应如父控件调整大小或由于本地化引起的文本长度更改等。任何 Windows 窗体控件都可以是 TableLayoutPanel 控件的子控件，包括 TableLayoutPanel 的其他实例，这允许开发人员构造复杂布局以适应运行时地更改。图 9.77 所示为 TableLayoutPanel 控件。

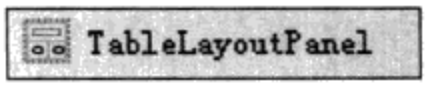


图 9.77　TableLayoutPanel 控件

2．属性

TableLayoutPanel 控件常用属性及说明如表 9.47 所示。

表 9.47　　TableLayoutPanel 控件常用属性及说明

属　性	说　明
CellBorderStyle	获取或设置单元格边框的样式
ColumnCount	获取或设置表中列的数目
ColumnStyles	获取 TableLayoutPanel 的列样式的集合
GrowStyle	获取或设置一个值，该值指示当现有的所有单元格都被占用时，TableLayoutPanel 控件是否应该扩展以容纳新单元格
RowCount	获取或设置表中行的数目
RowStyles	获取 TableLayoutPanel 的行样式的集合

下面对 TableLayoutPanel 中控件比较重要的属性进行详细介绍。

（1）CellBorderStyle 属性。该属性表示获取或设置单元格边框的样式。

语法：

```
public TableLayoutPanelCellBorderStyle CellBorderStyle { get; set; }
```

属性值：描述表中所有单元格的边框样式的 TableLayoutPanelCellBorderStyle 值之一。默

认为 None。

TableLayoutPanelCellBorderStyle 的可取值及说明如下。

- None：表示无边框。
- Single：表示单行边框。
- Inset：表示单线凹陷边框。
- InsetDouble：表示双线凹陷边框。
- Outset：表示单线凸起边框。
- OutsetDouble：表示双线凸起边框。
- OutsetPartial：表示包含凸起部分的单线边框。

例如，设置 TableLayoutPanel 控件的边框样式为单行边框。代码如下：

```
this.tableLayoutPanel1.CellBorderStyle=TableLayoutPanelCellBorderStyle.Single;
```

（2）ColumnCount 属性。该属性用于获取或设置表中列的数目。

语法：

```
public int ColumnCount { get; set; }
```

属性值：TableLayoutPanel 控件中列的数目，默认值为 0。

例如，下面设置 TableLayoutPanel 控件的 RowCount 属性和 ColumnCount 属性，代码如下：

```
this.tableLayoutPanel1.RowCount = 2;
tableLayoutPanel1.ColumnCount = 2;
```

（3）ColumnStyles 属性。该属性表示获取 TableLayoutPanel 的列样式的集合。

语法：

```
public TableLayoutColumnStyleCollection ColumnStyles { get; }
```

属性值：包含 TableLayoutPanel 控件中每个列的 ColumnStyle 的 TableLayoutColumnStyleCollection。

例如，通过单击 Button 按钮改变每一列的样式。代码如下：

```
private void button1_Click(System.Object sender,System.EventArgs e)
{
    //实例化一个 TableLayoutColumnStyleCollection 类的实例
    TableLayoutColumnStyleCollection styles = this.TableLayoutPanel1.ColumnStyles;
    foreach (ColumnStyle style in styles)                    //循环遍历集合中的每一项
    {
        if (style.SizeType == SizeType.Absolute)             //当该项的行或列的大小调整模式为确切的像素数时
        {
            style.SizeType = SizeType.AutoSize;//设置该项行和列的大小调整模式为自动调整大小
        }
        else if (style.SizeType == SizeType.AutoSize)   //当该项行和列的大小调整模式为自动调整大小时
        {
            style.SizeType = SizeType.Percent;//设置该项行和列的大小调整模式为按父容器的百分比调整大小
            style.Width = 33;                          //设置列的宽度值
        }
        else                                           //当为其他情况时
        {
            style.SizeType = SizeType.Absolute; //设置该项的行或列的大小调整模式为确切的像素数
            style.Width = 50;                          //设置列的宽度值
        }
    }
}
```

（4）RowCount 属性。该属性表示获取或设置表中行的数目。

语法：

```
public int RowCount { get; set; }
```

属性值：TableLayoutPanel 控件中行的数目。默认值为 0。

（5）RowStyles 属性。该属性表示获取 TableLayoutPanel 的行样式的集合。

语法：

```
public TableLayoutRowStyleCollection RowStyles { get; }
```

属性值：包含 TableLayoutPanel 控件中每行的 RowStyle 的 TableLayoutRowStyleCollection。

说明：RowStyles 属性的用法与 ColumnStyles 属性的用法相似，有关 RowStyles 的用法详见 ColumnStyles 属性。

3．方法

TableLayoutPanel 控件常用方法及说明如表 9.48 所示。

表 9.48　　TableLayoutPanel 控件常用方法及说明

方　法	说　明
GetCellPosition	获取表示单元格的行和列的 TableLayoutPanelCellPosition
GetColumn	此方法返回指定子控件的列位置
GetColumnSpan	返回指定子控件跨的列数
GetColumnWidths	返回一个数组，该数组表示 TableLayoutPanel 中的列的宽度（以像素为单位）
GetControlFromPosition	返回占据指定位置的子控件
GetRow	此方法返回指定子控件的行位置
GetRowSpan	返回指定子控件跨的行数
SetColumn	设置指定子控件的列位置
SetColumnSpan	设置子控件跨的列数
SetDisplayRectLocation	将显示窗口定位到指定的值
SetRow	此方法设置指定子控件的行位置
SetRowSpan	设置子控件跨的行数

下面对 TableLayoutPanel 控件中比较重要的方法进行详细介绍。

（1）GetRow 方法。该方法返回指定子控件的行位置。

语法：

```
public int GetRow (Control control)
```

参数说明如下。

control：TableLayoutPanel 的子控件。

返回值：control 的行位置，如果 control 的位置由 LayoutEngine 决定，则为-1。

例如，TableLayoutPanel 控件中有 Button 按钮，判断其所在的行位置。代码如下：

```
int I=this.tableLayoutPanel1.GetRow(this.button1);
```

（2）GetColumn 方法。该方法返回指定子控件的列位置。

语法：

```
public int GetColumn (Control control)
```

参数说明如下。

control：TableLayoutPanel 的子控件。

返回值：指定子控件的列位置，如果 control 的位置由 LayoutEngine 决定，则为-1。

例如，TableLayoutPanel 控件中有 Button 按钮，判断其所在的列位置。代码如下：

```
int j=this.tableLayoutPanel1.GetColumn(this.button1);
```

4．事件

（1）Click 事件。该事件在单击控件时发生。

语法：

```
public event EventHandler Click
```

（2）DoubleClick 事件。该事件在双击控件时发生。

语法：

```
public event EventHandler DoubleClick
```

示例 09-48　单击对应的按钮显示指定按钮的所在位置

本示例实现的功能为：当程序运行时，单击“风雷位置”按钮时，弹出信息提示框提示“风”、

“雷”按钮的位置；单击“雨电位置”按钮时，弹出信息提示框提示“雨”、“电”按钮的位置。单击 TableLayoutPanel 控件，隐藏所有 Button 控件；双击 TableLayoutPanel 控件，显示所有 Button 控件。代码如下：

代码 09-48　光盘位置：光盘\mingrisoft\sl\09\48

```
private void Form1_Load(object sender, EventArgs e)
{
    //设置 tableLayoutPanel1 控件的边框样式
    this.tableLayoutPanel1.CellBorderStyle = TableLayoutPanelCellBorderStyle.Outset ;
    //设置当 tableLayoutPanel1 控件内容空间不足时所采取的处理方式
    this.tableLayoutPanel1.GrowStyle = TableLayoutPanelGrowStyle.AddRows;
    this.tableLayoutPanel1.RowCount = 3;                    //设置 tableLayoutPanel1 控件的行数
    this.tableLayoutPanel1.ColumnCount = 2;                 //设置 tableLayoutPanel1 控件的列数
}
private void PositionOne_Click(object sender, EventArgs e)
{
    MessageBox.Show("按钮风在第" + this.tableLayoutPanel1.GetRow(Wind).ToString() + "行" + this.tableLayoutPanel1.
GetColumn(Wind).ToString() + "列\n 按钮雷在第" + this.tableLayoutPanel1.GetRow(Thunder).ToString() + "第" + this.tableLayoutPanel1.
GetColumn(Thunder).ToString() + "列", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);  //显示“风”、
“雷”按钮所处的位置
}
private void PositionTwo_Click(object sender, EventArgs e)
{
    MessageBox.Show("按钮雨在第" + this.tableLayoutPanel1.GetRow(Rain).ToString() + "行" + this.tableLayoutPanel1.
GetColumn(Rain).ToString() + "列\n 按钮电在第" + this.tableLayoutPanel1.GetRow(Electricity).ToString() + "第" + this.tableLayoutPanel1.
GetColumn(Electricity).ToString() + "列", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Information);    //显示
“雨”、“电”按钮所处的位置
}
private void tableLayoutPanel1_Click(object sender, EventArgs e)
{
    foreach (Control Temp in this.tableLayoutPanel1.Controls)  //循环遍历 tableLayoutPanel1 控件中的控件集合
    {
        if (Temp.GetType().ToString() == "System.Windows.Forms.Button")//当当前控件的类型为 Button 类型时
        {
            Temp.Visible = false;                           //修改当前控件的可见状态为不可见
        }
    }
}
private void tableLayoutPanel1_DoubleClick(object sender, EventArgs e)
{
    foreach (Control Temp in this.tableLayoutPanel1.Controls)  //循环遍历 tableLayoutPanel1 控件中的控件集合
    {
        if (Temp.GetType().ToString() == "System.Windows.Forms.Button")//当当前控件的类型为 Button 类型时
        {
            Temp.Visible = true;                            //修改当前控件的可见状态为可见
        }
    }
}
```

运行本示例，效果如图 9.78 所示。

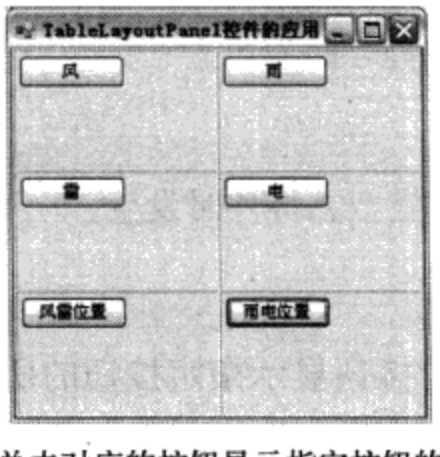


图 9.78　单击对应的按钮显示指定按钮的所在位置

9.10 组　　件

Windows 窗体应用程序提供了若干标准组件，包括 BackgroundWorker 组件、DirectoryEntry 组件、DirectorySearcher 组件、ErrorProvider 组件、EventLog 组件、FileSystemWatcher 组件、HelpProvider 组件、ImageList 组件、MessageQueue 组件、Process 组件、PerformanceCounter 组件、Serialport 组件、ServiceController 组件和 Timer 组件等，下面对主要的组件进行详细介绍。

9.10.1 BackgroundWorker 组件

1．功能

在运行程序时，有些操作可能需要很长的执行时间，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的响应，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”）执行耗时的操作。图 9.79 所示为 BackgroundWorker 组件。

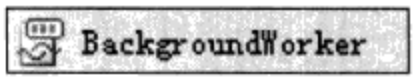


图 9.79　BackgroundWorker 组件

2．属性

BackgroundWorker 组件常用属性及说明如表 9.49 所示。

表 9.49　**BackgroundWorker 组件常用属性及说明**

属　性	说　明
CancellationPending	获取一个值，指示应用程序是否已请求取消后台操作
IsBusy	获取一个值，指示 BackgroundWorker 是否正在运行异步操作
WorkerReportsProgress	获取或设置一个值，该值指示 BackgroundWorker 能否报告进度更新
WorkerSupportsCancellation	获取或设置一个值，该值指示 BackgroundWorker 是否支持异步取消

下面对 BackgroundWorker 组件中比较重要的属性进行详细介绍。

（1）WorkerReportsProgress 属性。该属性表示获取或设置一个值，该值指示 BackgroundWorker 能否报告进度更新。

语法：

```
public bool WorkerReportsProgress { get; set; }
```

属性值：如果 BackgroundWorker 支持进度更新，则为 true；否则为 false。默认为 false。

例如，设置 BackgroundWorker 组件报告进度更新。代码如下：

```
this.backgroundWorker1.WorkerReportsProgress=true;
```

（2）WorkerSupportsCancellation 属性。该属性用于获取或设置一个值，该值指示 BackgroundWorker 是否支持异步取消。

语法：

```
public bool WorkerSupportsCancellation { get; set; }
```

属性值：如果 BackgroundWorker 支持取消，则为 True；否则为 False。默认值为 False。

例如，设置 BackgroundWorker 组件支持异步取消，代码如下：

```
this.backgroundWorker1.WorkerSupportsCancellation=true;
```

3．方法

（1）CancelAsync 方法。该方法用于请求取消挂起的后台操作。

语法：

```
public void CancelAsync ()
```

例如，使用 BackgroundWorker 组件取消挂起的后台操作。代码如下：

```
this.backgroundWorker1.CancelAsync();
```

（2）RunWorkerAsync 方法。该方法用于开始执行后台操作。

语法：

```
public void RunWorkerAsync()
```

例如，使用 BackgroundWorker 组件开始执行后台操作。代码如下：

```
this.backgroundWorker1.RunWorkerAsync();
```

（3）ReportProgress 方法。该方法用于引发 ProgressChanged 事件。

语法：

```
public void ReportProgress(int percentProgress)
```

参数说明如下。

percentProgress：表示已完成的后台操作所占的百分比，范围为 0～100%。

例如，当已完成的后台操作达到 0.01%时，引发 ProgressChanged 事件。代码如下：

```
int I=0.01%;
this.backgroundWorker1.ReportProgress(I*1000);
```

4．事件

（1）DoWork 事件。该事件用于操作在另一个线程上运行事件处理程序。

语法：

```
public event DoWorkEventHandler DoWork
```

说明：调用 RunWorkerAsync 方法时，将引发此事件。RunWorkerAsync 方法可以启动操作执行可能很耗时的工作。

（2）ProgressChanged 事件。该事件在调用 ReportProgress 时发生。

语法：

```
public event ProgressChangedEventHandler ProgressChanged
```

（3）RunWorkerCompleted 事件。该事件在后台操作已完成、被取消或引发异常时发生。

语法：

```
public event RunWorkerCompletedEventHandler RunWorkerCompleted
```

示例 09-49　BackgroundWorker 组件的应用

通过对 BackgroundWorker 组件的属性、方法和事件的应用，实现在计算斐波那契数时在进度条中显示计算进度。代码如下：

代码 09-49　光盘位置：光盘\mingrisoft\sl\09\49

```
private void Form1_Load(object sender, EventArgs e)
{
    this.backgroundWorker1.WorkerReportsProgress = true;      //设置 backgroundWorker1 控件可以报告进度更新
    this.backgroundWorker1.WorkerSupportsCancellation = true;//设置设置 backgroundWorker1 控件支持异步更新
    this.numericUpDown1.Minimum = 1;                          //设置 numericUpDown 控件的最小值
    this.numericUpDown1.Maximum = 91;                         //设置 numericUpDown 控件的最大值
    this.label1.Text = "没有结果";                            //设置 label1 控件的文本值
}
private int numberToCompute = 0;                              //设置进行计算的数字
private int highestPercentageReached = 0;                     //设置可以达到的最大百分比
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
    e.Result = ComputeFibonacci((int)e.Argument, backgroundWorker1, e); //设置异步操作的结果值
}
private void backgroundWorker1_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    this.progressBar1.Value = e.ProgressPercentage;           //设置 progressBar1 控件的进度值
}
private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    if (e.Error != null)                                      //当异步操作期间发生错误时
    {
```

```
            MessageBox.Show(e.Error.Message);                    //弹出错误提示信息
        }
        else if (e.Cancelled)                                    //当取消异步操作时
        {
            this.label1 .Text = "取消";                          //设置 label1 控件的文本为“取消”
        }
        else                                                     //当为其他情况时
        {
            this.label1 .Text = e.Result.ToString();             //设置 label1 中的文本为操作结果
        }
        this.numericUpDown1.Enabled = true;                      //设置 numericUpDown1 控件为可用状态
        Start.Enabled = true;                                    //设置“开始”按钮为可用状态
        Cancel.Enabled = false;                                  //设置“取消”按钮为不可用状态
    }
    private void startAsyncButton_Click(object sender, EventArgs e)
    {
        this.label1 .Text = String.Empty;                        //设置 label1 控件的文本内容为空
        this.numericUpDown1.Enabled = false;                     //设置 numericUpDown1 控件为不可用状态
        this.Start .Enabled = false;                             //设置“开始”按钮为不可用状态
        this.Cancel .Enabled = true;                             //设置“取消”按钮为不可用状态
        numberToCompute = (int)numericUpDown1.Value;             //保存 numericUpDown1 控件的当前值
        highestPercentageReached = 0;                            //设置 highestPercentageReached 的值为 0
        this.backgroundWorker1.RunWorkerAsync(numberToCompute);  //开始执行后台操作
    }
    private void cancelAsyncButton_Click(object sender, EventArgs e)
    {
        this.backgroundWorker1.CancelAsync();                    //调用取消挂起后的操作
        this.progressBar1.Value = 0;                             //设置 progressBar1 控件的当前值为 0
        this.Start .Enabled = true;                              //设置“开始”按钮为可用状态
        this.Cancel .Enabled = false;                            //设置“取消”按钮为不可用状态
        this.label1 .Text = "没有结果";                          //设置 label1 控件的文本内容
    }
    long ComputeFibonacci(int n, BackgroundWorker worker, DoWorkEventArgs e)
    {
        if ((n < 0) || (n > 91))                                 //当 n 的值不在有效值范围内时
        {
            throw new ArgumentException("数值必须在 0-91 之间", "n");     //抛出异常
        }
        long result = 0;                                         //定义一个 long 型变量
        if (worker.CancellationPending)                          //当请求取消后台操作时
        {
            e.Cancel = true;                                     //取消事件的值
        }
        else                                                     //当未请求取消后台操作时
        {
            if (n < 2)                                           //当 n 的值小于 2 时
            {
                result = 1;                                      //设置 result 的值为 1
            }
            else                                                 //当 n 的值为不小于 2 的值时
            {
                //求当前数之前的两个数的和
                result = ComputeFibonacci(n - 1, worker, e) + ComputeFibonacci(n - 2, worker, e);
            }
            int percentComplete = (int)((float)n / (float)numberToCompute * 100);     //保存百分比的值
            if (percentComplete > highestPercentageReached)                           //当百分比的值大于最大值时
            {
                highestPercentageReached = percentComplete;                           //设置最大值为当前百分比的值
                worker.ReportProgress(percentComplete);                               //引发 ProgressChanged 事件
            }
        }
        return result;                                           //返回 result 的值
    }
```

运行本示例，效果如图 9.80 所示。

图 9.80　BackgroundWorker 组件的应用

9.10.2　DirectoryEntry 组件

1．功能

DirectoryEntry 类封装 Active Directory 层次结构中的节点或对象，使用该类可以绑定到对象，或者读取和更新属性。图 9.81 所示为 DirectoryEntry 组件。

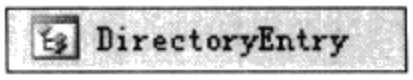


图 9.81　DirectoryEntry 组件

2．属性

DirectoryEntry 组件常用属性及说明如表 9.50 所示。

表 9.50　　DirectoryEntry 组件常用属性及说明

属　性	说　明
AuthenticationType	获取或设置要使用的身份验证的类型
Children	获取 Active Directory 域服务层次结构中此节点的子项
Guid	获取 DirectoryEntry 的 GUID
Name	获取用基础目录服务命名的对象的名称
NativeGuid	获取提供程序返回的 DirectoryEntry 的 GUID
NativeObject	获取本机 Active Directory 服务接口（ADSI）对象
ObjectSecurity	获取或设置此项的安全说明符
Options	获取此项的提供程序特定选项
Parent	获取 Active Directory 域服务层次结构中此项的父级
Password	设置客户端身份验证所使用的密码
Path	获取或设置此 DirectoryEntry 的路径
Properties	获取此 DirectoryEntry 对象的 Active Directory 域服务属性
SchemaClassName	获取用于此 DirectoryEntry 对象的架构类的名称
SchemaEntry	获取此项的架构对象
UsePropertyCache	获取或设置一个值，该值指示在每次运算之后是否应提交缓存
Username	获取或设置在对客户端进行身份验证时使用的用户名

下面对 DirectoryEntry 组件中比较重要的属性进行详细介绍。

（1）AuthenticationType 属性。该属性表示获取或设置要使用的身份验证的类型。

语法：

```
public AuthenticationTypes AuthenticationType { get; set; }
```

属性值：AuthenticationTypes 值之一。

例如，设置 DirectoryEntry 组件不使用身份验证类型。代码如下：

```
this.directoryEntry.AuthenticationType=AuthenticationTypes.Anonymous;
```

AuthenticationTypes 的成员名称及说明如表 9.51 所示。

表 9.51 AuthenticationTypes 的成员名称及说明

属　性	说　明
None	相当于零，表示在 LDAP 提供程序中使用基本身份验证（简单绑定）
Secure	请求安全身份验证。当设置此标记后，WinNT 提供程序将使用 NTLM 来对客户端进行身份验证。Active Directory 域服务使用 Kerberos（并可能使用 NTLM）来对客户端进行身份验证。当用户名和密码为 null 引用（Visual Basic 中为 Nothing）时，ADSI 将使用调用线程的安全上下文来绑定到对象，该上下文是应用程序运行的用户账户的安全上下文或调用线程模拟的客户端用户账户的安全上下文
Encryption	将加密签名附加到消息中，该签名既标识发送方，也确保消息未在传送中修改
SecureSocketsLayer	将加密签名附加到消息中，该签名既标识发送方，也确保消息未在传送中修改。Active Directory 域服务要求安装证书服务器来支持安全套接字层（SSL）加密
ReadonlyServer	对于 WinNT 提供程序，ADSI 尝试连接到域控制器。对于 Active Directory 域服务，此标记指示无服务器绑定不需要可写服务器
Anonymous	未执行任何身份验证
FastBind	指定 ADSI 将不会尝试查询 Active Directory 域服务 objectClass 属性。因此，只有所有 ADSI 对象支持的基接口才会公开。对象支持的其他接口将不可用。用户可以使用此选项来提高仅涉及基接口方法的一系列对象操作的性能。但是，ADSI 不验证服务器上是否实际存在任何请求对象
Signing	验证数据完整性，以确保接收的数据与发送的数据相同。Secure 标记还必须设置为使用签名
Sealing	使用 Kerberos 将数据加密。Secure 标记也必须设置为使用密封
Delegation	启用 Active Directory 服务接口(ADSI)来委托用户的安全上下文，它是在各个域之间移动对象所必需的
ServerBind	如果 ADsPath 包含服务器名称，请在使用 LDAP 提供程序时指定此标记。请勿将此标记用于包含域名的路径或无服务器的路径。如果在指定服务器名称的同时没有指定此标记，则会导致不必要的网络通信量

（2）Options 属性。该属性表示获取此项的提供程序特定选项。

语法：

```
public DirectoryEntryConfiguration Options { get; }
```

属性值：一个 DirectoryEntryConfiguration 对象，包含此项的提供程序特定选项。

例如，获取此项的提供程序的特定选项。代码如下：

```
this.textBox.Text=this.directoryEntry1.Options;
```

（3）Password 属性。该属性表示设置客户端身份验证所使用的密码。

语法：

```
public string Password { set; }
```

属性值：在对客户端进行身份验证时使用的密码。

例如，设置客户端身份验证所使用的密码为“123456”。代码如下：

```
this.directoryEntry1.Password=“123456”;
```

（4）Path 属性。该属性表示用于获取或设置 DirectoryEntry 对象的路径，默认值为空字符串（“”）。

语法：

```
public string Path { get; set; }
```

属性值：DirectoryEntry 对象的路径，默认值为空字符串（“”）。

例如，设置 DirectoryEntry 的路径为本地服务器。代码如下：

```
this.directoryEntry1 .Path = "WinNT://workgroup/Localhost";
```

Path 属性的语法将会随着提供程序的不同而不同。一些常见的情况如下。

● WinNT

① 连接到计算机上的组。例如，“WinNT://<域名>/<计算机名>/<组名>”。如果是连接到本地计算机，则为“WinNT://<计算机名>/<组名>”。

② 连接到计算机上的用户。例如，“WinNT://<域名>/<计算机名>/<用户名>”。如果是连接到本地计算机，则为“WinNT://<计算机名>/<用户名>”。

③ 连接到计算机上的服务。例如，“WinNT://<域名>/<计算机名>/<服务名>”。如果是连接到本地计算机，则为“WinNT://<计算机名>/<服务名>”。

④ 发现网络上的所有域。例如，“WinNT:”。通过枚举此项的子级可以找到这些域。

● LDAP

① 连接到域中的组。例如，“LDAP://CN=<组名>, CN =<用户>, DC=<域控制器 1>, DC=<域控制器 2>,...”。

② 连接到域中的用户。例如，“LDAP://CN=<完整用户名>, CN=<用户>, DC=<域控制器 1>, DC=<域控制器 2>,...”。

③ 连接到域中的计算机。例如，“LDAP://CN=<计算机名>, CN=<计算机>, DC=<域控制器 1>, DC=<域控制器 2>,...”。

● IIS

① 连接到 Web 目录。例如，“IIS://LocalHost/W3SVC/1/ROOT/<Web 目录名>”。

② 若要使用 LDAP 绑定到当前域，请使用路径“LDAP://RootDSE”，然后获取默认命名上下文，并重新绑定该项。

3．方法

DirectoryEntry 组件常用方法及说明如表 9.52 所示。

表 9.52　　DirectoryEntry 组件常用方法及说明

方　法	说　明
Close	关闭 DirectoryEntry 对象并释放与此组件关联的任何系统资源
CommitChanges	将目录项所作更改保存到基础目录存储中
DeleteTree	从 Active Directory 域服务层次结构中删除此项及其整个子树
Exists	确定指定路径是否表示目录服务实际项
Invoke	对本机 Active Directory 域服务对象调用方法
InvokeSet	设置本机 Active Directory 域服务对象上的属性
MoveTo	移动此 DirectoryEntry
RefreshCache	将此 DirectoryEntry 对象的属性值加载到属性缓存中
Rename	更改此 DirectoryEntry 对象的名称

下面对 DirectoryEntry 组件中比较重要的方法进行详细介绍。

（1）Close 方法。该方法表示关闭 DirectoryEntry 对象并释放与此组件关联的任何系统资源。

语法：

```
public void Close()
```

例如，关闭与 DirectoryEntry 对象并释放与此组件关联的任何系统资源。代码如下：

```
this.directoryEntry1.Close();
```

（2）DeleteTree 方法。该方法表示从 Active Directory 域服务层次结构中删除此项及其整个子树。

语法：

```
public void DeleteTree()
```

例如，从 ActiveDirectory 域服务层次结构中删除此项及其整个子树。代码如下：

```
this.directoryEntry1.DeleteTree();
```

（3）Exists 方法。该方法用于确定指定路径是否表示目录的实际项。

语法：

```
public static bool Exists (string path)
```

参数说明如下。

path：要验证项的路径。

返回值：如果指定路径表示目录服务的实际项，则为 true；否则为 false。

例如，在控制台环境下判断指定路径是否表示目录的实际项。代码如下：

```
string myADSPath = "LDAP://onecity/CN=Users,DC=onecity,DC=corp,DC=fabrikam,DC=com";
if (DirectoryEntry.Exists(myADSPath))
{
    Console.WriteLine("当前路径 {0} 是目录的实际项", myADSPath);
}
else
{
    Console.WriteLine("当前路径 {0} 不是目录的实际项", myADSPath);
}
```

（4）MoveTo 方法。该方法表示移动此 DirectoryEntry。

语法：

```
public void MoveTo(DirectoryEntry newParent)
```

参数说明如下。

newParent：要将此项移动到的父级。

例如，将当前 DirectoryEntry 对象移动到指定的父级。代码如下：

```
DirectoryEntry directoryEntry = new DirectoryEntry("WinNT://MINGRISOFT/Localhost");
this.directoryEntry1.MoveTo(directoryEntry);
```

（5）RefreshCache 方法。该方法表示将此 DirectoryEntry 对象的属性值加载到属性缓存中。

语法：

```
public void RefreshCache()
```

例如，将 DirectoryEntry 对象的属性值加载到属性缓存中。代码如下：

```
this.directoryEntry1.RegreshCache();
```

示例 09-50　DirectoryEntry 组件的应用

通过对 DirectoryEntry 组件的属性和方法的使用，借助于 TreeView 控件显示本机的用户、组以及服务。代码如下：

代码 09-50　光盘位置：光盘\mingrisoft\sl\09\50

```
private void Form1_Load(object sender, EventArgs e)
{
    //设置 directoryEntry1 控件所使用的验证类型
    this.directoryEntry1.AuthenticationType = System.DirectoryServices.AuthenticationTypes.Signing;
    this.directoryEntry1.UsePropertyCache = true; //设置 directoryEntry 组件每次运算后都提交缓存
    this.directoryEntry1 .Path = "WinNT://MINGRISOFT/Localhost"; //设置组件的路径
    TreeNode users = new TreeNode("用户");                    //实例化一个“Users”节点
    TreeNode groups = new TreeNode("组");                     //实例化一个“Groups”节点
    TreeNode services = new TreeNode("服务");                 //实例化一个“Services”节点
    //向 treeView1 控件集合中添加指定的一组树节点
    this.treeView1.Nodes.AddRange(new TreeNode[] { users, groups, services });
    foreach (DirectoryEntry child in this.directoryEntry1.Children)   //获取域服务层次结构中此节点的子项
    {
        TreeNode newNode = new TreeNode(child.Name);          //实例化一个 TreeNode 节点
        switch (child.SchemaClassName)                        //判断当前架构类的名称
        {
            case "User":                                      //当当前架构类的名称为 User 时
                users.Nodes.Add(newNode);                     //向 users 节点下添加内容
                break;                                        //跳出 switch 语句
            case "Group":                                     //当当前架构类的名称为 Group 时
                groups.Nodes.Add(newNode);                    //向 groups 节点下添加内容
                break;                                        //跳出 switch 语句
```

```
                case "Service":                                    //当当前架构类的名称为 Service 时
                    services.Nodes.Add(newNode);                   //向 Service 节点下添加内容
                    break;                                         //跳出 switch 语句
            }
            AddPathAndProperties(newNode, child);                  //为新添节点指定路径和属性
        }
        this.treeView1.ExpandAll();                                //展开 treeView1 控件中的所有节点
    }
    private void AddPathAndProperties(TreeNode node, DirectoryEntry entry)
    {
        node.Nodes.Add(new TreeNode("路径:" + entry.Path));         //向节点中添加路径节点
        TreeNode propertyNode = new TreeNode("属性");              //实例化一个属性节点
        node.Nodes.Add(propertyNode);                              //添加属性节点
        foreach (string propertyName in entry.Properties.PropertyNames) //循环遍历属性集中的每一条记录
        {
            string oneNode = propertyName + ":" + entry.Properties[propertyName][0].ToString();//保存属性的内容
            propertyNode.Nodes.Add(new TreeNode(oneNode));         //向属性节点中添加属性内容
        }
    }
```

运行本示例，效果如图 9.82 所示。

图 9.82　DirectoryEntry 组件的应用

注意：运行本示例时，如果出现错误，那么确认 DirectoryEntry 组件的 Path 属性是否正确。例如，本示例中，本地计算机上存在一个域服务器 MINGRISOFT，因此 Path 属性必须设置为“WinNT://MINGRISOFT/Localhost”，通常情况下该属性为“WinNT://workgroup/Localhost”。

9.10.3　DirectorySearcher 组件

1．功能

DirectorySearcher 组件用于对 Active Directory 执行查询。图 9.83 所示为 DirectorySearcher 组件。

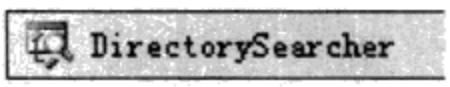


图 9.83　DirectorySearcher 组件

2．属性

DirectorySearcher 组件常用属性及说明如表 9.53 所示。

表 9.53　　DirectorySearcher 组件常用属性及说明

属　性	说　明
Asynchronous	获取或设置一个值，该值指示是否异步执行搜索
AttributeScopeQuery	获取或设置要在其中进行搜索的识别名属性的 LDAP 显示名称。只有一个属性可以用于此类型的搜索
CacheResults	获取或设置一个值，该值指示是否在客户计算机上缓存结果
ClientTimeout	获取或设置客户端等候服务器返回结果的最长时间。如果服务器在此时间内不响应，则终止搜索，不返回任何结果
DerefAlias	获取或设置一个值，该值指示应如何解析搜索过程中找到对象的别名

续表

属　性	说　明
DirectorySynchronization	获取或设置表示要与搜索一起使用的目录同步控件的对象
ExtendedDN	获取或设置指示识别名格式的值
Filter	获取或设置指示轻量目录访问协议（LDAP）格式筛选器字符串的值
PageSize	获取或设置指示分页搜索中的页大小的值
PropertiesToLoad	获取一个值，该值指示搜索过程中要检索的属性列表
PropertyNamesOnly	获取或设置一个值，该值指示搜索是否只检索已经为其赋值的属性的名称
ReferralChasing	获取或设置指示如何追踪转交（Referral）的值
SearchRoot	获取或设置一个值，该值指示 Active Directory 域服务层次结构中的节点，从该节点处开始搜索
SearchScope	获取或设置指示服务器遵循的搜索范围的值
SecurityMasks	获取或设置指示搜索应为指定属性返回哪些安全访问信息的值
ServerPageTimeLimit	获取或设置指示服务器应该搜索单页结果的最长时间的值。这不同于整个搜索的时间限制
ServerTimeLimit	ServerTimeLimit 属性获取或设置指示服务器在进行搜索时可花费的最长时间的值。如果达到时间限制，则只返回在该时间之前查找到的项
SizeLimit	获取或设置指示服务器在搜索中返回的最大对象数量的值
Sort	获取或设置指示对结果进行排序所依据的属性的值
Tombstone	获取或设置一个值，该值指示搜索是否还应返回与搜索筛选器匹配的已删除对象
VirtualListView	获取或设置指示搜索的虚拟列表视图选项的值

下面对 DirectorySearcher 组件中比较重要的属性进行详细介绍。

（1）Asynchronous 属性。该属性表示获取或设置一个值，该值指示是否异步执行搜索。

语法：

```
public bool Asynchronous { get; set; }
```

属性值：如果异步执行搜索，则为 true；否则为 false。

下面演示如何创建异步搜索，代码如下：

```
using System.DirectoryServices;
DirectoryEntry entry = new DirectoryEntry("LDAP://CN=users,DC=fabrikam,DC=com");
DirectorySearcher mySearcher = new DirectorySearcher(entry);
mySearcher.Asynchronous = True;
SearchResultCollection res = mySearcher.FindAll();
```

（2）CacheResults 属性。该属性表示获取或设置一个值，该值指示是否在客户计算机上缓存结果。

语法：

```
public bool CacheResults { get; set; }
```

属性值：如果在客户计算机上缓存结果，则为 true，否则为 false。默认为 true。

例如，设置在本地计算机上缓存结果。代码如下：

```
this.directorySearcher1.CacheResults=true;
```

注意：如果搜索返回一个大型结果集，则可通过将此属性设置为 false 来避免将结果集传输给客户端计算机，使搜索更有效。

（3）ClientTimeout 属性。该属性表示获取或设置客户端等候服务器返回结果的最长时间。如果服务器在此时间内不响应，则终止搜索，不返回任何结果。

语法：

```
public TimeSpan ClientTimeout { get; set; }
```

属性值：一个 TimeSpan 结构，包含客户端等待服务器返回结果的最长等待时间。默认值为-1s，即表示无限期地等候。

例如，设置客户端等候服务器返回结果的最长时间为 3s。代码如下：

```
this.searcher.ClientTimeout = new TimeSpan(0, 0, 3);
```

（4）DerefAlias 属性。该属性表示获取或设置一个值，该值指示应如何解析搜索过程中找到对象的别名。

语法：

```
public DereferenceAlias DerefAlias { get; set; }
```

属性值：DereferenceAlias 值，指定取消引用别名的行为。此属性的默认设置为 Never。

DereferenceAlias 的可取值及说明如下。

- Never：指示该别名不会被取消引用。
- InSerching：表示在搜索基对象的次级（而不是在定位基本身）时取消引用别名。
- FindingBaseObject：表示当查找该搜索的基对象，但不是在搜索其次级时，取消引用别名。
- Always：表示在搜索次级和定位搜索的基对象时取消引用别名。

例如，设置搜索次级和定位搜索的基对象时取消引用别名。代码如下：

```
this.directorySearcher1.DerefAlias = DereferenceAlias.Always;
```

（5）Filter 属性。该属性表示获取或设置指示轻量目录访问协议（LDAP）格式筛选器字符串的值。

语法：

```
public string Filter { get; set; }
```

属性值：以 LDAP 格式表示的搜索筛选器，如“(objectClass=user)”。默认值为“(objectClass=*)”，它检索所有对象。

例如，设置轻量目录访问协议（LDAP）格式筛选器字符串的值为查询所有的组。代码如下：

```
this.directorySearcher1.Filter= “(ObjectClass=group)”;
```

（6）PropertiesToLoad 属性。该属性表示获取一个值，该值指示搜索过程中要检索的属性列表。

语法：

```
public StringCollection PropertiesToLoad { get; }
```

属性值：一个 StringCollection 对象，包含搜索过程中要检索的属性集。默认值为空 StringCollection，即检索所有属性。

例如，循环遍历搜索过程中检索到的属性列表中的每一项内容，如果不存在记录，那么添加记录“天下无双”。代码如下：

```
foreach (StringCollection   item in this.directorySearcher1.PropertiesToLoad )
{
    if (item.Count==0)
    {
        item.Add("天下无双");
    }
}
```

（7）SearchRoot 属性。该属性表示获取或设置一个值，该值指示 Active Directory 域服务层次结构中的节点，从该节点处开始搜索。

语法：

```
public DirectoryEntry SearchRoot { get; set; }
```

属性值：Active Directory 域服务层次结构中的 DirectoryEntry 对象，从该对象处开始搜索。默认值为 null。

例如，设置在 Active Directory 域服务层次结构中的节点从指定节点处开始搜索。代码如下：

```
this.directorySearcher1.SearchRoot = new DirectoryEntry("LDAP://MINGRISOFT/LXY", "LXY", "19880825");
```

（8）SearchScope 属性。该属性表示获取或设置指示服务器遵循的搜索范围的值。

语法：

```
public SearchScope SearchScope { get; set; }
```

属性值：SearchScope 值之一。默认值为 Subtree。

SearchScope 的可取值及说明如下。

- Base：表示将搜索限于基对象。结果最多包含一个对象。当 AttributeScopeQuery 属性指定用于某一搜索时，搜索范围必须设置为 Base。
- OneLevel：表示搜索基对象的直接子对象，但不搜索基对象。
- Subtree：表示搜索整个子树，包括基对象及其所有子对象。如果未指定目录搜索的范围，则执行 Subtree 类型的搜索。

例如，设置服务器搜索基对象的直接子对象，但不搜索基对象。代码如下：

```
this.directorySearcher1.SearchScope = SearchScope.OneLevel;
```

3．方法

（1）FindAll 方法。该方法执行搜索并返回找到的项的集合。

语法：

```
public SearchResultCollection FindAll ()
```

返回值：包含搜索结果的 SearchResultCollection 对象。

例如，循环遍历找到的项集合中的每一个对象，以消息窗体的形式显示其路径信息。代码如下：

```
SearchResultCollection ResultCollection = this.directorySearcher1.FindAll();
foreach (SearchResult    result in ResultCollection )
{
    MessageBox.Show("路径："+result.Path ,"信息提示",MessageBoxButtons.OK,MessageBoxIcon.Asterisk );
}
```

（2）FindOne 方法。该方法表示执行搜索并只返回找到的第一项。

语法：

```
public SearchResult FindOne ()
```

返回值：一个 SearchResult 对象，该对象包含搜索过程中找到的第一项。

例如，实例化一个 SearchResult 实例用来接收查找到的结果。代码如下：

```
SearchResult result=this.searcher.FindOne();
```

示例 09-51　DirectorySearcher 组件的应用

通过对 DirectorySearcher 组件的属性和方法的使用，借助于 TreeView 控件显示域中的用户。代码如下：

代码 09-51　光盘位置：光盘\mingrisoft\sl\09\51

```
private void frmADTree_Load(object sender, EventArgs e)
{
    string LogonName = GetLogonName();                      //获取用户的登录名
    txtAD.Text = "LDAP://" + LogonName.Split('\\')[0];      //获取用户所在的域
    txtUserName.Text = LogonName.Split('\\')[1];            //获取当前用户的登录名称
}
private string GetLogonName()
{
    return System.Security.Principal.WindowsIdentity.GetCurrent().Name;   //获取当前用户的登录名
}
private void button1_Click(object sender, EventArgs e)
{
    ///   1.   objectClass=organizationalUnit    查询条件是所有的组织单元（OU）
    ///   2.   objectClass=group    查询条件是所有的组（GROUP）
    ///   3.   objectClass=user    查询条件是所有的用户（USER）
    string schemaClassNameToSearch = "organizationalUnit";                //保存计算机的查询条件
    string[] ADOU = GetOrganizationalUnit(txtAD.Text, txtUserName.Text, txtPwd.Text, schemaClassNameToSearch);
    // 得到组织结构树的字符串数组
    schemaClassNameToSearch = "user";                                     //设置查询条件
    // 得到用户的信息，包括“组织名,登录名,用户名,邮件地址”用“,”进行分隔
    string[] ADUser = GetUserUnit(txtAD.Text, txtUserName.Text, txtPwd.Text, schemaClassNameToSearch);
    foreach (string s in ADOU)                                  //循环遍历字符串数组 ADUser 中的每一条记录
    {
        TreeNode node = new TreeNode(s);                        //实例化一个 TreeNode 节点
        tree.Nodes.Add(node);                                   //向 TreeView 控件中添加 TreeNode 节点
```

```
            AddUser(ADUser, node);                          //添加新用户
        }
    }
    private void AddUser(string[] ADUser, TreeNode node)
    {
        if (ADUser.Length <= 0)                             //当字符串数组 ADUser 中不存在内容时
            throw new Exception("域中没有任何用户");        //抛出域中没有用户的提示
        if (node == null)                                   //当节点 node 为空时
            throw new ArgumentNullException("node 中不存在内容");      //抛出节点内容为空的异常
        foreach (string s in ADUser)                        //循环遍历数组 ADUSer 中的每一条记录
        {
            string[] sSplit = s.Split(',');                 //实例化一个字符串数组 sSplit
            if (sSplit.Length > 0)                          //当字符串数组 sSplit 中存在内容时
            {
                if (sSplit[0] == node.Text)                 //当数组中的第一条记录的内容等于节点中内容时
                {
                    TreeNode childrenNode = new TreeNode(sSplit[2]);      //实例化一个 TreeNode 对象
                    node.Nodes.Add(childrenNode);                         //向 node 节点中添加子节点
                }
            }
        }
    }
    private string[] GetUserUnit(string domainADsPath, string username, string password, string schemaClassNameToSearch)
    {
        //获取 SearchResult 实例的集合
        SearchResultCollection results = _ADHelper(domainADsPath, username, password, schemaClassNameToSearch);
        string[] sRe = GetGetUserUnitResults(results);      //实例化一个字符串数组
        results.Dispose();                                  //释放对象 results 占用的系统资源
        return sRe;                                         //返回数组 sRe
    }
    private string[] GetGetUserUnitResults(SearchResultCollection results)
    {
        string sRe = string.Empty;                          //当变量 sRe 为空时
        if (results.Count == 0)                             //当 SearchResult 对象集合中的对象数目为时
            throw new Exception("域中没有任何用户");        //抛出域中没有任何用户的异常
        else                                                //当存在 SearchResult 对象时
        {
            foreach (SearchResult result in results)        //循环遍历 results 集合中的每一条记录对象
            {
                string adPath = result.Path;                //保存对象的路径
                if (adPath.IndexOf("OU") < 0)               //当不存在信息时
                    continue;                               //退出此次循环进入下一个循环
                string[] sSplit = adPath.Split(',');        //实例化一个字符串数组 sSplit
                string orz = string.Empty;                  //实例化一个字符串变量 orz
                if (sSplit.Length > 2)                      //当 sSplit 中的元素总数不小于 3 时
                {
                    orz = sSplit[1].Split('=')[1];          //为变量 orz 赋值
                }
                ResultPropertyCollection propColl = result.Properties;//实例化一个用户访问 SearchResult 属性的对象
                if (propColl["samaccountname"].Count > 0)   //当 propColl["samaccountname"]中存在内容时
                    //获取 propColl["samaccountname"]中的首元素
                    orz += "," + propColl["samaccountname"][0].ToString();
                if (propColl["name"].Count > 0)             //当 propColl["name"]中存在内容时
                    orz += "," + propColl["name"][0].ToString(); //获取 propColl["name"]中的首元素
                if (propColl["mail"].Count > 0)             //当 propColl["mail"]中存在内容时
                    orz += "," + propColl["mail"][0].ToString(); //获取 propColl["mail"]中的首元素
                sRe += orz + "=";                           //为变量 sRe 赋值
            }
        }
        if (sRe.Length > 0)                                 //当 sRe 中存在内容时
            sRe = sRe.Substring(0, sRe.Length - 1);         //检索指定长度的子字符串
        return sRe.Split('=');                              //返回的字符串数组包含此实例中的子字符串
    }
```

```
    private SearchResultCollection _ADHelper(string domainADsPath, string username, string password, string
schemaClassNameToSearch)
    {
        //在域服务层次结构中设置起始搜索节点
        this.searcher.SearchRoot = new DirectoryEntry(domainADsPath, username, password);
        //设置指示轻量目录访问协议格式筛选器字符串的值
        this.searcher.Filter = "(objectClass=" + schemaClassNameToSearch + ")";
        this.searcher.SearchScope = SearchScope.Subtree;    //设置指示服务器遵循的搜索范围的值
        //设置指示对结果进行排序所依据的属性的值
        this.searcher.Sort = new SortOption("name", SortDirection.Ascending);
        this.searcher.PageSize = 512;                       //设置指示分页搜索中页大小的值
        //向 searcher 属性列表的末尾添加一组属性
        this.searcher.PropertiesToLoad.AddRange(new string[] { "name", "Path", "displayname", "samaccountname", "mail" });
        return this.searcher.FindAll();                     //执行搜索返回找到的所有项
    }
    private string[] GetOrganizationalUnit(string domainADsPath, string username, string password,string schemaClassNameToSearch)
    {
        //保存查询到的所有的 SearchResult 实例
        SearchResultCollection results = _ADHelper(domainADsPath, username, password, schemaClassNameToSearch);
        string[] sRe = GetGetOrganizationalUnitResults(results);    //返回组织单元中的内容
        results.Dispose();                                  //释放 results 对象占用的资源
        return sRe;                                         //返回 sRe 对象
    }
    private string[] GetGetOrganizationalUnitResults(SearchResultCollection results)
    {
        string sRe = string.Empty;                          //实例化一个字符串变量 sRe 并为其赋空值
        if (results.Count == 0)                             //当 results 集合中的元素个数为 0 时
            throw new Exception("域中没有任何组织结构");    //弹出“域中没有任何组织结构”的信息提示
        else                                                //当 results 集合中的元素个数不为 0 时
        {
            foreach (SearchResult result in results)        //循环遍历集合 results 中的每一个 SearchResult 实例
            {
                if (result.Path.IndexOf("OU") < 0)          //当 result 中不存在用户时
                    continue;                               //跳出此次循环进入下一层循环
                ResultPropertyCollection propColl = result.Properties;   //实例化一个 SearchResult 实例的属性对象
                sRe += propColl["name"][0].ToString() + ',';             //重新为变量 sRe 赋值
            }
        }
        if (sRe.Length > 0)                                 //当 sRe 的长度大于 0 时
            sRe = sRe.Substring(0, sRe.Length - 1);         //检索指定长度的字符串
        return sRe.Split(',');                              //返回分割后的字符串
    }
```

运行本示例，效果如图 9.84 所示。

图 9.84　DirectorySearcher 组件的应用

注意：运行本示例时，本地计算机上必须存在一个域控制器，例如运行本示例时，本地的域服务器为 MINGRISOFT。有关创建域服务器的方法详见 Windows 自带的帮助文档，可

以按<F1>键进行调用。

9.10.4 ErrorProvider 组件

1．功能

ErrorProvider 组件可以在不打扰用户的情况下向用户显示有错误发生。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。图 9.85 所示为 ErrorProvider 组件。

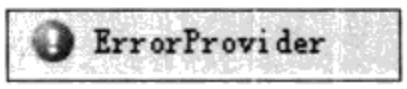


图 9.85 ErrorProvider 组件

2．属性

ErrorProvider 组件常用属性及说明如表 9.54 所示。

表 9.54 **ErrorProvider 组件常用属性及说明**

属　性	说　明
BlinkRate	获取或设置错误图标的闪烁速率
BlinkStyle	获取或设置一个值，通过该值指示错误图标的闪烁时间
ContainerControl	获取或设置一个值，通过该值指示 ErrorProvider 的父控件
DataMember	获取或设置数据源中要监视的列表
DataSource	获取或设置 ErrorProvider 监视的数据源
Icon	获取或设置 Icon，当为控件设置了错误说明字符串时，该图标显示在控件旁边

下面对 ErrorProvider 组件中比较重要的属性进行详细介绍。

（1）BlinkRate 属性。该属性获取或设置错误图标的闪烁速率。

语法：

```
public int BlinkRate { get; set; }
```

属性值：错误图标的闪烁速率（以 ms 为单位）。默认值为 250ms。

例如，设置错误图标的闪烁速率为 0.5s。代码如下：

```
this.errorProvider1.BlinkRate=500;
```

（2）BlinkStyle 属性。该属性表示获取或设置一个值，通过该值指示错误图标的闪烁时间。

语法：

```
public ErrorBlinkStyle BlinkStyle { get; set; }
```

属性值：ErrorBlinkStyle 值之一。默认为 BlinkIfDifferentError。

ErrorBlinkStyle 的可取值及说明如下。

- AlwaysBlink：表示当错误图标第一次显示时，或者当为控件设置了错误说明字符串并且错误图标已经显示时，总是闪烁。
- BlinkIfDifferentError：表示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：表示错误图标从不闪烁。

例如，设置错误图标为总是闪烁。代码如下：

```
this.errorProvider1.BlinkStyle=ErrorBlinkStyle.AlwaysBlink;
```

（3）Icon 属性。该属性表示获取或设置 Icon，当为控件设置了错误说明字符串时，该图标显示在控件旁边。

语法：

```
public Icon Icon { get; set; }
```

属性值：发出信号指出有错误发生的图标。默认图标由红色背景圆及其中的感叹号组成。

例如，当应用程序中存在一个“1.bmp”时，设置该图片为错误图标。代码如下：

```
this.errorProvider1.Icon=Properties.Resources._1;
```

3．方法

ErrorProvider 组件常用方法及说明如表 9.55 所示。

表 9.55 ErrorProvider 组件常用方法及说明

方　法	说　明
BindToDataAndErrors	提供在运行时同时设置 DataSource 和 DataMember 的方法
CanExtend	获取一个值，通过该值指示控件是否可以扩展
Clear	清除与此组件关联的所有设置
GetError	返回指定控件的当前错误说明字符串
GetIconAlignment	获取一个值，通过该值指示错误图标相对于控件的放置位置
SetError	设置指定控件的错误描述字符串
SetIconAlignment	设置错误图标相对于控件的放置位置
SetIconPadding	设置指定控件和错误图标之间应保留的额外空间量
RightToLeft	获取或设置一个值，该值指示组件是否可用于支持从右向左字体的区域设置

下面对 ErrorProvider 组件中比较重要的方法进行详细介绍。

（1）Clear 方法。该方法表示清除与此组件关联的所有设置。

语法：

```
public void Clear()
```

例如，清除与 ErrorProvider 组件关联的所有设置。代码如下：

```
this.errorProvider1.Clear();
```

（2）GetError 方法。该方法表示返回指定控件的当前错误说明字符串。

语法：

```
public string GetError(Control control)
```

参数说明如下。

control：表示要获取其错误描述字符串的项。

返回值：表示指定控件的错误描述字符串。

例如，将指定控件的当前错误说明字符串显示在指定的标签控件中。代码如下：

```
this.label1.Text=this.errorProvider1.GetError(this.button1);
```

（3）GetIconAlignment 方法。该方法表示获取一个值，通过该值指示错误图标相对于控件的放置位置。

语法：

```
public ErrorIconAlignment GetIconAlignment(Control control)
```

参数说明如下。

control：要获取其图标位置的控件。

返回值：ErrorIconAlignment 值之一。默认的图标对齐方式是 MiddleRight。

ErrorIconAlignment 的可取值及说明如下。

- TopLeft：表示图标与控件的顶部和控件的左边对齐。
- TopRight：表示图标与控件的顶部和控件的右边对齐。
- MiddleLeft：表示图标与控件的中间和控件的左边对齐。
- MiddleRight：表示图标与控件的中间和控件的右边对齐。
- BottomLeft：表示图标与控件的底部和控件的左边对齐。
- BottomRight：表示图标与控件的底部和控件的右边对齐。

例如，获取指定控件的错误图标对齐方式。代码如下：

```
this.textBox2.Text=this.errorProvider1.GetIconAlignment(this.textBox1);
```

（4）SetError 方法。该方法表示设置指定控件的错误描述字符串。

语法：

```
public void SetError(Control control,string value)
```

参数说明如下。

control：表示要为其设置错误描述字符串的控件。

value：表示错误描述字符串。

例如，设置文本框中的错误描述字符串为“此文本框必须输入数字”。代码如下：

```
this.errorProvider1.SetError(this.textBox1, "此文本框必须输入数字");
```

（5）SetIconAlignment 方法。该方法表示设置错误图标相对于控件的放置位置。

语法：

```
public void SetIconAlignment(Control control,ErrorIconAlignment value)
```

参数说明如下。

control：要为其设置图标位置的控件。

value：ErrorIconAlignment 值之一（参考 GetIconAlignment 方法的返回值）。

例如，设置指定控件的错误图标放置位置为左上对齐。代码如下：

```
this.errorProvider1.SetIconAlignment(this.textBox1,ErrorIconAlignment.TopLeft);
```

示例 09-52　ErrorProvider 组件的应用

本示例演示对 ErrorProvider 组件的属性和方法的应用，当用户输入非数字时，弹出错误提示信息。代码如下：

代码 09-52　光盘位置：光盘\mingrisoft\sl\09\52

```
private void Form1_Load(object sender, EventArgs e)
{
    this.errorProvider1.BlinkRate = 300;                          //设置错误图标的闪烁频率
    this.errorProvider1.BlinkStyle = ErrorBlinkStyle.AlwaysBlink;  //设置图标的闪烁方式
}
private void Identify_Click(object sender, EventArgs e)
{
    if (this.errorProvider1.GetError(this.textBox1) != "")        //当当前错误描述字符串不为空时
    {
        MessageBox.Show(this.errorProvider1.GetError(this.textBox1), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Error );                                           //弹出错误提示信息
    }
    else                                                          //当输入格式正确时
    {
        MessageBox.Show("祝贺你，输入格式正确…", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.
Asterisk);                                                        //弹出错误提示信息
    }
}
private void textBox1_Validating(object sender, CancelEventArgs e)
{
    try
    {
        int x = Int32.Parse(this.textBox1.Text);                  //将文本框的内容转换为 int 型的值
    }
    catch
    {
        this.errorProvider1.SetError(this.textBox1, "请输入一个整型数字");      //设置错误提示信息
    }
}
```

运行本示例，效果如图 9.86 所示。

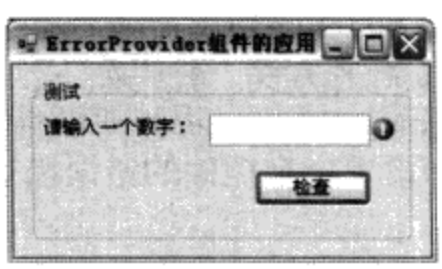


图 9.86　ErrorProvider 组件的应用

9.10.5 EventLog 组件

1．功能

使用 EventLog 组件可以轻松地连接到本地和远程计算机上的事件日志，并向这些日志中写入项；也可以从现有日志中读取条目并创建自己的自定义事件日志；还可以使用“服务器资源管理器”查看有访问权限的任何服务器的事件日志列表。用户可以展开“服务器资源管理器”中的“事件日志”节点以查看当前存在的日志列表，以及已写入其中的事件。此外，还可以从“服务器资源管理器”中将某些特定事件日志拖放到设计器上，以创建 EventLog 组件的实例，该实例自动将其属性设置为指向选定的日志。图 9.87 所示为 EventLog 组件。

图 9.87 EventLog 组件

2．属性

EventLog 组件的常用属性及说明如表 9.56 所示。

表 9.56 EventLog 组件常用属性及说明

属　性	说　明
EnableRaisingEvents	获取或设置一个值，用以指示 EventLog 是否接收 EntryWritten 事件通知
Entries	获取事件日志的内容
Log	获取或设置读取或写入的日志名称
LogDisplayName	获取事件日志的友好名称
MachineName	获取或设置在其上读取或写入事件的计算机名称
MaximumKilobytes	获取或设置最大事件日志大小（以 KB 为单位）
MinimumRetentionDays	获取要在事件日志中保留项的天数
OverflowAction	获取在事件日志达到其最大日志文件大小时，存储新项的已配置行为
Source	获取或设置在写入事件日志时要注册和使用的源名称
SynchronizingObject	获取或设置用于封送事件处理程序调用的对象，该调用在有 EventLog 项写入事件时发出

下面对 EventLog 组件中比较重要的属性进行详细介绍。

（1）Entries 属性。该属性获取事件日志的内容。

语法：

```
public EventLogEntryCollection Entries { get; }
```

属性值：EventLogEntryCollection 保留事件日志中的项。每个项均与 EventLogEntry 类的一个实例关联。

例如，获取事件日志的数目并以消息框的形式显示。代码如下：

```
MessageBox.Show(this.eventLog1.Entries.Count.ToString(),"信息提示");
```

（2）Log 属性。该属性获取或设置读取或写入的日志名称。

语法：

```
public string Log { get; set; }
```

属性值：日志的名称可以是 Application、System 和 Security，或是一个自定义的日志名称。默认值为空字符串（""）。

例如，向列表框控件中添加读取的日志名称。代码如下：

```
this.listBox1.Items.Add(this.eventLog1.Log);
```

（3）Source 属性。该属性是指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：

```
public string Source { get; set; }
```

属性值：在事件日志中，注册为项源的名称。默认值为空字符串（""）。

例如，设置在写入事件日志时要注册和使用的源名称为本地计算机。代码如下：

```
this.eventLog1.Source='.';
```

3．方法

EventLog 组件的常用方法及说明如表 9.57 所示。

表 9.57　　EventLog 组件常用方法及说明

方　法	说　明
BeginInit	开始初始化用于窗体或由其他组件使用的 EventLog。此初始化在运行时发生
Clear	从事件日志中清除所有的项
Close	关闭事件日志并释放读写句柄
CreateEventSource	建立一个能够将事件信息写入到系统的特定日志中的应用程序
Delete	移除日志资源
DeleteEventSource	从事件日志中移除应用程序的事件源注册
Dispose	释放由 EventLog 占用的资源
Exists	确定指定的日志是否存在
GetEventLogs	创建事件日志的数组
LogNameFromSourceName	获取指定的源注册到的日志名称
ModifyOverflowPolicy	更改已配置的行为，以便在事件日志达到其最大文件大小时写入新项
RegisterDisplayName	指定该事件日志的本地化名称，它显示在服务器的事件查看器中
SourceExists	在计算机的注册表中搜索给定的事件源
WriteEntry	将项写入事件日志
WriteEvent	向事件日志写入本地化事件项

下面对 EventLog 组件比较重要的方法进行详细介绍。

（1）Clear 方法。该方法表示从事件日志中清除所有的项。

语法：

```
public void Clear ()
```

例如，清空事件日志中的所有项。代码如下：

```
this.eventLog1.Clear();
```

（2）Close 方法。该方法表示关闭事件日志并释放读写句柄。

语法：

```
public void Close()
```

例如，关闭当前的事件日志并释放读写句柄。代码如下：

```
this.eventLog1.Close();
```

（3）Delete 方法。该方法表示移除日志资源。

语法：

```
public static void Delete(string logName)
```

参数说明如下。

logName：要删除的日志的名称。可能的值包括应用程序、安全性、系统和计算机上的任何自定义事件日志。

例如，删除日志资源中以“NewLog1”命名的日志。代码如下：

```
this.eventLog1.Delete("NewLog1");
```

（4）Exists 方法。该方法表示确定指定的日志是否存在。

语法：

```
public static bool Exists(string logName)
```

参数说明如下。

logName：要搜索的日志名称。可能的值包括应用程序、安全、系统、其他应用程序特定的日志（如与 Active Directory 关联的日志）或计算机上的任何自定义日志。

例如，判断日志资源中是否存在以“Log1”命名的日志。代码如下：

```
this.eventLog1.Exists("Log1");
```

（5）SourceExists 方法。该方法表示确定事件源是否已在本地计算机上注册。

语法：

```
public static bool SourceExists(string source)
```

参数说明如下。

source：事件源的名称。

返回值：如果事件源已在本地计算机上注册，则为 true；否则为 false。

例如，判断指定事件源是否在本地计算机上注册。代码如下：

```
this.eventLog1.SourceExists("Click");
```

（6）WriteEntry 方法。该方法表示将项写入事件日志。

语法：

```
public void WriteEntry(string message)
```

参数说明如下。

message：要写入事件日志的字符串。

例如，向事件日志中写入“这是用户自定义日志”。代码如下：

```
this.eventLog1.WriteEntry("这是用户自定义日志");
```

（7）WriteEvent 方法。该方法表示向事件日志写入本地化事件项。

语法：

```
public void WriteEvent(EventInstance instance,Object[] values)
```

参数说明如下。

instance：表示本地化事件日志项的 EventInstance 实例。

values：表示要合并到事件日志项的消息文本中的字符串数组。

例如，向事件日志中写入指定的本地化事件项。代码如下：

```
int SuccessId=124573;
EventInstance SuccessInstance = new EventInstance(SuccessId, 0, EventLogEntryType.SuccessAudit);
string[] InsertString = { "向事件日志中写入本地化事件项"};
this.eventLog1.WriteEvent(SuccessInstance, InsertString);
```

4．EntryWritten 事件

该事件在向本地计算机上的事件日志写入项时发生。

语法：

```
public event EntryWrittenEventHandler EntryWritten
```

示例 09-53　EventLog 组件的应用

运行本示例，在日志内容文本框中写入日志内容，然后单击“写入日志”按钮弹出写入成功的信息提示；单击“读取日志”按钮，显示日志文件中的所有内容；单击“清空日志”按钮时，日志文件中的内容被清空。代码如下：

代码 09-53　光盘位置：光盘\mingrisoft\sl\09\53

```
private void WriteIn_Click(object sender, EventArgs e)
{
    if (System.Diagnostics.EventLog.Exists("NewLog1"))                    //当本机上存在日志“NewLog1”时
    {
        if (textBox1.Text != "")                                          //当 textBox1 控件中内容不为空时
        {
            eventLog1.WriteEntry(textBox1.Text.ToString()); //向日志中写入文本框中的内容
            //弹出写入成功的信息提示
```

```
                MessageBox.Show("日志写成功", "提示信息", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                textBox1.Text = "";                                    //清空 textBox1 控件的内容
                this.textBox1.Focus();                                 //设置 textBox1 控件为当前焦点
            }
            else                                                       //当 textBox1 控件内容为空时
            {
                MessageBox.Show("日志内容不能为空", "提示信息", MessageBoxButtons.OK,
MessageBoxIcon.Warning);                                               //弹出日志内容不能为空的信息提示
            }
        }
        else                                                           //当本机中不存在日志“NewLog1”时
        {
            //弹出日志不存在的信息提示
            MessageBox.Show("日志不存在", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    private void ReadOut_Click(object sender, EventArgs e)
    {
        listBox1.Items.Clear();                                        //清空 listBox1 控件中的内容
        if (eventLog1.Entries.Count > 0)                               //当日志中存在内容时
        {
            foreach (System.Diagnostics.EventLogEntry entry in eventLog1.Entries)  //循环遍历日志中的内容
            {
                listBox1.Items.Add(entry.Message);                     //向 listBox1 控件中添加内容
            }
        }
        else                                                           //当日志中不存在内容时
        {
            //弹出信息提示
            MessageBox.Show("日志中没有记录", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Stop);
        }
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        this.textBox1.Focus();                                         //设置 textBox1 控件为当前的焦点
        //检查事件源是否存在，如果不存在将注册事件源
        if (System.Diagnostics.EventLog.SourceExists("EventSource"))
        {
            System.Diagnostics.EventLog.DeleteEventSource("EventSource");   //注册事件源
        }
        //为 NewLog1 日志名称注册事件源
        System.Diagnostics.EventLog.CreateEventSource("EventSource", "NewLog1");
        eventLog1.Log = "NewLog1";                                     //NewLog1 日志
        eventLog1.Source = "EventSource";                              //事件源名称
        this.eventLog1.MachineName = ".";                              //表示本机
    }
    private void ClearAll_Click(object sender, EventArgs e)
    {
        if (eventLog1.Entries.Count > 0)                               //当日志中存在内容时
        {
            this.eventLog1.Clear();                                    //清空日志内容
        }
        else                                                           //当日志中不存在内容时
        {
            //弹出日志中不存在内容的信息提示
            MessageBox.Show("日志中没有记录", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Stop);
        }
    }
```

运行本示例，效果如图 9.88 所示。

图 9.88　EventLog 组件的应用

9.10.6　FileSystemWatcher 组件

1．功能

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被创建、修改或删除时，快速方便地启动业务处理。图 9.89 所示为 FileSystemWatcher 组件。

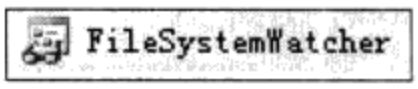


图 9.89　FileSystemWatcher 组件

2．属性

FileSystemWatcher 组件常用属性及说明如表 9.58 所示。

表 9.58　**FileSystemWatcher 组件常用属性及说明**

属　性	说　明
EnableRaisingEvent	获取或设置一个值，该值指示是否启用此组件
Filter	获取或设置筛选字符串，用于确定在目录中监视哪些文件
IncludeSubdirectories	获取或设置一个值，该值指示是否监视指定路径中的子目录
NotifyFilter	获取或设置要监视的更改类型
Path	获取或设置要监视的目录的路径
SynchronizingObject	获取或设置一个对象，该对象用于封送，因目录更改而发出的事件处理程序的调用

下面对 FileSystemWatcher 组件比较重要的属性进行详细介绍。

（1）Filter 属性。该属性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：

```
public string Filter {get;set;}
```

属性值：筛选器字符串，默认值为“*.*”（监视所有文件）。

例如，设置在目录中监视文本文件。代码如下：

```
this.fileSystemWatcher1.Filter = "*.txt";
```

说明：若要监视所有文件中的更改，请将 Filter 属性设置为空字符串（""）。若要监视特定的文件，请将 Filter 属性设置为该文件名。例如，若要监视文件 MyDoc.txt 中的更改，请将 Filter 属性设置为“MyDoc.txt”。若要监视特定类型文件中的更改，例如，监视任何文本文件中的更改，请将 Filter 属性设置为“*.txt”。

（2）NotifyFilter 属性。该属性表示获取或设置要监视的更改类型。

语法：

```
public NotifyFilters NotifyFilter { get; set; }
```

属性值：NotifyFilters 值之一。默认值为 LastWrite、FileName 和 DirectoryName 的按位“或”组合。

例如，设置要监视的更改类型为文件名、属性或者上一次打开的日期。代码如下：

```
this.fileSystemWatcher1.NotifyFilter = System.IO.NotifyFilters.FileName | System.IO.NotifyFilters.Attributes | System.IO.NotifyFilters.LastAccess;
```

说明：NotifyFilters 的成员有以下几种：FileName、DirectoryName、Attributes、Size、LastWrite、LastAccess、CreationTime 以及 Security。其中 FileName 表示文件名，DirectoryName 表示目录名，Attributes 表示文件或文件夹的属性，Size 表示文件或文件夹的大小，LastWrite 表示上一次向文件或文件夹中写入内容的日期，LastAccess 表示文件或文件夹上一次打开的日期，CreationTime 表示文件或文件夹的创建日期，Security 表示文件或文件夹的安全设置。

（3）Path 属性。该属性获取或设置要监视的目录的路径。

语法：

```
public string Path {get;set;}
```

属性值：要监视的路径。默认值为空字符串（""）。

例如，设置要监视的目录路径为应用程序下的 Test 文件。代码如下：

```
this.fileSystemWatcher1.Path = Application.StartupPath + "\\Test";
```

说明：这是某个目录的完全限定路径。如果 IncludeSubdirectories 属性为 true，则此目录为系统监视更改的根目录；否则此目录为被监视的惟一目录。若要监视特定文件，请将 Path 属性设置为完全限定的正确目录，并将 Filter 属性设置为文件名。Path 属性支持统一命名约定（UNC）路径。

3．方法

FileSystemWatcher 组件常用方法及说明如表 9.59 所示。

表 9.59　　FileSystemWatcher 组件常用方法及说明

方　法	说　明
BeginInit	开始初始化在窗体上使用或由另一个组件使用的 FileSystemWatcher 初始化。初始化发生在运行时
EndInit	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
WaitForChanged	返回结构的同步方法，该结构包含有关已发生的更改的特定信息

下面详细介绍 EndInit 方法，该方法用于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：

```
public void EndInit ()
```

例如，结束当前 FileSystemWatcher 组件的初始化。代码如下：

```
this.fileSystemWatcher.EndInit();
```

说明：Visual Studio 2008 设计环境使用此方法终止对用于窗体的组件或由其他组件使用的组件的初始化。BeginInit 方法开始此初始化。使用 BeginInit 和 EndInit 方法防止在尚未完全初始化控件时使用。

4．事件

FileSystemWatcher 组件常用事件及说明如表 9.60 所示。

表 9.60　　FileSystemWatcher 组件常用事件及说明

事　件	说　明
Changed	当更改指定 Path 中的文件和目录时发生
Created	当在指定 Path 中创建文件和目录时发生
Deleted	删除指定 Path 中的文件或目录时发生
Renamed	重命名指定 Path 中的文件或目录时发生

下面对 FileSystemWatcher 组件比较重要的事件进行详细介绍。

（1）Changed 事件。该事件当更改指定 Path 中的文件和目录时发生。

语法：

```
public event FileSystemEventHandler Changed
```

说明：当对所监视的目录中的文件或目录的大小、系统属性、上次写入时间、上次访问时间或安全权限进行更改时，将引发 Changed 事件。

（2）Created 事件。该事件当在指定 Path 中创建文件和目录时发生。

语法：

```
public event FileSystemEventHandler Created
```

（3）Deleted 事件。该事件当删除指定 Path 中的文件或目录时发生。

语法：

```
public event FileSystemEventHandler Deleted
```

（4）Renamed 事件。该事件重命名指定 Path 中的文件或目录时发生。

语法：

```
public event RenamedEventHandler Renamed
```

示例 09-54　FileSystemWatcher 组件的应用

运行本示例，选择项目文件中监视文件下的 TXT 文本文件，当该文本文件存在时，“删除”按钮处于可用状态。单击“删除”按钮删除该文本文件，此时触发 FileSystemWatcher 的 Deleted 事件从而显示删除成功的提示。代码如下：

代码 09-54　光盘位置：光盘\mingrisoft\sl\09\54

```
private void Form1_Load(object sender, EventArgs e)
{
    this.fileSystemWatcher1.EnableRaisingEvents = true;                    //设置一个值表示启用该控件
    this.fileSystemWatcher1.Filter = "*.txt";                              //设置监视的文件类型
    this.fileSystemWatcher1.IncludeSubdirectories = false;                 //设置该组件不监视指定路径下的目录
    this.fileSystemWatcher1.Path = Application.StartupPath + "\\Test";     //设置监视的文件目录
    this.fileSystemWatcher1.NotifyFilter = System.IO.NotifyFilters.FileName | System.IO.NotifyFilters.Attributes |
System.IO.NotifyFilters.LastAccess;                                        //设置组件的监视更改类型
    this.fileSystemWatcher1.EndInit();                                     //结束组件在窗体上的初始化
    this.Delete.Enabled = false;                                           //设置删除按钮为不可用状态
}
private void Open_Click(object sender, EventArgs e)
{
    OpenFileDialog TxtDialog = new OpenFileDialog();                       //实例化一个打开文本框
    TxtDialog.Filter = "文本文件|*.txt";                                   //设置打开的文件类型
    if (TxtDialog.ShowDialog() == DialogResult.OK)                         //当单击“打开”按钮时
    {
        this.FilePath.Text = TxtDialog.FileName;                           //将将要打开的文件路径显示在文本框中
    }
}
private void Delete_Click(object sender, EventArgs e)
{
    System.IO.File.Delete(this.FilePath.Text);                             //删除指定目录下的文件
    this.FilePath.Clear();                                                 //清空文本框中的内容
    this.FilePath.Focus();                                                 //设置文本框中为当前的焦点
}
private void FilePath_TextChanged(object sender, EventArgs e)
{
    if (System.IO.File.Exists(this.FilePath.Text))                         //当指定文件存在时
    {
        this.Delete.Enabled = true;                                        //设置删除按钮处于可用状态
    }
}
private void fileSystemWatcher1_Deleted(object sender, System.IO.FileSystemEventArgs e)
{
    MessageBox.Show("文件删除成功", "提示信息", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.DefaultDesktopOnly);    //弹出删除成功的信息提示
}
```

运行本示例，效果如图 9.90 所示。

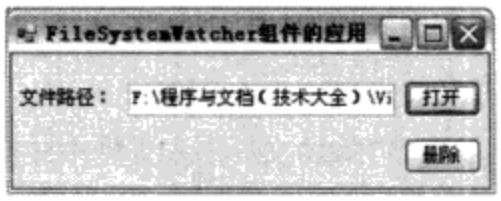


图 9.90　FileSystemWatcher 组件的应用

9.10.7　HelpProvider 组件

1．功能

HelpProvider 组件的功能如下。

- 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。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。
- 为特定对话框或对话框中的特定控件提供区分上下文的帮助。
- 打开帮助文件到特定部分，如目录、索引或搜索功能的主页。

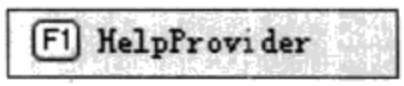


图 9.91　HelpProvider 组件

图 9.91 所示为 HelpProvider 组件。

2．属性

（1）HelpNamespace 属性。该属性用于获取或设置一个值，该值指定与此 HelpProvider 对象关联的帮助文件名。

语法：

```
public virtual string HelpNamespace { get; set; }
```

属性值：帮助文件的名称，其形式可以是“C:\path\sample.chm”或“/folder/file.htm”。

例如，获取应用程序下存在的“Help.chm”帮助文件。代码如下：

```
this.helpProvider1.HelpNamespace = Application.StartupPath + "Help.chm";
```

（2）Tag 属性。该属性获取或设置包含有关 HelpProvider 的补充数据的对象。

语法：

```
public Object Tag { get; set; }
```

属性值：一个 Object，它包含有关 HelpProvider 的数据。

例如，获取包含有关 HelpProvider 的补充对象信息并显示在指定文本框控件中。代码如下：

```
this.textBox1.Text=this.helpProvider1.Tag.ToString();
```

3．方法

HelpProvider 组件常用方法及说明如表 9.61 所示。

表 9.61　HelpProvider 组件常用方法及说明

方　法	说　明
CanExtend	指定此对象是否可以将其扩展程序属性提供给指定的对象
GetHelpKeyword	返回指定控件的帮助关键字
GetHelpNavigator	返回指定控件的当前 HelpNavigator 设置
GetHelpString	返回指定控件的弹出帮助窗口的内容
GetShowHelp	返回一个值，此值指示是否显示指定控件的帮助
ResetShowHelp	移除与指定的控件关联的帮助
SetHelpKeyword	指定在用户调用指定控件的帮助信息时用于检索帮助的关键字
SetHelpNavigator	指定在从帮助文件中检索指定控件的帮助信息时使用的帮助命令
SetHelpString	指定与指定的控件关联的帮助字符串
SetShowHelp	指定是否显示指定控件的帮助信息

下面对 HelpProvider 组件中比较重要的方法进行详细介绍。

（1）GetHelpKeyword 方法。该方法返回指定控件的帮助关键字。

语法：

```
public virtual string GetHelpKeyword (Control ctl)
```

参数说明如下。

ctl：检索帮助主题的 Control。

返回值：与此控件关联的帮助关键字，如果 HelpProvider 组件当前配置为显示整个帮助文件或配置为提供帮助字符串，则为空引用。

例如，获取与指定的文本框控件有关的帮助关键字。代码如下：

```
this.helpProvider1.GetHelpKeyword(this.textBox1);
```

（2）GetHelpNavigator 方法。该方法表示返回指定控件的当前 HelpNavigator 设置。

语法：

```
public virtual HelpNavigator GetHelpNavigator(Control ctl)
```

参数说明如下。

ct1：从中检索帮助导航器的控件。

返回值：指定控件的 HelpNavigator 设置。默认为 AssociateIndex。

例如，获取指定按钮控件的 HelpNavigator 设置。代码如下：

```
this.helpProvider1.GetHelpNavigator(this.button1);
```

说明：HelpNavigator 的可取值有 Topic、TableOfContents、Index、Find、AssociateIndex、KeywordIndex 以及 TopicId。其中 Topic 表示如果指定的主题存在，则帮助文件将打开到指定的主题，TableOfContents 表示帮助文件将打开到目录，Index 表示帮助文件将打开到索引，Find 表示帮助文件将打开到搜索页，AssociateIndex 表示帮助文件将打开到特定主题首字母的索引项，KeywordIndex 表示如果存在具有指定索引项的主题，则帮助文件打开到该主题；否则将显示与指定关键字最接近的索引项，TopicId 表示帮助文件将打开到由数字主题标识符指示的主题。

（3）GetHelpString 方法。该方法表示返回指定控件的弹出帮助窗口中的内容。

语法：

```
public virtual string GetHelpString(Control ctl)
```

参数说明如下。

ctl：从中检索帮助字符串的控件。

返回值：与此控件关联的帮助字符串。默认为 null。

例如，获取和指定按钮关联的帮助字符串显示在标签控件中。代码如下：

```
this.label1.Text=this.helpProvider1.GetHelpString(this.button1);
```

（4）ResetShowHelp 方法。该方法表示移除与指定控件关联的帮助。

语法：

```
public virtual void ResetShowHelp(Control ctl)
```

参数说明如下。

ctl：要从中移除帮助的控件。

例如，当需要更换和某一控件关联的帮助信息时，首先需要将它原来的去掉。代码如下：

```
this.helpProvider1.ResetShowHelp(this.button1);
```

（5）SetHelpKeyWord 方法。该方法表示指定在用户调用指定控件的帮助信息时用于检索帮助的关键字。

语法：

```
public virtual void SetHelpKeyword(Control ctl,string keyword)
```

参数说明如下。

ctl：指定要为其设置帮助主题的控件。

keyword：与控件关联的帮助关键字。

例如，设置 Button 控件的帮助关键字为“洁净”。代码如下：

```
this.helpProvider1.SetHelpKeyWord(this.button1, "洁净");
```

(6) SetHelpNavigator 方法。该方法指定在从帮助文件中检索指定控件的帮助信息时使用的帮助命令。

语法：

```
public virtual void SetHelpNavigator(Control ctl,HelpNavigator navigator)
```

参数说明如下。

ctl：要为其设置帮助关键字的控件。

navigator：HelpNavigator 值之一。其可选值可以参考 GetHelpNavigator 方法中的说明。

例如，设置从帮助文件中检索与文本框控件关联的帮助信息时使用的帮助命令。代码如下：

```
this.helpProvider1.SetHelpNavigator(this.textBox1,HelpNavigator.Topic);
```

(7) SetHelpString 方法。该方法指定与指定的控件关联的帮助字符串。

语法：

```
public virtual void SetHelpString(Control ctl,string helpString)
```

参数说明如下。

ctl：与帮助字符串关联的控件。

helpString：与控件关联的帮助字符串。

例如，设置与指定按钮控件关联的帮助字符串。代码如下：

```
this.helpProvider1.SetHelpString(this.button1, "单击此按钮可以进行登录操作");
```

(8) SetShowHelp 方法。该方法指定是否显示指定控件的帮助信息。

语法：

```
public virtual void SetShowHelp (Controlctl,bool value)
```

参数说明如下。

ctl：Control 其帮助信息已打开或关闭。

Value：如果显示控件的帮助信息，则为 true；否则为 false。

例如，设置显示当前窗体中的所有 Button 按钮的帮助信息。代码如下：

```
foreach (Control   item in this.Controls)
{
    if (item.GetType().ToString()=="System.Windows.Forms.Button")
    {
        this.helpProvider1.SetShowHelp(item, true);
    }
}
```

示例 09-55　HelpProvider 组件的应用

运行本示例，当鼠标处于窗体中的各控件上时，程序会自动弹出信息提示。代码如下：

代码 09-55　光盘位置：光盘\mingrisoft\sl\09\55

```
private void Form1_Load(object sender, EventArgs e)
{
    this.helpProvider1.SetHelpString(this.UserID, "账号");                    //设置账号文本框的提示字符串
    this.helpProvider1.SetHelpString(this.UserPetName, "用户昵称");           //设置用户昵称文本框的提示字符串
    this.helpProvider1.SetHelpString(this.UserPwd, "用户密码");               //设置用户密码文本框的提示字符串
    this.helpProvider1.SetHelpString(this.ReUserPwd, "确认密码");             //设置确认密码文本框的提示字符串
    this.helpProvider1.SetHelpString(this.Register , "注册");                 //设置注册按钮的提示字符串
    this.helpProvider1.SetHelpString(this.Cancel , "取消");                   //设置取消按钮的提示字符串
}
private void UserID_MouseHover(object sender, EventArgs e)
{
    //弹出用户账号的提示信息框
    MessageBox.Show(this.helpProvider1.GetHelpString(this.UserID), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
}
private void UserPetName_MouseHover(object sender, EventArgs e)
{
```

```
        //弹出用户昵称的提示信息框
        MessageBox.Show(this.helpProvider1.GetHelpString(this.UserPetName), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
    }
    private void UserPwd_MouseHover(object sender, EventArgs e)
    {
        //弹出用户密码的提示信息框
        MessageBox.Show(this.helpProvider1.GetHelpString(this.UserPwd), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
    }
    private void ReUserPwd_MouseHover(object sender, EventArgs e)
    {
        //弹出确认密码的提示信息框
        MessageBox.Show(this.helpProvider1.GetHelpString(this.ReUserPwd), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
    }
    private void Register_MouseHover(object sender, EventArgs e)
    {
        //弹出注册按钮的提示信息框
        MessageBox.Show(this.helpProvider1.GetHelpString(this.Register), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
    }
    private void Cancel_MouseHover(object sender, EventArgs e)
    {
        //弹出取消按钮的提示信息框
        MessageBox.Show(this.helpProvider1.GetHelpString(this.Cancel), "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.ServiceNotification);
    }
```

运行本示例，效果如图 9.92 所示。

图 9.92 HelpProvider 组件的应用

9.10.8 ImageList 组件

1．功能

ImageList 组件用于存储图像，这些图像可由控件显示。图像列表能够为一致的单个图像目录编写代码。图 9.93 所示为 ImageList 组件。

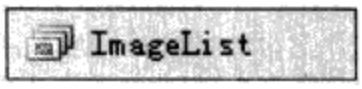


图 9.93 ImageList 组件

2．属性

ImageList 组件常用属性及说明如表 9.62 所示。

表 9.62 ImageList 组件常用属性及说明

属 性	说 明
ColorDepth	此属性确定呈现图像时所使用的颜色数量
Images	此属性表示图像列表中包含图像的集合
ImageSize	此属性表示获取或设置图像列表中图像的大小
ImageStream	获取与此图像列表关联的 ImageListStreamer 对象
TransparentColor	获取或设置被视为透明的颜色

下面对 ImageList 组件比较重要的属性进行详细介绍。

（1）Images 属性。该属性表示图像列表中包含图像的集合。

语法：

```
public ImageCollection Images { get; }
```

属性值：表示图像集合。

例如，获取图像列表中图像的总数目并显示在标签控件中。代码如下：

```
this.label1.Text=this.imageList1.Images.Count;
```

（2）ImageSize 属性。该属性表示获取或设置图像列表中图像的大小。

语法：

```
public Size ImageSize { get; set; }
```

属性值：定义列表中的图像高度和宽度（以像素为单位）。默认大小是 16 × 16。最大大小是 256 × 256。

例如，设置图像的大小为 18 × 18。代码如下：

```
this.imageList1.ImageSize = new Size(18,18);
```

3．Draw 方法

该方法主要用于绘制指定的图像。

语法：

```
public void Draw(Graphics g,int x,int y,int width,int height,int index)
```

参数说明如表 9.63 所示。

表 9.63　　Draw 方法的参数及说明

参　数	说　明
g	要进行绘制的 Graphics
x	图像绘制处的水平位置
y	图像绘制处的垂直位置
width	目标图像的宽度（以像素为单位）
heitht	目标图像的高度（以像素为单位）
index	要绘制的 ImageList 中的图像的索引

例如，利用此方法绘制指定的图像。代码如下：

```
Graphics g = this.CreateGraphics();
this.imageList1.Draw(g, 0, 0, 90, 90, 0);
```

示例 09-56　ImageList 组件的应用

运行本示例，单击“替换”按钮，PictureBox 控件中的图片自动替换。代码如下：

代码 09-56　光盘位置：光盘\mingrisoft\sl\09\56

```
internal static bool flag = true ;                          //设置一个全局变量标识
private void Form1_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.imageList1.Images[1];     //设置 pictureBox2 控件中显示的图片
}
private void ChangeEachOther_Click(object sender, EventArgs e)
{
    if (flag)                                               //当 flag 的值为 true 时
    {
        this.pictureBox1.Image = this.imageList1.Images[1]; //设置 pictureBox1 控件中显示的图片
        this.pictureBox2.Image = this.imageList1.Images[0]; //设置 pictureBox2 控件中显示的图片
        flag = false;                                       //设置 flag 的值为 false
    }
    else                                                    //当 flag 的值为 false 时
    {
        this.pictureBox1.Image = this.imageList1.Images[0]; //设置 pictureBox1 控件中显示的图片
```

```
            this.pictureBox2.Image = this.imageList1.Images[1];   //设置 pictureBox2 控件中显示的图片
            flag = true;                                          //设置 flag 的值为 true
        }
    }
```

运行本示例，效果如图 9.94 所示。

图 9.94　ImageList 组件的应用

9.10.9　MessageQueue 组件

1．功能

当需要应用程序的内置消息处理功能时，可以创建 MessageQueue 组件。使用 MessageQueue 组件，可以连接到现有队列，发送和接收消息，以及使用非常少的代码为应用程序添加通信服务。图 9.95 所示为 MessageQueue 组件。

图 9.95　MessageQueue 组件

2．属性

MessageQueue 组件常用属性及说明如表 9.64 所示。

表 9.64　　MessageQueue 组件常用属性及说明

属　性	说　明
AccessMode	获取一个值，该值指示队列的访问模式
Authenticate	获取或设置一个值，该值指示队列是否仅接收，已经过身份验证的消息
BasePriority	获取或设置基优先级，“消息队列”使用该基优先级，在网络上传送公共队列的消息
CanRead	获取一个值，该值指示 MessageQueue 是否可读
CanWrite	获取一个值，该值指示 MessageQueue 是否可写
Category	获取或设置队列类别
CreateTime	获取在“消息队列”中创建队列的时间和日期
EnableConnectionCache	获取或设置一个值，该值指示是否由应用程序维护连接缓存
EncryptionRequired	获取或设置一个值，该值指示队列是否只接受非私有（非加密）消息
FormatName	获取“消息队列”在创建队列时生成的惟一队列名
Formatter	获取或设置格式化程序，该格式化程序用于将对象序列化为从队列读取或写入队列的消息体，或者用于将从队列读取或写入队列的消息体反序列化为对象
Id	获取队列的唯一“消息队列”标识符
Label	获取或设置队列说明
LastModifyTime	获取队列属性的最近修改时间
MachineName	获取或设置“消息队列”队列所在的计算机的名称
MaximumJournalSize	获取或设置日记队列的最大大小
MaximumQueueSize	获取或设置队列的最大大小
MessageReadPropertyFilter	获取或设置接收或查看消息的属性筛选器
MulticastAddress	获取或设置与队列关联的多路广播地址
Path	获取或设置队列的路径
QueueName	获取或设置标识队列的友好名称
ReadHandle	获取用于从消息队列读取消息的本机句柄
UseJournalQueue	获取或设置一个值，该值指示接收的消息是否复制到日记队列

下面对 MessageQueue 组件比较重要的属性进行详细介绍。

（1）AccessMode 属性。该属性表示获取一个值，该值指示队列的访问模式。

语法：

```
public QueueAccessMode AccessMode{get;}
```

属性值：QueueAccessMode 值之一。QueueAccessMode 的值及说明如表 9.65 所示。

表 9.65　　QueueAccessMode 的值及说明

属　性　值	说　　明
Peek	队列仅可以查看消息
PeekAndAdmin	队列可以查看消息。消息也可以被清除
Receive	队列可以查看或接收消息
ReceiveAndAdmin	队列可以接收消息。消息也可以被清除
Send	队列仅可以发送消息
SendAndReceive	队列可以查看、接收或发送消息

例如，设置当前队列为可以查看、接收或发送消息的队列。代码如下：

```
this.messageQueue1.AccessMode=QueueAccessMode.SendAndReceive;
```

（2）Authenticate 属性。该属性表示获取或设置一个值，该值指示队列是否仅接收已经通过身份验证的消息。

语法：

```
public bool Authenticate { get; set; }
```

属性值：如果队列仅接收已经通过身份验证的消息，则为 true；否则为 false。默认值为 false。

例如，下面获取并设置消息队列的 Authenticate 属性值，代码如下：

```
queue.Authenticate = True;
Console.WriteLine("MessageQueue.Authenticate: {0}", queue.Authenticate);
```

（3）BasePriority 属性。该属性表示获取或设置基优先级，“消息队列”使用该基优先级，在网络上传送公共队列的消息。

语法：

```
public short BasePriority { get; set; }
```

属性值：所有发送到（公共）队列的消息的单个基优先级。默认值为零(0)。

例如，设置消息队列的基优先级为 10。代码如下：

```
this.messageQueue1.BasePriority=10;
```

（4）Category 属性。该属性表示获取或设置队列类别。

语法：

```
public Guid Category { get; set; }
```

属性值：Guid，表示队列类别（“消息队列”类型标识符），它允许应用程序将其队列分类。默认值为 Guid.empty。

例如，设置消息队列类别为指定类型。代码如下：

```
this.messageQueue1.Category =new System.Guid("00000000-0000-0000-0000-000000000001");
```

（5）FormatName 属性。该属性表示获取“消息队列”在创建队列时生成的惟一队列名。

语法：

```
public string FormatName { get; }
```

属性值：队列的名称，该名称在网络上是惟一的。

例如，获取消息队列的列名并显示在指定标签控件中。代码如下：

```
this.label1.Text=this.messageQueue1.FormatName;
```

（6）Label 属性。该属性表示获取或设置队列说明。

语法：

```
public string Label { get; set; }
```

属性值：消息队列的标签。默认值为空字符串("")。

例如，以信息框的形式显示消息队列的说明。代码如下：

```
MessageBox.Show("队列说明：" +this.messageQueue1.Label, "信息提示");
```

（7）MachineName 属性。该属性表示获取或设置“消息队列”所在的计算机的名称。

语法：

```
public string MachineName { get; set; }
```

属性值：队列所在的计算机的名称。“消息队列”默认值为“.”，即本地计算机。

例如，设置消息队列所在的计算机为 N，N 为消息队列所在的任意计算机。代码如下：

```
this.messageQueue1.MachineName="N";
```

（8）MaximumQueueSize 属性。该属性表示获取或设置队列的最大大小。

语法：

```
public long MaximumQueueSize { get; set; }
```

属性值：队列的最大大小，以千字节为单位。“消息队列”默认不存在任何限制。

例如，设置消息队列的最大大小为 1M。代码如下：

```
this.messageQueue1.MaximumQueueSize=1048.576;
```

（9）MessageReadPropertyFilter 属性。该属性表示获取或设置接收或查看消息的属性筛选器。

语法：

```
public MessagePropertyFilter MessageReadPropertyFilter { get; set; }
```

属性值：由队列使用的 MessagePropertyFilter，用来筛选队列所接收或查看的每个消息的属性集。

例如，设置接收或查看消息的属性筛选器为不接收消息。代码如下：

```
MessagePropertyFilter Filter = new MessagePropertyFilter();
Filter.ClearAll();
this.messageQueue1.MessageReadPropertyFilter=Filter ;
```

（10）Path 属性。该属性表示获取或设置队列的路径。

语法：

```
public string Path { get; set; }
```

属性值：由 MessageQueue 引用的队列。默认值取决于所使用的 MessageQueue 构造函数，它或者是 null，或者是由构造函数的 path 参数指定的。

例如，将队列的路径设置为指定路径。代码如下：

```
this.messageQueue1.Path = ".\\MyQueue";
```

3．方法

MessageQueue 组件常用方法及说明如表 9.66 所示。

表 9.66　　MessageQueue 组件常用方法及说明

方　法	说　明
BeginPeek	通过通知“消息队列”开始查看消息，并在完成查看操作后通知事件处理程序，启动一个异步查看操作
BeginReceive	通过通知“消息队列”开始接收消息并在完成后通知事件处理程序，启动一个异步接收操作
ClearConnectionCache	清除连接缓存
Create	在“消息队列”服务器上的指定路径中创建新队列
CreateCursor	为当前消息队列创建新的 Cursor
Delete	删除“消息队列”服务器上的队列
EndPeek	完成指定的异步查看操作
EndReceive	完成指定的异步接收操作
Exists	确定指定的路径中是否存在“消息队列”
GetAllMessages	返回位于队列中的所有消息

续表

方　法	说　明
GetMachineId	获取计算机的标识符
GetPrivateQueuesByMachine	检索指定计算机上的所有专用队列
GetPublicQueues	检索网络上的所有公共队列
Peek	返回队列中第一条消息的副本，而不从队列中移除该消息
PeekByCorrelationId	查看匹配给定相关标识符的消息
PeekByLookupId	查看队列中特定的消息。消息可通过查询标识符来指定，也可以通过其相对于队首或队尾的位置来指定
Purge	删除队列中包含的所有消息
Receive	接收队列中的第一条消息，但不将它从队列中移除
Refresh	刷新 MessageQueue 所显示的属性以反映资源的当前状态
Send	向队列发送对象
SetPermissions	将权限添加到当前集合中。这样可控制谁对队列中的队列属性和消息具有访问权限

下面对 MessageQueue 组件比较重要的方法进行详细介绍。

（1）ClearConnectionCache 方法。该方法表示清除连接缓存。

语法：

```
public static void ClearConnectionCache()
```

例如，清除当前队列的连接缓存。代码如下：

```
this.messageQueue1.ClearConnectionCache();
```

（2）Create 方法。该方法表示在“消息队列”服务器上的指定路径中创建新队列。

语法：

```
public static MessageQueue Create(string path,bool transactional)
```

参数说明如下。

path：要创建的队列的路径。

transactional：如果创建事务性队列，为 true；如果创建非事务性队列，则为 false。

返回值：表示新队列的 MessageQueue。

例如，在指定路径下创建非事务性队列。代码如下：

```
this.messageQueue1.Create(".\\MyQueue",false);
```

（3）Delete 方法。该方法表示删除“消息队列”服务器上的队列。

语法：

```
public static void Delete(string path)
```

参数说明如下。

path：要删除的队列的位置。

说明：path 参数的语法取决于队列类型。

例如，删除消息队列服务器上的指定队列。代码如下：

```
this.messageQueue1.Delete(".\\MyQueue");
```

（4）Exists 方法。该方法表示确定指定的路径中是否存在“消息队列”。

语法：

```
public static bool Exists(string path)
```

参数说明如下。

path：要查找的队列的位置。

例如，判断当前应用程序中是否存在指定路径下的消息队列。代码如下：

```
if(! MessageQueue.Exists(".\\MyQueue"))
{
    MessageBox.Show(".\\MyQueue 路径下不存在消息队列！","信息提示");
}
```

（5）GetAllMessages 方法。该方法表示返回位于队列中的所有消息。

语法：

```
public Message[] GetAllMessages()
```

返回值：类型 Message 的数组，表示队列中所有消息，其顺序与这些消息在“消息队列”中的出现顺序相同。

例如，在控制台环境中，获取指定路径下队列中的所有消息。代码如下：

```
MessageQueue queue = new MessageQueue(".\\exampleQueue");        //从指定路径中获取消息队列
Message[] msgs = queue.GetAllMessages();                          //获取指定队列中的所有消息
foreach (Message msg in msgs)                                     //循环遍历消息集合中的每一条消息
{
    Console.WriteLine(msg.Label);                                 //输出消息的内容
}
```

（6）GetPrivateQueuesByMachine 方法。该方法表示检索指定计算机上的所有专用队列。

语法：

```
public static MessageQueue[] GetPrivateQueuesByMachine(string machineName)
```

参数说明如下。

machineName：计算机，从其中检索专用队列。

返回值：引用检索的专用队列的 MessageQueue 对象数组。

例如，获取本地计算机上的所有专有队列并循环输出各自的路径。代码如下：

```
MessageQueue[] QueueList =MessageQueue.GetPrivateQueuesByMachine(".");    //获取本机中的所有消息队列
foreach (MessageQueue queueItem in QueueList)                             //循环遍历队列中的所有消息
{
    Console.WriteLine(queueItem.Path);                                    //输出消息的路径
}
```

（7）Peek 方法。该方法表示返回队列中第一条消息的副本，而不从队列中移除该消息。

语法：

```
public Message Peek(TimeSpan timeout,Cursor cursor,PeekAction action)
```

参数说明如下。

timeout：TimeSpan，指示等待队列包含消息所用的最长时间。

cursor：维持消息队列中特定位置的 Cursor。

action：PeekAction 值之一。指示是查看队列中的当前消息还是下一条消息。

返回值：一个 Message，表示队列中的消息。

例如，获取队列中第一条消息。代码如下：

```
System.Messaging.Cursor cursor = new System.Messaging.Cursor();
this.messageQueue1.Peek(new TimeSpan(30),cursor , PeekAction.Current);
```

（8）Purge 方法。该方法表示删除队列中包含的所有消息。

语法：

```
public void Purge()
```

例如，删除队列中包含的所有消息。代码如下：

```
this.messageQueue1.Purge();
```

（9）Receive 方法。该方法表示使用指定的游标接收队列中的当前消息。如果没有可用的消息，此方法将等待，直到有可用的消息或超时到期为止。

语法：

```
public Message Receive(TimeSpan timeout,Cursor cursor, MessageQueueTransactionType transactionType)
```

参数说明如下。

timeout：TimeSpan，指示在有新消息可用于检查之前等待的时间。

cursor：维持消息队列中特定位置的 Cursor。

transactionType：MessageQueueTransactionType 值之一，它描述与消息关联的事务上下文类型。

返回值：一个 Message，它引用队列中的一条消息。

例如，使用指定的游标接收队列中的当前消息。代码如下：

```
System.Messaging.Cursor cursor = new System.Messaging.Cursor();
this.messageQueue1.Receive(new TimeSpan(30),cursor,MessageQueueTransactionType.Single );
```

说明：MessageQueueTransactionType 的可选值有 None、Automatic 以及 Single。其中 None 表示操作不是事务性的，Automatic 表示用于 Microsoft Transaction Server（MTS）或 COM + 1.0 服务的事务类型。如果已有 MTS 事务上下文，将在发送或接收消息时使用它，Single 表示用于单个内部事务的事务类型。

（10）Refresh 方法。该方法表示刷新 MessageQueue 所显示的属性以反映资源的当前状态。

语法：

```
public void Refresh()
```

例如，对 MessageQueue 组件的属性进行设置后需要获取其当前的状态。代码如下：

```
this.messageQueue1.Refresh();
```

4．事件

（1）PeekCompleted 事件。该事件在消息未从队列移除的情况下读取该消息时发生。

语法：

```
public event PeekCompletedEventHandler PeekCompleted
```

（2）ReceiveCompleted 事件。该事件在从队列移除某条消息后发生。

语法：

```
public event ReceiveCompletedEventHandler ReceiveCompleted
```

示例 09-57　MessageQueue 组件的应用

运行本示例，在发送者文本框中输入发送者昵称或者姓名，在消息内容中写入消息体，然后单击“发送”按钮，消息即被发送至消息队列。当用户需要获取消息队列中的内容时，单击“读取”按钮，即可在 ListBox 控件中读取所有曾经发送的消息。代码如下：

代码 09-57　光盘位置：光盘\mingrisoft\sl\09\57

```
    private void SendButton_Click(object sender, EventArgs e)
    {
        if (MessageQueue.Exists(".\\MyQueue"))                          //当服务器中存在 MyQueue 队列时
        {
            MessageQueue MsgQueue = new MessageQueue(".\\MyQueue");   //实例化一个消息队列对象
            //实例化一个访问消息队列属性的对象
            System.Messaging.Message Msg = new System.Messaging.Message();
            Msg.Label = this.Sender.Text + "    " + DateTime.Now.ToString();   //设置消息的标题
            Msg.Body = this.MessageInformation.Text;                        //设置消息的内容
            MsgQueue.Send(Msg);                                             //发送消息
            //弹出消息发送成功的信息提示
            MessageBox.Show("消息发送成功！", "友情提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            this.Sender.Clear();                                            //清空发送者中的内容
            this.Sender.Focus();                                            //设置当前窗体的焦点位于发送者文本框中
            this.MessageInformation.Clear();                                //清空消息内容中的原有内容
        }
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        if (!MessageQueue.Exists(".\\MyQueue"))                          //判断是否存在消息队列 MyQueue
        {
            this.messageQueue1 = MessageQueue.Create(".\\MyQueue");          //创建非事务性消息队列
            MessageBox.Show("队列 MyQueue 创建成功！", "友情提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk);                                                   //弹出队列创建成功的信息提示
            this.messageQueue1.EncryptionRequired = EncryptionRequired.None;//设置该队列接收非私有消息
        }
    }
    private void ReadAll_Click(object sender, EventArgs e)
```

```
{
    MessageQueue MsgQueue = new MessageQueue(".\\MyQueue");          //实例化一个消息队列实例
    System.Messaging.Message[] Msg;                    //实例化一个用于访问消息队列属性的 Message 类型数组
    Msg = MsgQueue.GetAllMessages();                   //获取 MsgQueue 中的每一条消息记录
    this.listBox1.Items.Clear();                       //清空 listBox1 控件中的原有内容
    for (int i = 0; i < Msg.Length; i++)               //循环遍历 Msg 中的每一条消息记录
    {
        //将对象序列化到消息体中
        Msg[i].Formatter = new XmlMessageFormatter(new string[] { "System.String,mscorlib" });
        this.listBox1.Items.Add(Msg[i].Label);                          //向 listBox1 控件中添加消息的标题
        this.listBox1.Items.Add(System.Environment.NewLine);            //添加一个空行
        this.listBox1.Items.Add(Msg[i].Body);                           //向 listBox1 控件中添加消息的内容部分
        this.listBox1.Items.Add(System.Environment.NewLine);            //添加一个空行
    }
}
```

运行本示例，效果如图 9.96 所示。

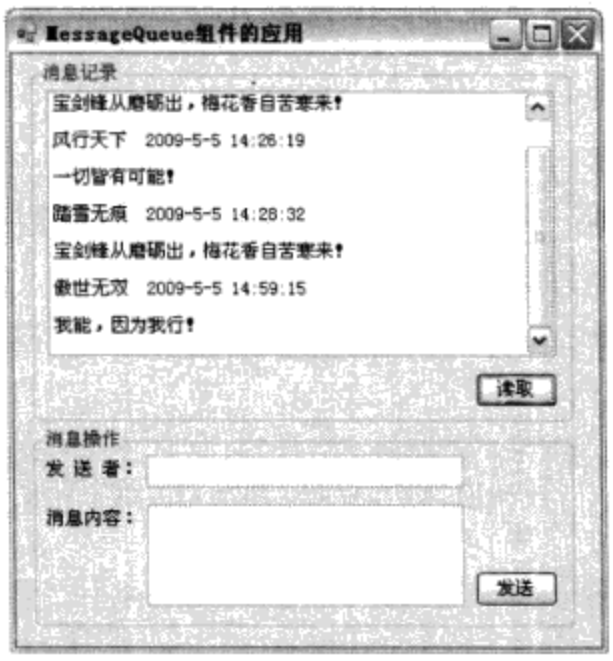


图 9.96 MessageQueue 组件的应用

9.10.10 PerformanceCounter 组件

1．功能

PerformanceCounter 组件可以方便地连接到本地和远程计算机上的性能计数器中，并从这些计数器中检索值。此外，还可以将值写入到本地计算机上现有的自定义计数器（用.NET Framework 创建的计数器），并在安装 Windows 的计算机上创建自己的自定义计数器。图 9.97 所示为 PerformanceCounter 组件。

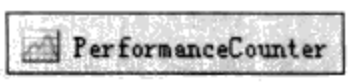


图 9.97 PerformanceCounter 组件

2．属性

PerformanceCounter 组件常用属性及说明如表 9.67 所示。

表 9.67 **PerformanceCounter 组件常用属性及说明**

属　性	说　明
CategoryName	获取或设置此性能计数器的性能和计数器类别的名称
CounterHelp	获取此性能计数器的说明
CounterName	获取或设置与 PerformanceCounter 实例相关联的性能计数器名称
CounterType	获取关联的性能计数器的计数器类型
InstanceLifetime	获取或设置进程的生存期
InstanceName	获取或设置此性能计数器的实例名称

续表

属　性	说　明
MachineName	获取或设置此性能计数器的计算机名
RawValue	获取或设置此计数器的原始值（即未经过计算的值）
ReadOnly	获取或设置一个值，该值指示 PerformanceCounter 实例是否处于只读模式

下面对 PerformanceCounter 组件中比较重要的属性进行详细介绍。

（1）CategoryName 属性。该属性表示获取或设置此性能计数器的性能和名称。

语法：

```
public string CategoryName { get; set; }
```

属性值：与此性能计数器相关联的性能计数器类别（性能对象）名称。

例如，设置此性能计数器的别名为“Apple”。代码如下：

```
this.performanceCounter.CategoryName="Apple";
```

（2）CounterName 属性。该属性表示获取或设置与 PerformanceCounter 实例相关联的性能计数器的名称。

语法：

```
public string CounterName { get; set; }
```

属性值：计数器的名称，它一般说明计数的数量，此名称显示在性能计数器、管理器 MMC 和管理单元的“添加计数器”对话框的计数器列表中。

例如，获取与 PerformanceCounter 实例相关联的性能计数器名称显示在标签控件中。代码如下：

```
this.label1.Text=this.performanceCounter1.CounterName;
```

（3）InstanceLifetime 属性。该属性表示获取或设置进程的生存期。

语法：

```
public PerformanceCounterInstanceLifetime InstanceLifetime { get; set; }
```

属性值：PerformanceCounterInstanceLifetime 值之一，默认值为 Global。Performance Counter InstanceLifetime 有两个值，其值的名称及作用如下。

- Global：表示在没有计数器使用进程类别时，移除该性能计数器实例。
- Process：表示关闭进程时，移除该性能计数器实例。

例如，当关闭进程时，移除该性能计数器实例。代码如下：

```
this.performanceCounter1.InstanceLisetime=PerformanceCounterInstanceLifetime.Process;
```

（4）MachineName 属性。该属性表示获取或设置此性能计数器的计算机名。

语法：

```
public string MachineName { get; set; }
```

属性值：性能计数器及其关联的类别所在的服务器。

例如，获取当前性能计数器所在的计算机以信息框的形式显示。

```
MessageBox.Show("性能计数器所在的计算机为："+this.performanceCounter1.MachineName ,"信息提示");
```

（5）RawValue 属性。该属性表示获取或设置此计数器的原始值（即未经过计算的值）。

语法：

```
public long RawValue { get; set; }
```

属性值：计数器的原始值。

例如，设置此性能计数器的原始值为 18。代码如下：

```
this.performaceCounter.RawValue=18;
```

（6）ReadOnly 属性。该属性表示获取或设置一个值，该值指示此 PerformanceCounter 实例是否处于只读模式。

语法：

```
public bool ReadOnly { get; set; }
```

属性值：如果 PerformanceCounter 实例为只读模式（即使计数器本身为自定义.NET Framework 计数器），则为 true；如果它为读/写模式，则为 false。默认值为构造函数设置的值。

例如，设置当前的性能计时器处于只读模式。代码如下：

```
this.performaceCounter1.ReadOnly=true;
```

3．方法

PerformanceCounter 组件常用方法及说明如表 9.68 所示。

表 9.68　PerformanceCounter 组件常用方法及说明

方　法	说　明
BeginInit	初始化在窗体上使用或由另一个组件使用的 PerformanceCounter 实例。此初始化在运行时发生
Close	关闭性能计数器并释放由此性能计数器实例分配的所有资源
CloseSharedResources	释放由计数器分配的性能计数器库共享状态
Decrement	通过有效的原子操作使关联的性能计数器减一
EndInit	结束在窗体上使用或由另一组件使用的 PerformanceCounter 实例的初始化。此初始化在运行时发生
Increment	通过有效的原子操作使关联的性能计数器加 1
IncrementBy	通过有效的原子操作，使关联的性能计数器的值增加或减少指定的量
NextSample	获取计数器样本，并为其返回原始值（即未经过计算的值）
NextValue	获取计数器样本并为其返回计算所得值
RemoveInstance	删除由 PerformanceCounter 对象的 InstanceName 属性指定的类别实例

下面对 PerformanceCounter 组件比较重要的方法进行详细介绍。

（1）Close 方法。该方法表示关闭性能计数器并释放由此性能计数器实例分配的所有资源。

语法：

```
public void Close()
```

例如，关闭性能计数器并释放由此性能计数器实例分配的所有资源。代码如下：

```
this.performanceCounter1.Close();
```

（2）Decrement 方法。该方法表示通过有效的原子操作使关联的性能计数器减 1。

语法：

```
public long Decrement()
```

返回值：减少后的计数器值。

例如，通过有效的原子操作使关联的性能计数器减 1。代码如下：

```
this.performanceCounter1.Decrement();
```

（3）Increment 方法。该方法表示通过有效的原子操作使关联的性能计数器加 1。

语法：

```
public long Increment()
```

返回值：增加后的计数器值。

例如，通过有效的原子操作使关联的性能计数器加 1。代码如下：

```
this.performanceCounter1.Increment();
```

（4）IncrementBy 方法。该方法表示通过有效的原子操作，使关联的性能计数器的值增加指定的量。

语法：

```
public long IncrementBy(long value)
```

参数说明如下。

value：要增加的值（负值使计数器减少）。

返回值：新的计数器值。

例如，通过有效的原子操作使关联的性能计数器增加指定的量。代码如下：

```
this.performanceCounter1.IncrementBy(10);
```

（5）NextValue 方法。该方法用于获取计数器样本，并为其返回计算所得值。

语法：

```
public float NextValue ()
```

返回值：系统为此计数器获取的下一计算所得值。

例如，获取计算器样本并为其返回计算所得值。代码如下：

```
this.performanceCounter1.NextValue();
```

示例 09-58　PerformanceCounter 组件的应用

运行本示例，在左侧的 GroupBox 控件中，单击“获取”按钮，在相应的 ListBox 控件中显示计数器的类别。选择一个任意的计数器类别，在右侧的 GroupBox 控件中，单击“获取”按钮，在相应的 ListBox 控件中显示该类别下的所有计数器。代码如下。

代码 09-58　光盘位置：光盘\mingrisoft\sl\09\58

```
private void btnGetCounters_Click(object sender, EventArgs e)
{
    string[] instanceNames;                                                    //声明一个字符串数组
    System.Collections.ArrayList counters = new System.Collections.ArrayList(); //实例化一个 ArrayList 对象
    if (this.lstCategories.SelectedIndex != -1)                                //当 lstCategories 中存在内容时
    {
        //实例化一个性能计数器类别对象
        System.Diagnostics.PerformanceCounterCategory mycat = new
System.Diagnostics.PerformanceCounterCategory(this.lstCategories.SelectedItem.ToString());
        this.lstCounters.Items.Clear();                        //清空 lstCounter 中的原有内容
        try
        {
            instanceNames = mycat.GetInstanceNames();          //检索此类别下的性能实例对象
            if (instanceNames.Length == 0)                     //当 instanceNames 中不存在内容时
            {
                counters.AddRange(mycat.GetCounters());        //向 counters 中添加性能计数器类别中的计数器列表
            }
            else                                               //当 instanceNames 中存在内容时
            {
                for (int i = 0; i < instanceNames.Length; i++)//循环遍历数组 instanceNames 中的每一条记录
                {
                    //向 counters 中添加指定性能计数器类别中的计数器列表
                    counters.AddRange(mycat.GetCounters(instanceNames[i]));
                }
            }
            //循环遍历计数器集合中的每一个计数器
            foreach (System.Diagnostics.PerformanceCounter counter in counters)
            {
                this.lstCounters.Items.Add(counter.CounterName);          //向 lstCounters 中添加内容
            }
        }
        catch (System.Exception ex)                                       //捕获异常
        {
            MessageBox.Show("不能显示该类别的计数器：\n" + ex.Message);//显示异常信息
        }
    }
}
private void btnGetCategories_Click(object sender, EventArgs e)
{
    System.Diagnostics.PerformanceCounterCategory[] myCat2;               //声明一个性能计数器对象的数组
    this.lstCategories.Items.Clear();                                     //清空 lstCategories 中的原有内容
    myCat2 = System.Diagnostics.PerformanceCounterCategory.GetCategories(); //实例化对象 myCat2
    for (int i = 0; i < myCat2.Length; i++)                               //循环遍历 myCat2 中的每一条记录
    {
        this.lstCategories.Items.Add(myCat2[i].CategoryName);             //向 lstCategories 中添加内容
    }
}
```

运行本示例，效果如图 9.98 所示。

说明：在向窗体中添加 PerformanceCounter 组件后需要设置其 CategoryName 属性、CounterName 属性以及 InstanceName 属性，属性值为任意的系统中的任意一个对象。本实例的设置如图 9.99 所示。

图 9.98　PerformanceCounter 组件的应用

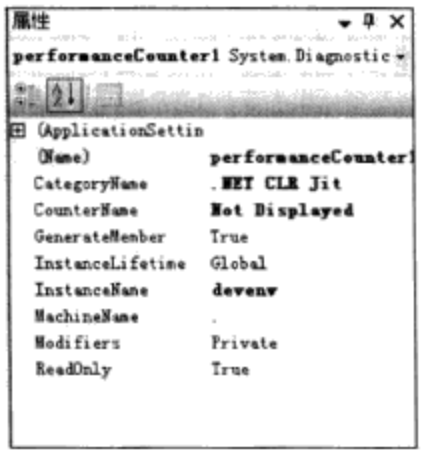


图 9.99　PerformanceCounter 组件的属性设置

9.10.11　ServiceController 组件

1．功能

ServiceController 组件实际上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。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，而不等待服务控制管理器将请求传递给服务。因此，应该在服务类自身而不是在 ServiceController 类中处理异常。图 9.100 所示为 ServiceController 组件。

图 9.100　ServiceController 组件

2．属性

ServiceController 组件常用属性及说明如表 9.69 所示。

表 9.69　　ServiceController 组件常用属性及说明

属　性	说　明
CanPauseAndContinue	获取一个值，该值指示是否可以暂停和继续服务
CanShutdown	获取一个值，该值指示在系统关闭时是否应通知服务
CanStop	获取一个值，该值指示服务在启动后是否可以停止
DependentServices	获取依赖于与此 ServiceController 实例关联的服务的服务集
DisplayName	获取或设置服务的友好名称
MachineName	获取或设置此服务所驻留的计算机的名称
ServiceHandle	获取服务的句柄
ServiceName	获取或设置对此实例引用的服务进行标识的名称
ServiceType	获取此对象引用的服务类型
Status	获取由此实例引用的服务状态

下面对 ServiceController 组件中比较重要的属性进行详细介绍。

（1）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：

```
public bool CanStop { get;}
```

属性值：如果服务可以停止则为 true；否则为 false。

例如，获取一个值判断服务器启动后是否可以停止。代码如下：

```
if (this.serviceController1.CanStop)
{
    MessageBox.Show("当前服务器可以在启动后停止！","信息提示");
}
```

（2）DependentServices 属性。该属性表示获取依赖于与此 ServiceController 实例关联的服务的服务集。

语法：

```
public ServiceController[] DependentServices { get; }
```

属性值：ServiceController 实例的数组，其中每个数组均与依赖于此服务的一个服务关联。

例如，判断服务集上是否有服务，如果存在，那么输出服务的名称。代码如下：

```
ServiceController sc = new ServiceController("Event Log");          //实例化一个 ServiceController 实例
ServiceController[] scServices = sc.DependentServices;              //获取一个服务集
if (scServices.Length == 0)                                         //当服务集中不存在内容时
{
    Console.WriteLine("在{0}上没有服务信息",sc.ServiceName);        //输出信息提示
}
else                                                                //当服务集中存在内容时
{
    Console.WriteLine("基于{0}的服务有:",sc.ServiceName);           //输出信息提示
    foreach (ServiceController scTemp in scServices)                //循环遍历集合中的每一个实例
    {
        Console.WriteLine(" {0}", scTemp.DisplayName);              //输出服务的名称
    }
}
```

（3）ServiceName 属性。该属性表示获取或设置对此实例引用的服务进行标识的名称。

语法：

```
public string ServiceName { get; set; }
```

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（""）。

例如，获取此示例引用的服务进行标识的名称显示在标签控件中。代码如下：

```
this.label1.Text=this.serviceController1.ServiceName;
```

（4）Status 属性。该属性表示获取由此实例引用的服务状态。

语法：

```
public ServiceControllerStatus Status { get; }
```

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。

说明：ServiceControllerStatus 的可选值有 ContinuePending、Paused、PausePending、Running、StartPending、Stopped 以及 StopPending。其中 ContinuePending 表示服务即将继续，Paused 表示服务已暂停，PausePending 表示服务即将暂停，Running 表示服务正在运行，StartPending 表示服务正在启动，Stopped 表示服务未运行，StopPending 表示服务正在停止。

例如，判断服务所处的状态并以信息框的形式进行显示。代码如下：

```
switch (this.serviceController1.Status )
{
    case ServiceControllerStatus.ContinuePending:
        MessageBox.Show("服务即将继续！","信息提示");
        break;
    case ServiceControllerStatus.PausePending:
        MessageBox.Show("服务即将暂停！", "信息提示");
        break;
    case ServiceControllerStatus.Paused:
        MessageBox.Show("服务已暂停！", "信息提示");
        break;
    case ServiceControllerStatus.Running:
        MessageBox.Show("服务正在运行！", "信息提示");
        break;
    case ServiceControllerStatus.StartPending:
        MessageBox.Show("服务正在启动！", "信息提示");
        break;
    case ServiceControllerStatus.StopPending:
        MessageBox.Show("服务正在停止！", "信息提示");
```

```
            break;
        case ServiceControllerStatus.Stopped:
            MessageBox.Show("服务未运行！", "信息提示");
            break;
    }
```

3．方法

ServiceController 组件常用方法及说明如表 9.70 所示。

表 9.70　　ServiceController 组件的常用方法及说明

方　法	说　明
Close	断开此 ServiceController 实例与服务的连接，并释放此实例分配的所有资源
Continue	在服务暂停后继续该服务
ExecuteCommand	对服务执行自定义命令
GetDevices	检索计算机上的设备驱动程序服务
GetServices	检索计算机上的非设备驱动程序服务和不是驱动程序的服务
Pause	挂起服务的操作
Refresh	通过将属性重置为其当前值来刷新属性值
Start	开启服务
Stop	停止服务
WaitForStatus	等待服务达到指定状态

下面对 ServiceController 组件中比较重要的方法进行详细介绍。

（1）Close 方法。该方法表示断开此 ServiceController 实例与服务的连接，并释放此实例分配的所有资源。

语法：

```
public void Close()
```

例如，断开此 ServiceController 实例与服务的连接，同时释放此实例分配的所有资源。

```
this.serviceController1.Close();
```

（2）Continue 方法。该方法表示在服务暂停后继续该服务。

语法：

```
public void Continue()
```

例如，在服务暂停后继续服务。代码如下：

```
this.serviceController1.Continue();
```

（3）ExecuteCommand 方法。该方法表示对服务执行自定义命令。

语法：

```
public void ExecuteCommand(int command)
```

参数说明如下。

command：应用程序定义的命令标志，指示要执行的自定义命令。

例如，定义一个枚举变量 SimpleServiceCustomCommands，然后使用枚举值作为自定义命令。代码如下：

```
private enum SimpleServiceCustomCommands
{
    StopWorker = 128,
    RestartWorker,
    CheckWorker
};
this.serviceController1.ExecuteCommand(((int)SimpleServiceCustomCommands.StopWorker ));
```

（4）Pause 方法。该方法表示挂起服务的操作。

语法：

```
public void Pause()
```

例如，设置服务为挂起状态。代码如下：

```
this.serviceController1.Pause();
```

（5）Start 方法。该方法用于开启服务。

语法：

```
public void Start ()
public void Start (string[] args)
```

参数说明如下。

args：在服务启动时传递给它的参数数组。

例如，使用 Start 方法开启服务，使用 Stop 方法停止服务，代码如下：

```
this.serviceController1.Start();
this.serviceController1.Stop();
```

（6）Stop 方法。该方法用于停止服务以及任何依赖该服务的服务。

语法：

```
public void Stop ()
```

（7）WaitForStatus 方法。该方法表示等待服务达到指定状态或等待指定的超时过期。

语法：

```
public void WaitForStatus(ServiceControllerStatus desiredStatus,TimeSpan timeout)
```

参数说明如下。

desiredStatus：要等待的状态。

timeout：一个 TimeSpan 对象，指定服务在到达指定状态前需要等待的时间量。

例如，设置达到暂停需要等待的时间量为 3s。代码如下：

```
this.serviceController1.WaitForStatus(ServiceControllerStatus.Paused ,new TimeSpan(0,0,3));
```

示例 09-59　ServiceController 组件的应用

本示例主要通过对 ServiceController 属性和方法的运用，实现对 IIS 服务器的开启、暂停继续以及停止操作。代码如下。

代码 09-59　光盘位置：光盘\mingrisoft\sl\09\59

```
    private void Form1_Load(object sender, EventArgs e)
    {
        this.JustifyStatus();                          //判断当前服务器状态从而改变不同按钮的可用状态
    }.
    private void TurnOn_Click(object sender, EventArgs e)
    {
        //当 IIS 的当前状态是停止时
        if (this.serviceController1.Status == System.ServiceProcess.ServiceControllerStatus.Stopped)
        {
            this.serviceController1.Start();                                    //启动 IIS 服务器
            MessageBox.Show("启动服务成功", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);      //弹出开启成功的信息提示
            Application.Exit();                                                 //退出应用程序
        }
    }
    private void Pause_Click(object sender, EventArgs e)
    {
        if (this.serviceController1.CanPauseAndContinue)                        //当可以暂停和继续服务时
        {
            //当 IIS 的当前状态是正在运行时
            if (this.serviceController1.Status == System.ServiceProcess.ServiceControllerStatus.Running)
            {
                this.serviceController1.Pause();                                //暂停 IIS 服务器
                MessageBox.Show("暂停服务成功", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);      //弹出暂停服务成功的信息提示
                Application.Exit();                                             //退出应用程序
            }
        }
        else                                                                    //当不能暂停和继续服务时
        {
```

```
            MessageBox.Show("无法暂停服务", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk, MessageBox
DefaultButton.Button1, MessageBoxOptions.ServiceNotification);                    //弹出无法暂停的信息提示
        }
    }
    private void Continue_Click(object sender, EventArgs e)
    {
        if (this.serviceController1.CanPauseAndContinue)                          //当可以暂停和继续服务时
        {
            //当服务处于暂停状态时
            if (this.serviceController1.Status == System.ServiceProcess.ServiceControllerStatus.Paused)
            {
                this.serviceController1.Continue();                               //继续 IIS 服务
                MessageBox.Show("继续服务成功", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);          //弹出继续服务成功的信息提示
                Application.Exit();                                               //退出应用程序
            }
        }
        else                                                                      //当无法暂停和继续服务时
        {
            MessageBox.Show("无法继续服务", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);          //弹出无法继续服务的信息提示
        }
    }
    private void Stop_Click(object sender, EventArgs e)
    {
        //当 IIS 服务处于运行状态时
        if (this.serviceController1.Status == System.ServiceProcess.ServiceControllerStatus.Running)
        {
            this.serviceController1.Stop();                                       //停止 IIS 服务
            MessageBox.Show("停止服务成功", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);          //弹出停止 IIS 服务成功的信息提示
            Application.Exit();                                                   //退出应用程序
        }
        else                                                                      //当 IIS 服务处于其它状态时
        {
            MessageBox.Show("无法停止服务", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk,
MessageBoxDefaultButton.Button1, MessageBoxOptions.ServiceNotification);          //弹出无法停止 IIS 服务的信息提示
        }
    }
    public void JustifyStatus()
    {
        this.serviceController1.MachineName = ".";  //设置 IIS 服务所在的计算机，“.” 表示为本机
        this.serviceController1.ServiceName = "IISAdmin";                         //设置实例的服务名
        switch (this.serviceController1.Status)                                   //选择 IIS 所处的不同状态
        {
            case System.ServiceProcess.ServiceControllerStatus.Paused:            //当服务处于暂停状态时
                this.Continue.Enabled = true;                                     //设置“继续”按钮为可用状态
                this.TurnOn.Enabled = false;                                      //设置“开启”按钮为不可用状态
                this.Stop.Enabled = false;                                        //设置“停止”按钮为不可用状态
                this.Pause.Enabled = false;                                       //设置“暂停”按钮为不可用状态
                break;
            case System.ServiceProcess.ServiceControllerStatus.Running:           //当服务处于正在运行的状态时
                this.Pause.Enabled = true;                                        //设置“暂停”按钮为可用状态
                this.Stop.Enabled = true;                                         //设置“停止”按钮为可用状态
                this.TurnOn.Enabled = false;                                      //设置“开启”按钮为不可用状态
                this.Continue.Enabled = false;                                    //设置“继续”按钮为不可用状态
                break;
            case System.ServiceProcess.ServiceControllerStatus.Stopped:           //当服务处于停止状态时
                this.TurnOn.Enabled = true;                                       //设置“开启”按钮为可用状态
                this.Continue.Enabled = false;                                    //设置“继续”按钮为不可用状态
                this.Stop.Enabled = false;                                        //设置“停止”按钮为不可用状态
```

```
                this.Pause.Enabled = false;                    //设置“暂停”按钮为不可用状态
                break;
            }
        }
```

运行本示例，效果如图 9.101 所示。

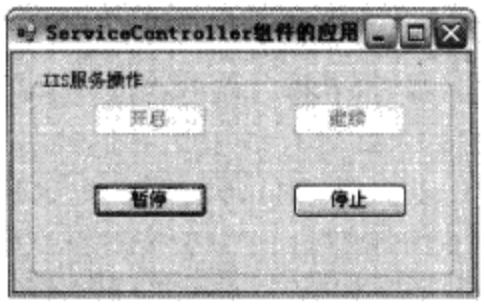


图 9.101　ServiceController 组件的应用

9.10.12　Timer 组件

1．功能

Timer 是定期引发事件的组件，该组件是为 Windows 窗体环境设计的，图 9.102 所示为 Timer 组件。

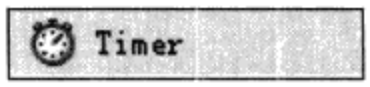


图 9.102　Timer 组件

2．属性

（1）Enable 属性。该属性用于指定计时器是否运行，此属性设置为 true，则表示可以启动计时器。

语法：

```
public bool Enabled { get; set; }
```

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。

例如，设置计时器为不可用状态。代码如下：

```
this.timer1.Enable=false;
```

（2）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）。

语法：

```
public double Interval { get; set; }
```

属性值：Int32 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。

例如，设置计时器发生 Tick 事件的时间间隔为 1s。代码如下：

```
this.timer1.Interval=1000;
```

3．方法

（1）Start 方法。该方法用来启动计时器。

语法：

```
public void Start()
```

例如，开启计时器。代码如下：

```
this.timer1.Start();
```

（2）Stop 方法。该方法用来停止计时器。

语法：

```
public void Stop()
```

例如，停止计时器。代码如下：

```
this.timer1.Stop();
```

4．Tick 事件

该事件在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。

语法：

```
public event EventHandler Tick
```

示例 09-60 显示并控制系统时间

本示例是在 Timer 组件的 Tick 事件下显示系统时间。代码如下：

代码 09-60 光盘位置：光盘\mingrisoft\sl\09\60

```
private void Form1_Load(object sender, EventArgs e)
{
    this.timer1.Interval = 1000;                          //设置 timer1 控件的 Interval 属性
}
private void timer1_Tick(object sender, EventArgs e)
{
    this.textBox1.Text = System.DateTime.Now.ToString();  //在 textBox1 控件中显示当前系统时间
}
private void Start_Click(object sender, EventArgs e)
{
    this.timer1.Start();                                  //开启计时器 timer1
}
private void Stop_Click(object sender, EventArgs e)
{
    this.timer1.Stop();                                   //停止计时器 timer1
}
```

运行本示例，效果如图 9.103 所示。

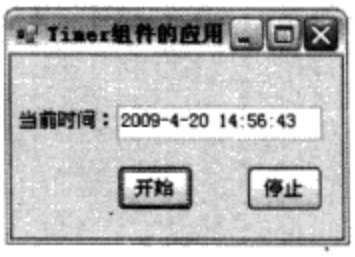


图 9.103 Timer 组件的应用

第 10 章

用户控件

- 用户控件的概述
- 用户控件的应用

在 Windows 窗体应用程序中用户除了可以使用基本控件外，还可以自己创建控件，即利用现有的控件创建可重复使用的用户控件集，这些控件集称为用户控件。通过本章的学习，读者可以进一步了解如何在 Windows 窗体应用程序中创建用户控件的相关知识。

10.1 用户控件的概述

10.1.1 用户控件的概述

在进行 Windows 应用程序开发时，开发工具的功能虽然强大，但可能不符合程序功能的需求。在这种情况下，需要用到用户控件。用户控件在形式上同单个控件一样，程序开发人员可以在这种控件中创建自定义的属性和事件。用户控件一般只在同一个项目中使用，并提供了一种创建和重用自定义图形的方法，用户控件本质上是具有可视化表示形式的组件，而且提供了大量的默认键盘处理功能，不需要任何额外的开发。

10.1.2 用户控件的优点

用户控件可以包含一个或多个 Windows 窗体控件、组件或代码块。它们能够通过验证用户输入、修改显示属性或执行用户所需的其他任务来扩展功能。用户可以将用户控件作为 Windows 桌面应用程序项目的一部分创建，且只在该项目的窗体上使用它。用户也可以在 Windows Control Library 项目中创建它，将该项目编译成一个程序集，在其他项目中使用这些控件，甚至可以继承它们。

10.2 用户控件的应用

用户控件依据性质的不同可以分为两类：一类是组合控件；另一类是自定义控件。

- 组合控件：这种控件是根据现有控件的功能，依据某一种特殊的需要将它们组合在一起，构成一个整体改善控件的界面。它的优点在对于某一种架构的需求量比较大时显得很突出，在这种情况下，不用重复书写相同的代码，节省了大量的资源和不必要的开销。它的缺点是灵活性差，一旦设置了组合控件中的各个属性，在以后的使用过程中就很难再更改。
- 自定义控件：以代码的形式将需要的功能写出来，可以满足用户的绝大多数需求。这种控件的优点是灵活性高，能最大限度地满足用户的需求；它的缺点是对开发者的要求比较高，书写的代码量比较大。

10.2.1 组合控件

下面着重讲解创建组合控件的操作步骤及应用。

1．创建组合控件的操作步骤如下所示。

（1）打开一个 Windows 应用程序。

（2）在菜单栏中选择“项目”→“添加用户控件”，如图 10.1 所示。

（3）弹出“添加新项”对话框，在该对话框中选择“用户控件”选项，并在“名称”文本框中为用户控件命名，如图 10.2 所示。单击“添加”按钮即可在当前项目中添加一个用户控件。

（4）向用户控件设计界面中添加控件，该界面主要用到的控件、控件属性设置及用途如表 10.1 所示，其设计结果如图 10.3 所示。

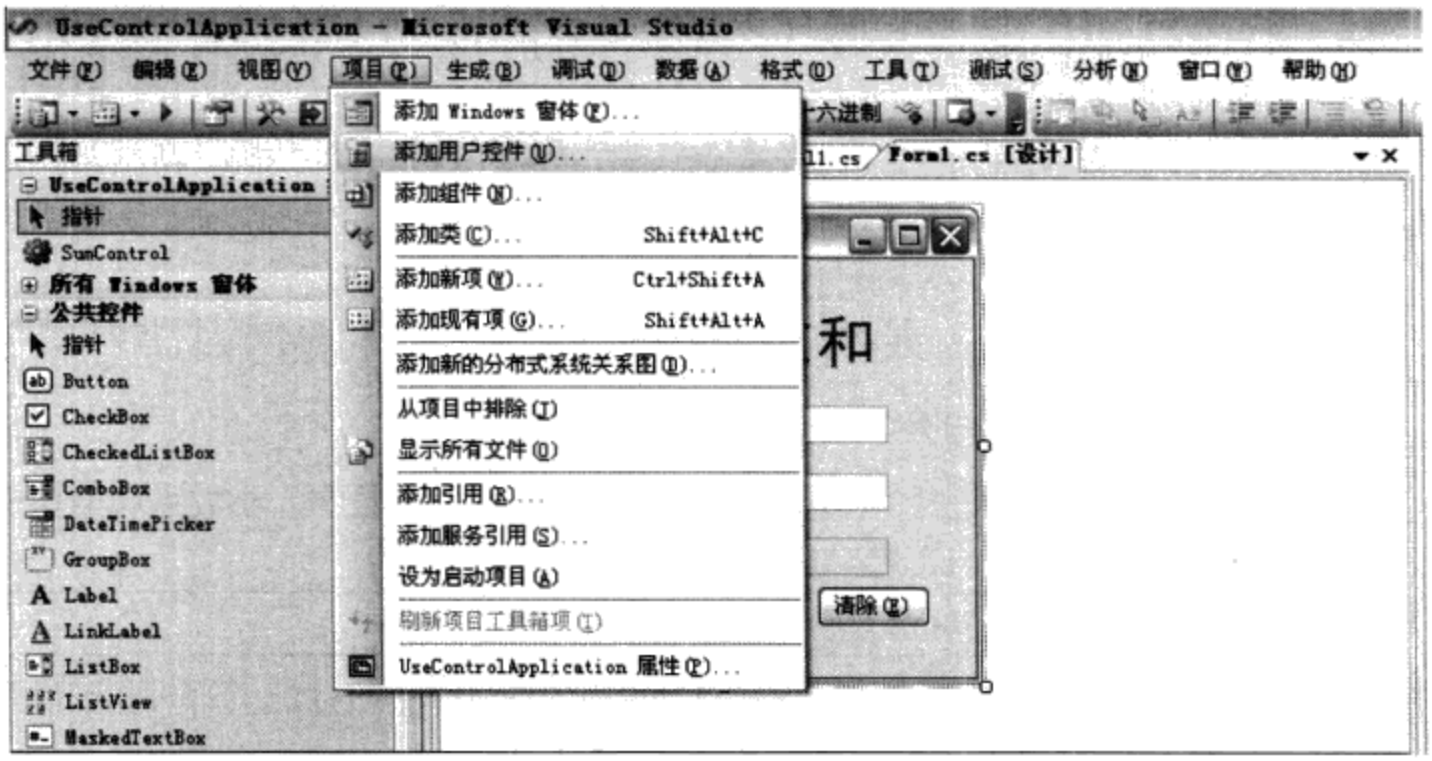


图 10.1 添加项

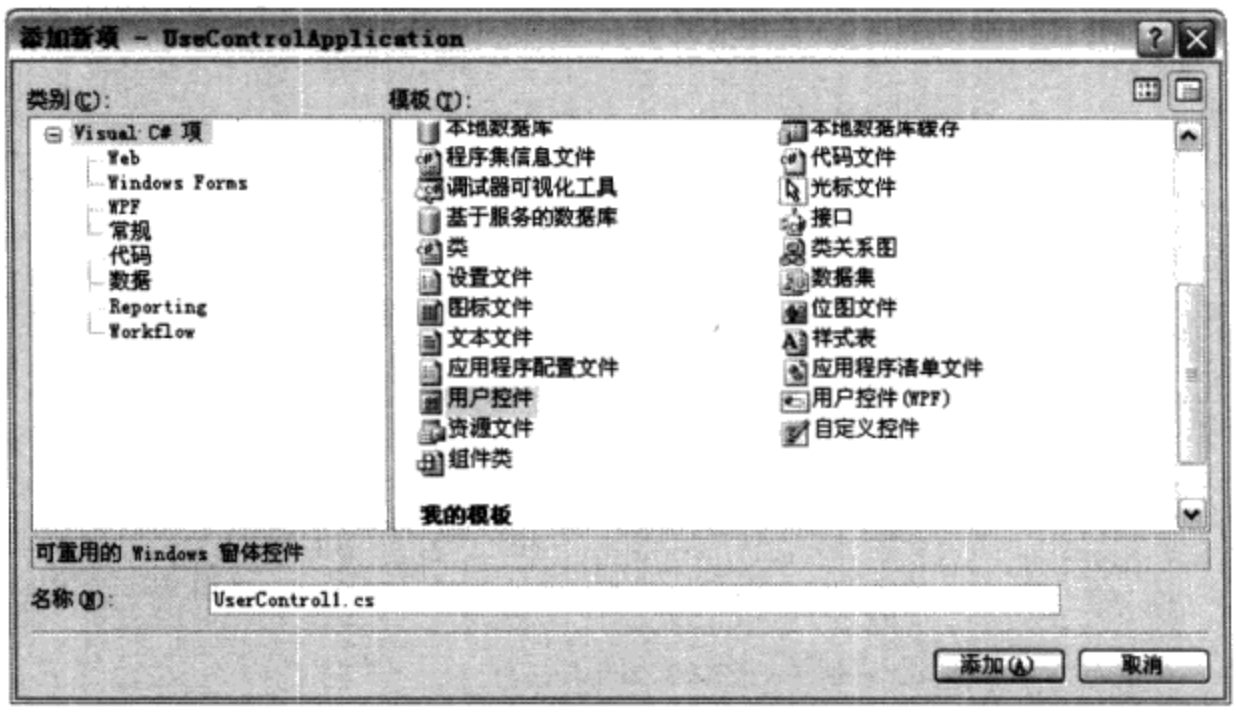


图 10.2 添加用户控件

表 10.1 用户控件主要用到的控件、控件属性设置及用途

控件类型	控件名称	主要属性设置	用途
A Label	Labone	将其 AutoSize 属性设置为“True”	标签“数字一”
	Labsum	将其 AutoSize 属性设置为“True”	标签“合数”
	Labtile	将其 AutoSize 属性设置为“True”	标签“两个数求合”
abl TextBox	Txtone	将其 ReadOnly 属性设置为“False”	输入数字一
	Txttwo	将其 ReadOnly 属性设置为“False”	输入数字二
	Txtsum	将其 ReadOnly 属性设置为“True”	显示合数
ab Button	Bntclose	将其 AutoSize 属性设置为“True”	退出窗体
	Bntsum	将其 AutoSize 属性设置为“True”	求和
	Bntclear	将其 AutoSize 属性设置为“True”	清除文本框

图 10.3　用户控件运行结果

示例 10-01　组合控件的创建

本示例对文本框的输入字符进行判断，判断输入的字符是否为数字、小数点和退格键，如果不是这些字符，示例将会给出与之相对应的提示信息。代码如下：

代码 10-01　光盘位置：光盘\mingrisoft\sl\10\01

```
private void bntSum_Click(object sender, EventArgs e)
{
    if (txtOne.Text == "")                    //当文本框一中的内容为空时
    {
        //弹出加数一不能为空的信息提示
        MessageBox.Show("加数一不能为空", "友情提示", MessageBoxButtons.OK, MessageBoxIcon.Stop);
        this.txtOne.Focus();                  //设置文本框一为当前窗体的焦点
        return;
    }
    if (txtTwo.Text == "")                    //当文本框二中的内容为空时
    {
        //弹出加数二不能为空的信息提示
        MessageBox.Show("加数二不能为空", "友情提示", MessageBoxButtons.OK, MessageBoxIcon.Stop);
        this.txtTwo.Focus();                  //设置文本框二为当前窗体的焦点
        return;
    }
    Decimal i = Convert.ToDecimal(string.Format("{0:G}", this.txtOne.Text));      //实例化一个十进制变量 i
    Decimal j = Convert.ToDecimal(string.Format("{0:G}", this.txtTwo.Text));      //实例化一个十进制变量 j
    Decimal f = i + j;                                                            //计算两个数的和
    this.txtSum.Text = f.ToString();                                              //在结果文本框中显示计算后的内容
}
private void txtOne_KeyPress(object sender, KeyPressEventArgs e)
{
    //判定按下的键不是退格、负号及小数点
    if ((e.KeyChar != 8 && !char.IsDigit(e.KeyChar)) && (e.KeyChar != 46 && e.KeyChar != 45))
    {
        MessageBox.Show("只允许输入数字,退格,小数点,负号", "友情提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);                         //实例化一个提示对话框
        e.Handled = true;                     //设置已经处理过 KeyPress 事件
    }
}
private void bntClose_Click(object sender, EventArgs e)
{
    DialogResult stresce = MessageBox.Show("是否要退出", "友情提示", MessageBoxButtons.YesNo,
MessageBoxIcon.Question);                     //实例化一个提示对话框
    if (stresce.ToString() == "Yes")          //当单击“是”按钮时
    {
        Application.Exit();                   //退出应用程序
    }
}
private void txtTwo_KeyPress(object sender, KeyPressEventArgs e)
{
    //判定按下的键不是退格、负号及小数点
```

```
        if ((e.KeyChar != 8 && !char.IsDigit(e.KeyChar)) && (e.KeyChar != 46 && e.KeyChar != 45))
        {
            MessageBox.Show("只允许输入数字,退格,小数点,负号","友情提示",MessageBoxButtons.OK,MessageBoxIcon.Stop);
        //弹出可视的信息提示
            e.Handled = true;                          //设置已经处理过 KeyPress 事件
        }
    }
    private void bntClear_Click(object sender, EventArgs e)
    {
        this.txtOne.Clear();                           //清空文本框一中的内容
        this.txtSum.Clear();                           //清空结果框中的文本内容
        this.txtTwo.Clear();                           //清空文本框二中的内容
        this.txtOne.Focus();                           //设置当前窗体的焦点在文本框一中
    }
```

2．组合控件的应用

组合控件创建完成之后，需要生成项目，以便使组合控件显示在工具箱中，以供使用，生成步骤如下。

（1）在菜单栏中选择“生成”→“生成 UseControlApplication”选项，编译组合控件。

（2）组合控件成功编译后，新建一个 Windows 窗体，则在“工具箱”窗口中多了一个名为“UseControlApplication 组件”的控件分组，单击“+”，会出现新建组合控件的名字“SumControl”，如图 10.4 所示。

（3）将组合控件添加到窗体中。窗体运行结果如图 10.5 所示。

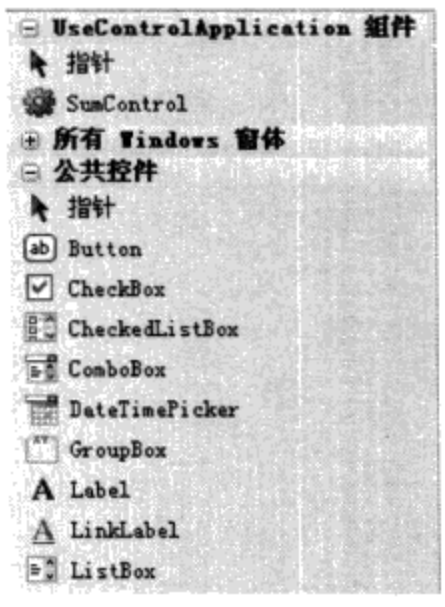


图 10.4　生成组合控件

图 10.5　组合控件运行结果

10.2.2　自定义控件

下面以创建下划线控件为例来讲解怎样创建自定义控件。

自定义控件创建方法的前半部分与组合控件创建方法的前三步类似。下面详细讲解后续部分。

（1）更改自定控件 UnderLineTextBox 的默认继承类 UserControl 为 TextBox 类。在 UnderLineTextBox 控件的构造方法中，定义它的边框样式和大小。代码如下：

```
public UnderLineTextBox()
{
    this.Width = 180;                          //设置控件的宽度
    this.Height = 150;                         //设置控件的高度
    this.BorderStyle = BorderStyle.None;       //设置控件为无边框状态
}
```

（2）由于本示例需要用到获取设备场景函数 GetWindowDC 和释放设备场景函数 ReleaseDC，因此在进行其他操作前需要对它们进行声明。代码如下：

```
public const int WM_PAINT = 0x000F;            //该变量表示绘制 TextBox 控件
public const int WM_CTLCOLOREDIT = 0x0133;     //该变量表示开始编辑 TextBox 控件的颜色
```

```
[DllImport("user32.dll")]
///说明：获取整个窗口的设备场景
///返回值：执行成功则为窗口设备场景，失败则为 0
///参数说明：将获取其设备场景的窗口
///备注：使用后一定要用 ReleaseDC 函数释放场景
public static extern IntPtr GetWindowDC(IntPtr hWnd);
[DllImport("user32.dll")]
///说明：释放由调用 GetDC 或 GetWindowDC 函数获取的指定设备场景，它对类或私有设备场景无效
///返回值：执行成功返回 1，否则，返回 0
///参数说明：
///hWnd：要释放的设备场景相关的窗口句柄
///hDC：要释放的设备场景句柄
public static extern int ReleaseDC(IntPtr hWnd, IntPtr hDC);
```

（3）为了实现下划线效果，需要捕获 Windows 的绘制消息。当进行绘制时，对于本示例而言，只需绘制 TextBox 控件的下边框即可，在本示例中通过调用 DrawBottomLines 方法来实现。对于制作其他类型的用户控件，依据各自的性质进行不同的处理。代码如下：

```
protected override void WndProc(ref Message m)
{
    base.WndProc(ref m);                    //处理消息
    switch (m.Msg)                          //截获有关 TextBox 控件的绘制信息
    {
        case WM_CTLCOLOREDIT:               //当开始编辑 TextBox 控件的颜色时
            goto case WM_PAINT;             //跳转到 TextBox 控件的绘制
        case WM_PAINT:                      //当开始绘制 TextBox 控件时
            IntPtr hDC = GetWindowDC(this.Handle);          //获取当前窗口的设备场景
            if (hDC.ToInt32() != 0)                         //当此场景存在时
            {
                using (Graphics g = Graphics.FromHdc(hDC))  //声明一个 GDI+绘图图面类对象 g
                {
                    DrawBottomLines(g);     //绘制 TextBox 控件的底端横线
                    g.Dispose();            //释放 GDI+绘图图面类对象 g 所占用的资源
                }
            }
            m.Result = IntPtr.Zero;         //指定在当前条件下的返回值
            ReleaseDC(m.HWnd, hDC);         //释放指定设备的场景
            break;
    }
}
public void DrawBottomLines(Graphics g)
{
    Pen p = new Pen(this.BackColor, 2);         //定义一个用于绘制直线和曲线的对象并设定它的颜色和宽度
    g.DrawRectangle(p, 1, 1, this.Width - 2, this.Height - 2);    //绘制由坐标对、宽度和高度指定的矩形
    p = new Pen(Color.FromArgb(0, 0, 0), 1);    //定义一个用于绘制直线和曲线的对象并设定它的颜色和宽度
    g.DrawLine(p, 0, this.Height - 1, this.Width, this.Height - 1);//绘制 TextBox 控件的底端横线
    p.Dispose();                                //释放画笔 p 所占用的资源
}
```

说明：有关制作下划线控件的详细代码见光盘“mingrisoft\sl\10\02”。

运行本示例，效果如图 10.6 所示。

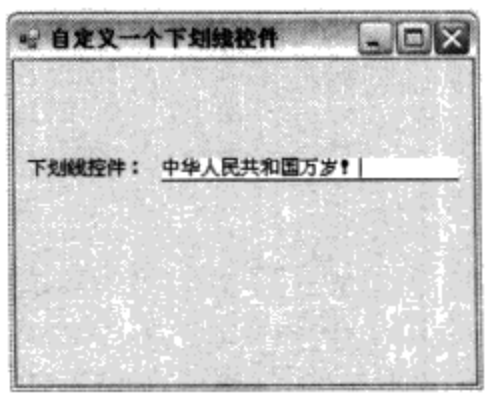


图 10.6　使用下划线控件的运行结果

第 11 章

菜单栏、工具栏和状态栏的设计

- 菜单栏
- 工具栏
- 状态栏

菜单栏、工具栏和状态栏是构成 Windows 应用程序的基本元素，本章将详细介绍它们的用途及设计过程。

11.1 菜 单 栏

11.1.1 菜单栏概念和用途

菜单栏是可视化编程的重要组成部分，是一种方便地给命令分组并访问这些命令的方法，如图 11.1 所示。菜单通常用来显示程序的各项功能，以方便用户选择执行。通过对菜单栏的访问，用户可以快速地进入需要的界面，因此，在开发 Windows 应用程序时，菜单仍然是组织大量选项最常用的方法。

菜单通常分为下拉式菜单和弹出式菜单两种，下面分别对这两种菜单进行介绍。

1．下拉式菜单

下拉式菜单主要由菜单栏、主菜单、子菜单和快捷键组成，如图 11.1 所示。

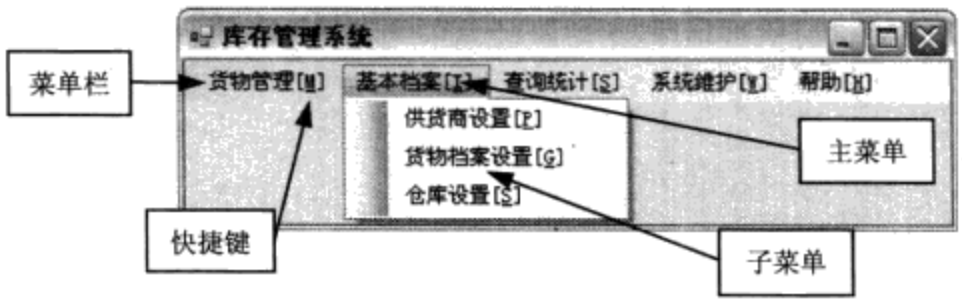


图 11.1 下拉式菜单的组成

- 菜单栏：菜单栏通常都紧挨着窗体的标题栏，它由一个或多个菜单标题构成，菜单栏中包含菜单的所有内容。
- 主菜单：窗体运行时始终显示的菜单，它通常包含一个以上的菜单项。
- 子菜单：主菜单下面的菜单统称为子菜单。
- 快捷键：为了方便程序的操作，可以为每个菜单项设置一个快捷键。用户可以直接通过快捷键执行相应的功能。

2．弹出式菜单

弹出式菜单，也称浮动菜单，其位置是由鼠标所在的位置决定的。弹出式菜单的结构与下拉式菜单基本相同，区别是该菜单不是固定在窗体上面，而是通过单击鼠标右键来显示的。图 11.2 所示为一个弹出式菜单。

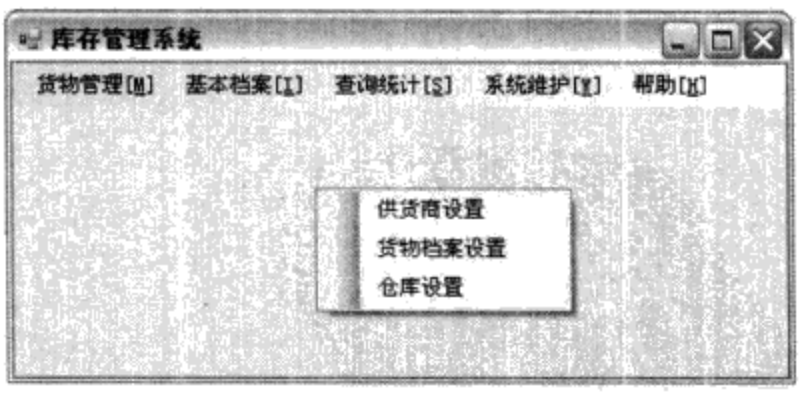


图 11.2 弹出式菜单

11.1.2 菜单栏的设计

1．下拉式菜单设计

在 Windows 应用程序中可以使用 MenuStrip 控件设计下拉式菜单。MenuStrip 控件可以设计成 Microsoft Office 中的菜单样式。使用 MenuStrip 控件设计下拉式菜单的步骤如下。

（1）在 Windows 窗体中添加一个 MenuStrip 控件。

（2）单击该控件 Items 属性中的…按钮，弹出如图 11.3 所示的“项集合编辑器”对话框。

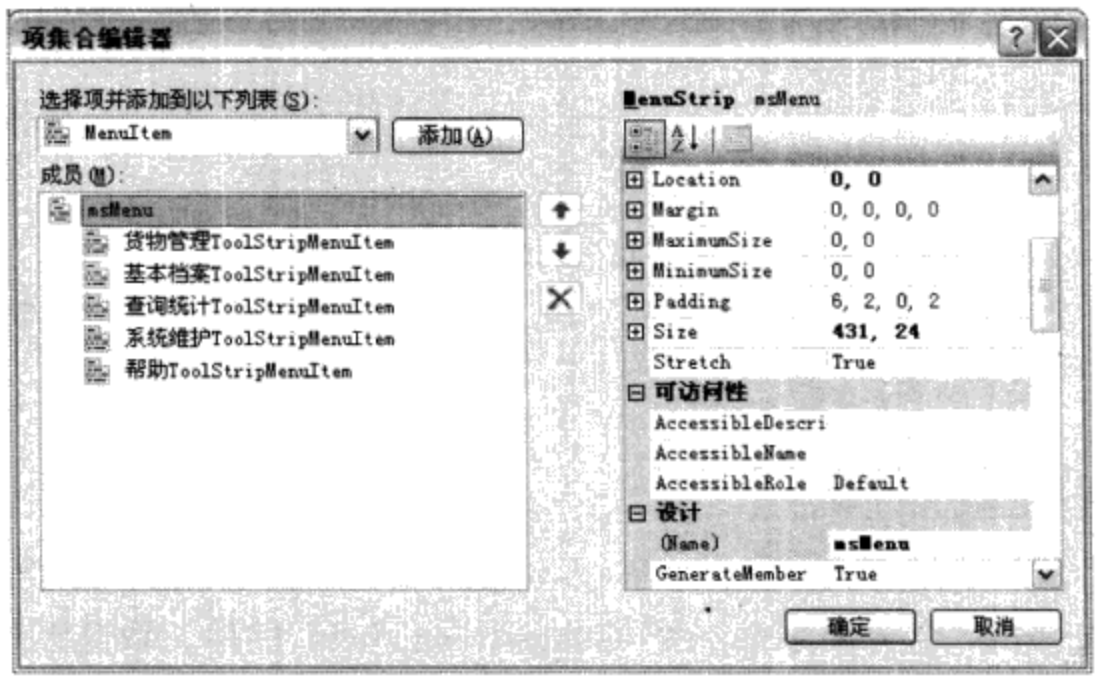


图 11.3 “项集合编辑器”对话框

(3) 在“项集合编辑器”对话框中可以添加 MenuItem 子项。如果要为已经添加的子项添加子菜单，可以选择 MenuItem 子项，单击其 DropDownItems 属性后面的…按钮，弹出如图 11.4 所示的“项集合编辑器”对话框（子项）。

图 11.4 “项集合编辑器”对话框（子项）

(4) 在“项集合编辑器”（子项）对话框中可以给指定子项添加子菜单，依次类推，即可完成下拉式菜单的设计。

(5) 为了更方便地使用菜单，可以给菜单项设置快捷键。选中菜单项，在其 Text 属性中使用样式“[&A]”为指定菜单项设置快捷键，其中，A 可以是任意字母，如图 11.5 所示。

另外，还有一种为菜单设置快捷键的方式，即选中菜单项，在“属性”窗口中通过设置 ShortcutKeys 属性为菜单添加快捷键，如图 11.6 所示。

知识扩展：程序开发人员如果要制作带有分割线的菜单，可以通过在 MenuStrip 控件中添加“Separator”菜单项来实现。

2．弹出式菜单设计

在 Windows 应用程序中设计弹出式菜单时，可以使用 ContextMenuStrip 控件。Windows 应用程序中的窗体和控件都有一个 ContextMenuStrip 属性，该属性用来指定当用户右键单击该窗体或控件时显示的快捷菜单。使用 ContextMenuStrip 控件设计弹出式菜单的步骤如下。

图 11.5　为菜单项设置快捷键

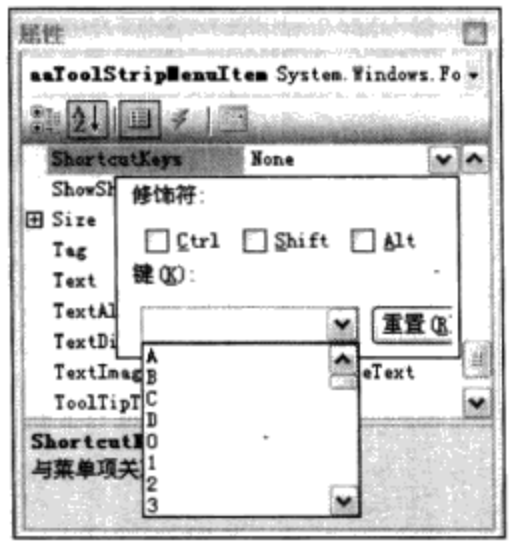


图 11.6　设置菜单项的 ShortcutKeys 属性

（1）在 Windows 窗体中添加一个 ContextMenuStrip 控件，取其默认名字“context MenuStrip1”。

（2）单击该控件 Items 属性后面的…按钮，为其设置菜单项，菜单项的设置方式及快捷键的添加跟下拉式菜单的基本相同，这里不再赘述，其详细设置步骤请参见“下拉式菜单设计”。

（3）选中单击鼠标右键时要显示弹出菜单的窗体或控件，设置其 ContextMenuStrip 属性为“contextMenuStrip1”，运行程序，当用户在窗体或控件上单击右键时，程序便会弹出已经设置的快捷菜单。

11.2　工　具　栏

11.2.1　工具栏概念和用途

工具栏是 Windows 应用程序的一个重要组成部分，它为用户提供了应用程序中常用菜单命令的快速访问方式。工具栏通常情况下位于菜单栏的下方，由许多命令按钮组成，每个按钮上都有一个代表该按钮功能的小图标。由于工具栏直观易用，因此它被广泛用于各种应用软件的主界面当中。

11.2.2　工具栏的设计

在 Windows 应用程序中设计工具栏时，可以使用 ToolStrip 控件或 BindingNavigator 控件，这两种控件在设计工具栏时的步骤大致相同，下面从 ToolStrip 控件为例介绍设计工具栏的方法。

使用 ToolStrip 控件，可以设计像 Windows XP 或 Microsoft Office 那样的工具栏，其步骤如下。

（1）向 Windows 窗体中添加一个 ToolStrip 控件。

（2）单击该控件 Items 属性后面的…按钮，弹出如图 11.7 所示的“项集合编辑器”对话框。

（3）在“项集合编辑器”对话框中，用户可以根据需要添加各种控件，在这里添加 4 个 Button 控件，然后单击“确定”按钮，如图 11.8 所示。

（4）为了使工具栏更加直观易用，在其 Text 属性中为其设置要显示的文本，然后选中工具栏，单击鼠标右键，选择“设置图像”选项，在弹出的“选择资源”对话框中，通过单击“导入”按钮，为其设置要显示的图像。“设置图像”选项和“选择资源”对话框分别如图 11.9 和图 11.10 所示。

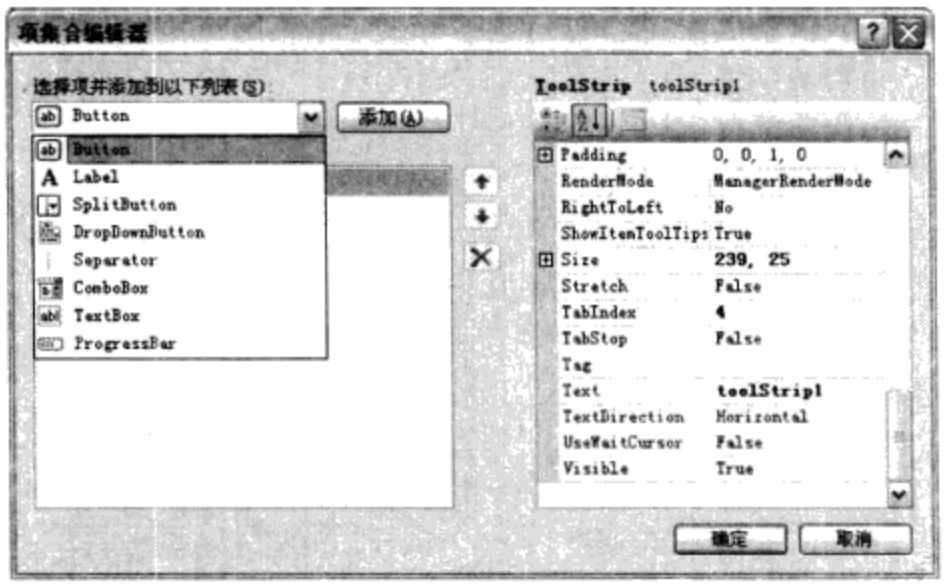


图 11.7 “项集合编辑器”对话框

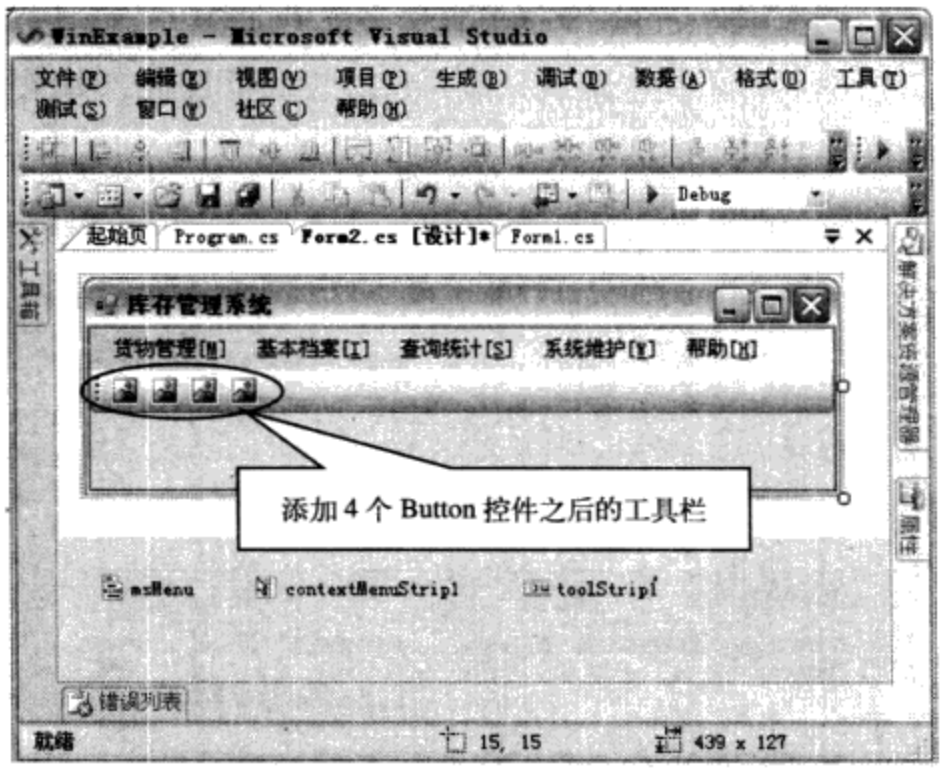


图 11.8 在工具栏中添加控件

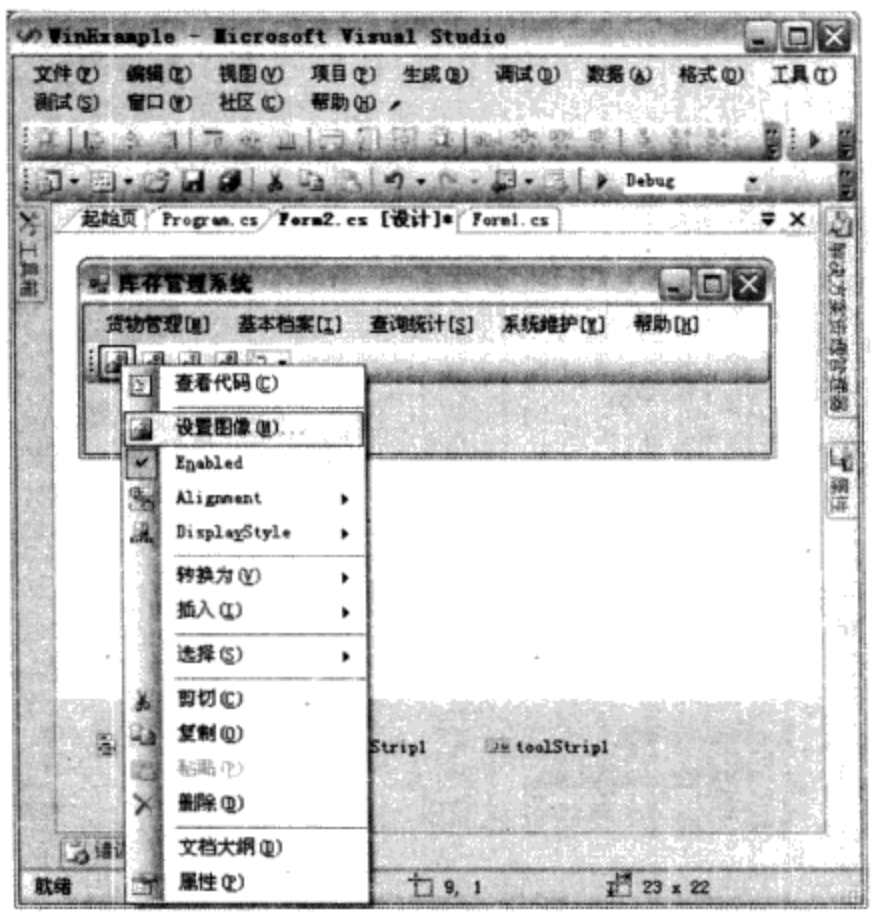


图 11.9 “设置图像”选项

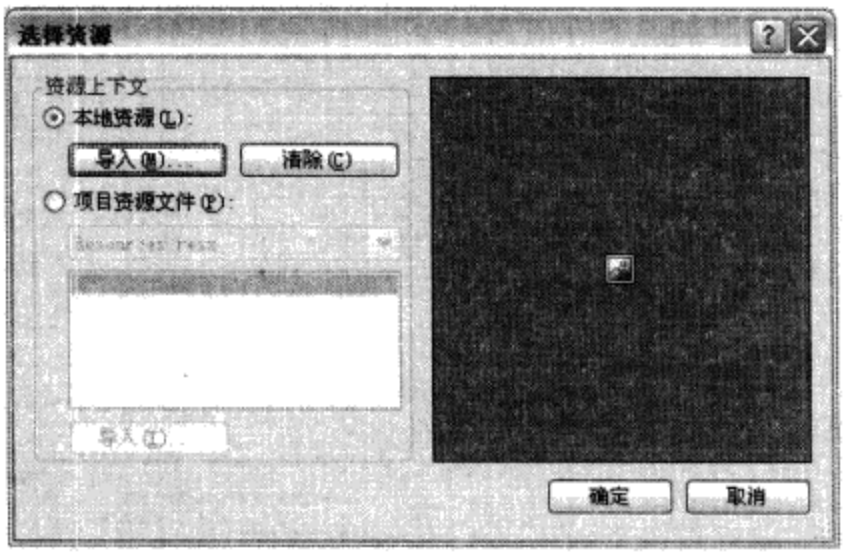


图 11.10 “选择资源”对话框

（5）为工具栏设置图像后，工具栏默认只显示已经设置的图像，如果要同时显示文字和图像，选中工具栏，单击鼠标右键，依次选择“DisplayStyle”→“ImageAndText”选项即可，如图 11.11 所示。

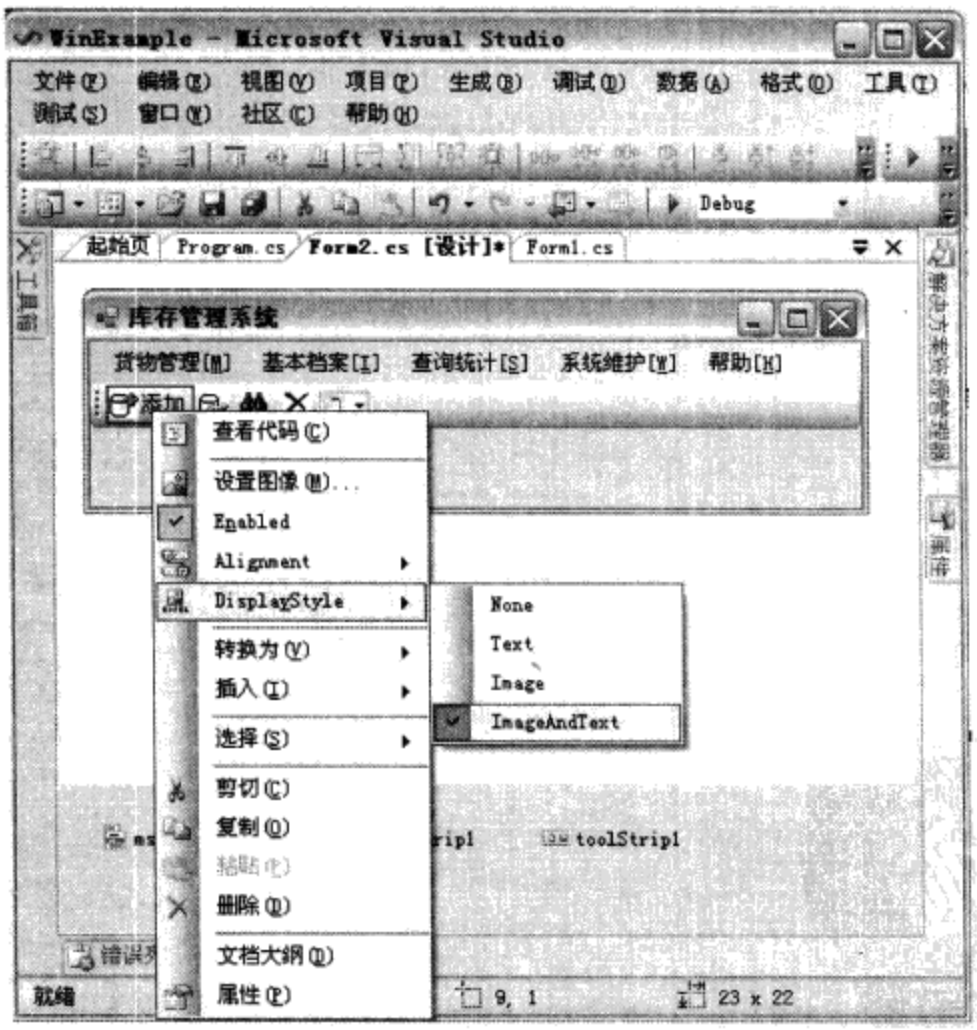


图 11.11 设置工具栏中同时显示文字和图像

（6）工具栏设计完成之后的运行结果如图 11.12 所示。

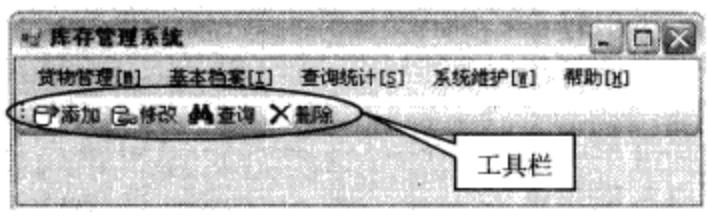


图 11.12 工具栏运行结果

11.3 状 态 栏

11.3.1 状态栏概念和用途

状态栏一般用来显示应用管理系统的系统信息和软件操作员信息，如软件版本号、当前日期与时间和系统操作员等。状态栏在应用软件的设计当中是必不可少的。

11.3.2 状态栏的设计

在 Windows 应用程序中设计状态栏时，可以使用 StatusStrip 控件。StatusStrip 控件可以显示正在操作当前 Windows 窗体的用户的相关信息，或当前系统的一些信息（如系统操作员名称、当前系统时间等）。使用 StatusStrip 控件设计状态栏的步骤如下。

（1）向 Windows 窗体中添加一个 StatusStrip 控件。

（2）单击该控件 Items 属性后面的按钮，弹出如图 11.13 所示的“项集合编辑器”对话框。

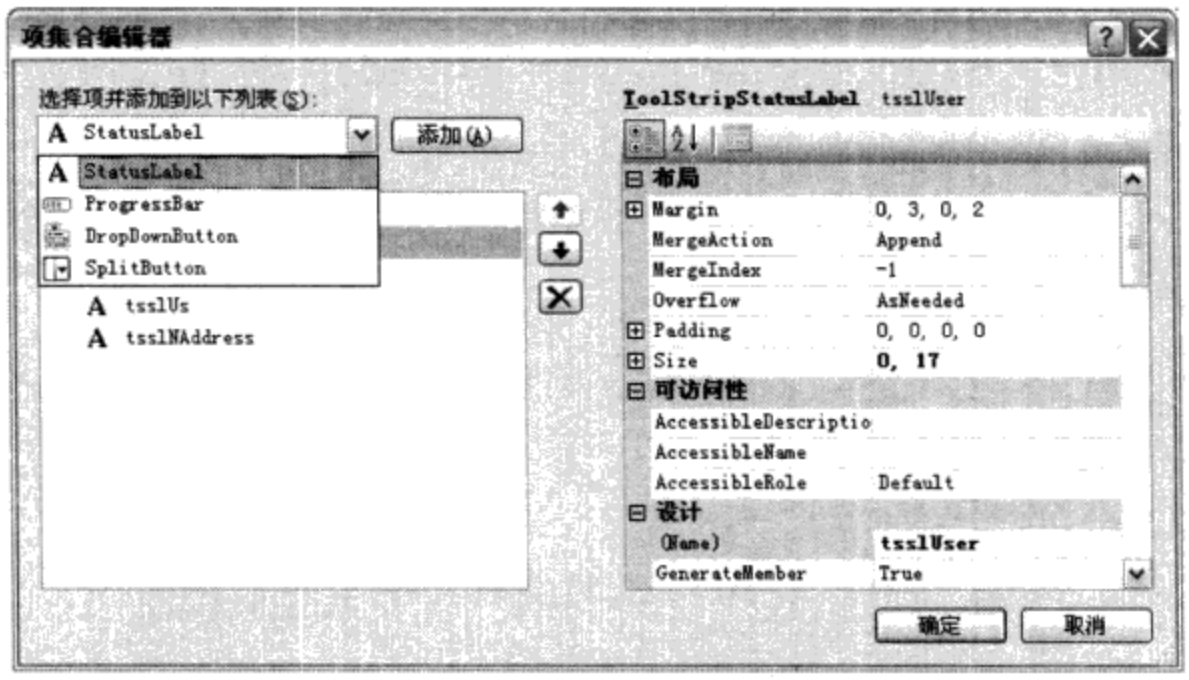


图 11.13 “项集合编辑器”对话框

（3）“项集合编辑器”对话框中，用户可以根据需要选择添加 StatusLabel 控件、ProgressBar 控件、DropDownButton 控件和 SplitButton 控件，单击“确定”按钮，即可完成状态栏的设计。

示例 11-01 状态栏的设计

下面介绍如何设计状态栏。程序开发步骤如下。

（1）在 Visual Studio 2008 开发环境中新建一个 Windows 应用程序，命名为 StatusBarApplication。

（2）在该应用程序中添加一个 Windows 窗体，命名为 Form2，然后在 Form2 窗体上添加一个 StatusStrip 控件，并在其中添加 7 个 StatusLabel 控件，分别用来显示当前用户姓名、登录时间、系统时间和版权归属等信息。

（3）在 Form2 窗体中添加一个 Timer 控件，用来在状态栏中显示系统时间。

代码 11-01 光盘位置：光盘\mingrisoft\sl\11\01

Form1 窗体中主要用来实现登录功能，当用户键入正确的用户名和密码时，即可进入 Form2 窗体中。代码如下：

```
private void btnLogin_Click(object sender, EventArgs e)
{
```

```
        //当在用户名文本框中输入“mr”且在密码文本框中输入“mrsoft”时
        if (txtUName.Text == "mr" && txtPwd.Text == "mrsoft")
        {
            strUName = txtUName.Text;        //保存用户名文本框中的内容
            Form2 frm = new Form2();         //实例化一个 Form2 的实例
            this.Hide();                     //隐藏当前窗体
            frm.Show();                      //显示 Form2 窗体
        }
        else                                 //当输入的用户名或密码不正确时
        {
            //弹出信息提示
            MessageBox.Show("用户名或密码错误！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    private void btnExit_Click(object sender, EventArgs e)
    {
        Application.Exit();                  //退出应用程序
    }
```

当 Form2 窗体成为活动窗体时，在状态栏的各个文本框中显示对应的信息内容，代码如下：

```
    private void Form2_Load(object sender, EventArgs e)
    {
        timer1.Start();                                          //启动计时器
        tsslUName.Text = "当前用户：" + Form1.strUName;            //在文本框中显示当前用户
        tsslLTime.Text = "登录时间：" + DateTime.Now.ToShortTimeString();  //在文本框中显示登录时间
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        tsslSTime.Text = "系统时间：" + DateTime.Now.ToString();   //在文本框中显示系统时间
    }
    private void Form2_FormClosed(object sender, FormClosedEventArgs e)
    {
        this.timer1.Stop();                                      //停止计时器
        Application.Exit();                                      //退出应用程序
    }
```

运行本示例，效果如图 11.4 所示。

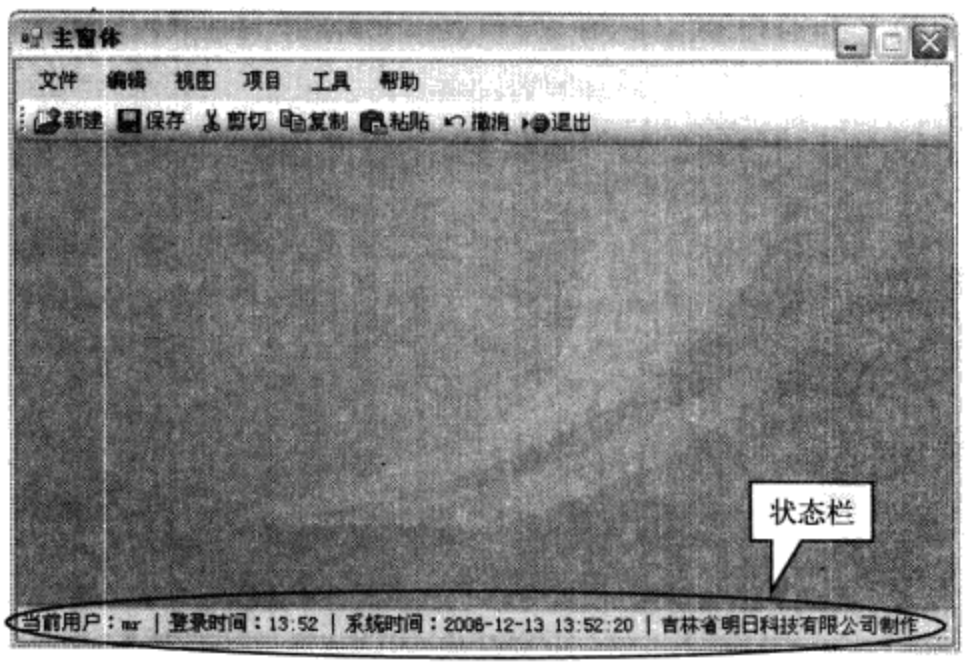


图 11.14　状态栏运行效果

第 12 章

对话框

- “消息“对话框
- “关于“对话框
- “打开“对话框
- “另存为“对话框
- “颜色“对话框
- “字体“对话框
- “打印“对话框
- “页面设置“对话框
- “打印预览“对话框
- “浏览文件夹”对话框

常用的对话框操作（如打开文件、选择字体、保存文件等）都是通过 Windows 提供的标准对话框实现的，在 C#中也可以利用这些对话框来实现相应的功能。

12.1 “消息”对话框

12.1.1 “消息”对话框简介

向用户显示信息和消息的对话框称为消息对话框。例如，在登录窗口中，当输入的账号或者密码有错误时，弹出的消息提示框；当在某些具有限制输入字符类型的文本框中，输入无效的字符时弹出的警告信息等，这些都被称为“消息”对话框。

12.1.2 “消息”对话框的应用

常用的消息对话框是通过 MessageBox 来实现的，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来自用户的信息。

显示消息对话框可以使用下面的语法：

```
MessageBox.Show("[Message]");
```

参数 Message 是要显示给用户的消息，Show 方法用于显示消息框。MessageBox.Show 的重载方法如表 12.1 所示。

表 12.1　　MessageBox.Show 的重载方法

重　载	描　述
MessageBox.Show（String）	显示具有指定文本的消息框
MessageBox.Show（IWin32Window，String）	在指定对象的前面显示具有指定文本的消息框
MessageBox.Show（String，String）	显示具有指定文本和标题的消息框
MessageBox.Show（IWin32Window，String，String）	在指定对象的前面显示具有指定文本和标题的消息框
MessageBox.Show（String，String，MessageBoxButtons）	显示具有指定文本、标题和按钮的消息框
MessageBox.Show（IWin32Window，String，String，Message BoxButtons）	在指定对象的前面显示具有指定文本、标题和按钮的消息框
MessageBox.Show（String，String，MessageBoxButtons，Message BoxIcon）	显示具有指定文本、标题、按钮和图标的消息框
MessageBox.Show（IWin32Window，String，String，Message BoxButtons，MessageBoxIcon）	在指定对象的前面显示具有指定文本、标题、按钮和图标的消息框
MessageBox.Show（String，String，MessageBoxButtons，MessageBoxIcon，　MessageBoxDefaultButton）	显示具有指定文本、标题、按钮、图标和默认按钮的消息框
MessageBox.Show(IWin32Window,String,String,Message BoxButtons,MessageBoxIcon,MessageBoxDefaultButton)	在指定对象的前面显示具有指定文本、标题、按钮、图标和默认按钮的消息框
MessageBox.Show（String，String，MessageBoxButtons，Message BoxIcon，MessageBoxDefaultButton，Message BoxOptions）	显示具有指定文本、标题、按钮、图标、默认按钮和选项的消息框
MessageBox.Show（IWin32Window，String，String，Message BoxButtons，MessageBoxIcon，MessageBoxDefault Button，MessageBoxOptions）	在指定对象的前面显示具有指定文本、标题、按钮、图标、默认按钮和选项的消息框
MessageBox.Show（String，String，MessageBoxButtons，Message BoxIcon,MessageBoxDefaultButton,MessageBox Options，Boolean）	显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（String，String，MessageBoxButtons，Message，BoxIcon，MessageBoxDefaultButton，Message Box Options，String）	使用指定的帮助文件显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框

续表

重　载	描　述
MessageBox.Show（IWin32Window，String，String，Message BoxButtons，MessageBoxIcon，MessageBoxDefault Button，MessageBoxOptions，String）	使用指定的帮助文件显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（String，String，MessageBoxButtons，Message BoxIcon，MessageBoxDefaultButton，MessageBox Options，String，HelpNavigator）	使用指定的帮助文件和HelpNavigator显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（String,String，MessageBoxButtons，Message BoxIcon，MessageBoxDefaultButton，MessageBox Options，String，String）	使用指定的帮助文件和帮助关键字显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（IWin32Window，String，String，Message BoxButtons，MessageBoxIcon，MessageBoxDefault Button，MessageBoxOptions，String，HelpNavigator）	使用指定的帮助文件和HelpNavigator显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show (IWin32Window, String, String, Message BoxButtons，MessageBoxIcon，MessageBoxDefault Button，MessageBoxOptions，String，String）	使用指定的帮助文件和帮助关键字显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（String,String，MessageBoxButtons，Message BoxIcon，MessageBoxDefaultButton，MessageBox Options，String，HelpNavigator，Object）	使用指定的帮助文件、HelpNavigator和帮助主题显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框
MessageBox.Show（IWin32Window，String，String，Message BoxButtons，MessageBoxIcon，MessageBoxDefault Button，MessageBoxOptions，String，HelpNavigator, Object）	使用指定的帮助文件、HelpNavigator和帮助主题显示一个具有指定文本、标题、按钮、图标、默认按钮、选项和“帮助”按钮的消息框

在表 12.1 的重载方法中，Iwin32Window 表示在消息框后面的对象或控件，Message BoxButtons、MessageBoxIcon、MessageBoxDefaultButton 和 MessageBoxOptions 枚举表示在对话框中显示的常量。

MessageBoxButtons 枚举指定若干常数，定义在 MessageBox 对话框中将显示哪些按钮。MessageBoxButtons 枚举值如表 12.2 所示。

表 12.2　　MessageBoxButtons 枚举值

枚 举 值	描　述
AbortRetryIgnore	消息框包含“中止”、“重试”和“忽略”按钮
OK	消息框包含“确定”按钮
OKCancel	消息框包含“确定”和“取消”按钮
RetryCancel	消息框包含“重试”和“取消”按钮
YesNo	消息框包含“是”和“否”按钮
YesNoCancel	消息框包含“是”、“否”和“取消”按钮

DialogResult 也是枚举类型，它的值用于表示对话框返回值，其枚举值包括 Abort、Cancel、Ignore、No、None、Ok、Retry 和 Yes。

示例 12-01　测试 MessageboxButtons 消息对话框的返回值

运行本示例，效果如图 12.1 所示。

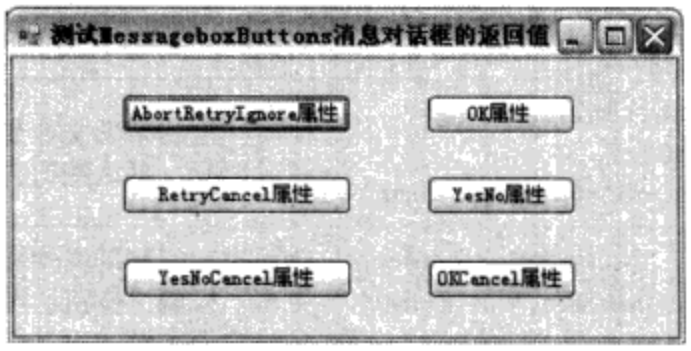


图 12.1　测试 MessageboxButtons 消息对话框的返回值

代码 12-01　光盘位置：光盘\mingrisoft\sl\12\01

运行本示例，单击窗体中的“AbortRetryIgnore 属性”按钮，弹出如图 12.2 所示的消息框，测试 MessageBox Buttons 枚举中的 AbortRetryIgnore 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

```
private void AbortRetryIgnore_Click(object sender, EventArgs e)
{
    DialogResult strAbortRetryIgnore = MessageBox.Show("枚举返回值：\n1.终止\n2.重试\n3.忽略", "友情提示",
MessageBoxButtons.AbortRetryIgnore);                    //实例化一个 DialogResult 对象
    if (strAbortRetryIgnore.ToString() == "Abort")      //当单击“终止”按钮时
    {
        MessageBox.Show("终止", "Abort");               //弹出“终止”信息提示
    }
    if (strAbortRetryIgnore.ToString() == "Retry")      //当单击“重试”按钮时
    {
        MessageBox.Show("重试", "Retry");               //弹出“重试”信息提示
    }
    if (strAbortRetryIgnore.ToString() == "Ignore")     //当单击“忽略”按钮时
    {
        MessageBox.Show("忽略", "Ignore");              //弹出“忽略”信息提示
    }
}
```

单击窗体中的“RetryCancel 属性”按钮，弹出如图 12.3 所示的消息框，测试 MessageBoxButtons 枚举中的 RetryCancel 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

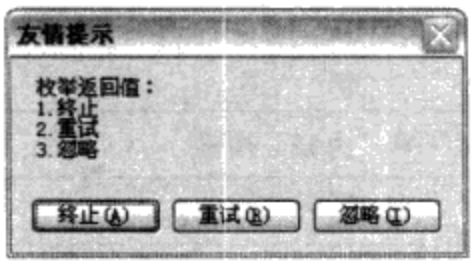


图 12.2　AbortRetryIgnore 成员

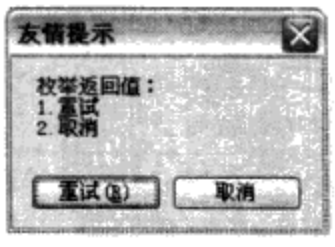


图 12.3　RetryCancel 成员

```
private void RetryCancel_Click(object sender, EventArgs e)
{
    DialogResult strRetryCancel = MessageBox.Show("枚举返回值：\n1.重试\n2.取消", "友情提示", MessageBox
Buttons.RetryCancel);                                   //实例化一个 DialogResult 对象
    if (strRetryCancel.ToString() == "Retry")           //当单击“重试”按钮时
    {
        MessageBox.Show("重试", "Retry");               //弹出“重试”信息提示
    }
    if (strRetryCancel.ToString() == "Cancel")          //当单击“取消”按钮时
    {
        MessageBox.Show("取消", "Cancel");              //弹出“取消”信息提示
    }
}
```

单击窗体中的“YesNoCancel 属性”按钮，弹出如图 12.4 所示的消息框，测试 MessageBoxButtons 枚举中的 YesNoCancel 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

```
private void YesNoCancel_Click(object sender, EventArgs e)
{
    DialogResult strYesNoCancel = MessageBox.Show("枚举返回值：\n1.是\n2.否\n3.取消", "友情提示", Message
BoxButtons.YesNoCancel);                                    //实例化一个 DialogResult 对象
    if (strYesNoCancel.ToString() == "Yes")                 //当单击“是”按钮时
    {
        MessageBox.Show("是", "Yes");                       //弹出“是”信息提示
    }
    if (strYesNoCancel.ToString() == "No")                  //当单击“否”按钮时
    {
        MessageBox.Show("否", "No");                        //弹出“否”信息提示
    }
    if (strYesNoCancel.ToString() == "Cancel")              //当单击“取消”按钮时
    {
        MessageBox.Show("取消", "OK");                      //弹出“取消”信息提示
    }
}
```

单击窗体中的“OK 属性”按钮，弹出如图 12.5 所示的消息框，测试 MessageBoxButtons 枚举中的 OK 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

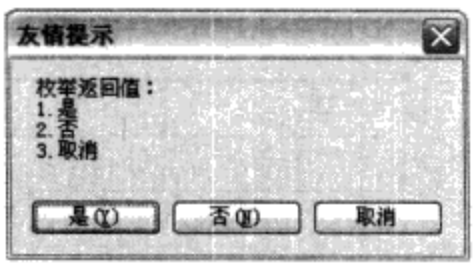


图 12.4　YesNoCancel 成员

图 12.5　OK 成员

```
private void OK_Click(object sender, EventArgs e)
{
    //实例化一个 DialogResult 对象
    DialogResult strOK = MessageBox.Show("枚举返回值：\n 确定", "友情提示", MessageBoxButtons.OK);
    if (strOK.ToString() == "OK")                           //当单击“确定”按钮时
    {
        MessageBox.Show("确定", "OK");                      //弹出“确定”信息提示
    }
}
```

单击窗体中的“YesNo 属性”按钮，弹出如图 12.6 所示的消息框，测试 MessageBoxButtons 枚举中的 YesNo 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

```
private void YesNo_Click(object sender, EventArgs e)
{
    DialogResult strYesNo = MessageBox.Show("枚举返回值：\n1.是\n2.否", "友情提示", MessageBoxButtons.YesNo);
                                                            //实例化一个 DialogResult 对象
    if (strYesNo.ToString() == "Yes")                       //当单击“是”按钮时
    {
        MessageBox.Show("是", "Yes");                       //弹出“是”信息提示
    }
    if (strYesNo.ToString() == "No")                        //当单击“否”按钮时
    {
        MessageBox.Show("否", "No");                        //弹出“否”信息提示
    }
}
```

单击窗体中的“OKCancel 属性”按钮，弹出如图 12.7 所示的消息框，测试 MessageBoxButtons 枚举中的 OKCancel 成员的用法，并用 DialogResult 枚举接收返回值。代码如下：

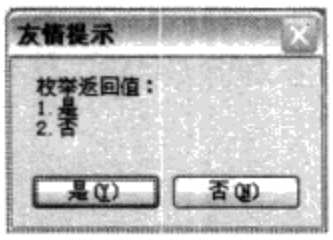


图 12.6　YesNo 成员

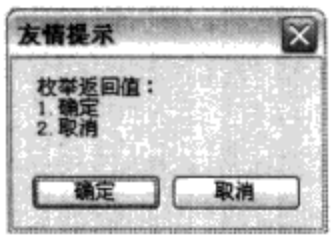


图 12.7　OKCancel 成员

```
    private void OKCancel_Click(object sender, EventArgs e)
    {
        DialogResult strOKCancel = MessageBox.Show("枚举返回值：\n1.确定\n2.取消", "友情提示", MessageBox
Buttons.OKCancel);                                          //实例化一个 DialogResult 对象
        if (strOKCancel.ToString() == "OK")                 //当单击“确定”按钮时
        {
            MessageBox.Show("确定", "OK");                  //弹出“确定”信息提示
        }
        if (strOKCancel.ToString() == "Cancel")             //当单击“取消”按钮时
        {
            MessageBox.Show("取消", "Cancel");              //弹出“取消”信息提示
        }
    }
```

MessageBoxIcon 枚举用来显示一个与提示信息相应的图标。常用的 MessageBoxIcon 枚举值如表 12.3 所示。

表 12.3　　MessageBoxIcon 枚举

枚　举　值	描　　述
Asterisk	该消息框包含一个符号，该符号是由一个圆圈及其中的小写字母 i 组成
Error	该消息框包含一个符号，该符号是由一个红色背景的圆圈及其中的白色 X 组成
Exclamation	该消息框包含一个符号，该符号是由一个黄色背景的三角形及其中的一个感叹号组成
Hand	该消息框包含一个符号，该符号是由一个红色背景的圆圈及其中的白色 X 组成
Information	该消息框包含一个符号，该符号是由一个圆圈及其中的小写字母 i 组成
None	消息框未包含符号
Question	该消息框包含一个符号，该符号是由一个圆圈和其中的一个问号组成
Stop	该消息框包含一个符号，该符号是由一个红色背景的圆圈及其中的白色 X 组成
Warning	该消息框包含一个符号，该符号是由一个黄色背景的三角形及其中的一个感叹号组成

示例 12-02　测试 MessageboxButtons 消息对话框 MessageBoxIcon 枚举值

示例运行效果如图 12.8 所示。

图 12.8　测试 MessageboxButtons 消息对话框 MessageBoxIcon 枚举值

代码 12-02　光盘位置：光盘\mingrisoft\sl\12\02

单击图 12.8 中的“Asterisk 图标”按钮，显示 MessageBoxIcon 枚举的 Asterisk 成员，效果

如图 12.9 所示，代码如下：

```
private void Asterisk_Click(object sender, EventArgs e)
{
    //弹出 Asterisk 样式图标的消息框
    MessageBox.Show("内存不足", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
}
```

单击图 12.8 中的“Exclamation 图标”按钮，显示 MessageBoxIcon 枚举的 Exclamation 成员，效果如图 12.10 所示，代码如下：

图 12.9　Asterisk 图标

图 12.10　OKCancel 成员

```
private void Exclamation_Click(object sender, EventArgs e)
{
    //弹出 Exclamation 样式图标的消息框
    MessageBox.Show("危险品", "有情提示", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
```

单击图 12.8 中的“Information 图标”按钮，显示 MessageBoxIcon 枚举的 Information 成员，效果如图 12.11 所示，代码如下：

```
private void Information_Click(object sender, EventArgs e)
{
    //弹出 Information 样式图标的消息框
    MessageBox.Show("信息提示", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

单击图 12.8 中的“Question 图标”按钮，显示 MessageBoxIcon 枚举的 Question 成员，效果如图 12.12 所示，代码如下：

```
private void Question_Click(object sender, EventArgs e)
{
    //弹出 Question 样式图标的消息框
    MessageBox.Show("是否要格式化 C 盘", "系统提示", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
}
```

图 12.11　Information 图标

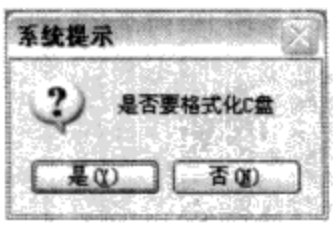


图 12.12　Question 成员

单击图 12.8 中的“Warning 图标”按钮，显示 MessageBoxIcon 枚举的 Warning 成员，效果如图 12.13 所示，代码如下：

```
private void Warning_Click(object sender, EventArgs e)
{
    //弹出 Warning 样式图标的消息框
    MessageBox.Show("请等待", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
```

单击图 12.8 中的“None 图标”按钮，显示 MessageBoxIcon 枚举的 None 成员，效果如图 12.14 所示，代码如下：

```
private void None_Click(object sender, EventArgs e)
{
```

```
        //弹出 None 样式图标的消息框
        MessageBox.Show("有病毒", "系统提示", MessageBoxButtons.YesNo, MessageBoxIcon.None);
    }
```

图 12.13　Warning 图标

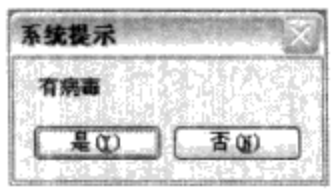


图 12.14　None 成员

单击图 12.8 中的“Error 图标”按钮，显示 MessageBoxIcon 枚举的 Error 成员，效果如图 12.15 所示，代码如下：

```
    private void Error_Click(object sender, EventArgs e)
    {
        //弹出 Error 样式图标的消息框
        MessageBox.Show("系统错误", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
```

单击图 12.8 中的“Stop 图标”按钮，显示 MessageBoxIcon 枚举的 Stop 成员，效果如图 12.16 所示，代码如下：

图 12.15　Error 图标

图 12.16　Stop 成员

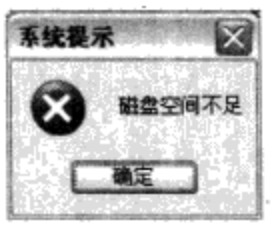


图 12.17　Hand 图标

```
    private void Stop_Click(object sender, EventArgs e)
    {
        //弹出 Stop 样式图标的消息框
        MessageBox.Show("停止操作", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Stop);
    }
```

单击图 12.8 中的“Hand 图标”按钮，显示 MessageBoxIcon 枚举的 Hand 成员，效果如图 12.17 所示，代码如下：

图 12.18　测试 MessageboxButtons 消息对话框中的 MessageBox Default Button 枚举值

```
    private void Hand_Click(object sender, EventArgs e)
    {
        //弹出 Hand 样式图标的消息框
        MessageBox.Show("磁盘空间不足", "系统提示", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    }
```

MessageBoxDefaultButton 枚举指定若干常数，用于定义 MessageBox 上的默认按钮。MessageBoxDefaultButton 枚举值如表 12.4 所示。

表 12.4　　MessageBoxDefaultButton 枚举值

枚　举　值	描　　述
Button1	消息框上的第一个按钮是默认按钮
Button2	消息框上的第二个按钮是默认按钮
Button3	消息框上的第三个按钮是默认按钮

示例 12-03 测试 MessageboxButtons 消息对话框中的 MessageBox Default Button 枚举值

示例运行，效果如图 12.18 所示。

代码 12-03 光盘位置：光盘\mingrisoft\sl\12\03

单击图 12.18 中的“Button1 按钮”按钮，程序弹出带有 3 个按钮的对话框，其中第一个按钮（“终止”按钮）为默认选中的按钮，效果如图 12.19 所示，代码如下：

```
private void Button1_Click(object sender, EventArgs e)
{
    //实例化一个 DialogResult 对象
    DialogResult button1 = MessageBox.Show("默认为第一个按钮", "Button1", MessageBoxButtons.AbortRetryIgnore,
MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button1);
    if (button1.ToString() == "Abort")      //当单击“终止”按钮时
    {
        MessageBox.Show("你单击的是：\n 第一个按钮", "友情提示"); //弹出单击的是第一个按钮的信息提示
    }
    if (button1.ToString() == "Retry")//当单击“重试”按钮时
    {
        MessageBox.Show("你单击的是：\n 第二个按钮", "友情提示"); //弹出单击的是第二个按钮的信息提示
    }
    if (button1.ToString() == "Ignore")      //当单击“忽略”按钮时
    {
        MessageBox.Show("你单击的是：\n 第三个按钮", "友情提示"); //弹出单击的是第三个按钮的信息提示
    }
}
```

单击图 12.18 中的“Button2 按钮”按钮，弹出带有 3 个按钮的对话框，其中第二个按钮（“重试”按钮）为默认选中的按钮，效果如图 12.20 所示。

单击图 12.18 中的“Button3 按钮”按钮，弹出带有 3 个按钮的对话框，其中第三个按钮（“忽略”按钮）为默认选中的按钮，效果如图 12.21 所示。

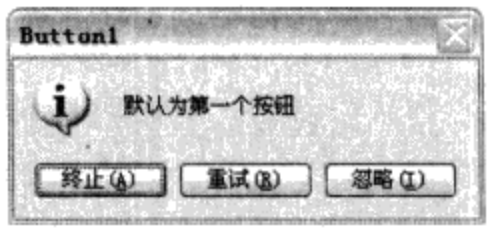


图 12.19 第一个按钮

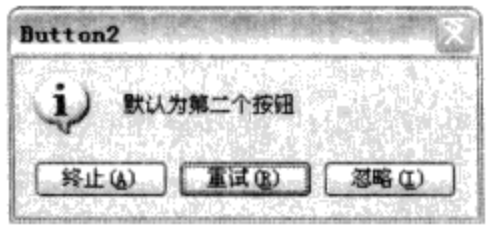


图 12.20 第二个按钮

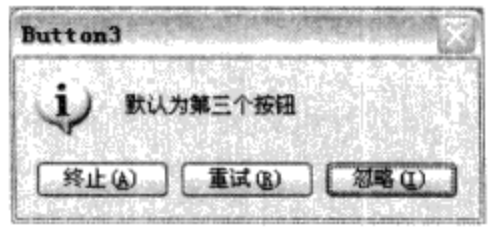


图 12.21 第三个按钮

说明：默认效果的不同仅与 MessageBoxDefaultButton 的枚举值有关，可以通过对该枚举值进行设置从而实现不同的效果。

MessageBoxOptions 枚举用来指定 MessageBox 对话框中的选项，其枚举值如表 12.5 所示。

表 12.5 MessageBoxOptions 枚举值

成员名称	描述
DefaultDesktopOnly	使消息框显示在活动桌面上，此参数与 ServiceNotification 相同
RightAlign	消息框文本右对齐
RtlReading	指定消息框文本按从右到左的阅读顺序显示
ServiceNotification	消息框显示在活动桌面上调用方是一种服务，用于将事件通知用户。即使没有用户登录到计算机，该功能也会在当前活动桌面上显示一个消息框

示例 12-04 测试 MessageboxButtons 消息对话框中的 MessageBoxOptions 枚举值

本示例主要用来测试 MessageboxButtons 消息对话框中的 MessageBoxOptions 枚举值。示例

运行效果如图 12.22 所示。

图 12.22　测试 MessageboxButtons 消息对话框中的 MessageBoxOptions 枚举值

代码 12-04　光盘位置：光盘\mingrisoft\sl\12\04

单击图 12.22 中的“DefaultDesktopOnly 属性”按钮，弹出如图 12.23 所示的消息框。代码如下：

```
private void DefaultDesktopOnly_Click(object sender, EventArgs e)
{
    //弹出 DefaultDesktopOnly 类型的对话框
    MessageBox.Show("DefaultDesktopOnly 成员", "MessageBoxOptions 枚举", MessageBoxButtons.YesNo, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button1, MessageBoxOptions.DefaultDesktopOnly);
}
```

单击图 12.22 中的“ServiceNotification 属性”按钮，弹出如图 12.24 所示的消息框。代码如下：

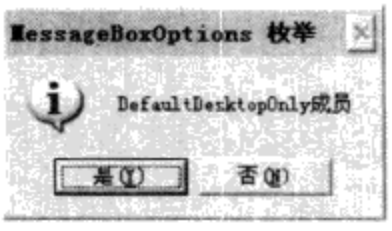


图 12.23　DefaultDesktopOnly 属性

图 12.24　ServiceNotification 属性

```
private void ServiceNotification_Click(object sender, EventArgs e)
{
    //弹出 ServiceNotification 类型的对话框
    MessageBox.Show("ServiceNotification 成员", "MessageBoxOptions 枚举", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button3, MessageBoxOptions.ServiceNotification);
}
```

单击图 12.22 中的“RightAlign 属性”按钮，弹出如图 12.25 所示的消息框。代码如下：

```
private void RightAlign_Click(object sender, EventArgs e)
{
    //弹出 RightAlign 类型的对话框
    MessageBox.Show("RightAlign 成员", "MessageBoxOptions 枚举", MessageBoxButtons.OKCancel, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button2, MessageBoxOptions.RightAlign);
}
```

单击图 12.22 中的“RtlReading 属性”按钮，弹出如图 12.26 所示的消息框。代码如下：

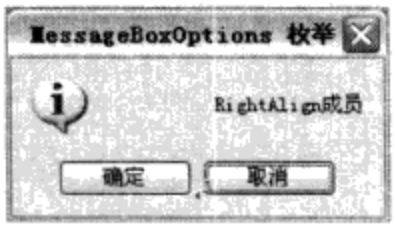


图 12.25　RightAlign 属性

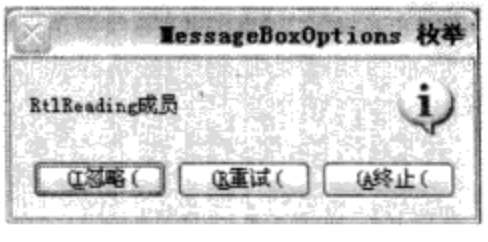


图 12.26　RtlReading 属性

```
private void RtlReading_Click(object sender, EventArgs e)
{
    //弹出 RtlReading 类型的对话框
    MessageBox.Show("RtlReading 成员", "MessageBoxOptions 枚举", MessageBoxButtons.AbortRetryIgnore, MessageBoxIcon.Asterisk, MessageBoxDefaultButton.Button3, MessageBoxOptions.RtlReading);
}
```

12.2 “关于”对话框

12.2.1 “关于”对话框简介

“关于”对话框一般用来介绍一款软件的相关信息，例如版本号、版权以及作者等信息。通常情况下，微软公司的产品中，调用该对话框的项位于“帮助”菜单下。图 12.27 所示为 SQL Server 2005 的“关于”对话框。

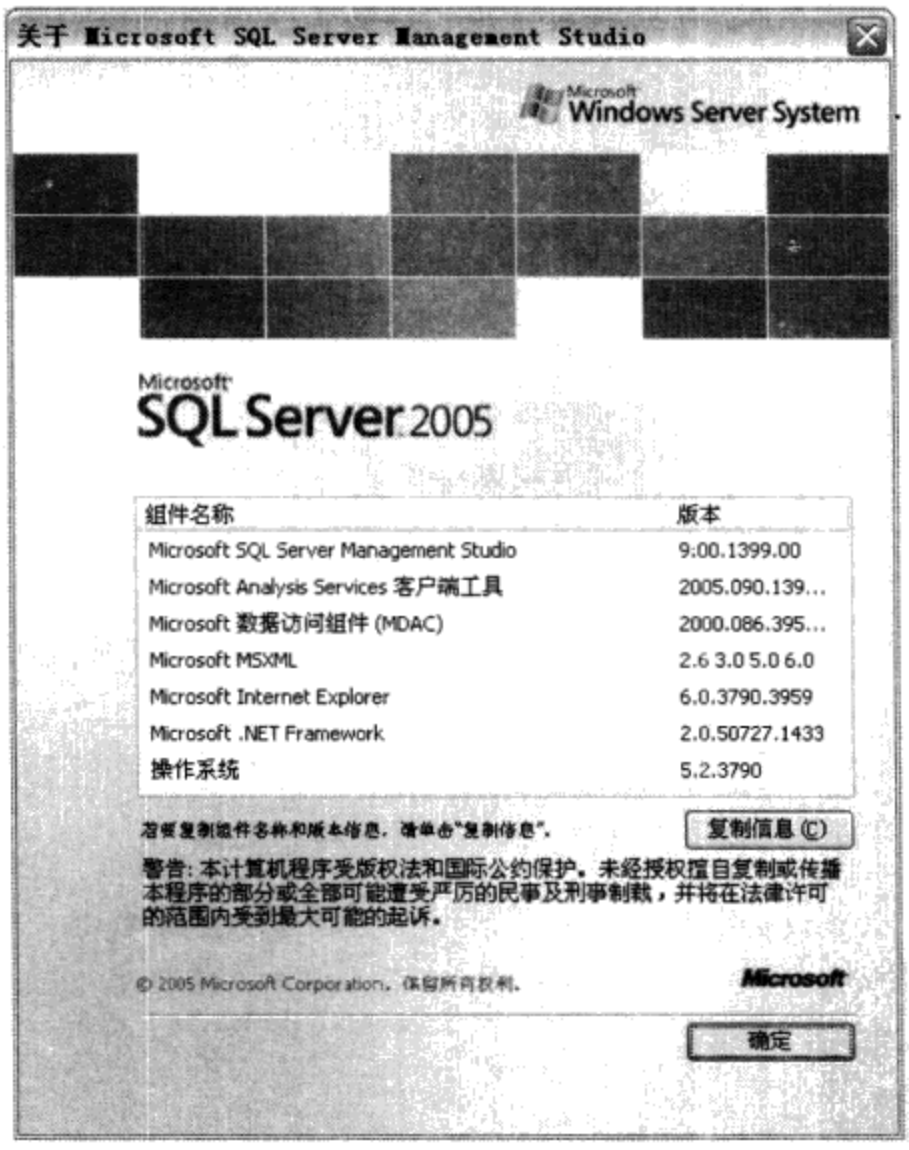


图 12.27　SQL Server 2005 的“关于”对话框

12.2.2 “关于”对话框的应用

创建“关于”对话框的步骤如下所示。

(1) 打开一个 Windows 应用程序。

(2) 在菜单栏中选择“项目”→“添加新项”命令，如图 12.28 所示。

(3) 在弹出的“添加新项”对话框中选择“‘关于’框”选项，并将其命名为 frmAboutbox.cs，如图 12.29 所示。

(4) 单击“添加”按钮，在当前项目中出现一个系统自带的关于窗体，如图 12.30 所示。

图 12.28 添加项

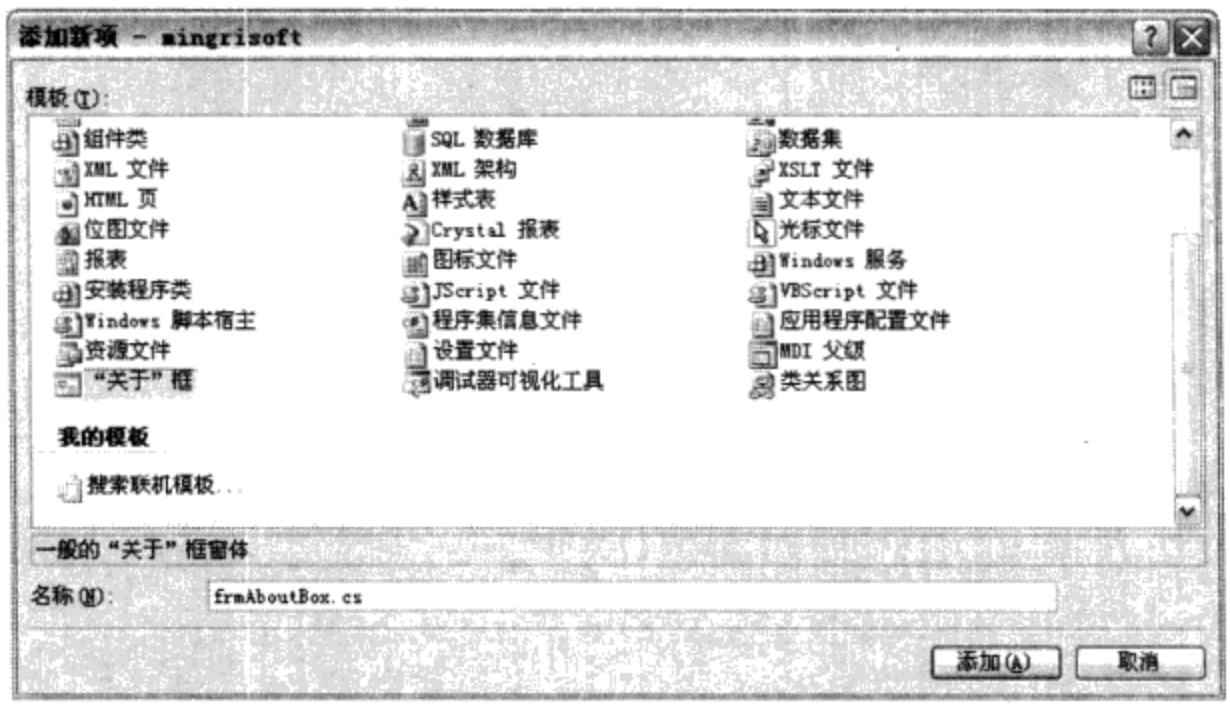


图 12.29 创建"关于"对话框

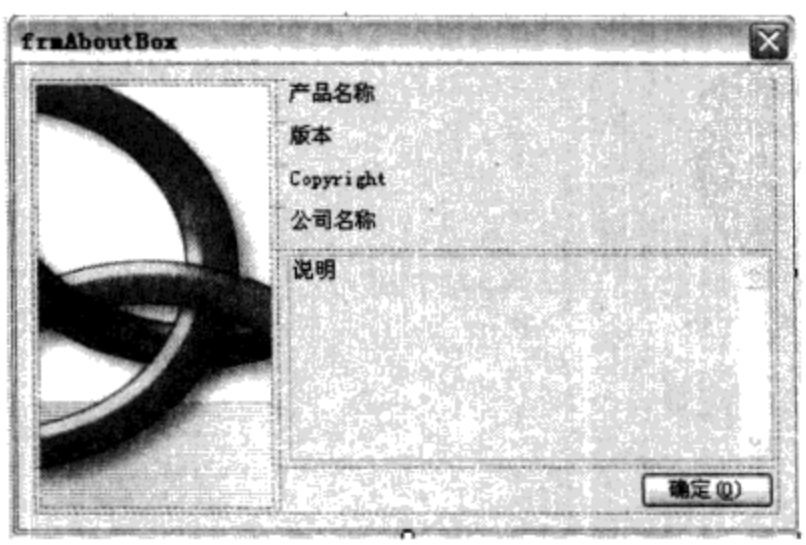


图 12.30 关于对话框"关于"对话框界面

示例 12-05 自定义"关于"对话框

本示例通过添加"关于"对话框来显示程序的一些常用信息，运行结果如图 12.31 所示。

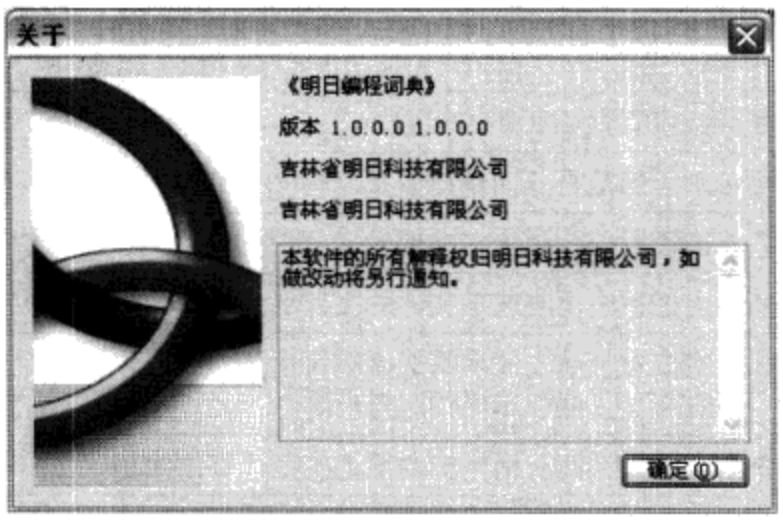


图 12.31 “关于”对话框

代码如下：

代码 12-05 光盘位置：光盘\mingrisoft\sl\12\05

```
private void AboutBox1_Load(object sender, EventArgs e)
{
    this.Text = "关于";                                          //设置标题栏文本
    this.labelCompanyName.Text = "吉林省明日科技有限公司";       //设置公司名称
    this.labelCopyright.Text = "吉林省明日科技有限公司";         //设置版权
    //设置说明文字
    this.textBoxDescription.Text = "本软件的所有解释权归明日科技有限公司，如做改动将另行通知。";
    this.labelProductName.Text = "《明日编程词典》";             //设置产品名称
}
private void okButton_Click(object sender, EventArgs e)
{
    Application.Exit();                                          //退出应用程序
}
```

12.3 “打开”对话框

12.3.1 “打开”对话框简介

“打开”对话框通常用于打开某一指定类型的文件，文件的类型可以通过相关属性的设置实现。例如，当用户在上传文件时，系统都会提供一个选择文件位置的按钮和一个显示文件路径的文本框以及一个选择打开文件类型的下拉框。

12.3.2 “打开”对话框的应用

“打开”对话框的实现主要是通过 OpenFileDialog 控件来实现，下面对该控件进行详细说明。

1．功能

OpenFileDialog 控件主要用于打开文件，图 12.32 所示为 Windows 应用程序中的 OpenFileDialog 控件。

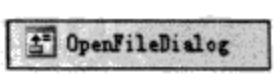


图 12.32 OpenFileDialog 控件

2．属性

OpenFileDialog 控件常用属性及说明如表 12.6 所示。

表 12.6 **OpenFileDialog 控件常用属性及说明**

属　性	说　明
AddExtension	指示如果用户省略扩展名，对话框是否自动在文件名后添加扩展名
CheckFileExists	指示如果用户指定不存在的文件名，对话框是否显示警告
CheckPathExists	指示如果用户指定不存在的路径，对话框是否显示警告

续表

属　性	说　明
DefaultEtxt	获取或设置默认文件扩展名
DereferenceLinks	指示对话框是否返回快捷方式引用文件的位置，或者是否返回快捷方式（.lnk）的位置
FileName	获取或设置一个包含在文件对话框中选定的文件名的字符串
FileNames	获取对话框中所有选定文件的文件名
Filter	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”框中出现的选择内容
FilterIndex	获取或设置文件对话框中当前选定筛选器的索引
InitialDirectory	获取或设置文件对话框显示的初始目录
Multiselect	获取或设置一个值，该值指示对话框是否允许选择多个文件
ReadOnlyChecked	获取或设置一个值，该值指示是否选定只读复选框
RestoreDirectory	获取或设置一个值，该值指示对话框在关闭前是否还原当前目录
ShowHelp	获取或设置一个值，该值指示文件对话框中是否显示“帮助”按钮
ShowReadOnly	获取或设置一个值，该值指示对话框是否包含只读复选框
Title	获取或设置文件对话框标题
ValidateNames	获取或设置一个值，该值指示对话框是否只接收有效的 Win32 文件名

下面对 OpenFileDialog 控件比较重要的属性进行详细介绍。

（1）InitialDirectory 属性。该属性用于获取或设置文件对话框显示的初始目录。

语法：

```
object. InitialDirectory[=string]
```

属性值：object 表示对象表达式，string 表示初始文件的目录。

（2）Filter 属性。该属性用于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”框中出现的选择内容。

语法：

```
object.Filter [=string]
```

属性值：object 表示对象表达式，string 表示文件的后缀。

例如，设置“打开”对话框只能打开相关的图片文件（.bmp、.gif、.jpg）。代码如下：

```
penFileDialog1.Filter = "bmp 文件(*.bmp)|*.bmp|gif 文件(*.gif)|*.gif|jpg 文件(*.jpg)|*.jpg";
```

（3）RestoreDirectory 属性。该属性用于获取或设置一个值，该值指示对话框在关闭前是否还原当前目录。

语法：

```
object.RestoreDirectory [=boolean]
```

属性值：object 表示对象表达式，boolean 表示一个布尔表达式，指示对话框在关闭前是否还原当前目录，当设置值为 true 时，对话框在关闭前还原当前目录；当设置值为 False 时，对话框在关闭前不还原当前目录。

（4）Multiselect 属性。该属性表示获取或设置一个值，该值指示对话框是否允许选择多个文件。

语法：

```
public bool Multiselect { get; set; }
```

属性值：如果对话框允许同时选定多个文件，则为 true；反之，则为 false。默认值为 false。

3．方法

OpenFileDialog 控件常用方法及说明如表 12.7 所示。

表 12.7　　OpenFileDialog 控件常用方法及说明

方　法	说　明
OpenFile	此方法以只读模式打开用户选择的文件
ShowDialog	此方法显示 OpenFileDialog

下面对 OpenFileDialog 控件的 ShowDialog 方法进行详细介绍。该方法用于运行通用对话框。

语法：

```
object. ShowDialog()
```

参数说明如下。

object：代表对象表达式。

4．事件

OpenFileDialog 控件常用事件及说明如表 12.8 所示。

表 12.8　　OpenFileDialog 控件常用事件及说明

事　件	描　述
FileOk	当用户单击对话框中“打开”或“保存”按钮时触发
HelpRequest	当用户单击对话框中的“帮助”按钮时触发

下面对 OpenFileDialog 控件的 HelpRequest 事件进行说明，该事件表示当用户单击对话框中的“帮助”按钮时触发。

语法：

```
public event EventHandler HelpRequest
```

示例 12-06　OpenFileDialog 控件的应用

本示例通过对上传文件的模拟来实现 OpenFileDialog 控件的应用，代码如下：

代码 12-06　光盘位置：光盘\mingrisoft\sl\12\06

```
private void Form1_Load(object sender, EventArgs e)
{
    this.openFileDialog1.CheckFileExists = true;            //当指定的文件不存在时显示警告信息
    this.openFileDialog1.CheckPathExists = true;            //当指定的路径不存在时显示警告信息
    this.openFileDialog1.Multiselect = true;                //指示允许单次选定多个文件
    this.openFileDialog1.ReadOnlyChecked = true;            //指示选定只读复选框
    this.openFileDialog1.ShowHelp = true;                   //在通用对话框中显示“帮助”按钮
    this.openFileDialog1.Filter = "所有文件（*.*）|*.*";     //设置打开文件的类型
}
private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
{
    this.textBox1.Text = this.openFileDialog1.FileName;     //在文本框中显示相对应的文件名
}
private void openFileDialog1_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show("如需帮助请登录下面的网站查询相关信息:\nhttp://www.mingrisoft.com", "友情提示", Message
BoxButtons.OK, MessageBoxIcon.Information);             //弹出友情提示
}
private void UpLoad_Click(object sender, EventArgs e)
{
    this.openFileDialog1.ShowDialog();                      //显示通用对话框
}
```

运行本示例，效果如图 12.33 所示。

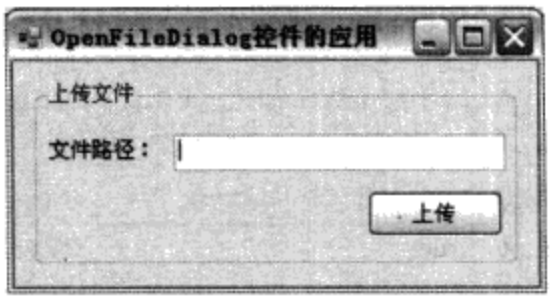


图 12.33　OpenFileDialog 控件的应用

12.4　“另存为”对话框

12.4.1　“另存为”对话框简介

“另存为”对话框通常用于对当前文件的保存以及改变当前文件的类型。它的外观同 OpenFileDialog 控件打开的对话框类似，仅在按钮上有些不同。

12.4.2　“另存为”对话框的应用

“另存为”对话框是通过 SaveFileDialog 控件来实现的，下面对该控件进行详细的讲解。

1．功能

SaveFileDialog 控件显示一个预先配置的对话框，用户可以使用该对话框将文件保存到指定的位置。图 12.34 所示为 Windows 应用程序中的 SaveFileDialog 控件。

2．属性

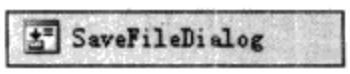


图 12.34　SaveFileDialog 控件

SaveFileDialog 控件继承了 OpenFileDialog 控件的大多数属性、方法和事件。

下面对 SaveFileDialog 控件的其他属性进行详细介绍。

（1）CreatePrompt 属性。该属性用于获取或设置一个值，该值指示如果用户指定不存在的文件，对话框提示用户是否允许创建该文件。

语法：

```
object. CreatePromp [= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，该值指示如果用户指定不存在的文件，对话框是否提示用户允许创建该文件。当值为 true 时，可以创建该文件；当值为 false 时，不创建该文件。

（2）OverwritePrompt 属性。该属性表示获取或设置一个值，该值指示如果用户指定的文件名已存在，另存为对话框是否显示警告。

语法：

```
object. OverwritePrompt[= boolean]
```

3．方法

OpenFile 方法用于打开用户选定的具有读/写权限的文件。

语法：

```
object. OpenFile()
```

示例 12-07　SaveFileDialog 控件的应用

本示例通过对 SaveFileDialog 控件属性、方法以及事件的应用实现对文件的另存为操作，代码如下：

代码 12-07　光盘位置：光盘\mingrisoft\sl\12\07

```
private void Form1_Load(object sender, EventArgs e)
{
    this.richTextBox1.Focus();                                  //设置当前窗体的焦点在 RichTextBox 控件中
    this.saveFileDialog1.ShowHelp = true;                       //设置在窗体中显示“帮助”按钮
    this.saveFileDialog1.Filter = "Word 文档(*.doc)|*.doc";     //设置保存文件的类型
    this.saveFileDialog1.CreatePrompt = true;                   //设置当文件不存在时对文件进行创建
}
private void 另存为 AToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.saveFileDialog1.ShowDialog();                          //显示另存为对话框
}
private void saveFileDialog1_FileOk(object sender, CancelEventArgs e)
{
    this.richTextBox1.SaveFile(this.saveFileDialog1.FileName);  //保存当前文本框中的内容
    this.richTextBox1.Clear();                                  //清空文本框中的原有内容
}
private void saveFileDialog1_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show("如有疑问，欢迎登录下面的网站进行查询：\nhttp://www.mingrisoft.com", "友情提示",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);                 //弹出信息提示
}
```

运行本示例，效果如图 12.35 所示。

图 12.35　SaveFileDialog 控件的应用

12.5 “颜色”对话框

12.5.1 “颜色”对话框简介

“颜色”对话框通常情况下用于改变选定对象的颜色，例如，常用的 Word 软件中，当需要改变字体的颜色时，首先选定将要修改的文字，然后选择“格式”菜单下的“字体”子菜单，在“字体颜色”选项下选择需要使用的颜色。

12.5.2 “颜色”对话框的应用

“颜色”对话框的调用是通过 ColorDialog 控件实现的，下面详细介绍该控件。

1．功能

ColorDialog 控件用于选择颜色，它允许用户从调色板选择颜色或自定义颜色。图 12.36 所示为 Windows 应用程序中的 ColorDialog 控件。

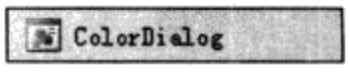


图 12.36　ColorDialog 控件

2．属性

ColorDialog 控件常用属性及说明如表 12.9 所示。

表 12.9　　ColorDialog 控件常用属性及说明

属　性	说　明
AllowFullOpen	获取或设置一个值，该值指示用户是否可以使用该对话框自定义颜色
AnyColor	获取或设置一个值，该值指示对话框是否显示基本颜色集
Color	获取或设置用户选定的颜色
CustomColors	获取或设置对话框中显示的自定义颜色集
FullOpen	获取或设置一个值，该值指示用于创建自定义颜色的控件在对话框打开时是否可见
ShowHelp	获取或设置一个值，该值指示在颜色对话框中是否显示“帮助”按钮
SolidColorOnly	获取或设置一个值，该值指示对话框是否限制用户只选择纯色

下面对 ColorDialog 控件比较重要的属性进行详细介绍。

（1）AllowFullOpen 属性。该属性表示获取或设置一个值，该值指示用户是否可以使用该对话框定义自定义颜色。

语法：

```
object. AllowFullOpen [= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，指示用户是否可以使用该对话框定义自定义颜色。当值为 true 时，用户可以使用该对话框定义自定义颜色；当值为 false 时，用户不可以使用该对话框定义自定义颜色。

（2）AnyColor 属性。该属性表示获取或设置一个值，该值指示对话框是否显示基本颜色集中可用的。

语法：

```
object. AnyColo [= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，指示对话框是否显示基本颜色集中可用的。当值为 true 时，基本颜色集中可用；当值为 false 时，基本颜色集中不可用。

（3）Color 属性。该属性表示获取或设置用户选定的颜色。

语法：

```
“object. Color [= Color]
```

属性值：object 表示一个对象表达式，Color 表示用户选定的颜色。

（4）FullOpen 属性。该属性表示获取或设置一个值，该值指示用于创建自定义颜色的控件在对话框打开时是否可见。

语法：

```
object. FullOpen [= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，指示用于创建自定义颜色的控件在对话框打开时是否可见。当值为 true 时，自定义颜色的控件在对话框打开时可见；当值为 false 时，自定义颜色的控件在对话框打开时不可见。

3．方法

ShowDialog 方法表示用默认的所有者运行通用对话框。

语法：

```
public DialogResult ShowDialog()
```

返回值：如果用户在对话框中单击“确定”按钮，则为 DialogResult.OK；否则为 DialogResult.Cancel。

4．事件

HelpRequest 事件在当用户单击通用对话框中的“帮助”按钮时发生。

语法：

```
public event EventHandler HelpRequest
```

示例 12-08　ColorDialog 控件的应用

本示例中，当程序运行时，单击“改变”按钮，弹出 ColorDialog 对话框，选择颜色，单击“确定”按钮，Label 标签中文字的颜色改变。代码如下：

代码 12-08　光盘位置：光盘\mingrisoft\sl\12\08

```
private void colorDialog1_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show("如有疑问，欢迎登录下面的网站咨询：\nhttp://www.mingrisoft.com", "友情提示", Message
BoxButtons.OK, MessageBoxIcon.Asterisk);                    //弹出提示性信息
}
private void Form1_Load(object sender, EventArgs e)
{
    this.colorDialog1.ShowHelp = true;                      //设置在颜色对话框中显示“帮助”按钮
    this.colorDialog1.SolidColorOnly = false;               //设置在颜色对话框中不限制用户使用纯颜色
    this.colorDialog1.FullOpen = true;                      //设置用于创建自定义颜色的控件在对话框打开时可见
    this.colorDialog1.AnyColor = true;                      //显示对话框中的所有基本颜色
    this.colorDialog1.AllowFullOpen = true;                 //允许用户使用自定义颜色
}
private void Change_Click(object sender, EventArgs e)
{
    this.colorDialog1.ShowDialog();                         //显示颜色对话框
    this.label1.ForeColor = this.colorDialog1.Color;        //设置当前 Label 控件中的文本颜色为设置的颜色
}
```

运行本示例，效果如图 12.37 所示。

图 12.37　ColorDialog 控件的应用

12.6　“字体”对话框

12.6.1　“字体”对话框简介

“字体”对话框通常用于对选定文字的样式进行更改，例如，更改文字的字体、字号、是否加粗或者加下划线等。图 12.38 所示的是 Word 的字体设置对话框。

图 12.38　Word 中的字体对话框

12.6.2 “字体”对话框的应用

“字体”对话框的实现是通过 FontDialog 控件实现的，下面详细介绍一下该控件。

1．功能

FontDialog 对话框用于设置公开系统上当前安装的字体。默认情况下，“字体”对话框显示字体、字体样式和字体大小的列表框、删除线和下划线等效果的复选框、脚本（脚本是指给定字体可用的不同字符脚本，如希伯来语或日语等）的下拉列表以及字体外观等选项。图 12.39 所示为 Windows 应用程序中的 FontDialog 控件。

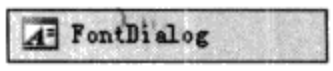


图 12.39 FontDialog 控件

2．属性

FontDialog 控件常用属性及说明如表 12.10 所示。

表 12.10 FontDialog 控件常用属性及说明

属　性	说　明
AllowScriptChange	此属性指定用户能否更改“脚本”组合框中指定的字符集，以显示除了当前所显示字符集以外的字符集
AllowSimulations	此属性指定对话框是否允许图形设备接口（GDI）字体模拟
AllowVectorFonts	此属性指定对话框是否允许选择矢量字体
AllowVerticalFonts	此属性指定对话框是既显示垂直字体又显示水平字体，还是只显示水平字体
Color	此属性获取或设置选定字体的颜色
FixedPitchOnly	此属性指定对话框是否只允许选择固定间距字体
Font	此属性获取或设置选定的字体
FontMustExist	此属性指定对话框是否指定当用户试图选择不存在的字体或样式时的错误条件
MaxSize	此属性获取或设置用户可选择的最大磅值
MinSize	此属性获取或设置用户可选择的最小磅值
ScriptsOnly	此属性获取或设置一个值，该值指示对话框是否允许为所有非 OEM 和 Symbol 字符集以及 ANSI 字符集选择字体
ShowApply	此属性获取或设置一个值，该值指示对话框是否包含“应用”按钮
ShowColor	此属性获取或设置一个值，该值指示对话框是否显示颜色选择
ShowEffects	此属性获取或设置一个值，该值指示对话框是否包含允许用户指定删除线、下划线和文本颜色选项的控件
ShowHelp	此属性获取或设置一个值，该值指示对话框是否显示“帮助”按钮

下面对 FontDialog 控件比较重要的属性进行详细介绍。

（1）AllowVectorFonts 属性。该属性表示指定对话框是否允许选择矢量字。

语法：

```
object. AllowVectorFonts [= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，指定对话框是否允许选择矢量字。当值为 true 时，允许选择矢量字；当值为 false 时，不允许选择矢量字。

（2）FixedPitchOnly 属性。该属性表示指定对话框是否只允许选择固定间距字体。

语法：

```
object. FixedPitchOnly[= boolean]
```

属性值：object 表示一个对象表达式，boolean 表示一个布尔表达式，指定对话框是否只允许选择固定间距字体。当值为 true 时，允许选择固定间距字体；当值为 false 时，不允许选择固定间距字体。

3．方法

ShowDialog 方法表示用默认的所有者运行通用对话框。

语法：

```
public DialogResult ShowDialog()
```

返回值：如果用户在对话框中单击“确定”按钮，则为 DialogResult.OK；否则为 DialogResult.Cancel。

4．事件

（1）Apply 事件。该事件当用户单击字体对话框中的“应用”按钮时发生。

语法：

```
public event EventHandler Apply
```

（2）HelpRequest 事件。该事件当用户单击字体对话框中的“帮助”按钮时发生。

语法：

```
public event EventHandler HelpRequest
```

示例 12-09　FontDialog 控件的应用

本示例中，当程序运行时，单击“字体”按钮，弹出字体设置对话框，设置完成后单击“确定”按钮或者“应用”按钮应用当前设置。代码如下：

代码 12-09　光盘位置：光盘\mingrisoft\sl\12\09

```
private void Form1_Load(object sender, EventArgs e)
{
    this.fontDialog1.AllowScriptChange = true; //设置显示除了当前所显示字符集以外的字符集
    this.fontDialog1.AllowSimulations = true;    //设置允许土型设备接口字体模拟
    this.fontDialog1.AllowVectorFonts = true;    //设置对话框允许访问矢量字体
    this.fontDialog1.AllowVerticalFonts = true;  //设置允许显示竖直字体和水平字体
    this.fontDialog1.FixedPitchOnly = false;     //设置对话框可以选择任意间距的字体
    this.fontDialog1.FontMustExist = true;       //用户只能选择当前电脑中存在的字体
    this.fontDialog1.MaxSize = 100;              //设置字体的最大磅值
    this.fontDialog1.MinSize = 1;                //设置字体的最小磅值
    this.fontDialog1.ScriptsOnly = true;         //可以为所有非 OEM 和 Symbol 字符集以及 ANSI 字符集选择字体
    this.fontDialog1.ShowApply = true;           //设置在对话框中显示“应用”按钮
    this.fontDialog1.ShowColor = true;           //设置在对话框中显示颜色设置
    this.fontDialog1.ShowEffects = true;         //设置在对话框中显示下划线、删除线以及文本颜色选项的控件
    this.fontDialog1.ShowHelp = true;            //设置在对话框中显示“帮助”按钮
}
private void button1_Click(object sender, EventArgs e)
{
    this.fontDialog1.ShowDialog();               //显示字体设置对话框
    this.label1.Font = this.fontDialog1.Font;    //设置 Label 控件中的字体为所设置的字体
    this.label1.ForeColor = this.fontDialog1.Color;//设置 Label 控件中的文本颜色为所设置的颜色
}
private void fontDialog1_Apply(object sender, EventArgs e)
{
    this.label1.Font = this.fontDialog1.Font;    //设置 Label 控件中的字体为所设置的字体
    this.label1.ForeColor = this.fontDialog1.Color;//设置 Label 控件中的文本颜色为所设置的颜色
}
private void fontDialog1_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show("如对本程序有疑问，欢迎登录下面的网站进行反馈：\nhttp://www.mingrisoft.com", "信息提示", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);//弹出信息提示
}
```

运行本示例，效果如图 12.40 所示。

图 12.40　FontDialog 控件的应用

12.7　“打印”对话框

12.7.1　“打印”对话框简介

现实生活中，当需要保存一份文档时，经常会将该文档进行打印然后再保存。打印的功能是通过“打印”对话框实现的，如图 12.41 所示是一个常见的打印对话框。

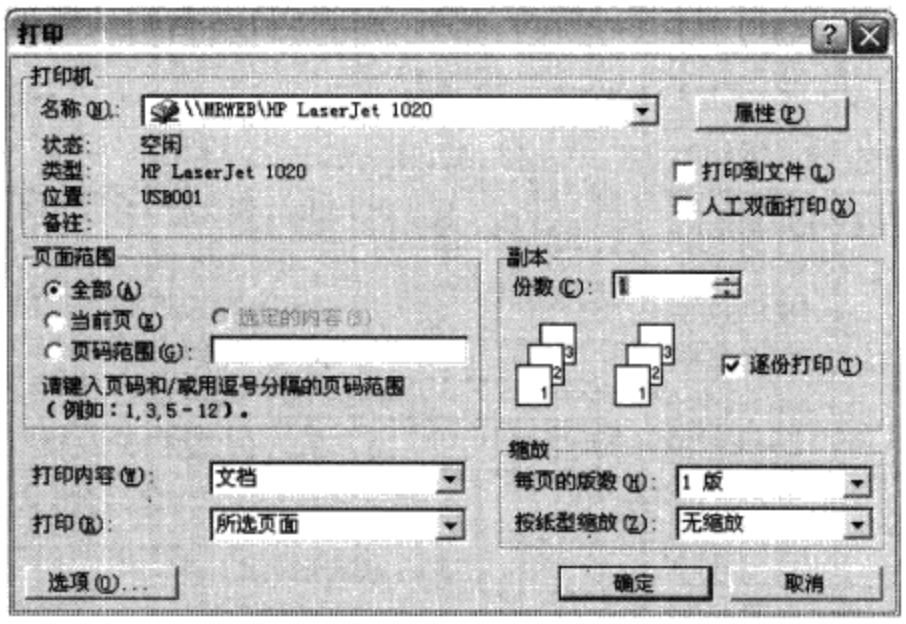


图 12.41　常见的“打印”对话框

12.7.2　“打印”对话框应用

“打印”对话框是通过 PrintDialog 控件实现的，下面将详细讲解该控件。

1．功能

PrintDialog 控件允许用户从 Windows 窗体应用程序中选择一台打印机，并选择文档中要打印的部分。该控件如图 12.42 所示。

图 12.42　PrintDialog 控件

2．属性

PrintDialog 控件常用属性及说明如表 12.11 所示。

表 12.11　　PrintDialog 控件常用属性及说明

属　性	说　明
AllowCurrentPage	获取或设置一个值，该值指示是否显示“当前页”选项按钮
AllowPrintToFile	获取或设置一个值，该值指示是否启用“打印到文件”复选框
AllowSelection	获取或设置一个值，该值指示是否启用“选择”选项按钮
AllowSomePages	获取或设置一个值，该值指示是否启用“页”选项按钮
Document	获取或设置一个值，指示用于获取 PrinterSettings 的 PrintDocument
PrinterSettings	获取或设置对话框修改的打印机设置
PrintToFile	获取或设置一个值，该值指示是否选中“打印到文件”复选框

续表

属　性	说　明
ShowHelp	获取或设置一个值，该值指示是否显示“帮助”按钮
ShowNetwork	获取或设置一个值，该值指示是否显示“网络”按钮
UseEXDialog	获取或设置一个值，该值指示在运行 Windows XP Home Edition、Windows XP Professional、Windows Server 2003 或更高版本的系统上，此对话框是否应当以 Windows XP 样式显示

下面对 PrintDialog 控件比较重要的属性进行详细讲解。

（1）AllowCurrentPage 属性。该属性表示获取或设置一个值，该值指示是否显示“当前页”选项按钮。

语法：

```
public bool AllowCurrentPage { get; set; }
```

属性值：如果显示“当前页”选项按钮，为 true；否则为 false。默认为 false。

（2）AllowPrintToFile 属性。该属性表示获取或设置一个值，该值指示是否启用“打印到文件”复选框。

语法：

```
public bool AllowPrintToFile { get; set; }
```

属性值：如果启用“打印到文件”复选框，为 true；否则为 false。默认为 true。

（3）AllowSelection 属性。该属性表示获取或设置一个值，该值指示是否启用“选择”选项按钮。

语法：

```
public bool AllowSelection { get; set; }
```

属性值：如果启用“选择”选项按钮，为 true；否则为 false。默认为 false。

（4）AllowSomePages 属性。该属性表示获取或设置一个值，该值指示是否启用“页”选项按钮。

语法：

```
public bool AllowSomePages { get; set; }
```

属性值：如果启用“页”选项按钮，为 true；否则为 false。默认为 false。

（5）Document 属性。该属性表示获取或设置一个值，指示用于获取 PrinterSettings 的 PrintDocument。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：用于获取 PrinterSettings 的 PrintDocument。默认为 null。

3．方法

（1）Reset 方法。该方法表示将所有选项、最后选定的打印机和页面设置重新设置为其默认值。

语法：

```
public override void Reset()
```

（2）ShowDialog 方法。该方法表示用默认的所有者运行通用对话框。

语法：

```
public DialogResult ShowDialog()
```

属性值：如果用户在对话框中单击“确定”按钮，则为 DialogResult.OK；否则为 DialogResult.Cancel。

4．事件

HelpRequest 事件当用户单击通用对话框中的“帮助”按钮时发生。

语法：

```
public event EventHandler HelpRequest
```

示例 12-10　PrintDialog 控件的应用

本示例中，当程序运行时，选择“文件”菜单下的“打印”选项即可实现对打印对话框的调用。代码如下：

代码 12-10　光盘位置：光盘\mingrisoft\sl\12\10

```
private void 打印 PToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (this.printDialog1.ShowDialog()==DialogResult.OK)//当单击“打印”选项时
        {
            this.printDocument1.Print();                    //开始文档的打印进程
        }
        if (this.richTextBox1.SelectedText != "")           //当 richTextBox1 控件中的选定内容不为空时
        {
            this.printDialog1.AllowSelection = true;        //启用“选择”选项按钮
            this.printDialog1.AllowPrintToFile = true;      //启用“打印到文件”复选框
            this.printDialog1.AllowSomePages = true;        //启用“页”选项按钮
            this.printDialog1.AllowCurrentPage = true;      //在窗体中显示“当前页”按钮
            this.printDialog1.PrintToFile = true;           //设置“打印到文件”复选框为选中状态
            this.printDialog1.ShowNetwork = true;           //设置显示“网络”按钮
            this.printDialog1.UseEXDialog = true;           //无论在什么系统中都以 Windows XP 样式显示
        }
    }
    catch (InvalidPrinterException ex)                      //捕获异常
    {
        MessageBox.Show(ex.Message, "编辑器", MessageBoxButtons.OK, MessageBoxIcon.Error);//弹出异常信息
    }
}
```

运行本示例，效果如图 12.43 所示。

图 12.43　PrintDialog 控件的应用

12.8　“页面设置”对话框

12.8.1　“页面设置”对话框简介

在打印的过程中，如果不对页面进行设置，那么打印结果有时候常常令人不满意，比如说，

最后一页只有几个字或者几行字，这样会造成极大的资源浪费。通过对页面的设置，可以尽可能地减少这种浪费。下面将详细讲解“页面设置”对话框的应用，图 12.44 是一个常见的页面设置对话框。

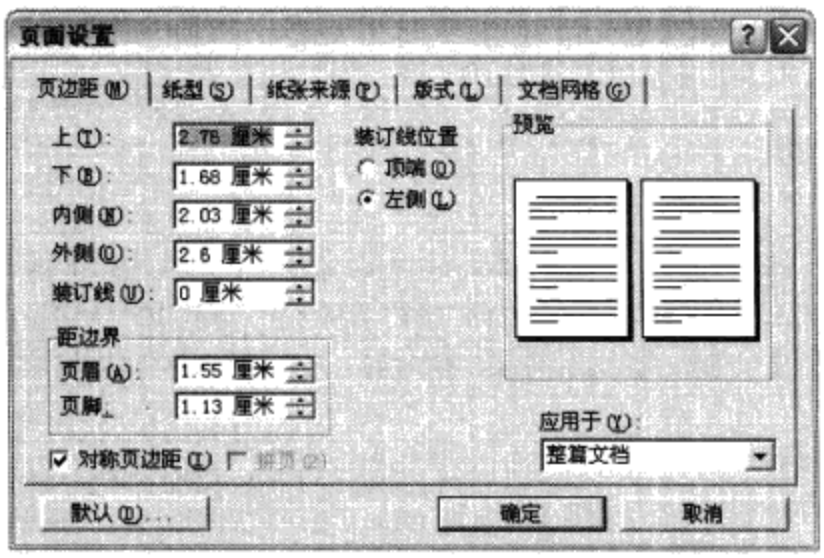


图 12.44　常见的“页面设置”对话框

12.8.2　“页面设置”对话框应用

“页面设置”对话框是通过 PageSetupDialog 控件来实现的，下面将详细讲解 PageSetupDialog 控件。

1．功能

PageSetupDialog 控件允许用户更改与页面相关的打印设置，包括边距和纸张方向。如图 12.45 所示为 PageSetupDialog 控件。

图 12.45　PageSetupDialog 控件

2．属性

PageSetupDialog 控件常用属性及说明如表 12.12 所示。

表 12.12　　PageSetupDialog 控件常用属性及说明

属　性	说　明
AllowMargins	获取或设置一个值，该值指示是否启用对话框的边距部分
AllowOrientation	获取或设置一个值，该值指示是否启用对话框的方向部分（横向对纵向）
AllowPaper	获取或设置一个值，该值指示是否启用对话框的纸张部分（纸张大小和纸张来源）
AllowPrinter	获取或设置一个值，该值指示是否启用“打印机”按钮
Document	获取或设置一个值，指示从中获取页面设置的 PrintDocument
EnableMetric	获取或设置一个值，该值指示以毫米为单位显示边距设置时，是否自动将边距设置转换为以百分之一英寸为单位（或相反）
MinMargins	获取或设置一个值，该值指示允许用户选择的最小边距（以百分之一英寸为单位）
PageSettings	获取或设置一个值，该值指示要修改的页设置
PrinterSettings	获取或设置用户单击对话框中“打印机”按钮时修改的打印机设置
ShowHelp	获取或设置一个值，该值指示“帮助”按钮是否可见
ShowNetwork	获取或设置一个值，该值指示“网络”按钮是否可见

下面对 PageSetupDialog 控件比较重要的属性进行详细说明。

（1）AllowMargins 属性。该属性表示获取或设置一个值，该值指示是否启用对话框的边距部分。

语法：

```
public bool AllowMargins { get; set; }
```

属性值：如果启用了对话框的边距部分，则为 true；否则为 false。默认为 true。

（2）AllowOrientation 属性。该属性表示获取或设置一个值，该值指示是否启用对话框的方向部分（横向对纵向）。

语法：

```
public bool AllowOrientation { get; set; }
```

属性值：如果启用了对话框的方向部分，则为 true；否则为 false。默认为 true。

（3）AllowPaper 属性。该属性表示获取或设置一个值，该值指示是否启用对话框的纸张部分（纸张大小和纸张来源）。

语法：

```
public bool AllowPaper { get; set; }
```

属性值：如果启用了对话框的纸张部分，则为 true；否则为 false。默认为 true。

（4）AllowPrinter 属性。该属性表示获取或设置一个值，该值指示是否启用“打印机”按钮。

语法：

```
public bool AllowPrinter { get; set; }
```

属性值：如果启用“打印机”按钮，则为 true；否则为 false。默认为 true。

3．方法

ShowDialog 方法表示用默认的所有者运行通用对话框。

语法：

```
public DialogResult ShowDialog()
```

返回值：如果用户在对话框中单击“确定”按钮，则为 DialogResult.OK；否则为 DialogResult.Cancel。

4．事件

HelpRequest 事件当用户单击通用对话框中的“帮助”按钮时发生。

语法：

```
public event EventHandler HelpRequest
```

示例 12-11　PageSetupDialog 控件的应用

本示例中，当程序运行时，单击“文件”菜单下的“页面设置”选项，弹出页面设置对话框。代码如下：

代码 12-11　光盘位置：光盘\mingrisoft\sl\12\11

```
private void 页面设置 UToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.pageSetupDialog1.ShowDialog();                    //使用默认的所有者运行通用对话框
}
private int linesPrinted;                                  //实例化一个整型变量
private void printDocument1_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    char[] param = { '\n' };                               //实例化一个字符型数组
    string[] lines = this.richTextBox1.Text.Split(param);  //实例化一个字符串型数组
    int x = e.MarginBounds.Left;                           //获取边距以内的矩形区域左边缘的 x 坐标
    int y = e.MarginBounds.Top;                            //获取边距以内的矩形区域上边缘的 y 坐标
    while (linesPrinted < lines.Length)                    //当打印的元素数小于数组的总长度时
    {
        //绘制指定的文本内容
        e.Graphics.DrawString(lines[linesPrinted++], new Font("宋体", 10), Brushes.Black, x, y);
        y += 15;                                           //递增 y 的值
        if (y > e.PageBounds.Bottom)                       //当 y 的值超出当前页的底部时
        {
            e.HasMorePages = true;                         //打印附加页
            return;                                        //返回
        }
    }
    linesPrinted = 0;                                      //重新为 linesPrinted 赋值
```

```
        e.HasMorePages = false;                                    //不打印附加页
    }
```

运行本示例，效果如图 12.46 所示。

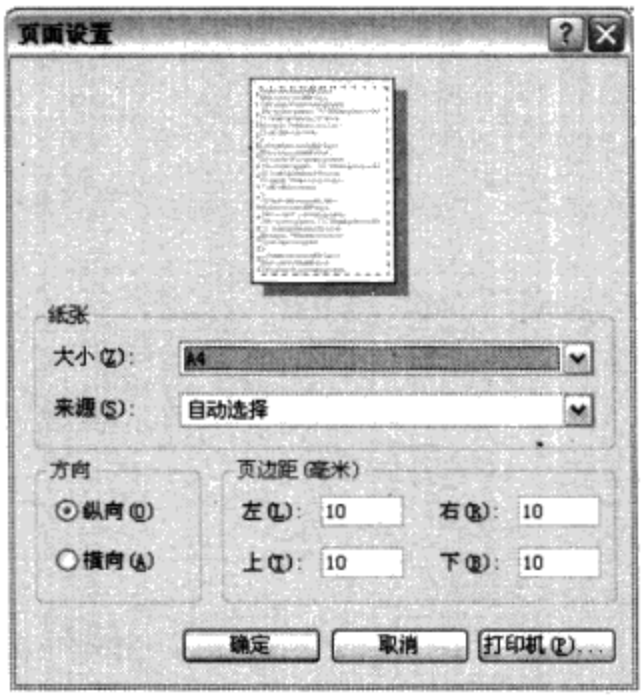


图 12.46　PageSetupDialog 控件的应用

12.9　“打印预览”对话框

12.9.1　“打印预览”对话框简介

在打印的过程中，如果用户想提前看到打印效果，那么可以通过打印预览来实现。通过打印预览对话框，用户可以对自己将要打印的文件有一个直观的了解，当发现不满意的地方时可以做出及时的调整。

12.9.2　“打印预览”对话框应用

“打印预览”对话框是通过 PrintPreviewDialog 控件来实现的，下面将详细讲解 PrintPreviewDialog 控件。

1．功能

PrintPreviewDialog 控件表示包含用于从 Windows 窗体应用程序中进行打印的 PrintPreviewControl 对话框窗体。如图 12.47 所示为 PrintPreviewDialog 控件。

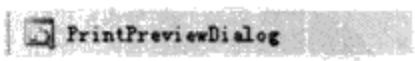


图 12.47　PrintPreviewDialog 控件

2．属性

PrintPreviewDialog 控件常用属性及说明如表 12.13 所示。

表 12.13　　PrintPreviewDialog 控件常用属性及说明

属　　性	说　　明
AcceptButton	获取或设置当用户按 Enter 键时所单击的窗体上的按钮
ActiveControl	获取或设置容器控件上的活动控件
ActiveMdiChild	获取当前活动的多文档界面（MDI）子窗口
AllowDrop	获取或设置一个值，该值指示控件是否可以接受用户拖放到它上面的数据
AllowTransparency	获取或设置一个值，该值指示能否调整窗体的不透明度
AutoScale	获取或设置一个值，该值指示窗体是否调整其大小以适合该窗体上使用的字体高度，以及是否缩放其控件

续表

属　　性	说　　明
AutoScaleMode	获取或设置控件的自动缩放模式
AutoScroll	获取或设置一个值，该值指示窗体是否实现自动滚动
AutoScrollMargin	获取或设置自动滚动边距的大小
AutoScrollMinSize	获取或设置自动滚动条的最小大小
AutoSize	获取或设置一个值，该值指示 PrintPreviewDialog 是否自动调整大小以完整显示其内容
AutoSizeMode	获取或设置窗体自动调整自身大小的模式
CancelButton	获取或设置 PrintPreviewDialog 的取消按钮
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
CurrentAutoScaleDimensions	获取屏幕的当前运行时尺寸
DialogResult	获取或设置窗体的对话框结果
Document	获取或设置要预览的文档
HelpButton	获取或设置一个值，该值指示是否应在窗体的标题框中显示“帮助”按钮
Icon	获取或设置窗体的图标
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
Menu	获取或设置在窗体中显示的 MainMenu
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
PrintPreviewControl	获取一个值，该值指示此窗体中包含的 PrintPreviewControl 控件
ShowIcon	获取或设置一个值，该值指示是否在窗体的标题栏中显示图标
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
StartPosition	获取或设置运行时对话框的起始位置
Text	获取或设置显示在控件上的文本
UseAntiAlias	获取或设置一个值，该值指示打印是否使用操作系统的防锯齿功能
WindowState	获取或设置窗体的窗口状态

下面对 PrintPreviewDialog 控件比较重要的属性进行详细介绍。

（1）AutoScaleMode 属性。该属性表示获取或设置控件的自动缩放模式。

语法：

```
public AutoScaleMode AutoScaleMode { get; set; }
```

属性值：表示当前缩放模式的 AutoScaleMode。默认为 None。

AutoScaleMode 的可取值及说明如下。

- None：表示禁用自动缩放。
- Font：表示根据类使用的字体（通常为系统字体）的维度控制缩放。
- Dpi：表示根据显示分辨率控制缩放。常用分辨率为 96DPI 和 120DPI。
- Inherit：表示根据类的父类的缩放模式控制缩放。如果不存在父类，则禁用自动缩放。

（2）AutoSizeMode 属性。该属性表示获取或设置窗体自动调整自身大小的模式。

语法：

```
public AutoSizeMode AutoSizeMode { get; set; }
```

属性值：AutoSizeMode 枚举值。默认为 GrowOnly。

AutoSizeMode 的可取值及说明如下。

- GrowAndShrink：表示控件根据它的内容增大或缩小。不能手动调整该控件的大小。
- GrowOnly 表示控件可以根据其内容任意增大，但不会缩小至小于它的 Size 属性值。窗

体可以调整大小，但不能缩小到它所包含的任意控件被隐藏。

(3) Document 属性。该属性表示获取或设置要预览的文档。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：PrintDocument，它表示要预览的文档。

(4) WindowState 属性。该属性表示获取或设置窗体的窗口状态。

语法：

```
public FormWindowState WindowState { get; set; }
```

属性值：FormWindowState，表示窗体的窗口状态。默认为 FormWindowState.Normal。

FormWindowState 的可取值及说明如下。

- Normal：表示默认大小的窗口。
- Minimized：表示最小化窗口。
- Maximized：表示最大化窗口。

3．方法

(1) AddOwnedForm 方法。该方法表示向此窗体添加附属窗体。

语法：

```
public void AddOwnedForm(Form ownedForm)
```

参数说明如下。

ownedForm：此窗体将拥有的 Form。

(2) Close 方法。该方法表示关闭窗体。

语法：

```
public void Close()
```

(3) Focus 方法。该方法表示为控件设置输入焦点。

语法：

```
public bool Focus()
```

参数说明如下。

返回值：如果输入焦点请求成功，则为 true；否则为 false。

(4) Hide 方法。该方法表示对用户隐藏控件。

语法：

```
public void Hide()
```

(5) RemoveOwnedForm 方法。该方法表示从此窗体移除附属窗体。

语法：

```
public void RemoveOwnedForm(Form ownedForm)
```

参数说明如下。

ownedForm：Form 类型，表示要从此窗体的附属窗体列表中移除的窗体。

4．事件

(1) Click 事件。该事件在单击窗体时发生。

语法：

```
public event EventHandler Click
```

(2) Load 事件。该事件在第一次显示窗体前发生。

语法：

```
public event EventHandler Load
```

示例 12-12　PrintPreviewDialog 控件的应用

本示例中，当程序运行时，单击"文件"菜单下的"打印预览"子菜单即可实现打印预览功能。代码如下：

代码 12-12　光盘位置：光盘\mingrisoft\sl\12\12

```
private void 打印预览 VToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.printPreviewDialog1.ShowDialog();              //显示打印预览对话框
    this.printPreviewControl1.Visible = true;           //显示 printPreviewControl1 控件
}
private void Form1_Load(object sender, EventArgs e)
{
    this.printPreviewControl1.Visible = false;          //隐藏 printPreviewControl1 控件
}
private void printDocument1_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    char[] param = { '\n' };                            //实例化一个字符数组 param
    string[] lines = richTextBox1.Text.Split(param);    //实例化一个字符串数组 lines
    int i = 0;                                          //实例化一个整型变量 i
    char[] trimparam = { '\r' };                        //实例化一个字符数组 trimparam
    foreach (string s in lines)                         //循环遍历数组 lines 中的内容
    {
        lines[i++] = s.TrimEnd(trimparam);              //移除每一行的换行符
    }
    int x = 20;                                         //实例化一个整型变量 x
    int y = 20;                                         //实例化一个整型变量 y
    foreach (string line in lines)                      //循环遍历数组 lines 中的内容
    {
        e.Graphics.DrawString(line, new Font("宋体", 10), Brushes.Black, x, y);  //绘制内容
        y += 15;                                                                 //递增 y 的值
    }
}
```

运行本示例，效果如图 12.48 所示。

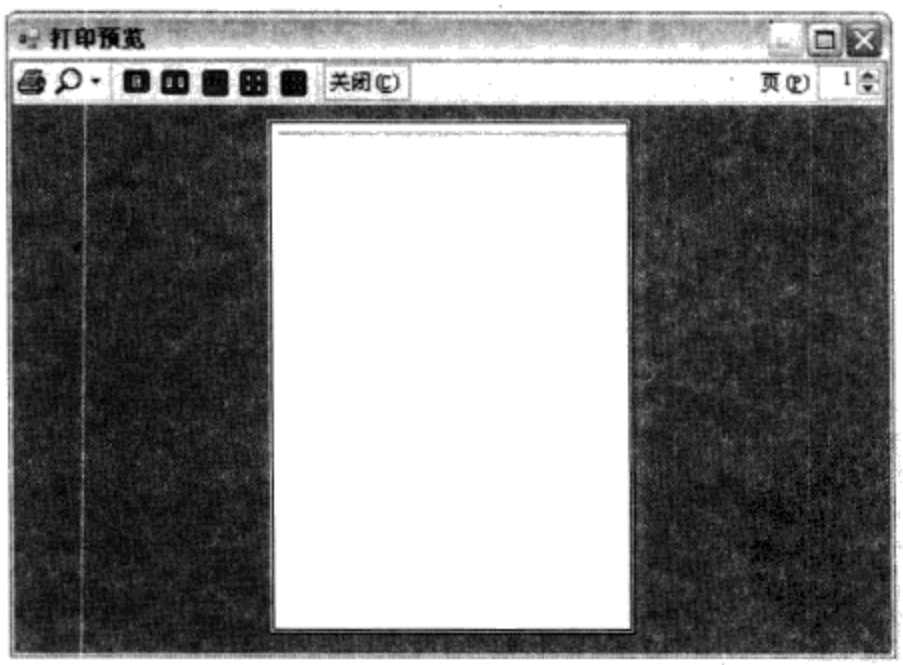


图 12.48　PrintPreviewDialog 控件的应用

12.10　“浏览文件夹”对话框

12.10.1　“浏览文件夹”对话框简介

在安装软件的过程中，都会遇到安装路径的更改问题，系统默认的路径并不是最有利于用户使用的路径。在这个时候，“浏览文件夹”对话框帮助解决了这个难题，如图 12.49 所示是一个常用的“浏览文件夹”对话框。

图 12.49　PageSetupDialog 控件的应用

12.10.2 "浏览文件夹"对话框应用

"浏览文件夹"对话框的实现主要是通过 FolderBrowserDialog 控件实现的，下面详细讲解该控件。

1．功能

FolderBrowserDialog 控件的主要功能是提示用户选择文件夹，如图 12.50 所示。

图 12.50　FolderBrowserDialog 控件

2．属性

FolderBrowserDialog 控件常用属性及说明如表 12.14 所示。

表 12.14　　FolderBrowserDialog 控件常用属性及说明

属　性	说　明
Description	获取或设置对话框中在树视图控件上显示的说明文本
RootFolder	获取或设置从其开始浏览的根文件夹
SelectedPath	获取或设置用户选定的路径
ShowNewFolderButton	获取或设置一个值，该值指示"新建文件夹"按钮是否显示在文件夹浏览对话框中

下面对 FolderBrowserDialog 控件比较重要的属性进行详细的说明。

（1）Description 属性。该属性表示获取或设置对话框中在树视图控件上显示的说明文本。

语法：

```
public string Description { get; set; }
```

属性值：要显示的说明。默认值为空字符串("")。

（2）RootFolder 属性。该属性表示获取或设置从其开始浏览的根文件夹。

语法：

```
public Environment.SpecialFolder RootFolder { get; set; }
```

属性值：Environment.SpecialFolder 值之一。默认为 Desktop。

SpecialFolder 的可选值及说明如表 12.15 所示。

表 12.15　　SpecialFolder 的可选值及说明

可　选　值	说　明
ApplicationData	目录，它用作当前漫游用户的应用程序特定数据的公共储存库
CommonApplicationData	目录，它用作所有用户使用的应用程序特定数据的公共储存库
LocalApplicationData	目录，它用作当前非漫游用户使用的应用程序特定数据的公共储存库
Cookies	用作 Internet Cookie 的公共储存库的目录
Desktop	逻辑桌面，而不是物理文件系统位置

续表

可 选 值	说 明
Favorites	用作用户收藏夹项的公共储存库的目录
History	用作 Internet 历史记录项的公共储存库的目录
InternetCache	用作 Internet 临时文件的公共储存库的目录
Programs	包含用户程序组的目录
MyComputer	“我的电脑”文件夹
MyMusic	“My Music”文件夹
MyPictures	“My Pictures”文件夹
Recent	包含用户最近使用过的文档的目录
SendTo	包含“发送”菜单项的目录
StartMenu	包含“开始”菜单项的目录
Startup	对应于用户的“启动”程序组的目录
System	“System”目录
Templates	用作文档模板的公共储存库的目录
DesktopDirectory	用于物理上存储桌面上的文件对象的目录
Personal	用作文档的公共储存库的目录
MyDocuments	“我的电脑”文件夹
ProgramFiles	“Program files”目录
CommonProgramFiles	用于应用程序间共享的组件的目录

(3) SelectedPath 属性。该属性表示获取或设置用户选定的路径。

语法:

```
public string SelectedPath { get; set; }
```

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("")。

3．方法

ShowDialog 方法表示用默认的所有者运行通用对话框。

语法:

```
public DialogResult ShowDialog()
```

属性值：如果用户在对话框中单击“确定”按钮，则为 DialogResult.OK；否则为 DialogResult.Cancel。

4．事件

HelpReques 事件当用户单击字体对话框中的“帮助”按钮时发生。

语法:

```
public event EventHandler HelpRequest
```

示例 12-13 FolderBrowserDialog 控件的应用

本示例中，当程序运行时，单击“浏览”按钮显示浏览文件夹对话框。代码如下：

代码 12-13 光盘位置：光盘\mingrisoft\sl\12\13

```
private void Open_Click(object sender, EventArgs e)
{
    this.folderBrowserDialog1.ShowDialog();                    //使用默认的所有者运行“浏览文件夹”对话框
    this.Position.Text = this.folderBrowserDialog1.SelectedPath;//在文本框中显示选择文件夹的路径
}
```

```
private void Form1_Load(object sender, EventArgs e)
{
    this.folderBrowserDialog1.ShowNewFolderButton = true;  //在运行的通用对话框中显示“新建文件夹”按钮
}
```

运行本示例，效果如图 12.51 所示。

图 12.51　FolderBrowserDialog 控件的应用

第 3 篇

ASP.NET 网站开发篇

C#

第 13 章

ASP.NET 3.5 网站开发基础

- ASP.NET 3.5 运行环境配置与管理
- ASP.NET 3.5 基本设计步骤
- ASP.NET 3.5 网页语法
- Web.config 文件配置

ASP.NET 是一个统一的 Web 开发模型，它包括用户使用尽可能少的代码生成企业级 Web 应用程序所必须的各种服务。ASP.NET 的开发运行是基于.NET Framework 的，当用户编写 ASP.NET 应用程序的代码时，可以访问.NET Framework 中的类及其成员，用户可以使用与公共语言运行库兼容的任何语言来编写应用程序的代码。

13.1 ASP.NET 3.5 运行环境配置与管理

Internet 信息服务（IIS）为用户提供了一种可以添加、配置 Web 站点、Web 文件夹和文件的方法。用 ASP.NET 3.5 开发的网页要通过 Internet 信息服务系统来访问。Internet 信息服务包含在 Windows 的操作系统中，所以无论采用哪种操作系统（Microsoft），其安装方式均类似。

13.1.1 安装 IIS 服务器

建立 Web 应用程序必须要用到 Internet 信息服务。IIS 的主要功能包括：为响应使用者的请求，将所要浏览的网页内容传输给使用者；管理及维护 Web 站点、FTP 站点、SMTP 虚拟服务器。目前 IIS 的最新版本是 6.0 版，它是 Windows 2003 的内置组件，下面将介绍 Windows Server 2003 系统中 IIS 的安装，安装步骤如下。

（1）将 Windows Server 2003 系统安装盘放到光盘驱动器中。

（2）依次选择“开始”/“设置”/“控制面板”/“添加或删除程序”选项，弹出“添加或删除程序”对话框，如图 13.1 所示。

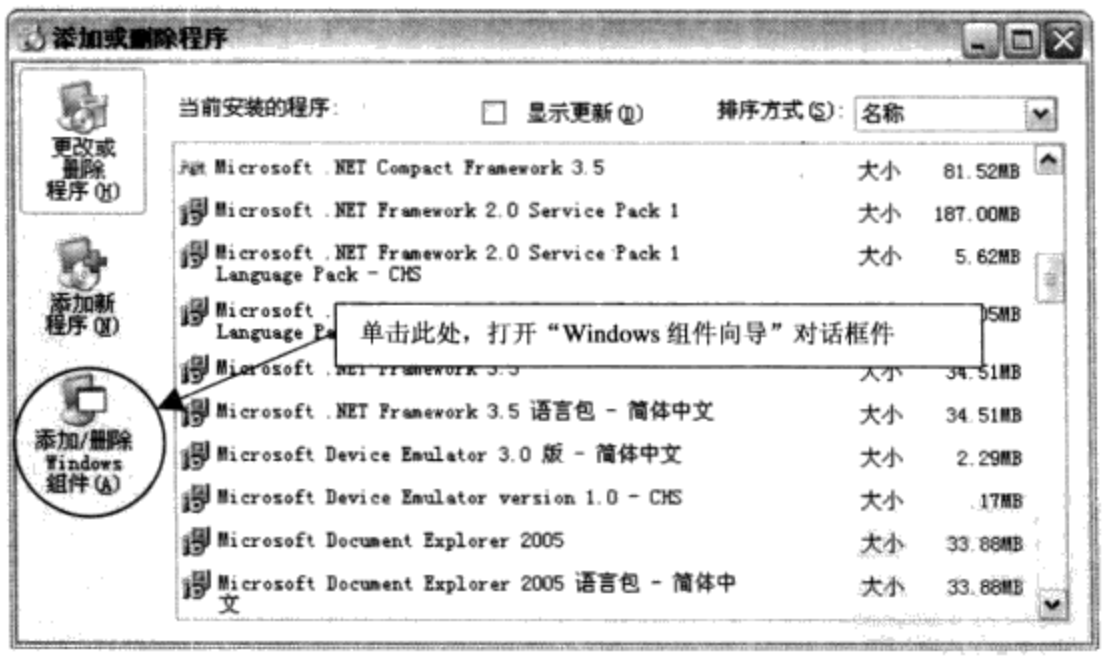


图 13.1 “添加或删除程序”对话框

（3）单击对话框左侧的“添加/删除 Windows 组件”命令，弹出“Windows 组件向导”对话框，如图 13.2 所示。

（4）选中“应用程序服务器”复选框，单击【详细信息】按钮，弹出“应用程序服务器”对话框，如图 13.3 所示。

（5）选中“Internet 信息服务（IIS）”复选框，单击【详细信息】按钮，选择 IIS 服务器需要安装的组件，如图 13.4 所示。

（6）选择完成后，单击【确定】按钮，然后单击【下一步】按钮，开始安装 IIS 信息服务，如图 13.5 所示。

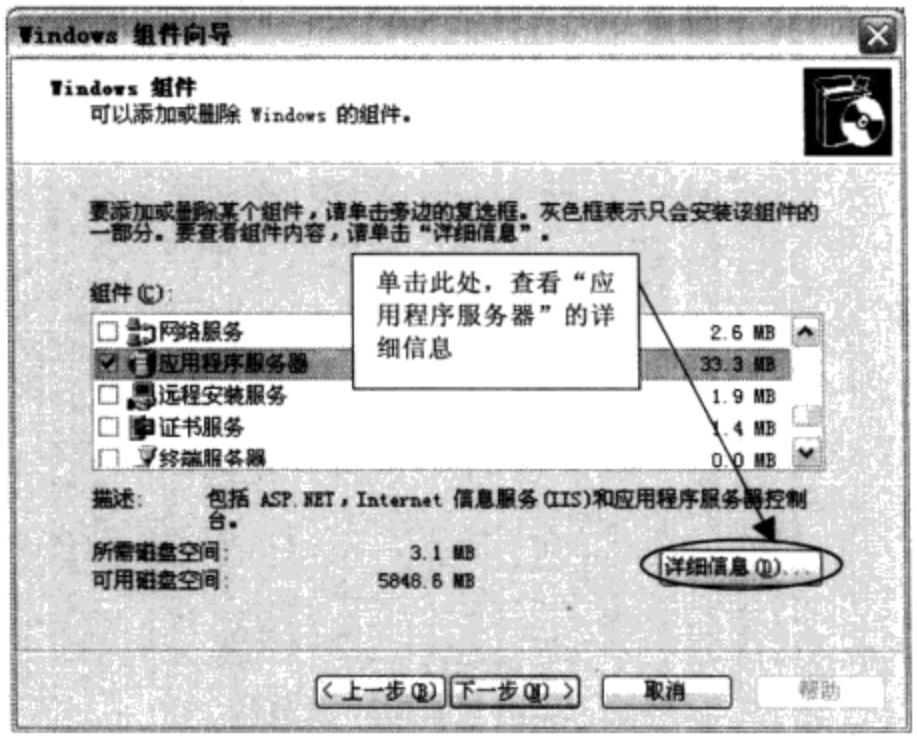


图 13.2 “Windows 组件向导”对话框

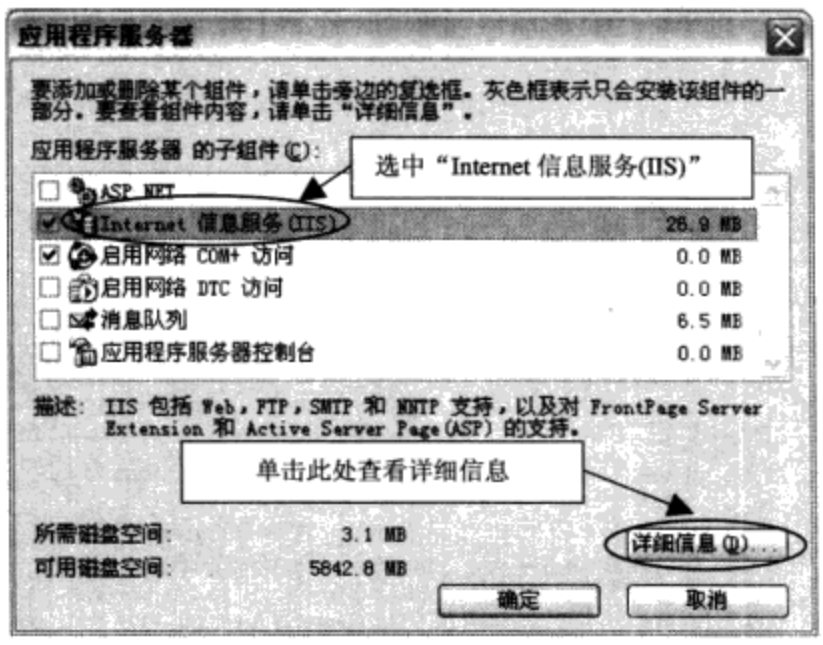


图 13.3 “应用程序服务器”对话框

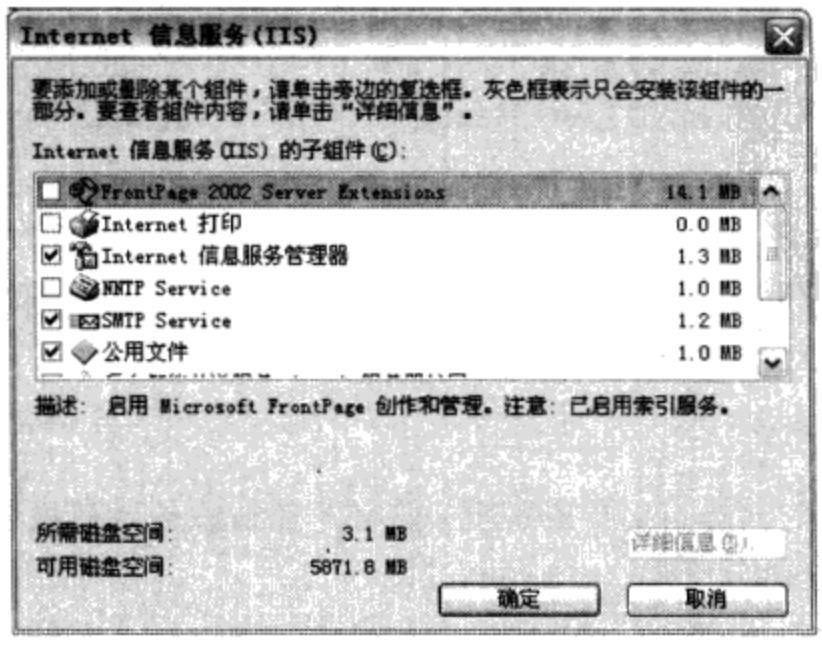


图 13.4 “Internet 信息服务(IIS)”对话框

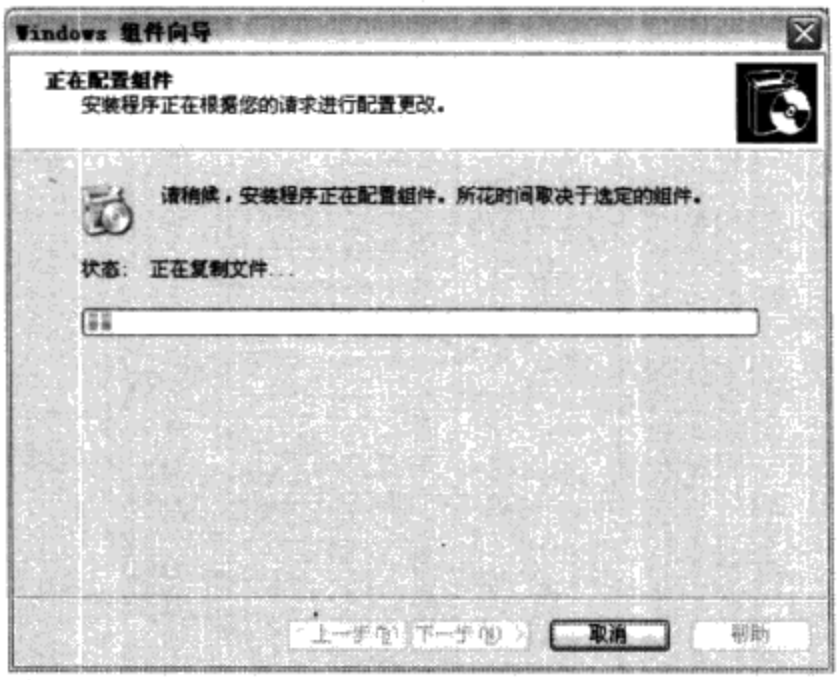


图 13.5 “正在配置组件”对话框

(7) 单击【完成】按钮，完成 IIS 信息服务的安装。

(8) IIS 安装完成之后，选择“开始”/“程序”/“管理工具”/“服务”选项，弹出“服务”窗口，如图 13.6 所示。

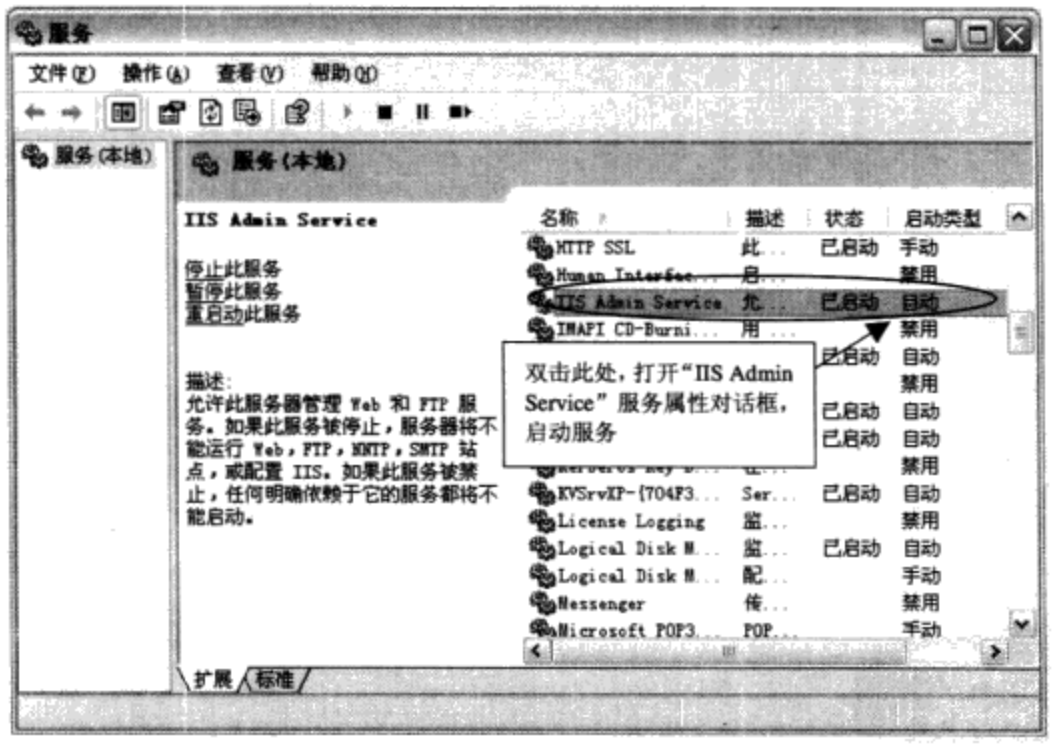


图 13.6 “服务”窗口

(9) 在“服务”窗口中，选择“IIS Admin Service”服务，双击鼠标左键，打开“IIS Admin Service”服务属性对话框，启动服务。

13.1.2 配置 IIS 服务器

IIS 安装启动后就要对其进行必要的配置，这样才能使服务器在最优的环境下运行，下面介绍 IIS 服务器配置的具体操作步骤。

(1) 依次选择“开始”/“设置”/“控制面板”/“管理工具”/“Internet 信息服务(IIS)管理器”选项，弹出“Internet 信息服务(IIS)管理器”窗口，如图 13.7 所示。

(2) 选择“网站”选项卡。在此选项卡中可以设置网站标识、连接和日志记录，在“IP 地

址”下拉列表框中，默认选项为“(全部未分配)”，用户可以使用此默认选项，也可以选择下拉列表框中本机的 IP 地址；“TCP 端口”文本框中的默认值为 80，建议不要修改此参数；用户还可以设置站点的连接超时时间，默认为 120s，如图 13.8 所示。

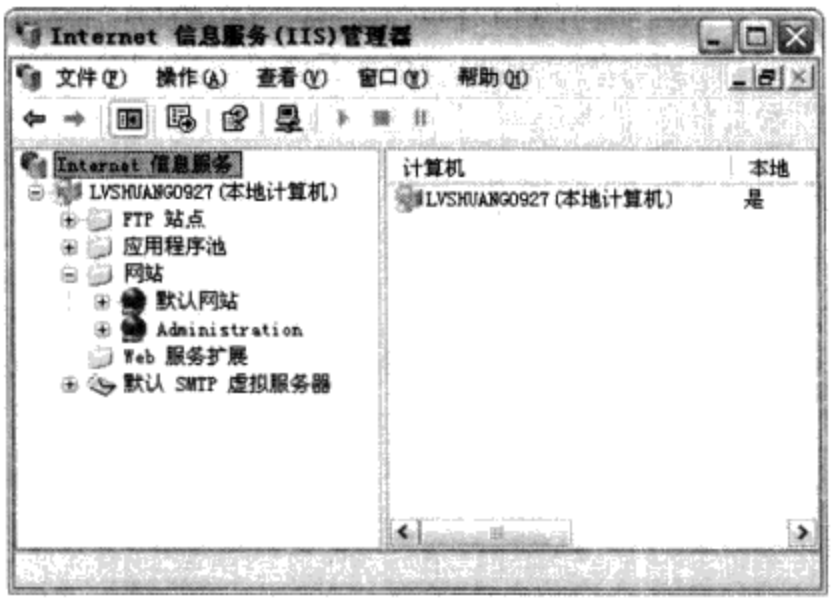


图 13.7 “Internet 信息服务（IIS）管理器”窗口

（3）选择“主目录”选项卡。设置连接到 Web 站点的内容来源，默认为“此计算机上的目录”，默认的 Web 站点存放文件的路径为“C:\Inetpub\wwwroot”，对此目录的默认权限只有“读取”、“记录访问”、“索引资源”等 3 项，如果没有特殊要求，此选项卡中的内容不需要改动，如图 13.9 所示。

图 13.8 “网站”选项卡

图 13.9 “主目录”选项卡

（4）选择“文档”选项卡。选中“启用默认内容文档”复选框，这样当运行 Web 程序后，不需要在地址栏中填写此文件名，系统会默认读取默认文档中文件，Web 程序会自动寻找此路径下的 Default.aspx 文件，用户可以添加新的默认文档或删除默认文档，如图 13.10 所示。

（5）选择“目录安全性”选项卡。设置目录的安全性，共有 3 种方法可以控制目录的安全性，分别为“身份验证和访问控制”、“IP 地址和域名限制”和“安全通信”，通过这 3 种方法可以有效地控制目录的安全性，如图 13.11 所示。

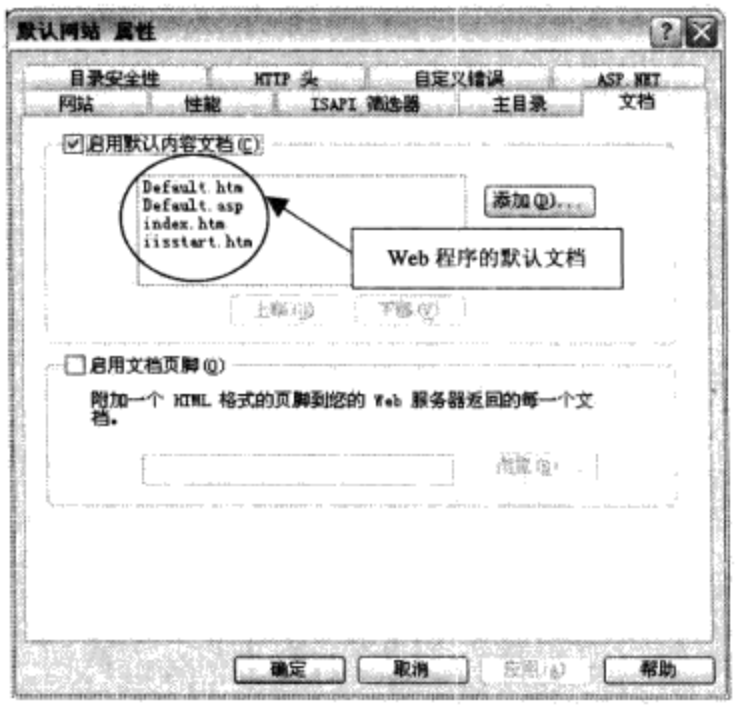


图 13.10 “文档”选项卡

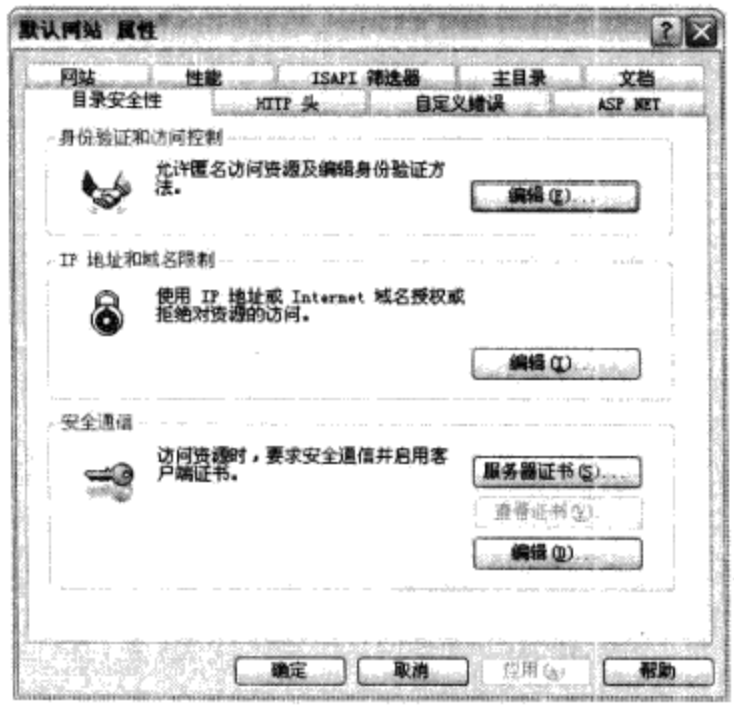


图 13.11 “目录安全性”选项卡

13.2 ASP.NET 3.5 基本设计步骤

13.2.1 创建 ASP.NET 3.5 网站

基本设计步骤如下。

(1) 运行“Visual Studio 2008”集成开发环境，如图 13.12 所示。

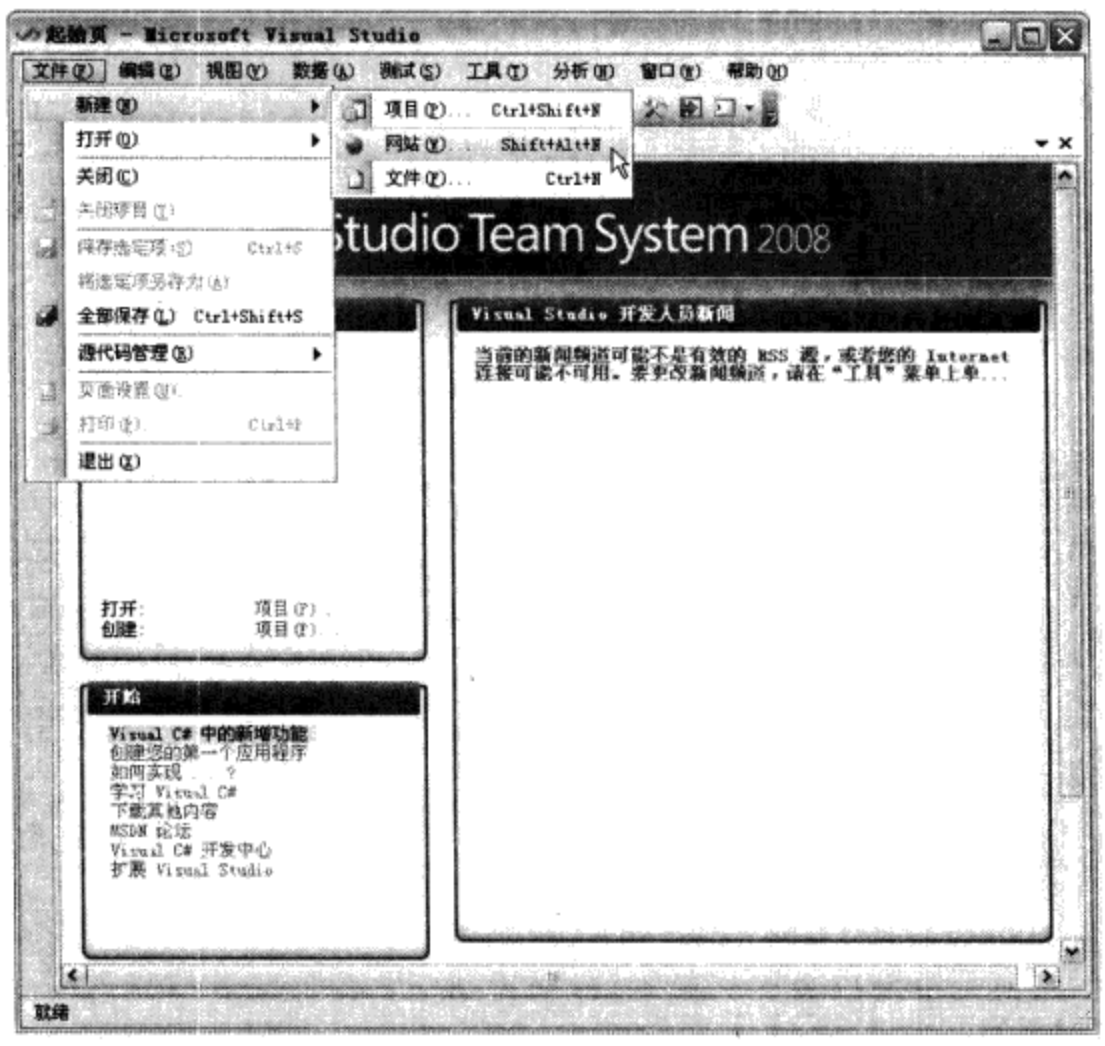


图 13.12 创建 ASP.NET 3.5 网站

(2) 选择“文件”/“新建”/“网站”菜单，弹出“新建网站”对话框。

（3）选取“ASP.NET 网站”并为其指定存储的路径，如图 13.13 所示，然后单击【确定】按钮，这样一个 ASP.NET 网站便创建成功了。

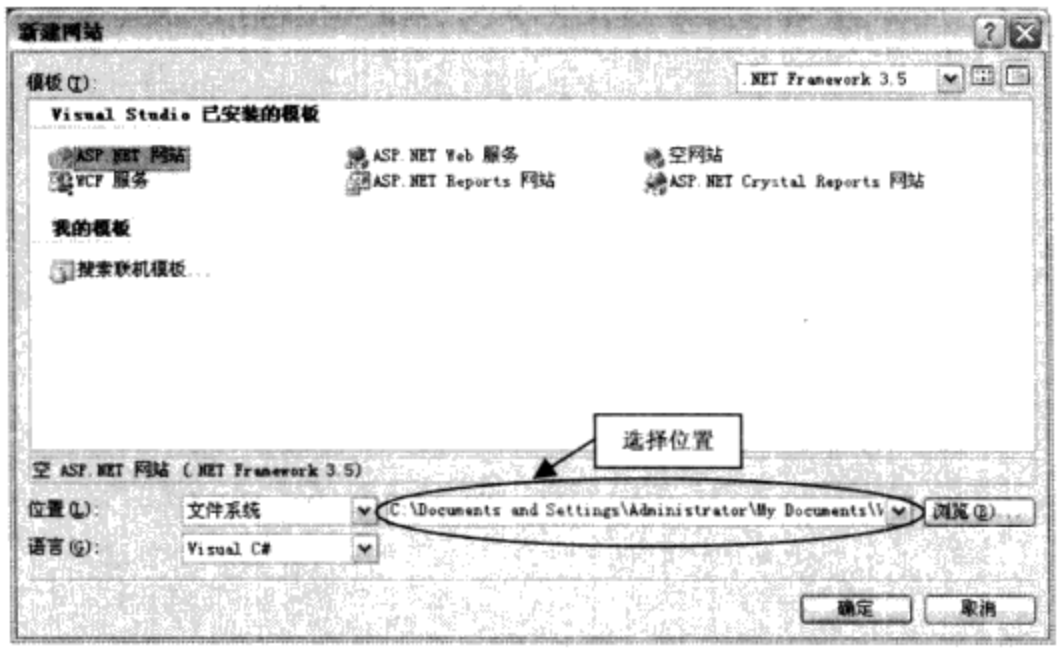


图 13.13 “新建网站”对话框

13.2.2 创建 Web 窗体

当用户创建 ASP.NET 3.5 网站后，“Visual Studio 2008”集成开发环境默认创建一个 Web 窗体（Default.aspx），然而在实际开发中，仅仅一个默认窗体是不够的，下面将演示如何创建一个新的 Web 窗体。

（1）展开“解决方案资源管理器”选项卡，用鼠标右键单击网站名，在弹出的右键菜单中选择“添加新项”命令，打开“添加新项”对话框，如图 13.14 所示。

（2）在“添加新项”对话框中选择 Web 窗体，单击【添加】按钮，如图 13.15 所示。

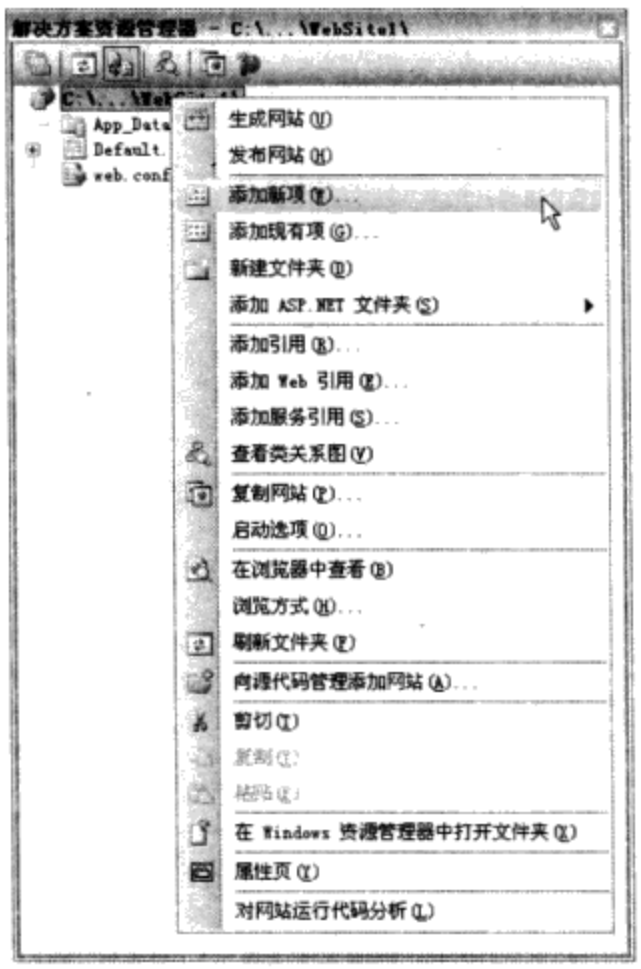


图 13.14 添加新项

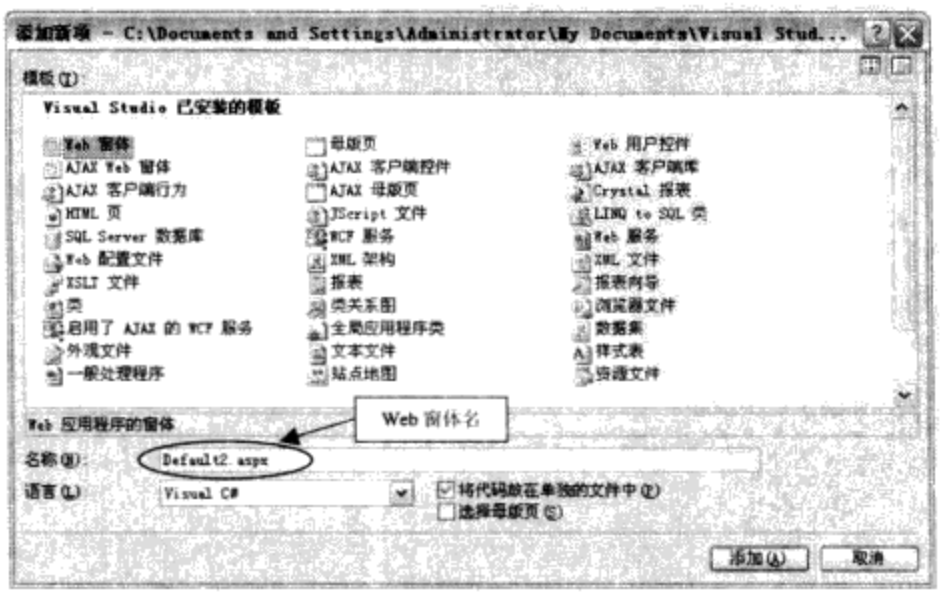


图 13.15 “添加新项”对话框

13.2.3 设计 Web 窗体界面

在“Visual Studio 2008”集成开发环境中，设计 Web 窗体界面通常借助“表格”来完成，如图 13.16 所示。

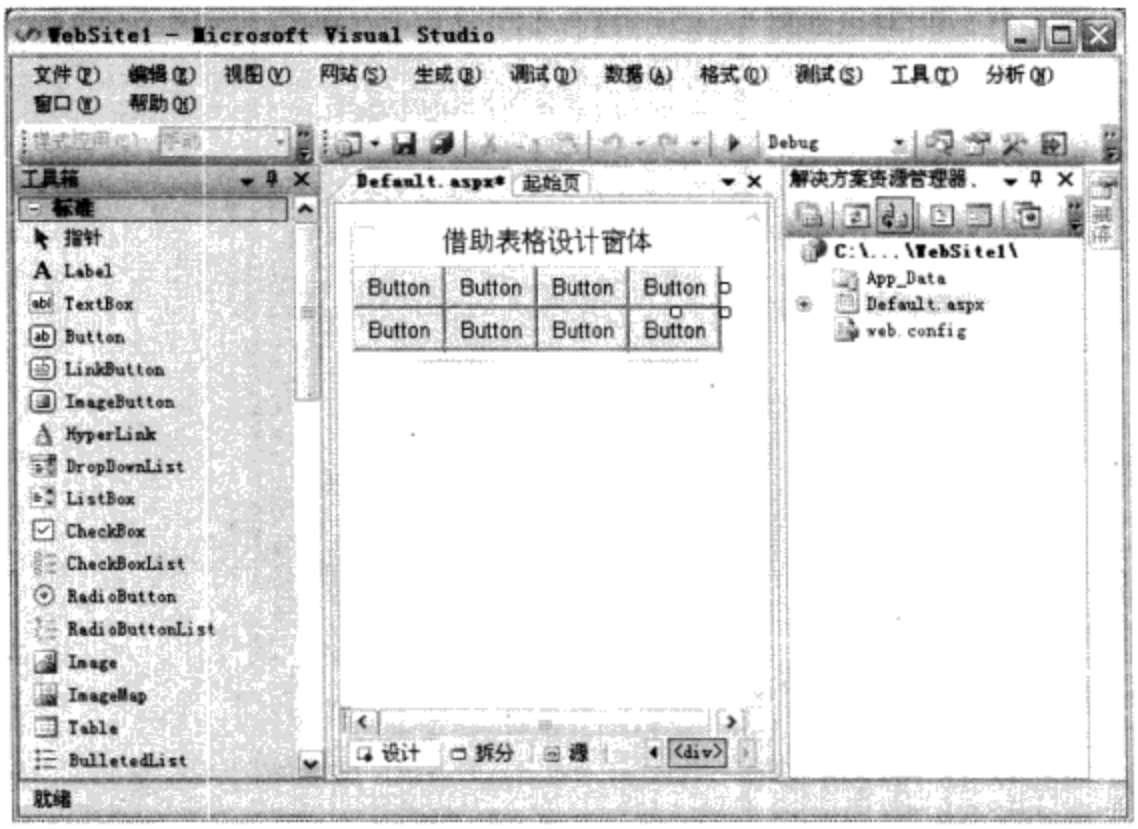


图 13.16 Web 窗体设计窗口

13.2.4 编写代码

ASP.NET 3.5 也是事件驱动的编程语言，例如，在“设计”窗体中添加一个按钮，双击这个按钮便会进入“代码编辑区”，如图 13.17 所示（这种模式便是“代码隐藏页”）。同时，ASP.NET 3.5 也支持另一种模式即“单文件页”模型，例如，单击“源”选项卡，如图 13.18 所示。在单文件页模型中，页的标记及其编程代码位于同一个.aspx 文件中。

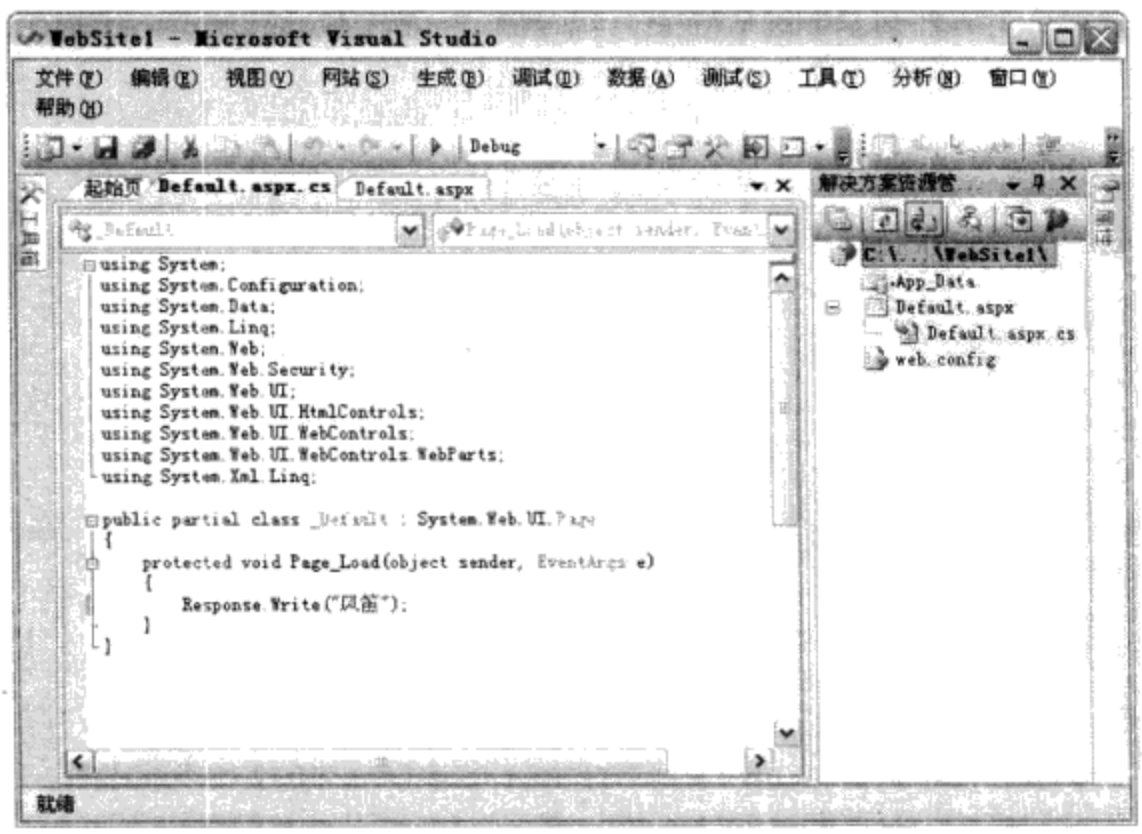


图 13.17 “代码编辑器”窗口

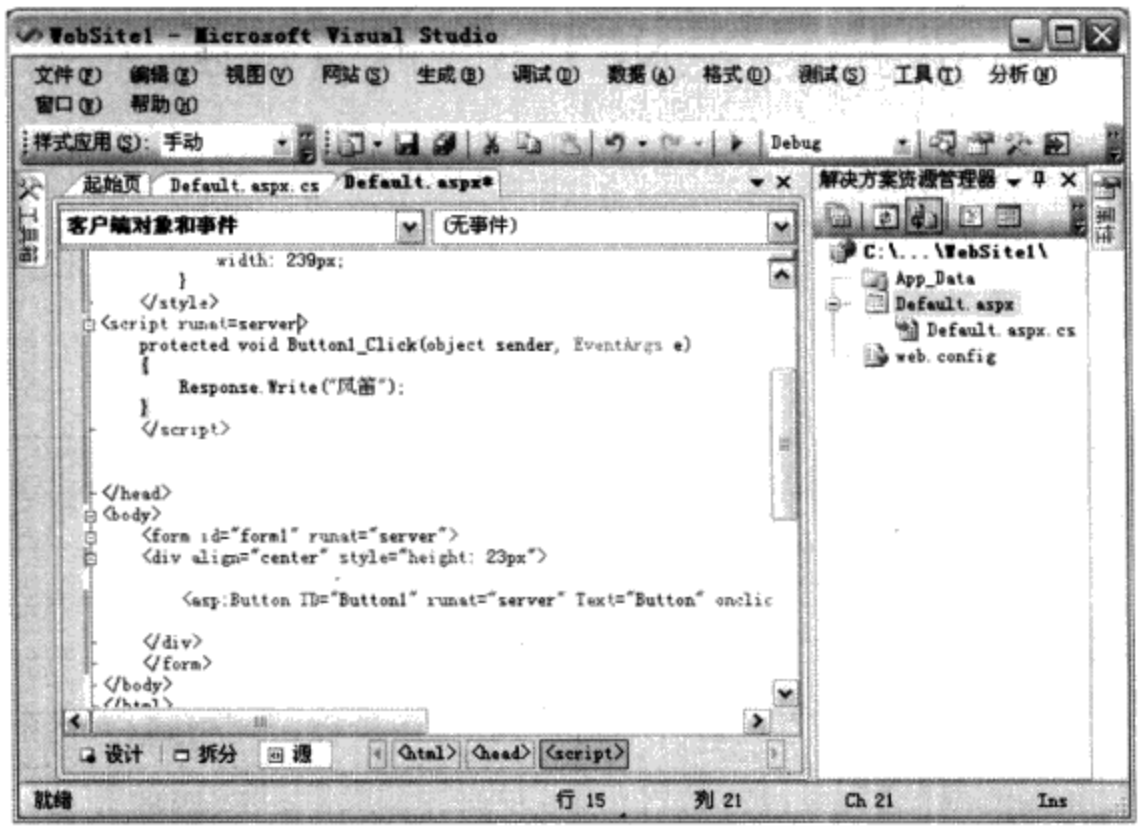


图 13.18 “HTML 代码编辑器”窗口

13.2.5 运行程序

当首次运行程序时，首先要为站点设置起始页，用鼠标右键单击需要启动的窗体，然后选择“设为起始页”选项，如图 13.19 所示，其次是为站点添加 Web.config 文件，因为“Visual Studio 2008”集成开发环境在创建时并没有添加 Web.config 文件，所以在首次启动时系统会给出提示，如图 13.20 所示。用户只需选择【确定】按钮就会自动在程序中添加一个 Web.config 文件。

图 13.19 设置起始页

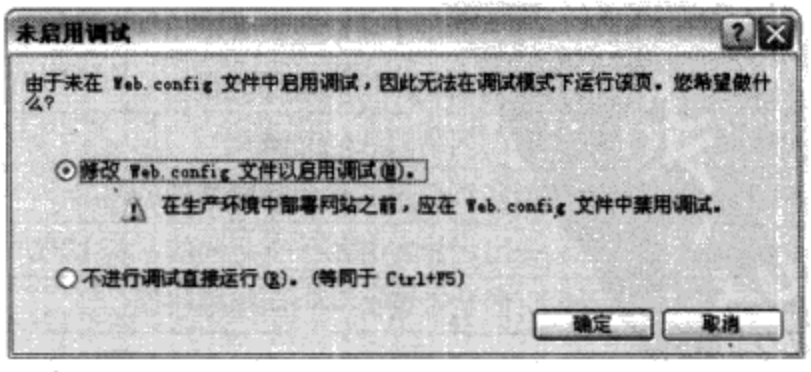


图 13.20 系统提示窗体

注意：ASP.NET 3.5 中包含启动执行（快捷键 F5）、开始执行不调试（快捷键 Ctrl+F5）两种运行方式。两者的区别是前者要在确保程序正确的情况下才能进行，而后者则是直接运行，跳过程序中的错误。

13.3 ASP.NET 3.5 网页语法

13.3.1 ASP.NET 3.5 网页扩展名

每种程序都有自己的扩展名，例如，可执行文件的扩展名为.exe、记事本的扩展名为.txt，ASP.NET 3.5 网页也有自己的扩展名，具体描述如表 13.1 所示。

表 13.1　　ASP.NET 网页扩展名

文　件	扩　展　名
Web 用户控件	.ascx
HTML 页	.htm
XML 页	.xml
母版页	.master
Web 服务	.asmx
全局应用程序类	.asax
Web 配置文件	.config
网站地图	.sitemap
外观文件	.skin
样式表	.css

13.3.2 Page 指令

Page 指令是 ASP.NET 页面中必须有的指令，而且每个页面只能拥有一个 Page 指令。主要用于定义 ASP.NET 页面分析器和编译器使用的特定属性。可以放在 ASP.NET 页面任何合理位置，但一般都是将其放置在页的顶端。下面的代码就是一个 Page 指令。

```
<%@ Page Language="C#" AutoEventWireup="True" CodeFile="TryHtml.aspx.cs" Inherits="TryHtml" %>
```

Page 指令的一些常用属性如表 13.2 所示。

表 13.2　　Page 指令的常用属性

属　性	说　明
AspCompat	当设为 true 时，使页面可以使用 ASP 内置对象，默认值为 false
AutoEventWireup	指示页面控件与事件是否自动关联，如果启用事件自动关联，则设为 true；否则，设为 false
Buffer	确定是否启用 HTTP 响应缓冲，如果启用页缓冲，则为 true；否则为 false
CodePage	用来设置页面使用的编码
ContentType	该属性主要用于获取或设置输出文件流的文件类型
Language	该页面代码所使用的编程语言
Inherits	定义供页继承的代码隐藏类
CodeFile	指定指向页引用的代码隐藏文件的路径
ErrorPage	定义在出现异常时用于重定向的目标 URL
Debug	设置页面是否调试其页面程序错误

13.3.3 代码注释

代码注释就是在代码中对程序进行说明或解释，这些说明或解释并不影响代码的执行，代码注释在编程过程中是必不可少的，其主要功能是为了方便程序员理解代码，下面将详细介绍 ASP.NET 中比较重要的注释方式。

（1）HTML 注释。在 HTML 代码中，可以通过“<!—注释内容-->”的形式进行注释，可以对单行或多行进行注释，例如，在 ASP.NET 的 HTML 代码中为代码添加注释，代码如下：

```
<!--下面的代码用于声明页面的加载事件-->                    //HTML 代码中的注释方式
<script runat="server" language="c#">
void Page_Load(object sender, EventArgs e)
{
    Label1.Text = System.DateTime.Now.ToString();
}
</script>
```

（2）代码注释。在 ASP.NET 程序的.cs 代码中，可以使用“//”为代码添加注释，这样可以使程序员更好地理解代码的作用。例如，在.cs 代码中添加注释，代码如下：

```
public partial class _Default : System.Web.UI.Page
{
    //页面的加载事件
    protected void Page_Load(object sender, EventArgs e)
    {
        //向页面输出“风笛”
        Response.Write("风笛");
    }
}
```

13.3.4 HTML 服务器控件语法

默认情况下，ASP.NET 3.5 文件中的 HTML 元素作为文本进行处理，并且不能在服务器端代码中引用这些元素。若要使这些元素能以编程方式进行访问，可以通过添加 runat="server"属性将 HTML 元素作为服务器控件进行处理；还可以设置元素的 id 属性，以通过编程方式引用控件。

图 13.21 查询结果

示例 13-01 HTML 服务器控件编程

通过设置 HTML 控件的 runat 属性，使其成为服务器控件，同时添加事件，使 HTML 控件如同服务器控件一样使用，如图 13.21 所示。

程序主要代码如下。

代码 13-01 光盘位置：光盘\mingrisoft\sl\13\13_01

```
<%@ Page Language="C#" AutoEventWireup="true"   CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>runat 属性</title>
    <script runat="server">                                          //声明代码是在服务端运行的
     string[] name = new string[] { "风笛","芸烨湘枫","叶子","风飘飘"};  //声明一个名称数组
     string[] birthday = new string[] { "1981-09-27","1982-01-12","1982-01-01","1988-01-11"};//声明一个生日数组
     string[] Addr = new string[] { "吉林长岭","吉林公主岭","吉林公主岭","吉林长岭"};     //声明一个地址数组
     protected void Button1_Click(object sender, EventArgs e)        //声明按钮的单击事件
     {
         if (this.Text1.Value == "")                                 //如果没有输入姓名
         {
             Response.Write("请输入姓名");                           //提示要输入姓名
         }
         else                                                        //否则
         {
             string strName = this.Text1.Value;                      //获取输入的姓名
             for (int i = 0; i < name.Length; i++)                   //遍历姓名数组查找是否存在
             {
```

```
                    if (name[i] == strName)                                    //如果数组中包含输入的姓名
                    {
                        //输出姓名
                        Response.Write("***************" + name[i] + "***************<br/>");
                        Response.Write("出生日期：" + birthday[i] + "<br/>");//输出生日
                        Response.Write("出生地点：" + Addr[i] + "<br/>");//输出出生地点
                    }
                }
            }
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <asp:Label ID="Label1" runat="server" Text="输入姓名："></asp:Label>      //Label 控件提示输入姓名
 <input id="Text1" runat="server" type="text" />                   //HTML 文本框设置 runat 属性
    //HTML 按钮，设置 runat 属性使其成为服务器端控件，单击后指定 Button1_Click 事件下的代码
    <input id="Button1" runat ="server" type="button" value="查找"   onserverclick="Button1_Click"/><br />
    </form>
</body>
</html>
```

注意：HTML 服务器控件必须驻留在具有 runat="server"属性包含的 form 标记中。

13.3.5 代码块语法

HTML 页面一般都是通过 HTML 语言进行编程的，但是在 ASP.NET 程序中，可以通过“<%%>”在<form id="form1" runat="server"></form>区域添加后台代码，也可以通过<Script></Script>在<head runat="server"></head>区域中添加后台代码，下面通过示例演示如何通过这两种方式向 HTML 页面中添加代码。

图 13.22 嵌入式代码遍历指定文件夹

示例 13-02 向 HTML 页面中添加代码

遍历指定目录下的所有文件名，并将文件名显示出来，如图 13.22 所示。

程序主要代码如下。

代码 13-02 光盘位置：光盘\mingrisoft\sl\13\13_02

```
<%@ Page Language="C#" AutoEventWireup="true"   CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>嵌入式代码遍历指定文件夹</title>
    <script runat="server">                                          //<Script></Script>标记程序在后台运行
        protected void Page_Load(object sender, EventArgs e)         //声明窗体 Load 事件
        {
            Response.Write("指定目录下的文件名");                     //显示提示信息
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <%string[] fileName = System.IO.Directory.GetFiles(@"D:\图片");//<%%>标记程序在后台运行
      for (int i = 0; i < fileName.Length; i++)                      //遍历指定目录下的所有文件名
      {
          Response.Write(fileName[i]+"<br/>");                       //输出文件名
      }
```

```
        %>
      </div>
      </form>
  </body>
  </html>
```

13.3.6 数据绑定语法

数据绑定就是将数据源中的数据填充到控件的方法，ASP.NET 3.5 包含一个新的数据绑定模型。无需编写代码，数据源控件即可以连接到各种数据源并从中检索数据，将数据应用于要绑定的控件。在绑定数据时可以配合 DataSource 属性、DataBind 方法、XPath、Eval 函数来实现。

1．通过 Eval 函数绑定数据

数据绑定表达式包含在<%#和%>分隔符之内，并使用 Eval 函数进行绑定。Eval 函数用于定义单向（只读）绑定。下面通过示例演示如何使用 Eval 函数绑定数据。

示例 13-03　通过 Eval 函数绑定数据

通过 Eval 函数在数据控件 DataList 的模板内绑定员工信息表中的数据字段，以显示员工信息，如图 13.23 所示。

员工信息	
员工编号:	1
姓名	风笛
性别	男
年龄	29
地址	长春
薪金	1500.0000
工龄	1
奖金	100.0000

图 13.23　Eval 函数绑定员工信息

程序主要代码如下：

代码 13-03　光盘位置：光盘\mingrisoft\sl\13\13_03

在 Default.aspx 页面的 HTML 视图中，DataList 控件的模板内添加代码，通过 Eval 函数绑定数据库中员工信息表中的字段，代码如下。

```
    <body bgcolor ="#ffffcc">
        <form id="form1" runat="server">
        <div align =center >
            <asp:DataList ID="DataList1" runat="server"
                BorderStyle="None" BorderWidth="1px" CellPadding="3" CellSpacing="2" GridLines="Both" BackColor
="#DEBA84" BorderColor="#DEBA84"   >
                <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />

                <HeaderTemplate>
                    员工信息
                 </HeaderTemplate>
                <ItemTemplate >
                <table id="Table1"    runat =server>
                  <tr>
                  <td width =110 align =left >员工编号:</td>
                      <td width =110 align =left ><asp:Label ID="Label1" runat="server" Text="<%#Bind('empID')%
>"></asp:Label></td>
                  </tr>
                  <tr>
                  <td width =120 align =left>姓名</td>
                  <td width =120 align =left><%# Eval("EmpName") %> </td>        //绑定 EmpName 字段
                  </tr>
                  <tr>
                  <td width =60 align =left>性别</td>
                  <td width =60 align =left><%# Eval("EmpSex") %></td>            //绑定 EmpSex 字段
                  </tr>
                  <tr>
                  <td width =60 align =left>年龄</td>
                  <td width =60 align =left><%# Eval("EmpAge") %></td>            //绑定 EmpAge 字段
                  </tr>
                  <tr >
```

```
                <td width =200 align =left>地址</td>
                <td width =200 align =left><%# Eval("EmpAddress") %></td>     //绑定 EmpAddress 字段
                </tr>
                <tr>
                    <td width =60 align =left>薪金</td>
                    <td width =60 align =left><%# Eval("Salary") %></td>        //绑定 Salary 字段
                </tr>
                <tr>
                <td width =60 align =left>工龄</td>
                  <td width =60 align =left><%# Eval("GL") %></td>              //绑定 GL 字段
                </tr>
                <tr>
                  <td width =60 align =left>奖金</td>
                <td width =60 align =left><%# Eval("Prize") %></td>             //绑定 Prize 字段
                </tr>
              </table>

              <br />
              </ItemTemplate>
              <FooterTemplate>

              </table>
              </FooterTemplate>
              <SelectedItemStyle BackColor="#738A9C" Font-Bold="True" ForeColor="White" />
              <ItemStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
              <HeaderStyle BackColor="#A55129" Font-Bold="True" ForeColor="White" />

          </asp:DataList></div>
      </form>
  </body>
```

在 Default.aspx.cs 代码中，自定义一个 DBBind 方法，用于绑定数据库中的数据，代码如下。

```
public void DBBind()
{
    SqlConnection myConn = DBClass.GetConnection();                    //获得数据库连接
    myConn.Open();                                                     //打开数据库连接
    string SqlStr = "select * from tbEmp";                             //声明 sql 语句
    SqlDataAdapter da = new SqlDataAdapter(SqlStr, myConn);            //创建一个 SqlDataAdapter 对象
    DataSet ds = new DataSet();                                        //生成 DataSet 对象并填充数据
    da.Fill(ds, "Emp");                                                //填充数据
    DataList1.DataSource = ds.Tables["Emp"].DefaultView;               //将 DataList 控件进行数据绑定
    DataList1.DataBind();                                              //绑定数据
}
```

2．通过 DataSource 属性与 DataBind 方法绑定数据

有些控件，如 GridView、FormView、DropDownList 和 DetailsView 控件，当它们通过 DataSource 属性绑定到数据源控件时，会通过调用 DataBind 方法来执行绑定。

例如，将字符串数组绑定到 DropDownList 控件，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    string[] str = new string[4] { "风笛","芸烨湘枫","叶子","一生所爱"};
    this.DropDownList1.DataSource = str;
    this.DropDownList1.DataBind();
}
```

13.3.7 对象标记语法

开发人员可以使用对象标记语法“<object runat=“server”/>“来声明和创建变量。下面演示如何通过对象标记语法声明变量。

示例 13-04 通过对象标记实现数据绑定

使用嵌入式代码实现循环输出功能，通过对象标记语法声明 ArrayList 变量，然后通过 for

语句为变量赋值，最后将数据绑定到下拉框中，如图 13.24 所示。

图 13.24　通过对象标记实现数据绑定

程序主要代码如下：

代码 13-04　光盘位置：光盘\mingrisoft\sl\13\13_04

```
<%@ Page Language="C#" AutoEventWireup="true"    CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>无标题页</title>
    <script runat ="server">                                          //声明代码在服务器端运行
        protected void Page_Load(object sender, EventArgs e)          //页面 Load 事件
        {
            for (int i = 1949; i <= 1999; i++)                        //通过 for 语句添加年份
            {
                temp_arrayList.Add(i);                                //将年份添加到动态数组中
            }
            DropDownList1.DataSource = temp_arrayList;                //设置 DropDownList 的 DataSource 属性
            DropDownList1.DataBind();                                 //绑定数据
            for (int j = 1; j <= 12; j++)                             //通过 for 语句添加月份
            {
                temp_month.Add(j);                                    //将年份添加到动态数组中
            }
            DropDownList2.DataSource = temp_month;                    //设置 DropDownList 的 DataSource 属性
            DropDownList2.DataBind();                                 //绑定数据
            for (int k = 1; k <= 31; k++)                             //通过 for 语句添加天数
            {
                temp_Day.Add(k);                                      //将年份添加到动态数组中
            }
            DropDownList3.DataSource = temp_Day;                      //设置 DropDownList 的 DataSource 属性
            DropDownList3.DataBind();                                 //绑定数据
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            if(IsPostBack)                                            //显示选择的出生日期
            Response.Write("您的出生日期："+DropDownList1.SelectedItem.Text+"-"+DropDownList2.SelectedItem.
Text+"-"+DropDownList3.SelectedItem.Text);
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    <object runat="server" id="temp_arrayList" class="System.Collections.ArrayList" ></object>
    <object runat="server" id="temp_month" class="System.Collections.ArrayList" ></object>
    <object runat="server" id="temp_Day" class="System.Collections.ArrayList" ></object>
        <asp:Label ID="Label1" runat="server" Text="出生日期："></asp:Label>
        <asp:DropDownList ID="DropDownList1" runat="server">
```

```
        </asp:DropDownList>
        年<asp:DropDownList ID="DropDownList2" runat="server">
        </asp:DropDownList>
        月<asp:DropDownList ID="DropDownList3" runat="server">
        </asp:DropDownList>
        日
        <asp:Button ID="Button1" runat="server" Text="输入"   OnClick="Button1_Click"/>
    </div>
    </form>
</body>
</html>
```

13.4 Web.config 文件配置

ASP.NET 3.5 的配置数据被存储在 XML 文件中，每一个 XML 文件都命名为 Web.config。Web.config 文件可以出现在 ASP.NET 3.5 应用程序的多个目录中。它包含有关 ASP.NET 3.5 的格式和继承层次结构等信息。

Web.config 中包含多个配置节，相关的说明如表 13.3 所示。

表 13.3　　Web.config 的配置节

配　置　节	描　　述
customErrors 节	指示此应用程序域中的服务器通道是否将经筛选的或完整的异常信息返回给本地或远程调用方
sessionState 节	为当前应用程序配置会话状态设置
authentication 节	如果在执行请求的过程中，出现未处理的错误，则通过<customErrors>节可以配置相应的处理步骤。具体说来，开发人员通过该节可以配置要显示的 html 错误页以代替错误堆栈跟踪
trace 节	配置 ASP.NET 2.0 跟踪服务，主要用来程序测试判断哪里出错，包含对跟踪消息进行收集、存储和路由的侦听器
compilation 节	配置 ASP.NET 2.0 用于编译应用程序的所有编译设置
authorization 节	配置 Web 应用程序的授权，以控制客户端对 URL 资源的访问
pages 节	用于标识特定页的配置设置（例如，是否启用会话状态、视图状态和是否检测用户的输入等）
httpRuntime 节	配置 ASP.NET 2.0HTTP 运行时的设置，以确定如何处理对 ASP.NET 2.0 应用程序的请求

1．应用 authentication 节

<authentication>主要用于配置 ASP.NET 3.5 身份验证，在<authentication>节中，主要进行对 mode 属性的设置，此属性包含有 4 种参数，分别为 Windows、Forms、PassPort 和 None。下面分别对这 4 种参数进行介绍。

- Windows：Internet(IIS)信息服务根据应用程序的设置执行身份验证。在 IIS 中必须禁止匿名访问。
- Forms：需要工程创建者为用户提供一个输入凭据的自定义窗体，然后在应用程序中验证身份。用户的凭据标记存储于 Cookie 对象中。
- PassPort：身份验证通过 Microsoft 的集中身份验证服务执行，它为成员站点提供了不同用户访问此应用程序资源。
- None：不执行身份验证。

为了防止没有登录的用户访问需要身份验证的网页，完成网页自动跳转到登录网页，增强网站的安全性，可以使用下面的程序代码：

```
<authentication mode="Forms" >
  <forms>
    name=".ASPXAUTH"
    loginUrl="logPage.aspx"
  </forms>
</authentication>
```

注意：mode 是必选的属性，默认值为 Windows。其中元素 loginUrl 表示登录网页的名称，name 表示 Cookie 名称。

2．应用 customErrors 节

<customErrors>用于为 ASP.NET 3.5 应用程序提供有关自定义错误的信息。它不适用于 XML Web services 中发生的错误。在<customErrors>中主要设置 mode 属性，此属性共有 3 个属性值，其含义如下。

- On：一直显示自定义信息。
- Off：一直显示详细的 ASP.NET 3.5 错误信息。
- RemoteOnly：只对不在本地 Web 服务器上运行的用户显示自定义信息。

通过配置<customErrors>节实现当发生错误时，将网页跳转到自定义的错误页面，这样可以使程序在出错的情况下，仍给用户显示很友好的信息。程序代码如下：

```
<customErrors defaultRedirect="ErrorInfo.aspx"mode="RemoteOnly">
</customErrors>
```

3．应用 sessionState 节

sessionState 用于为当前应用程序配置会话状态（如设置是否启用会话状态及会话状态保存位置）。

sessionState 各参数的含义如下。

- Mode：设置存储会话状态。状态包括 Off、Inproc、StateServer 和 SqlServer。Off 表示禁用会话状态，Inproc 表示工作进程自身存储会话状态，StateServer 表示将把会话信息存放在一个单独的 ASP.NET 3.5 状态服务中，SqlServer 表示将把会话信息存放在 SQL Server 数据库中。
- StateConnectionString：用来设置 ASP.NET 3.5 应用程序存储远程会话状态的服务器名，默认名为本地。
- Cookieless：当 Cookieless 的属性设置为 true 时，表示不使用 Cookie 会话标识客户；反之设置为 false 时，表示启动 Cookie 会话状态。
- SqlConnectionString：设置 SQL Server 数据库连接。
- Timeout：设置会话时间，超过该期限，会自动中断会话，默认设置为 20mm。

对 sessionState 节的设置，程序代码如下：

```
<sessionState mode="InProc" cookieless="True" timeout="20"/>
</sessionState>
```

注意：mode="InProc"表示在本地储存会话状态（也可以选择储存在远程服务器或 SAL 服务器中或不启用会话状态），cookieless="True"表示如果用户浏览器不支持 Cookie 时，启用会话状态（默认为 False），timeout="20"表示会话可以处于空闲状态的分钟数。

4．应用 trace 节

此节中各属性的配置如下。

- enabled="False"表示不启用跟踪。
- requestLimit="10"表示指定在服务器上存储的跟踪请求的数目。
- pageOutput="False"表示只能通过跟踪实用工具访问跟踪输出。
- traceMode="SortByTime"表示以处理跟踪的顺序来显示跟踪信息。
- localOnly="True" 表示跟踪查看器 (trace.axd) 只用于宿主 Web 服务器。

Web.config 中的默认配置程序代码如下：

```
<trace enabled="False" requestLimit="10" pageOutput="False" traceMode="SortByTime" localOnly="True" />
```

5．应用 compilation 节

用于配置 ASP.NET 使用的所有编译设置。<compilation>主要对两个属性进行设置，分别为

defaultLanguage 和 debug，具体说明如下。

- defaultLanguage：定义了 ASP.NET 3.5 后台代码语言，可以选择 C#和 VB.NET 两者之一。
- debug：设置为“True”时，将启动 ASPX 调试；设置为“False”时，可以提高应用程序的运行性能。建议在程序开发调试时，设置 debug=“True”；在客户使用时，设置为“False”。

Web.config 中的默认配置，程序代码如下：

```
<compilation debug="True"/>
        <!--
          通过 <authentication> 节可以配置 ASP.NET 使用的
          安全身份验证模式，
          以标识传入的用户。
    -->
```

6．应用 authorization 节

用于控制对 URL 资源的客户端访问（如允许匿名用户访问）。此元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明，但必须与 authentication 节配合使用。authorization 节的属性的配置如下。

- allow：向授权规则映射添加一个规则，该规则允许对资源进行访问。
- deny：向授权规则映射添加一条规则，该规则拒绝对资源进行访问。

允许所有 admins 角色成员进行访问以及拒绝所有 users 角色成员进行访问，程序代码如下：

```
<authorization>
    <allow roles="Admins"/>
    <deny users="*"/>
</authorization>
```

7．应用 pages 节

标识特定页的配置设置（例如，是否启用会话状态、视图状态和是否检测用户的输入等）。该节可以在计算机、站点、应用程序和子目录级别声明。

通过配置<pages>节，实现不检测用户在浏览器输入的内容中是否存在潜在的危险数据，在从客户端回发页时，检查加密的视图状态，以验证视图状态是否已在客户端被篡改。程序代码如下：

```
<pages buffer="True" enableViewStateMac="True" validateRequest="False"/>
```

8．应用 httpRuntime 节

配置 ASP.NET 3.5HTTP 运行库设置，该节可以在计算机、站点、应用程序和子目录级别声明。

通过对<httpRuntime>的设置，指定用户上传文件最大为 4MB，最长时间为 60s，最多请求数为 100，程序代码如下：

```
<httpRuntime maxRequestLength="4096" executionTimeout="60" appRequestQueueLimit="100"/>
```

第 14 章

ASP.NET 3.5 常用服务器控件

- 标准服务器控件
- 验证控件
- 登录控件

14.1 标准服务器控件

14.1.1 TextBox 控件

1．功能

TextBox 控件用于获取用户输入的数据或显示文本，它通常用于可编辑文本，有时也可使其成为只读控件。另外，TextBox 控件还可以显示密码或多行，图 14.1 为 TextBox 控件。

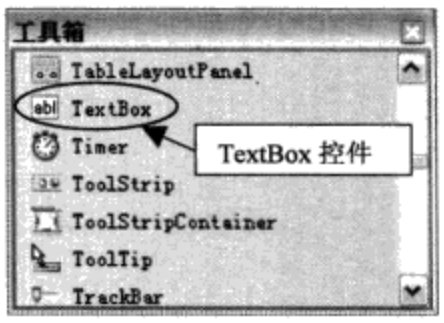


图 14.1 TextBox 控件

2．属性

TextBox 控件常用属性及描述如表 14.1 所示。

表 14.1 TextBox 控件常用属性及描述

属　性	描　述
AutoPostBack	控件失去焦点时是否发生自动回发到服务器的操作
CausesValidation	控件在回发发生时是否进行验证
ID	控件 ID
Text	控件要显示的文本
TextMode	控件的行为模式（单行、多行或密码）
Width	控件的宽度
Visible	控件是否可见
ReadOnly	获取或设置一个值，用于指示能否更改 TextBox 控件的内容
CssClass	控件呈现的样式
BackColor	控件的背景颜色
Enabled	控件是否可用
Rows	设置多行文本框中显示的行数
Columns	设置多行文本框中显示的宽度

下面对比较重要的属性进行详细介绍。

（1）Text 属性。设置或返回 TextBox 控件的文本。语法格式如下：

语法：

```
object.Text [= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 TextBox 控件显示文本内容的字符串。

例如，设置 TextBox 控件显示的文本，代码如下：

```
private void Form1_Load(object sender, EventArgs e)
```

```
{
    textBox1.Text = "风笛";                                    //设置 TextBox 控件显示的文本
}
```

（2）TextMode 属性。设置 TextBox 控件的行为模式。语法格式如下：

语法：

```
object. TextMode [=enum]
```

参数说明如下。

object：对象表达式。

enum：用来指定 TextBox 控件显示的模式。

TextBox 控件有 3 种行为模式：单行、多行和密码文本框。

- 单行（SingleLine）模式将 TextBox 控件显示为单行。如果用户输入的文本超过了 TextBox 控件的物理大小，则文本将沿水平方向滚动。
- 多行（MultiLine）模式基于 Rows 属性显示 TextBox 的高度，并且允许数据项位于多行上。如果 Wrap 属性设置为 true，则文本将自动换行。如果用户输入的文本超过了 TextBox 的物理大小，则文本将相应地沿垂直方向滚动，并且将出现滚动条。
- 密码（Password）将用户输入的字符用黑点（●）屏蔽，以隐藏这些信息。

示例 14-01　TextBox 控件 TextMode 属性的使用

本示例演示将 TextBox 控件的 TextMode 属性分别设为 SingleLine、MultiLine、Password 时的不同效果，如图 14.2 所示。

图 14.2　TextBox 控件的显示模式效果图

程序代码如下：

代码 14-01　光盘位置：光盘\mingrisoft\sl\14\14_01

```
protected void Page_Load(object sender, EventArgs e)
{
        this.txtSingleLine.TextMode = TextBoxMode.SingleLine;          //单行显示
        this.txtMultiLine.TextMode = TextBoxMode.MultiLine;            //多行显示
        this. txtPassword.TextMode = TextBoxMode.Password;             //密码框显示
}
```

（3）ReadOnly 属性。在程序开发过程中，经常需要将文本框设为只读状态，这时可以使用 ReadOnly 属性来指定能否更改 TextBox 控件的内容。将该属性设置为 true，将禁止用户输入或更改现有的值。语法格式如下：

语法：

```
object.ReadOnly[=bool]
```

参数说明如下。

object：对象表达式。

bool：用来指定 TextBox 控件是否为只读。如果不能更改 TextBox 控件的内容，则为 true；否则为 false，默认值为 false。

例如，禁止输入或更改 TextBox 控件中的内容，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    txtMultiLine.ReadOnly = true;                                      //设置文本框为只读状态
}
```

3．方法

TextBox 控件常用方法及描述如表 14.2 所示。

表 14.2 **TextBox 控件常用方法及描述**

方　法	描　述
DataBind	将数据源绑定到被调用的服务器控件及其所有子控件
Focus	为控件设置输入焦点
Dispose	使服务器控件得以在从内存中释放之前执行最后的清理操作
Equals	确定两个 Object 实例是否相等
FindControl	在当前的命名容器中搜索指定的控件 ID
HasControls	确定服务器控件是否包含任何子控件
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
RenderBeginTag	将控件的 HTML 开始标记呈现到指定的编写器中
RenderControl	输出服务器控件内容，并存储有关此控件的跟踪信息（如果已启用跟踪）
RenderEndTag	将控件的 HTML 结束标记呈现到指定的编写器中
ResolveClientUrl	获取浏览器可以使用的 URL
ResolveUrl	将 URL 转换为在请求客户端可用的 URL
ToString	返回表示当前对象的类型的完全限定名

在 TextBox 控件的众多方法中，最常用的是 Focus 方法，下面对该方法进行详细介绍。Focus 方法用于使文本框获得焦点，也就是光标在指定的文本框中闪动。语法格式如下：

语法：

```
object.Focus();
```

参数说明如下。

object：对象表达式。

例如，当窗体加载时，使文本框获得焦点，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox1.Focus();
}
```

4．事件

TextBox 控件常用事件及描述如表 14.3 所示。

表 14.3 **TextBox 控件常用事件及描述**

事　件	描　述
DataBinding	当服务器控件绑定到数据源时引发的事件
Disposed	当从内存释放服务器控件时发生，这是请求 ASP.NET 页时服务器控件生存期的最后阶段
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象时引发的事件
PreRender	在加载 Control 对象之后、呈现之前发生
Unload	当服务器控件从内存中卸载时发生
TextChanged	当用户更改 TextBox 的文本内容时发生

TextBox 控件的所有事件中最常用的是 TextChanged 事件，当用户更改 TextBox 控件中的内容时会触发 TextChanged 事件。语法格式如下：

语法：

```
public event EventHandler TextChanged
```

示例 14-02　TextBox 控件 TextChanged 事件的使用

本示例通过 TextBox 控件的 TextChanged 事件，实现当输入关键字时，立即检索数据库，查找与关键字相关的数据并显示出来，如图 14.3 所示。

图 14.3 TextChanged 事件的使用

程序主要代码如下：

代码 14-02 光盘位置：光盘\mingrisoft\sl\14\14_02

```
SqlConnection conn;                                                    //声明 SqlConnection 对象
protected void Page_Load(object sender, EventArgs e)                   //页面加载时触发 Load 事件
{
    conn = new SqlConnection("server=.;database=db_14;uid=sa;pwd=");   //实例化 SqlConnection 对象
}
protected void TextBox1_TextChanged(object sender, EventArgs e)        //内容改变时触发 TextChanged 事件
{
    string sql = "";                                                   //声明一个字符串变量
    if (TextBox1.Text == "")                                           //如果文本框为空
    {
        sql = "select * from tb_User";                                 //则检索所有数据
    }
    else                                                               //否则
    {
        //根据文本框中输入的内容查找与之相关的数据
        sql = "select * from tb_User where User_Name like '%" + TextBox1.Text.Trim() + "%'";
    }
    SqlDataAdapter sda = new SqlDataAdapter(sql, conn);                //声明一个 SqlDataAdapter 对象
    DataSet ds = new DataSet();                                        //声明一个 DataSet 对象
    sda.Fill(ds);                                                      //填充 DataSet 对象
    GridView1.DataSource = ds.Tables[0];                               //设置 GridView 控件数据源
    GridView1.DataBind();                                              //将数据绑定到 GridView 控件中
}
```

注意：在对 TextChanged 事件编程时，应先将该控件的 AutoPostBack 属性设为 true。AutoPostBack 属性用于控制 TextBox 控件的事件是否自动提交服务器，系统默认设置为 false。如果该属性设置为 true，若事件被触发则事件自动被提交到服务器；否则事件在下一次页面提交服务器时才被触发。

14.1.2 Button 控件

1．功能

Button 控件可使用户将页面数据发送到服务器，并触发页面上的事件，开发人员可以在事件中加入自定义代码。例如，将数据插入到数据库中，图 14.4 所示为 Button 控件。

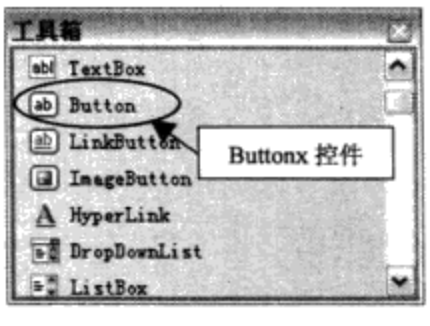


图 14.4 Button 控件

2．属性

Button 控件常用属性及描述如表 14.4 所示。

表 14.4 **Button 控件常用属性及描述**

属　性	描　述
ID	分配给服务器控件的编程标识符
Text	在 Button 控件中显示的文本标题
AccessKey	该控件使用的键盘快捷键
CssClass	控件呈现的样式
CausesValidation	在单击 Button 控件时是否执行了验证
CommandArgument	获取或设置可选参数，该参数与关联的 CommandName 一起被传递到 Command 事件
CommandName	设置命令名，该命令名与传递给 Command 事件的 Button 控件相关联
PostBackUrl	设置单击 Button 控件时从当前页发送到的网页的 URL

下面对比较重要的属性进行详细介绍。

（1）Text 属性。Text 属性用于设置在按钮上显示的文本。语法格式如下：

语法：

```
object.Text [= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 Button 控件显示文本内容的字符串。

例如，将 ID 属性为 btnButton 的按钮控件的显示文本设置为“Button 控件”，代码如下：

```
btnButton.Text=" Button 控件";
```

设计效果如图 14.5 所示。

图 14.5　Button 控件的 Text 属性

（2）AccessKey 属性。使用 AccessKey 属性为 Button 控件指定键盘快捷键。用户可以通过按键盘上的【Alt】键和指定的字符键快速触发该控件的 Click 事件。例如，如果将控件的访问键设置为字符串"A"，表明用户可以通过按 Alt+A 组合键触发该控件。

（3）PostBackUrl 属性。设置单击 Button 控件时从当前页发送到的网页的 URL。语法格式如下：

语法：

```
public virtual string PostBackUrl { get; set; }
```

属性值：单击 Button 控件时从当前页发送到的网页的 URL。默认值为空字符串(""),表示将页回发到自身。

例如，单击按钮转到指定的页面，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Button1.PostBackUrl = @"~/Default2.aspx";
}
```

3．方法

交叉链接：Button 控件常用方法同 TextBox 控件类似，请参见 TextBox 控件常用方法。

例如，通过 Button 控件的 Focus 方法，使 Button 控件获得焦点，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Button1.Focus();
}
```

4．事件

Button 控件常用事件及描述如表 14.5 所示。

表 14.5　　Button 控件常用事件及描述

事　件	描　述
Command	在单击 Button 控件时引发的事件（当命令名与控件关联时，通常使用该事件）
Click	在单击 Button 控件时引发的事件（当没有与控件关联的命令名时，通常使用该事件）
DataBinding	当服务器控件绑定到数据源时引发的事件
Disposed	当从内存释放服务器控件时发生，这是请求 ASP.NET 2.0 页时服务器控件生存期的最后阶段
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象时引发的事件
PreRender	在加载 Control 对象之后、呈现之前发生
Unload	当服务器控件从内存中卸载时发生

Button 控件最常用的事件是 Click 事件，当用户单击按钮时将会触发按钮的该事件。

示例 14-03　Button 控件 Click 事件的使用

本示例在 Button 控件的 Click 事件中编写代码，实现将输入的员工信息插入数据库中。运行结果如图 14.6 所示。

图 14.6　Click 事件的示例运行结果图

程序代码如下：

代码 14-03　光盘位置：光盘\mingrisoft\sl\14\14_03

```
protected void Button2_Click(object sender, EventArgs e)                    //Button 控件的 Click 事件
{
    if (txtAddr.Text.Trim() == "" || txtName.Text.Trim() == "" || txtTel.Text.Trim() == "")//判断信息添加是否完整
    {
        Page.RegisterStartupScript("", "<script>alert('请将信息添加完整')</script>");//如果不完整则弹出提示信息
    }
    else                                                                    //如果信息添加完整
    {
        conn = new SqlConnection("server=.;database=db_14;uid=sa;pwd=");    //创建 SqlConnection 对象
        conn.Open();                                                        //打开数据库连接
        SqlCommand cmd = new SqlCommand("insert into tb_User(User_Name,User_Sex,User_Tel,User_Addr)
values('" + txtName.Text.Trim() + "','" + ddlSex.SelectedItem.Text + "','" + txtTel.Text.Trim() + "','" + txtAddr.Text.Trim() +
"')", conn);                                                                //创建 SqlCommand 对象
        int i = cmd.ExecuteNonQuery();                                      //执行 SQL 语句
        if (i > 0)                                                          //返回值大于 0 操作成功
        {
            Page.RegisterStartupScript("", "<script>alert('添加成功')</script>");   //弹出提示信息
        }
        else                                                                //否则
        {
            Page.RegisterStartupScript("", "<script>alert('添加失败')</script>");   //添加失败
        }
        conn.Close();                                                       //关闭数据库连接
    }
}
```

14.1.3 LinkButton 控件

1．功能

LinkButton 控件是一种在网页上显示超链接样式的按钮控件。该控件在功能上与 Button 控件相似，但在呈现样式上不同，LinkButton 控件以超级链接的形式显示，图 14.7 所示为 LinkButton 控件。

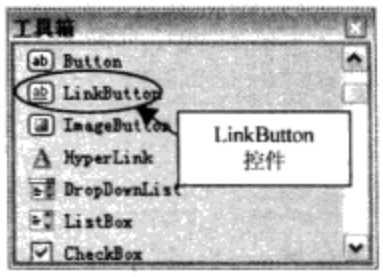


图 14.7　LinkButton 控件

2．属性

LinkButton 控件常用属性及描述如表 14.6 所示。

表 14.6　　LinkButton 控件常用属性及描述

属　性	描　述
ID	控件 ID
Text	设置在 LinkButton 控件中显示的文本标题
Width	控件的宽度
CausesValidation	单击 LinkButton 控件时是否执行了验证
Enabled	是否启用 Web 服务器控件
OnClientClick	在引发某个 LinkButton 控件的 Click 事件时所执行的客户端脚本
PostBackUrl	设置单击 LinkButton 控件时从当前页发送到的网页的 URL

PostBackUrl 属性可以使 LinkButton 控件执行跨页发送。将 PostBackUrl 属性设置为单击 LinkButton 控件时要发送到的网页的 URL。语法格式如下：

语法：

```
object.PostBackUrl[= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 LinkButton 控件发送到的网页的 URL。

示例 14-04　LinkButton 控件 PostBackUrl 属性的使用

本示例将 LinkButton 控件的 PostBackUrl 属性指定为 Default2.aspx，单击 LinkButton 控件时，页面会转向 Default2.aspx。如果未指定 PostBackUrl 属性的值，则回发到自身，运行结果如图 14.8 所示。

图 14.8　LinkButton 控件的 PostBackUrl 属性

程序代码如下：

代码 14-04　光盘位置：光盘\mingrisoft\sl\14\14_04

```
protected void Page_Load(object sender, EventArgs e)
{
    //设置 LinkButton 控件显示的文本
    LinkButton1.Text = "学习资源中心";
    LinkButton2.Text="建议与问题";
    LinkButton3.Text = "图书介绍 ";
    LinkButton4.Text = "图书排行";
    LinkButton5.Text = " 经验技巧";
    LinkButton1.PostBackUrl = @"～/Default2.aspx";              //设置当单击控件时页面的转向地址
}
```

3．方法

交叉链接：LinkButton 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

交叉链接：LinkButton 控件常用事件同 Button 控件类似，可参见 Button 控件常用事件。

14.1.4　ImageButton 控件

1．功能

使用 ImageButton 控件显示对单击鼠标作出响应的图像，它是与按钮功能相同的图像控件，图 14.9 所示为 ImageButton 控件。

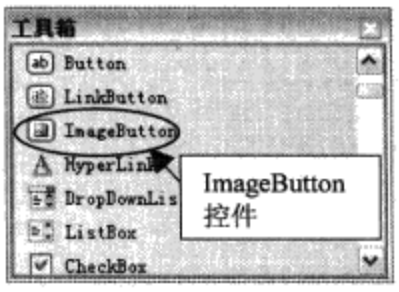


图 14.9　ImageButton 控件

2．属性

ImageButton 控件常用属性及描述如表 14.7 所示。

表 14.7　ImageButton 控件常用属性及描述

属　性	描　述
ID	控件 ID
Text	ImageButton 控件中显示的文本标题
CausesValidation	单击 ImageButton 控件时是否执行了验证
ImageUrl	ImageButton 控件中显示的图像的位置
Enabled	是否可以单击 ImageButton 以执行到服务器的回发
PostBackUrl	单击 ImageButton 控件时从当前页发送到的网页的 URL
AlternateText	当图像不可用时，ImageButton 控件中显示的替换文本

下面对比较重要的属性进行详细介绍。

（1）ImageUrl 属性。设置在 ImageButton 控件中显示的图像的位置。语法格式如下：

语法：

```
object.ImageUrl[= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 ImageButton 控件显示图像的 URL。

例如，设置 ImageButton 控件显示的图像，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    ImageButton1.ImageUrl = @"～/e-mail.png";                    //设置图像路径
}
```

说明：可以使用相对 URL 或绝对 URL。相对 URL 使图像的位置与网页的位置相关联，而不指定服务器上的完整路径，该路径是相对于网页的位置。由于不必更新代码，这使得将整个站点移动到服务器上的其他目录变得更加容易。绝对 URL 提供完整路径，因此将站点移到其他目录时需要更新代码。

（2）AlternateText 属性。该属性指定在 ImageUrl 属性中指定的图像不可用时 ImageButton 控件中显示的文本。语法格式如下：

语法：

```
object.AlternateText[= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 ImageButton 控件指定的图像不可用时显示的文本。

例如，项目根目录中不存在图像 e-mail.png，然后设置 AlternateText 属性为“图像加载失败”，这样，当加载图像时，由于不存在指定的图像，所以会显示提示信息“图像加载失败“，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    ImageButton1.ImageUrl = @"～/e-mail.png";                    //设置图像路径
    ImageButton1.AlternateText = "图像加载失败";                 //设置图像不可用时显示的文字
}
```

3．方法

交叉链接：ImageButton 控件常用方法同 TextBox 控件类似，请参见 TextBox 控件常用方法。

4．事件

交叉链接：ImageButton 控件常用事件及描述如表 14.8 所示。

表 14.8　ImageButton 控件常用事件及描述

事　件	描　述
Command	在单击 Button 控件时引发的事件（当命令名与控件关联时，通常使用该事件）
Click	在单击 Button 控件时引发的事件（当没有与控件关联的命令名时，通常使用该事件）
DataBinding	当服务器控件绑定到数据源时引发的事件
Disposed	当从内存释放服务器控件时发生，这是请求 ASP.NET 页时服务器控件生存期的最后阶段
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象时引发的事件
PreRender	在加载 Control 对象之后、呈现之前发生
Unload	当服务器控件从内存中卸载时发生

当用户单击 ImageButton 控件时将会触发该控件的 Click 事件。语法格式如下：

语法：

```
public event ImageClickEventHandler Click
```

示例 14-05　ImageButton 控件 Click 事件的使用

本示例首先设置 ImageButton 控件显示的图像，然后在其 Click 事件中编写代码，用于获取

当前鼠标的位置，运行结果如图 14.10 所示。

图 14.10　ImageButton 控件的 Click 事件示例结果图

程序代码如下：

代码 14-05　光盘位置：光盘\mingrisoft\sl\14\14_05

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    //显示鼠标当前的位置
    Response.Write("X 坐标："+e.X.ToString()+"<br/>"+"Y 坐标："+e.Y.ToString());
}
```

14.1.5　HyperLink 控件

1．功能

HyperLink 控件在网页上创建链接，使用户可以在不同页之间转换，该控件在功能上和 HTML 的“<a href="">”相似，它显示模式为超级链接的形式，图 14.11 所示为 HyperLink 控件。

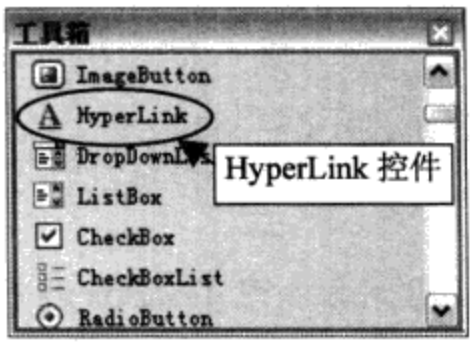


图 14.11　HyperLink 控件

2．属性

HyperLink 控件常用属性及描述如表 14.9 所示。

表 14.9　HyperLink 控件常用属性及描述

属　性	描　述
ID	控件 ID
Text	HyperLink 控件的文本标题
ImageUrl	HyperLink 控件显示的图像路径
NavigateUrl	单击 HyperLink 控件时链接到的 URL
Target	单击 HyperLink 控件时显示链接到的 Web 页内容的目标窗口或框架
Enabled	是否启用 Web 服务器控件

下面对比较重要的属性进行详细介绍。

（1）NavigateUrl 属性。用来设置单击 HyperLink 控件时要链接到的网页地址。语法格式如下：

语法：

```
object.NavigateUrl[= string]
```

参数说明如下。

object：对象表达式。

string：用来指定 HyperLink 控件指定链接到的网页的 URL。

例如，通过 HyperLink 控件的 Text 属性设置显示的文本，然后通过 NavigateUrl 属性设置链接的网页地址，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    HyperLink1.Text = "超链接";                          //设置控件显示的文本
    HyperLink1.NavigateUrl = @"～/Default2.aspx";        //设置单击控件时链接的网页地址
}
```

（2）Target 属性。Target 属性表示下一个框架或窗口显示样式，Target 属性值一般以下划线开头，其常用成员及描述如表 14.10 所示。

表 14.10　　Target 属性常用成员及描述

成　员	描　述
_blank	在没有框架的新窗口中显示链接页
_self	在具有焦点的框架中显示链接页
_top	在没有框架的全部窗口中显示链接页
_parent	在直接框架集父级窗口或页面中显示链接页
_search	在搜索窗格中呈现内容

例如，通过将 Target 属性设为“_blank”，实现当单击 HyperLink 控件后，在新窗口中打开链接窗口，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    HyperLink1.Text = "超链接";                          //设置控件显示的文本
    HyperLink1.Target = "_blank";                        //将 Target 属性设为“_blank”
    HyperLink1.NavigateUrl = @"～/Default2.aspx";        //设置链接网页地址
}
```

3．方法

交叉链接：HyperLink 控件常用方法同 TextBox 控件类似，请参见 TextBox 控件常用方法。

4．事件

交叉链接：HyperLink 控件常用事件同 Button 控件类似，请参见 Button 控件常用事件。

说明：Button、LinkButton、ImageButton 和 Hyperlink 控件都可用于在页面之间导航或将控制权转移到指定的页面。尽管这些控件执行相同的功能，但在页面设计时，还应选择最为适合的控件。下面介绍使用这 4 个控件的一般规则。

- Button 控件和 ImageButton 控件用于将数据传递回服务器。
- 当用户需要在页面之间导航时，主要使用 Hyperlink 控件。
- 虽然 LinkButton 控件与超级链接很相似，但是在单击该控件时，它会将数据传递回服务器。
- LinkButton 控件主要用于将数据保存到服务器或访问服务器上的数据。

14.1.6　ListBox 控件

1．功能

ListBox 控件允许用户从列表中选择单项或多项，图 14.12 所示为 ListBox 控件。

2．属性

ListBox 控件常用属性及描述如表 14.11 所示。

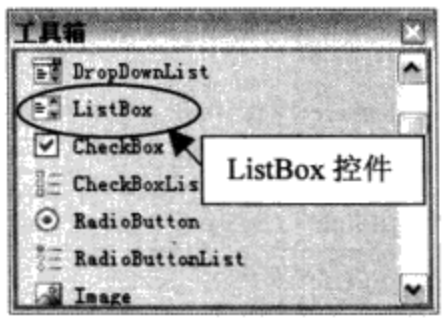


图 14.12 ListBox 控件

表 14.11 ListBox 控件常用属性及描述

属 性	描 述
Items	获取列表控件项的集合
SelectionMode	ListBox 控件的选择格式
SelectedIndex	列表中选定项的最低序号索引
SelectedItem	获取列表中索引最小的选中的项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
Rows	ListBox 控件中显示的行数
Text	ListBox 控件的 SelectedValue 属性

下面对比较重要的属性进行详细介绍。

（1）Items 属性。Items 属性表示列表对象中的项的集合。可以通过 Items 属性的方法添加、删除列表中的项。语法格式如下：

语法：

```
public virtual ListItemCollection Items { get; }
```

属性值：表示列表项的 ListItemCollection，默认值是空列表。

例如，向 ID 为 lbTest 的 ListBox 控件添加项，代码如下：

```
//使用 Items 属性的 Add 方法为 ListBox 控件添加项
lbTest.Items.Add("苹果");
lbTest.Items.Add("香蕉");
lbTest.Items.Add("西瓜");
lbTest.Items.Add("水蜜桃");
```

例如，从 ID 为 lbTest 的 ListBox 控件中移除项，代码如下：

```
this.lbTest.Items.Remove("苹果");                    //从列表中移除指定项
this.lbTest.Items.RemoveAt(0);                       //从列表中移除索引为 0 的项
```

（2）SelectionMode 属性。用户使用 SelectionMode 属性指定 ListBox 控件的模式行为。将该属性设置为 ListSelectionMode.Single 只能从 ListBox 控件中选择一项，而设定为 ListSelectionMode.Multiple 可从 ListBox 中选择多项。语法格式如下：

语法：

```
object.SelectionMode[=ListSelectionMode]
```

参数说明如下。

object：对象表达式。

ListSelectionMode：用来指定 ListBox 控件的选择模式的值。

例如，将 ListBox 控件的选择模式设为选择多项。代码如下：

```
this.lbTest.SelectionMode = ListSelectionMode.Multiple;
```

3．方法

ListBox 控件最常用方法及描述如表 14.12 所示。

表 14.12　　ListBox 控件的常用方法

方　法	描　述
GetSelectedIndices	获取当前在 ListBox 控件中选定的项的索引值数组
IsLiteralContent	确定服务器控件是否只包含文字内容
LoadControlState	从 SaveControlState 方法保存的上一个页请求还原控件状态信息
LoadPostData	如果列表控件的已发布内容不同于上次发布的内容，则加载该内容
LoadViewState	加载 DetailsView 控件的以前保存的视图状态
MapPathSecure	检索虚拟路径（绝对的或相对的）映射到的物理路径
MarkAsDataBound	将视图状态中的控件状态设置为成功绑定到数据

下面主要介绍比较常用的 GetSelectedIndices 方法，该方法返回 ListBox 控件中选定项的索引值数组。语法格式如下：

语法：

```
public virtual int[] GetSelectedIndices()
```

返回值：一个整数数组，其中每个整数表示列表框中一个选定项的索引。

示例 14-06　ListBox 控件 Items 属性的使用

本示例演示了在选择列表框中的多个项后，单击【获取选择的索引数组】按钮，以消息框的形式显示出选定列表项的索引，运行结果如图 14.13 所示。

图 14.13　获取选择项的索引数组

程序代码如下：

代码 14-06　光盘位置：光盘\mingrisoft\sl\14\14_06

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)                                          //页面第一次加载时
    {
        //向列表中添加项
        this.lbTest.Items.Add("风笛");
        this.lbTest.Items.Add("用一生下载你");
        this.lbTest.Items.Add("芸烨湘枫");
        this.lbTest.Items.Add("一生所爱");
    }
}
protected void btnTest_Click(object sender, EventArgs e)  //选择多个项目后，单击【获取选择的索引数组】按钮
{
    foreach (int i in this.lbTest.GetSelectedIndices())       //遍历获取的选择项的索引数组
    {
        Response.Write("<script>alert('" + i + "');</script>");//输出选择项的索引
    }
}
```

4．事件

ListBox 控件常用事件及描述如表 14.13 所示。

表 14.13　　ListBox 控件常用事件及描述

事　件	描　述
DataBinding	当服务器控件绑定到数据源时引发的事件
DataBound	在服务器控件绑定到数据源后发生
Disposed	当从内存释放服务器控件时发生
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象时引发的事件
PreRender	在加载 Control 对象之后、呈现之前发生
SelectedIndexChanged	当列表控件的选定项在信息发往服务器之间变化时发生
TextChanged	当 Text 和 SelectedValue 属性更改时发生
Unload	当服务器控件从内存中卸载时发生

下面对比较重要的事件进行详细介绍。

(1)SelectedIndexChanged 事件。当列表控件中的选择在向服务器的各次发送过程间更改时，会引发 SelectedIndexChanged 事件。语法格式如下：

语法：

```
public event EventHandler SelectedIndexChanged
```

示例 14-07　ListBox 控件 SelectedIndexChanged 事件的使用

本示例演示了在选择列表框中的某项时，将触发 ListBox 控件的 SelectedIndexChanged 事件。在该事件中获取选中项的值。程序代码如下：

代码 14-07　光盘位置：光盘\mingrisoft\sl\14\14_07

```
protected void lbTest_SelectedIndexChanged(object sender, EventArgs e)
{
  Response.Write("<script>alert('"+this.lbTest.SelectedValue+"');</script>");
}
```

注意：在运行此程序之前应先将该控件的 AutoPostBack 属性设置为 true，表示只要用户更改列表中的选定内容就自动产生向服务器的回发。

(2) TextChanged 事件。当用户对 ListBox 控件的 Text 属性进行更改时，引发 TextChanged 事件。TextChanged 事件发生在 SelectedIndexChanged 事件之后。语法格式如下：

语法：

```
public event EventHandler TextChanged
```

例如，当更改选择项后，如果 SelectedIndexChanged 事件中没有编写代码，那么会触发 TextChanged 事件中的代码，程序代码如下：

```
protected void ListBox1_TextChanged(object sender, EventArgs e)
{
    Response.Write("<script>alert('" + this.lbTest.SelectedValue + "');</script>");
}
```

14.1.7　DropDownList 控件

1．功能

DropDownList 控件与 ListBox 控件的使用方法类似，但 DropDownList 控件只允许用户每次从列表中选择一项。DropDownList 控件只在框中显示选定项，同时还显示下拉按钮，图 14.14 所示为 DropDownList 控件。

图 14.14　DropDownList 控件

2．属性

DropDownList 控件常用属性及描述如表 14.14 所示。

表 14.14　　DropDownList 控件常用属性及描述

属　性	描　述
Items	获取列表控件项的集合
SelectionMode	ListBox 控件的选择格式
SelectedIndex	列表中选定项的最低序号索引
SelectedItem	获取列表中索引最小的选中的项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
Text	获取或设置 ListBox 控件当前选择项的文本
AutoPostBack	当用户更改列表中的选定内容时是否自动产生向服务器的回发
DataSource	设置控件数据源

DropDownList 控件的属性大部分与 ListBox 控件相同，这里只介绍 DataSource 属性。

通过使用 DataSource 属性可以从数组或集合中获取列表项并将其添加到控件中。当编程人员希望从数组或集合中填充列表时，可以使用此属性。语法格式如下：

语法：

```
object1.DataSource[= object2]
```

参数说明如下。

object1：对象表达式。

object2：对象表达式，其值是被添加到列表中的数据源对象。

例如，先创建一个数组 arrList，代码如下：

```
ArrayList arrList = new ArrayList();                    //创建一个动态数组
//向动态数组中添加项目
arrList.Add("red");
arrList.Add("blue");
arrList.Add("green");
arrList.Add("yellow");
```

使用 DataSource 属性将该数组附加到要填充的 DropDownList 控件中，代码如下：

```
this.ddlTest.DataSource = arrList;                      //设置控件的 DataSource 属性绑定数据源
```

3．方法

DropDownList 控件最常用方法及描述如表 14.15 所示。

表 14.15　　DropDownList 控件常用方法及描述

方　法	描　述
DataBind	将数据源绑定到被调用的服务器控件及其所有子控件
Dispose	使服务器控件得以在从内存中释放之前执行最后的清理操作
ResolveUrl	将 URL 转换为在请求客户端可用的 URL

DataBind 方法是在 DropDownList 控件使用 DataSource 属性附加了数据源后，将数据源绑

定到被调用到 DropDownList 控件。语法格式如下：

语法：

```
object. DataBind（[bool]）;
```

参数说明如下。

object：必需的，是一个对象表达式。

bool：可选的，是一个布尔表达式，它指定绑定数据源后是否引发 DataBinding 事件。

例如，将上例中的 arrList 数组绑定到 DropDownList 控件，代码如下：

```
this.ddlTest.DataBind();
```

4．事件

DropDownList 控件常用事件及描述如表 14.16 所示。

表 14.16　　DropDownList 控件常用事件及描述

事　件	描　述
DataBinding	当服务器控件绑定到数据源时发生
DataBound	在服务器控件绑定到数据源后发生
SelectedIndexChanged	当列表控件的选定项在信息发往服务器过程中变化时发生
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象中时发生
PreRender	在加载 Control 对象之后、呈现之前发生
TextChanged	当 Text 和 SelectedValue 属性更改时发生
Unload	当服务器控件从内存中卸载时发生

DropDownList 的事件与 ListBox 的事件十分相似，这里仅对 SelectedIndexChanged 事件做详细介绍。与 ListBox 控件相同，当 DropDownList 控件的选项在向服务器发送过程中更改时，将引发 SelectedIndexChanged 事件。语法格式如下：

语法：

```
public event EventHandler SelectedIndexChanged
```

示例 14-08　DropDownList 控件 SelectedIndexChanged 事件的使用

本示例演示了在选取列表框中的某项时，将触发 DropDownList 控件 SelectedIndex 的 Changed 事件，从而改变 DropDownList 控件的背景色，运行结果如图 14.15 所示。

图 14.15　SelectedIndexChanged 事件示例运行结果

程序代码如下：

代码 14-08　光盘位置：光盘\mingrisoft\sl\14\14_08

```
protected void ddlTest_SelectedIndexChanged(object sender, EventArgs e)
{
    string color=this.ddlTest .SelectedValue ;                    //获取列表框中选择的值
    if(color =="red")                                             //如果选择红色
        this.ddlTest.BackColor=System .Drawing .Color.Red ;       //设置控件的背景为红色
```

```
        if (color == "blue")                                    //如果选择蓝色
            this.ddlTest.BackColor = System.Drawing.Color.Blue;  //设置控件的背景为蓝色
        if (color == "green")                                   //如果选择绿色
            this.ddlTest.BackColor = System.Drawing.Color.Green ;//设置控件的背景为绿色
        if (color == "yellow")                                  //如果选择黄色
            this.ddlTest.BackColor = System.Drawing.Color.Yellow;//设置控件的背景为黄色
    }
```

14.1.8 CheckBox 控件

1．功能

CheckBox 控件为用户提供了一种指定是/否（真/假）选择的方法，图 14.16 所示为 CheckBox 控件。

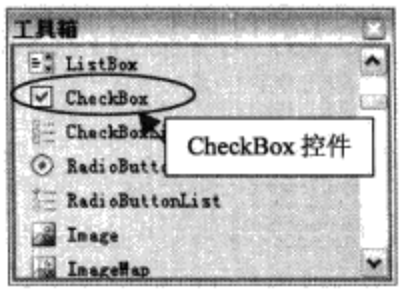


图 14.16　CheckBox 控件

2．属性

CheckBox 控件常用属性及描述如表 14.17 所示。

表 14.17　　CheckBox 控件常用属性及描述

属　性	描　述
AutoPostBack	单击时 CheckBox 状态是否自动回发到服务器
CausesValidation	单击 CheckBox 控件时是否执行验证
Checked	是否已选中 CheckBox 控件
InputAttributes	获取对呈现的 CheckBox 控件 INPUT 元素的属性集合的引用
LabelAttributes	获取对呈现的 CheckBox 控件 LABEL 元素的属性集合的引用
Text	与 CheckBox 关联的文本标签
TextAlign	与 CheckBox 控件关联的文本标签的对齐方式
ToolTip	当鼠标指针悬停在 Web 服务器控件上时显示的文本
ValidationGroup	在 CheckBox 控件回发到服务器时要进行验证的控件组

下面对比较重要的属性进行详细介绍。

（1）AutoPostBack 属性。指定在单击时是否将 CheckBox 控件的状态回发到服务器。语法格式如下：

语法：

```
public virtual bool AutoPostBack { get; set; }
```

属性值：如果该属性值为 true，则在单击 CheckBox 控件时自动将其状态发送到服务器，否则为 false，默认值为 false。

例如，允许 CheckBox 控件自动回发到服务器，程序代码如下：

```
this.chkTest1.AutoPostBack = True;
```

注意：将此属性设置为 true 会导致每次单击控件时发生到服务器的往返行程。

（2）Checked 属性。该属性可以确定是否已选中 CheckBox 控件。用户也可以使用该属性以编程方式设置 CheckBox 控件的状态。语法格式如下：

语法：

```
public virtual bool Checked { get; set; }
```

属性值：如果为 true，则设置选中状态；否则为 false。默认值为 false。

例如，将 CheckBox 控件设为选中状态，程序代码如下：

```
this.chkTest1.Checked = True;
```

（3）Text 属性。该属性指定与 CheckBox 控件关联的文本标签。该属性也可以以编程方式获取或设置与 CheckBox 控件关联的文本标签。语法格式如下：

语法：

```
public virtual string Text { get; set; }
```

属性值：与 CheckBox 关联的文本标签。

例如，设置 CheckBox 控件显示的文本，程序代码如下：

```
this.chkTest1.Text = "唱歌";
this.chkTest2.Text = "跳舞";
this.chkTest3.Text = "看书";
```

运行结果如图 14.17 所示。

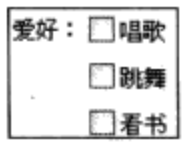


图 14.17　设置 CheckBox 控件的 Text 属性

3．方法

交叉链接：CheckBox 控件常用方法同 TextBox 控件类似，具体可参见 TextBox 控件常用方法。

4．事件

CheckBox 控件常用事件及描述如表 14.18 所示。

表 14.18　　CheckBox 控件常用事件及描述

事　件	描　述
CheckedChanged	当 Checked 属性的值在向服务器进行发送期间更改时发生
DataBinding	当服务器控件绑定到数据源时发生
Disposed	当从内存释放服务器控件时发生
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象中时发生
PreRender	在加载 Control 对象之后、呈现之前发生
Unload	当服务器控件从内存中卸载时发生

当 Checked 属性的值在向服务器的各次发送过程间更改时，将引发 CheckedChanged 事件。此事件不将页面回发到服务器，除非 AutoPostBack 属性被设置为 true。用户可以在 CheckBox 控件的 CheckedChanged 事件中指定和编写处理程序。语法格式如下：

语法：

```
public event EventHandler CheckedChanged
```

示例 14-09　CheckBox 控件 CheckedChanged 事件的使用

本示例在 CheckBox 控件 CheckedChanged 事件中编写代码，实现当选择 CheckBox 控件后，获取选择项的文本，运行结果如图 14.18 所示。

图 14.18　CheckBox 控件 CheckedChanged 事件的使用

程序代码如下：

代码 14-09　光盘位置：光盘\mingrisoft\sl\14\14_09

```
static System.Collections.ArrayList al = new System.Collections.ArrayList(); //创建一个动态数组
string str = "";                                                              //声明字符串
protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox1.Checked)                                                    //当第一个控件被选择时
        al.Add("上网");                                                       //向动态数组中添加“上网”
    else                                                                      //如果取消选择
        al.Remove("上网");                                                    //移除“上网”
}
protected void CheckBox2_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox2.Checked)                                                    //当第二个控件被选择时
        al.Add("看书");                                                       //向动态数组中添加“看书”
    else                                                                      //如果取消选择
        al.Remove("看书");                                                    //移除“看书”
}
protected void CheckBox3_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox3.Checked)                                                    //当第三个控件被选择时
        al.Add("旅游");                                                       //向动态数组中添加“旅游”
    else                                                                      //如果取消选择
        al.Remove("旅游");                                                    //移除“旅游”
}
protected void CheckBox4_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox4.Checked)                                                    //当第四个控件被选择时
        al.Add("交友");                                                       //向动态数组中添加“交友”
    else                                                                      //如果取消选择
        al.Remove("交友");                                                    //移除“交友”
}
protected void Button1_Click(object sender, EventArgs e)                      //选择之后，单击【确定】按钮
{
    str="";
    for (int i = 0; i<al.Count; i++)                                          //遍历动态数组
    {
        str += al[i].ToString() + ",";                                        //读取数组中的所有项
    }
    if (al.Count == 0)                                                        //如果数组中没有项
    {
        Response.Write("<script>alert('请选择爱好');</script>");              //弹出提示信息
    }
    else                                                                      //如果存在项目
    {
        //显示选择的项目
        Response.Write("<script>alert('您的爱好：" + str.Remove(str.LastIndexOf(",")) + "');</script>");
    }
}
```

14.1.9　RadioButton 控件

1．功能

RadioButton 控件是一种单选按钮控件，允许用户从给出的选项集中仅选出一个选项。添加的多个 RadioButton 控件必须有相同的 GroupName 属性，图 14.19 所示为 RadioButton 控件。

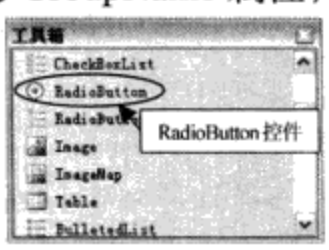


图 14.19　RadioButton 控件

2．属性

RadioButton 控件常用属性及描述如表 14.19 所示。

表 14.19　　RadioButton 控件常用属性及描述

属　性	描　述
AutoPostBack	在单击 RadioButton 控件时，RadioButton 状态是否自动回发到服务器
CausesValidation	单击 RadioButton 控件时是否执行验证
Checked	是否已选中 RadioButton 控件
GroupName	单选按钮所属的组名
InputAttributes	获取对呈现的 RadioButton 控件 INPUT 元素的属性集合的引用
LabelAttributes	获取对呈现的 RadioButton 控件 LABEL 元素的属性集合的引用
Text	与 RadioButton 关联的文本标签
TextAlign	与 RadioButton 控件关联的文本标签的对齐方式
ToolTip	当鼠标指针悬停在 Web 服务器控件上时显示的文本
ValidationGroup	设置在 RadioButton 控件回发到服务器时要进行验证的控件组

使用 GroupName 属性指定一组单选按钮，以创建一组互相排斥的控件。当只能从可用选项列表中选择一项时，可使用 GroupName 属性。设置该属性后，每次只能使该组中的一个 RadioButton 为选中状态。语法格式如下：

语法：

```
public virtual string GroupName { get; set; }
```

属性值：单选按钮所属的组名

例如，在网页上添加一个 RadioButton 控件，ID 属性分别设为 rbtnSex1 和 rbtnSex2，Text 属性分别设为“男”和“女”。由于对性别的选择只能选择一项，故将这两个 RadioButton 按钮的 GroupName 设置为同一个。代码如下：

```
//设置 RadioButton 控件显示的文本
this.rbtnSex1.Text = "男";
this.rbtnSex2.Text = "女";
//设置 RadioButton 控件所属的组
this.rbtnSex1.GroupName = "Sex";
this.rbtnSex2.GroupName = "Sex";
```

运行结果如图 14.20 所示。

性别：⊙男 ○女

图 14.20　RadioButton 控件的运行结果图

3．方法

交叉链接：RadioButton 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

交叉链接：RadioButton 控件常用事件同 CheckBox 控件类似，可参见 CheckBox 控件常用事件。

14.1.10　Calendar 控件

1．功能

Calendar 控件在 Web 页面显示一个单月份日历。用户可使用该日历查看和选择日期，图 14.20 所示为 Calendar 控件。

2．属性

Calendar 控件部分常用属性及描述如表 14.21 所示。

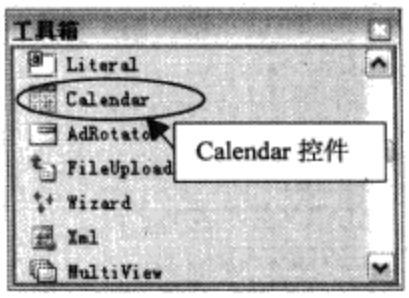


图 14.21　Calendar 控件

表 14.20　　Calendar 控件常用属性及描述

属　性	描　述
Caption	设置呈现日历标题的文本值
FirstDayOfWeek	设置将在日历的第一列中显示的一周中的某一天
NextMonthText	设置为下一月导航控件显示的文本
PrevMonthText	设置为前一月导航控件显示的文本
SelectedDate	设置当前选定的日期
SelectionMode	设置 Calendar 对象的选择模式
ShowDayHeader	确定控件是否在日历标头中显示一周中每一天的名称
ShowNextPrevMonth	指示 Calendar 控件是否在标题部分显示下个月和上个月导航元素

下面对比较重要的属性进行详细介绍。

(1) FirstDayOfWeek 属性。设置将在日历的第一列中显示的一周中哪一天。语法格式如下：

语法：

```
public FirstDayOfWeek FirstDayOfWeek { get; set; }
```

属性值：在日历的第一列中显示的一周中的某一天。

例如，将星期二设置为一周中的第一天，代码如下：

```
this.Calendar1.FirstDayOfWeek = FirstDayOfWeek.Tuesday;
```

(2) SelectionMode 属性。该属性设置了 Calendar 控件的选择模式。默认值为 Day。语法格式如下：

语法：

```
public CalendarSelectionMode SelectionMode { get; set; }
```

属性值：日历控件的选择模式。

Web 窗体 CalendarSelectionMode 枚举提供了其常用属性及描述，如表 14.21 所示。

表 14.21　　CalendarSelectionMode 控件常用属性及描述

属　性	描　述
Day	可以在 Calendar 控件上选择单个日期
DayWeek	可以在 Calendar 控件上选择单个日期或整周
DayWeekMonth	可以在 Calendar 控件上选择单个日期、周或整月
None	在 Calendar 控件上不能选择任何日期

例如，将选择模式设为选择整周，代码如下：

```
this.Calendar1.SelectionMode = CalendarSelectionMode.DayWeek;
```

(3) SelectedDate 属性。使用 SelectedDate 属性确定 Calendar 控件上选定的日期。通常，将此属性与 SelectionMode 属性一起使用以确定 Calendar 控件的选定日期。将 SelectionMode 属性设置为 Day，此设置只允许从 Calendar 控件选择一个日期。语法格式如下：

语法：

```
public DateTime SelectedDate { get; set; }
```

属性值：当前选定的日期。

例如，显示当前选中的日期，代码如下：

```
Response.Write(this.Calendar1 .SelectedDate.ToShortDateString());
```

3．方法

交叉链接：Calendar 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

Calendar 控件常用事件及描述如表 14.22 所示。

表 14.22　　Calendar 控件常用事件及描述

事　件	描　述
DataBinding	当服务器控件绑定到数据源时引发的事件
Disposed	当从内存释放服务器控件时发生，这是请求 ASP.NET 页时服务器控件生存期的最后阶段
Init	当服务器控件初始化时发生，初始化是控件生存期的第一步
Load	当服务器控件加载到 Page 对象时引发的事件
PreRender	在加载 Control 对象之后、呈现之前发生
SelectionChanged	每次用户在 Calendar 控件中选择一天、一周或一月时发生
Unload	当服务器控件从内存中卸载时发生

Calendar 控件中较常使用的是 SelectionChanged 事件，当用户选择 Calendar 控件的一天、一周或一月时将触发该事件。语法格式如下：

语法：

```
public event EventHandler SelectionChanged
```

例如，在该事件中获得用户所选择的日期，代码如下：

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    string date = this.Calendar1.SelectedDate.ToShortDateString();
    Response.Write("<script>alert('您所选择的日期为："+date+"');</script>");
}
```

14.1.11　FileUpload 控件

1．功能

FileUpload 控件显示一个文本框控件和一个浏览按钮，用户通过 FileUpload 控件可以在客户端上选择一个文件并将该文件上载到 Web 服务器。该控件允许用户上传图片、文本文件或其他文件，图 14.22 所示为 FileUpload 控件。

图 14.22　FileUpload 控件

2．属性

FileUpload 控件常用属性及描述如表 14.23 所示。

表 14.23　　FileUpload 控件常用属性及描述

属　性	描　述
ID	分配给服务器控件的编程标识符
Text	设置或返回控件的文本
Width	控件的宽度

续表

属　性	描　述
Visible	是否显示控件
CssClass	应用于该控件的 CSS 类名
BackColor	控件的背景颜色
Enabled	控件是否可用
FileBytes	从使用 FileUpload 控件指定的文件返回一个字节数组
FileContent	获取 Stream 对象，它指向要使用 FileUpload 控件上传的文件
FileName	获取客户端上使用 FileUpload 控件上传的文件的名称
HasFile	获取一个值，该值指示 FileUpload 控件是否包含文件
PostedFile	获取使用 FileUpload 控件上载的文件的基础 HttpPostedFile 对象

下面对比较重要的属性进行详细介绍。

（1）FileBytes 属性。该属性返回文件的内容。FileUpload 控件不会自动从客户端读取该文件。用户必须显式提供一个控件或机制，使用户能提交指定的文件。语法格式如下：

语法：

```
public byte[] FileBytes { get; }
```

属性值：包含指定文件内容的 Byte 数组。

例如，获得上载文件的大小，代码如下：

```
this.ful.FileBytes.Length;
```

（2）FileName 属性。该属性获取客户端上使用 FileUpload 控件上载的文件的名称，但 FileName 属性返回的文件名不包含此文件在客户端上的路径。语法格式如下：

语法：

```
public string FileName { get; }
```

属性值：该字符串指定客户端上使用 FileUpload 上载的文件的名称。

例如，获得上传文件的名称，代码如下：

```
this.ful.FileName;
```

（3）HasFile 属性。HasFile 属性获取一个值，该值指示 FileUpload 控件是否包含要上传的文件。在对文件执行操作之前，使用该属性来验证要上传的文件存在。如果 FileUpload 包含文件，则为 true；否则为 false。语法格式如下：

语法：

```
public bool HasFile { get; }
```

属性值：如果 FileUpload 包含文件，则为 true；否则为 false。

例如，判断文件是否存在，如果存在则输入文件名；如果不存在则提示文件不存在消息框。代码如下：

```
if (this.ful.HasFile)                                          //改过该属性为 true
{
   Response.Write("文件名为："+this.ful.FileName);              //显示包含的文件名
}
else                                                           //否则
{
   Response.Write("<script>alert('文件不存在！');</script>");   //提示文件不存在
}
```

3．方法

FileUpload 控件常用方法及描述如表 14.24 所示。

表 14.24　　FileUpload 控件常用方法及描述

方　法	描　述
ApplyStyle	将指定样式的所有非空白元素复制到 Web 控件，改写控件的所有现有的样式元素
ApplyStyleSheetSkin	将页样式表中定义的样式属性应用到控件
CopyBaseAttributes	将 Style 对象未封装的属性从指定的 Web 服务器控件复制到从中调用此方法的 Web 服务器控件
DataBind	将数据源绑定到被调用的服务器控件及其所有子控件
Focus	为控件设置输入焦点
Dispose	使服务器控件得以在从内存中释放之前执行最后的清理操作
Equals	确定两个 Object 实例是否相等
FindControl	在当前的命名容器中搜索指定的控件 ID
GetHashCode	用作特定类型的哈希函数
GetType	获取当前实例的 Type
HasControls	确定服务器控件是否包含任何子控件
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
RenderBeginTag	将控件的 HTML 开始标记呈现到指定的编写器中
RenderControl	输出服务器控件内容，并存储有关此控件的跟踪信息（如果已启用跟踪）
RenderEndTag	将控件的 HTML 结束标记呈现到指定的编写器中
ResolveClientUrl	获取浏览器可以使用的 URL
ResolveUrl	将 URL 转换为在请求客户端可用的 URL
SaveAs	使用 FileUpload 控件将上传的文件的内容保存到 Web 服务器上的指定路径
SetRenderMothodDelegate	分配事件处理程序委托，将服务器控件及其内容呈现到父控件中
ToString	返回表示当前对象的类型的完全限定名

下面主要介绍 SaveAs 方法，该方法将使用 FileUpload 控件上载的文件保存到 Web 服务器上的指定路径。语法格式如下：

语法：

```
public void SaveAs(string filename)
```

参数说明如下。

filename：用来指定服务器上保存上载文件的完整路径。

示例 14-10　FileUpload 控件 SaveAs 方法的使用

本示例将用户选择的文件保存在“D:\”，运行结果如图 14.23 所示。

图 14.23　使用 FileUpload 控件上传文件

程序代码如下：

代码 14-10　光盘位置：光盘\mingrisoft\sl\14\14_10

```
protected void btnUpload_Click(object sender, EventArgs e)
{
```

```
    if (this.ful.HasFile)                                          //如果控件包含文件
    {
        string savePath = @"D:\";                                  //设置保存的路径
        savePath += this.ful.FileName;
        this.ful.SaveAs(savePath);                                 // SaveAs 方法保存文件
        Response.Write("<script>alert('OK！ ');</script>");        //弹出提示信息
    }
    else                                                           //否则
    {
        Response.Write("<script>alert('文件不存在！ ');</script>");  //提示文件不存在
    }
}
```

4．事件

交叉链接：FileUpload 控件常用事件同 TextBox 控件类似，可参见 TextBox 控件常用事件。

14.1.12 标准服务器控件典型应用

实例位置：光盘\mingrisoft\dxyy\14\14_01

每个网页都是由许多的标准控件组成的，开发人员利用这些标准控件来实现各种功能。下面以网站会员注册信息页为例，对标准服务器控件进行介绍。运行结果如图 14.24 所示。

图 14.24 标准控件的应用示例运行结果

程序开发步骤如下。

（1）新建一个网站，将其命名为 14_01，默认主页名称为 Default.aspx。

（2）dxyy.aspx 页面涉及的控件如表 14.25 所示。

表 14.25 **dxyy.aspx 页面用到的控件**

控件类型	控件名称	数量	用途
HTML	Table	1	布局页面
标准	Button	3	注册按钮、取消按钮和加载照片
标准	TextBox	3	输入用户名、密码和简介信息
标准	RadioButton	2	选择性别
标准	FileUpLoad	1	上传照片
标准	Image	1	显示照片

页面中主要控件属性设置如下。

RadioButton 控件属性设置：将两个 RadioButton 控件的 GroupName 属性设为“sex”。

Button 控件属性设置：将 Text 属性设为“注册”、“取消”和“加载”。

（3）主要程序代码如下。

单击 FileUpLoad 控件的【浏览】按钮选择照片的路径，并单击【加载】按钮将照片保存在服务器中的指定位置并显示在 Image 控件上，代码如下：

```
static   string name = "";                                  //用于保存上传文件名
static string path = "";                                    //用于保存上传文件路径
protected void Button3_Click(object sender, EventArgs e)
{
    string fileType = FileUpload1.FileName;                 //获取选择的文件路径
    //获取文件类型
    fileType = fileType.Substring(fileType.LastIndexOf(".") + 1, fileType.Length - fileType.LastIndexOf(".") - 1);
    if (fileType.ToLower() == "bmp" || fileType.ToLower() == "jpeg" || fileType.ToLower()== "png" ||
fileType.ToLower()== "jpg" || fileType.ToLower() == "gif")  //如果上传文件符合指定的类型
    {
        name = this.FileUpload1.FileName;                   //获取上载文件的名称
        if (name != "")                                     //如果文件名不为空
        {
            path = Server.MapPath("./");                    //设置保存路径
            this.FileUpload1.PostedFile.SaveAs(path + name);//将文件保存在相应的路径下
            Image1.ImageUrl = "～/" + name;                 //将图片显示在 Image 控件上
        }
    }
    else                                                    //上传文件如果不符合指定的类型
    {
        RegisterStartupScript("", "<script>alert('请选择图像')</script>");//弹出提示信息
    }
}
```

单击【注册】按钮将用户填写的信息在网页上输出，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("**********注册信息**********<br/>");     //输出提示信息
    Response.Write("姓名："+txtName.Text+"<br/>");            //输出姓名
    Response.Write("密码：" + txtPwd.Text + "<br/>");         //输出密码
    if(RadioButton1.Checked)                                  //如果第一个 RadioButton 控件被选择
        Response.Write("性别：男<br/>");                      //则性别为男
    if(RadioButton2.Checked)                                  //如果第二个 RadioButton 控件被选择
        Response.Write("性别：女<br/>");                      //则性别为女
    Response.Write("头像：" + path+name + "<br/>");           //输出头像路径
    Response.Write("简介：" + txtSummary.Text+ "<br/>");      //输出简介信息
}
```

14.2 验 证 控 件

14.2.1 RequiredFieldValidator 控件

1．功能

RequiredFieldValidator 控件常用于验证文本输入框是否为空。当用户提交网页到服务器端时，系统自动检查被验证控件的输入是否为空；如果为空，则 RequiredFieldValidator 控件在网页中显示提示信息。图 14.25 所示为 RequiredFieldValidator 控件。

2．属性

RequiredFieldValidator 控件常用属性及描述如表 14.26 所示。

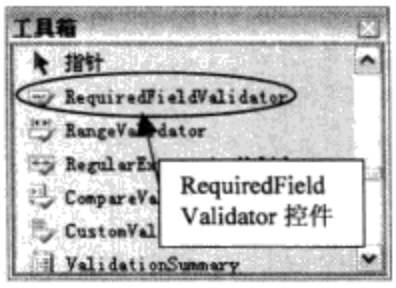


图 14.25　RequiredFieldValidator 控件

表 14.26　　RequiredFieldValidator 控件常用属性及描述

属　性	描　述
ID	控件 ID
ControlToValidate	表示要进行验证的控件 ID，此属性必须设置为输入控件 ID。如果没有指定有效输入控件，则在显示页面时引发异常。另外该 ID 的控件必须和验证控件在相同的容器中
ErrorMessage	表示当验证不合法时，出现错误的信息
IsValid	获取或设置一个值，该值指示控件验证的数据是否有效，默认值为 true
Display	设置错误信息的显示方式
Text	如果 Display 为 Static，不出错时，显示该文本

下面对比较重要的属性进行详细介绍。

（1）ControlToValidate 属性。该属性指定验证控件对哪一个控件的输入进行验证。语法格式如下：

```
public string ControlToValidate { get; set; }
```

属性值：要验证的控件的 ID。

例如，要验证 ID 属性为 txtUserName 的 TextBox 控件，只要将 RequiredFieldValidator 控件的 ControlToValidate 属性设置为 txtUserName，代码如下：

```
this.RequiredFieldValidator1.ControlToValidate = "txtUserName";
```

（2）ErrorMessage 属性。该属性用于指定页面中使用 RequiredFieldValidator 控件时显示的错误消息文本。语法格式如下：

```
public string ErrorMessage { get; set; }
```

属性值：要用于错误消息的文本。

例如，将 RequiredFieldValidator 控件的错误消息文本设为“*”，代码如下：

```
this.RequiredFieldValidator1.ErrorMessage = "*";
```

3．方法

交叉链接：RequiredFieldValidator 控件常用方法同 TextBox 控件类似，请参见 TextBox 控件常用方法。

4．事件

交叉链接：RequiredFieldValidator 控件常用事件同 TextBox 控件类似，可参见 TextBox 控件常用事件。

14.2.2　RegularExpressionValidator 控件

1．功能

该控件用于计算输入控件的值，以确定该值是否与某个正则表达式所定义的模式相匹配。图 14.26 所示为 RegularExpressionValidator 控件。

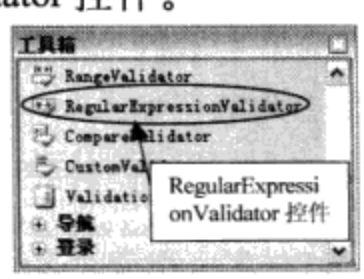


图 14.26　RegularExpressionValidator 控件

2．属性

RegularExpressionValidator 控件常用属性及描述如表 14.27 所示。

表 14.27　　RegularExpressionValidator 控件常用属性及描述

属　性	描　述
ID	控件 ID
ControlToValidate	表示要进行验证的控件 ID，此属性必须设置为输入控件 ID。如果没有指定有效输入控件，则在显示页面时引发异常。另外该 ID 的控件必须和验证控件在相同的容器中
ErrorMessage	表示当验证不合法时，出现错误的信息
IsValid	获取或设置一个值，该值指示控件验证的数据是否有效。默认值为 True
Display	设置错误信息的显示方式
Text	如果 Display 为 Static，不出错时，显示该文本
ValidationExpression	获取或设置被指定为验证条件的正则表达式。默认值为空字符串（""）

下面介绍 ValidationExpression 属性。该属性用于指定验证条件的正则表达式。常用的正则表达式字符及其含义如表 14.28 所示。

表 14.28　　常用正则表达式字符及其含义

正则表达式字符	含　义
[……]	匹配括号中的任何一个字符
[^……]	匹配不在括号中的任何一个字符
\w	匹配任何一个字符（a～z、A～Z 和 0～9）
\W	匹配任何一个空白字符
\s	匹配任何一个非空白字符
\S	与任何非单词字符匹配
\d	匹配任何一个数字（0～9）
\D	匹配任何一个非数字（^0～9）
[\b]	匹配一个退格键字母
{n,m}	最少匹配前面表达式 *n* 次，最大为 *m* 次
{n,}	最少匹配前面表达式 *n* 次
{n}	恰恰匹配前面表达式为 *n* 次
?	匹配前面表达式 0 或 1 次{0,1}
+	至少匹配前面表达式 1 次{1,}
*	至少匹配前面表达式 0 次{0,}
|	匹配前面表达式或后面表达式
(…)	在单元中组合项目
^	匹配字符串的开头
$	匹配字符串的结尾
\b	匹配字符边界
\B	匹配非字符边界的某个位置

下面列举几个常用的正则表达式。

- 验证电子邮件。
  - ✧ \w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
  - ✧ \S+@\S+\．\S+
- 验证网址。
  - ✧ HTTP：//\S+\．\S+

✧ HTTP：//\S+\．\S+

- 验证邮政编码。

  ✧ \d{6}

- [0-9]：表示 0～910 位数字。
- \d*：表示任意个数字。
- \d{3,4}-\d{7,8}：表示中国大陆的固定电话号码。
- \d{2}-\d{5}：验证由两位数字、一个连字符再加 5 位数字组成的 ID 号。
- <\s*(\S+)(\s[^>]*)?>[\s\S]*<\s*\/\l\s*>：匹配 HTML 标记。

3．方法

交叉链接：RegularExpressionValidator 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

交叉链接：RegularExpressionValidator 控件常用事件同 TextBox 控件类似，可参见 TextBox 控件常用事件。

14.2.3 CompareValidator 控件

1．功能

该控件将输入控件的值同常数或其他输入控件的值相比较，以确定这两个值是否与比较运算符（小于、等于、大于等）指定的关系相匹配。图 14.27 所示为 CompareValidator 控件。

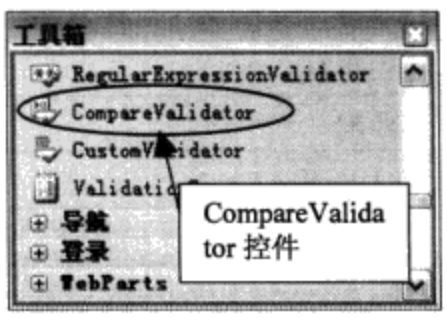


图 14.27　CompareValidator 控件

2．属性

CompareValidator 控件常用属性及描述如表 14.29 所示。

表 14.29　　CompareValidator 控件常用属性及描述

属　性	描　述
ID	控件 ID
ControlToCompare	设置用于比较的输入控件的 ID
ControlToValidate	表示要进行验证的控件 ID，此属性必须设置为输入控件 ID。如果没有指定有效输入控件，则在显示页面时引发异常。另外该 ID 的控件必须和验证控件在相同的容器中
ErrorMessage	表示当验证不合法时，出现错误的信息
IsValid	控件验证的数据是否有效，默认值为 true
Operator	设置验证中使用的比较操作，默认值为 Equal
Display	设置错误信息的显示方式
Text	如果 Display 为 Static，不出错时，显示该文本
Type	设置比较的两个值的数据类型，默认值为 string
ValueToCompare	设置要比较的值

下面对比较重要的属性进行详细介绍。

（1）ControlToCompare 属性。该属性指定要对其进行值比较的控件的 ID。语法格式如下：

语法：

```
public string ControlToCompare { get; set; }
```

属性值：要比较的输入控件的 ID。

例如，将 ID 属性为 txtPassword 的 TextBox 控件与 ID 属性为 txtAgain 的 TextBox 控件进行比较验证。代码如下：

```
this. CompareValidator1.ControlToCompare= "txtPassword";
this. CompareValidator1.ControlToCompare= "txtAgain";
```

（2）Operator 属性。该属性指定要在比较验证中使用的比较操作。ControlToValidate 属性必须位于比较运算符的左边，ControlToCompare 属性位于右边，才能进行计算。语法格式如下：

语法：

```
public ValidationCompareOperator Operator { get; set; }
```

属性值：验证中使用的比较操作。

例如，要验证 ID 属性为 txtPassword 的 TextBox 控件与 ID 属性为 txtAgain 的 TextBox 控件是否相等，代码如下：

```
this.CompareValidator1.Operator = ValidationCompareOperator.Equal;
```

（3）Type 属性。该属性指定要对其进行比较的两个值的数据类型。语法格式如下：

```
public ValidationDataType Type { get; set; }
```

属性值：比较的两个值的数据类型。

例如，要验证 ID 属性为 txtPassword 的 TextBox 控件与 ID 属性为 txtAgain 的 TextBox 控件的值类型为 string 类型，代码如下：

```
this.CompareValidator1.Type = ValidationDataType.String;
```

（4）ValueToCompare 属性。该属性指定要比较的值。如果 ValueToCompare 和 ControlToCompare 属性都存在，则使用 ControlToCompare 属性的值。语法格式如下：

```
public string ValueToCompare { get; set; }
```

属性值：要比较的值。

例如，设置比较的值为“你好”，代码如下：

```
this.CompareValidator1.ValueToCompare = "你好";
```

示例 14-11　CompareValidator 控件的使用

本示例演示了如何使用 CompareValidator 控件验证两次输入的数据是否一致，运行结果如图 14.28 所示。

图 14.28　CompareValidator 控件示例运行结果

程序代码如下：

代码 14-11　光盘位置：光盘\mingrisoft\sl\14\14_11

```
protected void Page_Load(object sender, EventArgs e)
{
    this.CompareValidator1.ControlToValidate = "txtpwd";                    //用于验证的控件名称
    this.CompareValidator1.ControlToCompare = "txtpwd1";                    //用于比较的控件名称
    this.CompareValidator1.ErrorMessage = "两次密码输入不一致";              //设置错误信息
    this.CompareValidator1.Operator = ValidationCompareOperator.Equal;      //比较的方式是判断是否相等
```

```
        this.CompareValidator1.Type = ValidationDataType.String;                    //设置比较值的类型为字符串
    }
```

3．方法

交叉链接：CompareValidator 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

交叉链接：CompareValidator 控件常用事件同 TextBox 控件类似，可参见 TextBox 控件常用事件。

14.2.4 ValidationSummary 控件

1．功能

ValidationSummary 控件本身不进行任何验证，它只是用于分组显示其他验证控件的错误信息。图 14.29 所示为 ValidationSummary 控件。

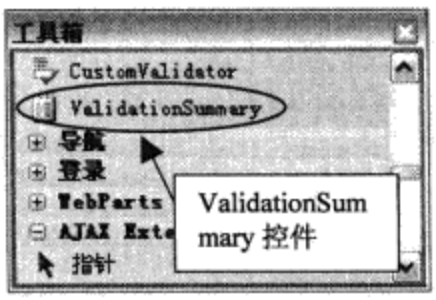


图 14.29　ValidationSummary 控件

2．属性

ValidationSummary 控件常用属性及描述如表 14.30 所示。

表 14.30　　ValidationSummary 控件常用属性及描述

属　性	描　述
HeaderText	控件汇总信息
DisplayMode	设置错误信息的显示格式
ShowMessageBox	是否以弹出方式显示每个被验证控件的错误信息
ShowSummary	是否使用错误汇总信息
EnableClientScript	是否使用客户端验证，系统默认值为 true
Validate	执行验证并且更新 IsValid 属性

下面对比较重要的属性进行详细介绍。

（1）DisplayMode 属性。该属性指定 ValidationSummary 控件的显示格式。摘要可以按列表、项目符号列表或单个段落的形式显示。语法格式如下：

```
public ValidationSummaryDisplayMode DisplayMode { get; set; }
```

例如，设置 ValidationSummary 的显示模式为项目符号列表，代码如下：

```
this.ValidationSummary1.DisplayMode = ValidationSummaryDisplayMode.BulletList;
```

（2）ShowMessageBox 属性。是否在消息框中显示验证摘要。语法格式如下：

```
public bool ShowMessageBox { get; set; }
```

属性值：如果在消息框中显示验证摘要，则为 true；否则为 false。默认值为 false。当 ShowMessageBox 属性设为 true 时，网页上的错误信息不在网页本身显示，而是以弹出对话框的形式来显示错误信息。如图 14.30 所示。

图 14.30　ShowMessageBox 属性

(3) ShowSummary 属性。除了 ShowMessageBox 属性外，该属性也可用于控制验证摘要的显示位置。如果该属性设置为 true，则在网页上显示验证摘要。语法格式如下：

```
public bool ShowSummary { get; set; }
```

属性值：如果内联显示验证摘要，则为 true；否则为 false，默认值为 true。

注意：如果 ShowMessageBox 和 ShowSummary 属性都设置为 true，则在消息框和网页上都显示验证摘要。

3．方法

交叉链接：ValidationSummary 控件常用方法同 TextBox 控件类似，可参见 TextBox 控件常用方法。

4．事件

交叉链接：ValidationSummary 控件常用事件同 TextBox 控件类似，可参见 TextBox 控件常用事件。

14.2.5 验证控件典型应用

实例位置：光盘\mingrisoft\dxyy\14\14_02

在 Web 网页中，经常会用验证控件对用户输入的信息进行验证。例如，用户是否输入信息、日期格式是否正确等。下面通过用户注册来介绍验证控件的作用。运行结果如图 14.31 所示。

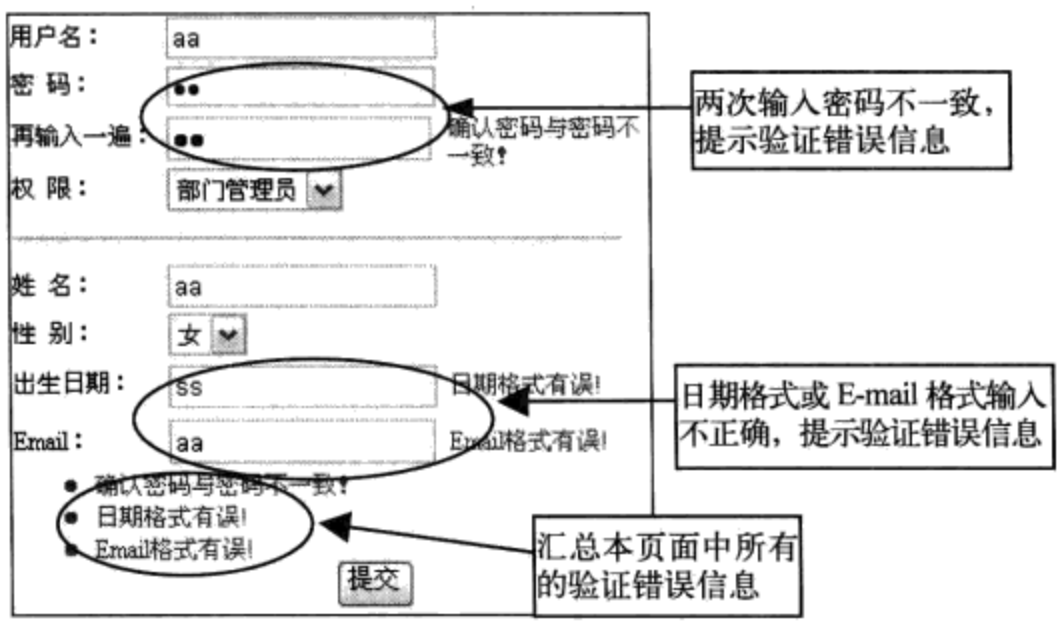


图 14.31 利用验证控件验证用户输入信息

程序开发步骤如下。

(1) 新建一个网站，将其命名为 14_02，默认主页名称为 User.aspx。

(2) User.aspx 页面涉及到的控件如表 14.31 所示。

表 14.31 User.aspx 页面用到的控件

控件类型	控件名称	数量	用途
HTML	Table	1	布局页面
标准	DropDownList	2	显示数据
标准	Button	1	提交按钮
标准	TextBox	6	数据录入
验证	RequiredFieldValidator	3	验证用户名、密码、姓名不为空
验证	CompareValidator	2	验证密码与确认密码一致以及日期输入格式
验证	RegularExpressionValidator	1	验证 E-mail 的输入格式
验证	ValidationSummary	1	显示验证控件的错误信息

(3) 主要设计如下。

- 将输入框的 ID 属性分别设置为 txtUserName、txtPassWord、txtAgain、ddlRole、txtName、ddlSex、txtBirthday 和 txtEmail。
- 该页面中，使用了 3 个 RequiredFieldValidator 控件来验证文本框中的内容不为空。它们的 ControlToValidate 属性分别设置为 txtUserName、txtPassWord 和 txtName，并将这 3 个 RequiredFieldValidator 控件的 ErrorMessage 属性设置为“*”。
- 本页中 ID 属性为 CompareValidator1 的 CompareValidator 控件是用来验证 ID 属性为 txtPassWord 和 txtAgain 文本框中的输入内容是否一致的。首先，将 CompareValidator 控件的 ControlToCompare 属性设为 txtAgain，然后将 ControlToValidate 属性设为 txtPassWord，最后将 ErrorMessage 属性设置为“确认密码与密码不一致!”

交叉链接: ID 属性 CompareValidator2 的 CompareValidator 控件。的使用。参见示例 13_12。

- 本页中还用到了一个 ValidationSummary 控件，用来分类显示本页中的验证控件的错误信息。

14.3 登 录 控 件

14.3.1 Login 控件

1．功能

Login 控件是一个复合控件，它主要提供用于登录到网站的用户界面元素，让用户输入用户名和密码。图 14.32 所示为 Login 控件。

图 14.32 Login 控件

2．属性

Login 控件常用属性及描述如表 14.32 所示。

表 14.32 Login 控件常用的属性及描述

属 性	描 述
CreateUserText	新用户注册页的链接文本
CreateUserUrl	新用户注册页的 URL
DestinationPageUrl	在登录尝试成功时向用户显示的页面的 URL
FailureAction	当登录尝试失败时发生的操作
FailureText	当登录尝试失败时显示的文本
HelpPageText	设置登录帮助页链接的文本
HelpPageUrl	置登录帮助页的 URL
InstructionText	用户的登录说明文本
LoginButtonText	设置 Login 控件的登录按钮的文本
MembershipProvider	设置控件使用的成员资格数据提供程序的名称

续表

属　性	描　述
Password	获取用户输入的密码
PasswordLabelText	设置 Password 文本框的标签文本
PasswordRecoveryText	设置密码恢复页链接的文本
PasswordRecoveryUrl	设置密码恢复页的 URL
PasswordRequiredErrorMessage	设置当密码字段为空时，在 ValidationSummary 控件中显示的错误信息
RememberMeSet	设置一个值，该值指示是否将持久性身份验证 Cookie 发送到用户的浏览器
RememberMeText	设置“记住我”复选框的标签文本
TitleText	设置 Login 控件的标题
UserName	获取用户输入的用户名
UserNameLabelText	设置 UserName 文本框的标签文本
UserNameRequiredErrorMessage	设置当用户名字段为空时，在 ValidationSummary 控件中显示的错误信息
VisibleWhenLoggedIn	设置一个值，该值指示在验证用户身份后是否显示 Login 控件

下面对比较重要的属性进行详细介绍。

(1) CreateUserText 属性。CreateUserText 属性包含站点注册页的链接文本。在 CreateUserUrl 属性中指定注册页的 URL。如果 CreateUserUrl 为空，则向用户显示 CreateUserText 属性中的文本，但不以链接的形式显示。如果 CreateUserText 属性为空，则不向用户提供注册页链接。

(2) CreateUserUrl 属性。该属性用来设置新用户注册页的 URL，它包含网站新用户注册页的 URL。CreateUserText 属性包含链接使用的文本。如果 CreateUserText 属性为空，则不向用户提供注册页链接。

(3) DestinationPageUrl 属性。DestinationPageUrl 属性指定当登录成功时显示的页面。它将重写 Login 控件的默认行为以及在配置文件中所做的 defaultUrl 设置。

(4) FailureAction 属性。该属性定义当用户没有成功登录到网站时 Login 控件的行为，默认行为为重新加载页并显示 FailureText 属性的内容，以提醒用户登录失败。当 FailureAction 设置为 RedirectToLoginPage 时，用户将被重定向到 Web.Config 文件中定义的登录页。

(5) Password 属性。该属性用来设置用户登录所需的密码，默认为空。设置该属性时，既可以在属性对话框中设置，也可以在后台代码中设置。密码为明文形式。

3．方法

交叉链接：Login 控件常用方法同 TextBox 控件类似，请参见 TextBox 控件常用方法。

4．事件

Login 控件常用事件及描述如表 14.33 所示。

表 14.33　　Login 控件常用事件及描述

事　件	描　述
Authenticate	验证用户的身份后出现
LoggedIn	在用户登录到网站并进行身份验证后出现
LoggingIn	在用户未进行身份验证而提交登录信息时出现
LoginError	当检测到登录错误时出现

当用户使用 Login 控件登录到网站时，引发 Authenticate 事件。自定义身份验证方案可以使用 Authenticate 事件对用户进行身份验证。定义身份验证方案应该将 Authenticated 属性设置为 true，以指示已验证用户的身份。

说明：在用户提交登录信息后，Login 控件先引发 LoggingIn 事件，然后引发 Authenticate

事件，最后引发 LoggedIn 事件。

14.3.2 CreateUserWizard 控件

1．功能

CreateUserWizard 控件提供用于创建新网站用户账户的用户界面。该对象与网站的用户数据存储区进行通信以便在数据存储区中创建新用户账户。默认情况下，CreateUserWizard 控件将接收网站访问者的用户名和密码。图 14.33 所示为 CreateUserWizard 控件。

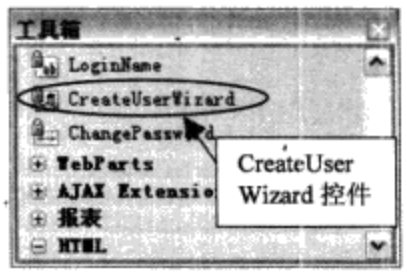


图 14.33 CreateUserWizard 控件

2．属性

CreateUserWizard 控件常用属性及描述如表 14.34 所示。

表 14.34 CreateUserWizard 控件常用属性及描述

属　性	描　述
AutoGeneratePassword	设置用于指示是否自动为新用户账户生成密码的值
CompleteStep	获取对最终用户账户创建步骤的引用
CompleteSuccessText	网站用户账户创建成功后所显示的文本
ConfirmPassword	获取用户输入的第二个密码
ConfirmPasswordCompareErrorMessage	当用户在密码文本框和确认密码文本框中输入两个不同的密码时所显示的错误信息
ConfirmPasswordLabelText	设置第二个密码文本框的标签文本
ConfirmPasswordRequiredErrorMessage	设置当用户将确认密码文本框留空时所显示的错误信息
ContinueButtonImageUrl	设置最终用户账户创建步骤上的【继续】按钮所用图像的 URL
ContinueButtonText	设置在【继续】按钮上显示的文本标题
ContinueDestinationPageUrl	设置在用户单击成功页上的【继续】按钮后将看到的页的 URL
DisableCreatedUser	设置一个值，该值指示是否应允许新用户登录到网站
DisplayCancelButton	设置一个布尔值，指示是否显示【取消】按钮
E-mail	设置用户输入的电子邮件地址
EmailRegularExpression	设置用于验证提供的电子邮件地址的正则表达式
FinishDestinationPageUrl	设置当用户单击【完成】按钮时将重定向到的 URL
MembershipProvider	设置为创建用户账户而调用的成员资格提供程序
Password	获取用户输入的密码
PasswordHintText	设置描述密码要求的文本
PasswordRegularExpression	设置用于验证提供的密码的正则表达式
PasswordRequiredErrorMessage	设置由于用户未输入密码而显示的错误信息的文本
Question	设置用户输入的密码恢复确认问题
RequireEmail	设置一个值，该值指示网站用户是否必须填写电子邮件地址

下面对比较重要的属性进行详细介绍。

（1）ActiveStep 属性。ActiveStep 属性返回当前显示在 Wizard 控件中的 WizardStep Base 对象。ActiveStep 属性是只读的，但用户可以使用 ActiveStep 属性访问当前 WizardStepBase 对象的属性。使用 ActiveStepIndex 属性可动态地将 ActiveStep 更改为不同的 WizardStepBase 对象。用户还可以使用 MoveTo 方法动态设置 ActiveStep 属性。

（2）MembershipProvider 属性。MembershipProvider 属性包含定义的 Membership Provider 类成员资格提供程序的名称，该提供程序用于存储用户信息。更改 Membership Provider 属性的

值就可以在不同的数据存储区中存储用户的成员资格信息。当 Membership Provider 属性设为 Empty 时，将使用 Web.config 文件中定义的默认成员资格提供程序。

（3）ContinueDestinationPageUrl 属性。ContinueDestinationPageUrl 属性包含用户在站点上成功完成注册后将看到的网页的 URL。通过设置 ContinueDestinationPageUrl 属性，可以控制新注册的用户将看到的第一个页面。

当 ContinueDestinationPageUrl 属性为 Empty 且用户单击【继续】按钮后，该页将刷新并清除表单中的所有值。

例如，注册完成后，单击【继续】按钮跳转到登录页，只需要在 CreateUserWizard 控件的属性窗口将 ContinueDestinationPageUrl 属性设为登录页的 URL，如图 14.34 所示。

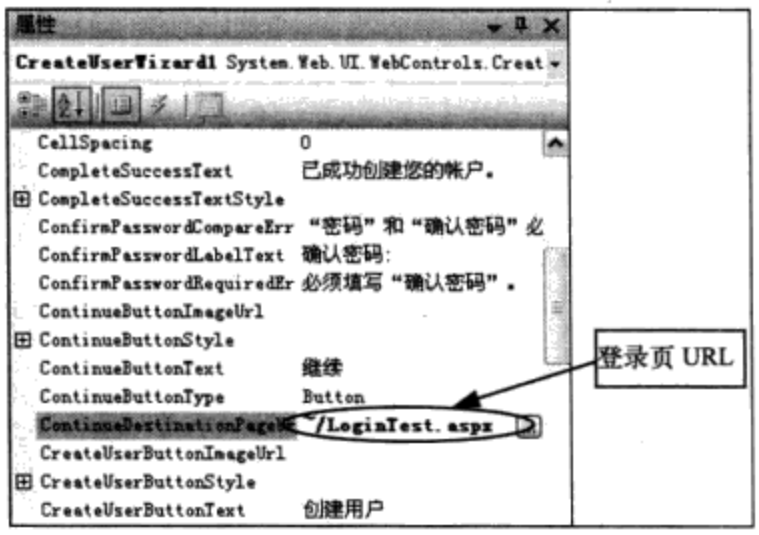


图 14.34 【继续】按钮的 URL

（4）PasswordRegularExpression 属性。使用 PasswordRegularExpression 属性可以定义密码的要求，这些密码用于验证网站上的用户。密码要求常用于确保用户在密码中包含除字母之外的符号，使攻击者更难猜出密码。

例如，要求用户输入的密码多于 6 个字符，至少包含一个数字和一个特殊字符（非字母数字字符）。代码如下：

```
CreateUserWizard1.PasswordRegularExpression="@\"(?=.{6,})(?=(.*\d){1,})(?=(.*\W){1,})";
```

3．方法

CreateUserWizard 控件常用方法及描述如表 14.35 所示。

表 14.35　　CreateUserWizard 控件的常用方法

方　法	描　述
ApplyStyle	将指定样式的所有非空白元素复制到 Web 控件，改写控件的所有现有的样式元素
ApplyStyleSheetSkin	将页样式表中定义的样式属性应用到控件
CopyBaseAttributes	将 Style 对象未封装的属性从指定的 Web 服务器控件复制到从中调用此方法的 Web 服务器控件
DataBind	将数据源绑定到被调用的服务器控件及其所有子控件
Focus	为控件设置输入焦点
Dispose	使服务器控件得以在从内存中释放之前执行最后的清理操作
Equals	确定两个 Object 实例是否相等
FindControl	在当前的命名容器中搜索指定的控件 ID
GetHashCode	用作特定类型的哈希函数
GetHistory	返回已经被访问过的 WizardStepBase 对象的集合
GetStepType	返回指定的 WizardStepBase 对象的 WizardStepType 值
GetType	获取当前实例的 Type

续表

方　法	描　述
HasControls	确定服务器控件是否包含任何子控件
MergeStyle	将指定样式的所有非空白元素复制到 Web 控件，但不改写该控件现有的任何样式元素
MoveTo	将指定的从 WizardStepBase 派生的对象设置为 Wizard 控件的 ActiveStep 属性的值
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
RenderBeginTag	将控件的 HTML 开始标记呈现到指定的编写器中
RenderControl	输出服务器控件内容，并存储有关此控件的跟踪信息（如果已启用跟踪）
RenderEndTag	将控件的 HTML 结束标记呈现到指定的编写器中
ResolveClientUrl	获取浏览器可以使用的 URL
ResolveUrl	将 URL 转换为在请求客户端可用的 URL
SetRenderMothodDelegate	分配事件处理程序委托，以将服务器控件及其内容呈现到父控件中
ToString	返回表示当前对象的类型的完全限定名

下面对比较重要的方法进行详细介绍。

（1）GetHistory 方法。GetHistory 方法返回一个包含已经被访问过的 WizardStepBase 对象的集合。WizardStepBase 对象按访问顺序排列，集合中的第一项是上一个 WizardStepBase 对象，集合中的第二项是在上一步骤之前访问的 WizardStepBase 对象，依此类推。

（2）GetStepType 方法。GetStepType 方法可以确定指定的 WizardStepBase 对象的 WizardStepType 值。

4．事件

CreateUserWizard 控件常用事件及描述如表 14.36 所示。

表 14.36　　CreateUserWizard 控件常用事件及描述

事　件	描　述
ActiveStepChanged	当用户切换到控件中的新步骤时发生
CancelButtonClick	当单击【取消】按钮时发生
ContinueButtonClick	当用户单击最终用户账户创建步骤中的【继续】按钮时发生
CreatedUser	在成员资格提供程序创建了新的网站用户账户后发生
CreateUserError	当成员资格提供程序无法创建指定的用户账户时发生
CreatingUser	在调用成员资格提供程序以创建新的网站用户账户前发生
FinishButtonClick	当单击【完成】按钮时发生
NextButtonClick	当单击【下一步】按钮时发生
PreviousButtonClick	当单击【上一步】按钮时发生
SendingMail	在向用户发送用于确认账户已创建的电子邮件前发生
SendMailError	在向新用户发送电子邮件的过程中出现 SMTP 错误时发生
SideBarButtonClick	当单击侧栏区域中的按钮时发生

下面对比较重要的事件进行详细介绍。

（1）ActiveStepChanged 事件。当 Wizard 控件中显示的当前步骤更改时，会引发 ActiveStepChanged 事件。当 Wizard 控件的当前步骤更改时，使用 ActiveStepChanged 事件可以提供其他处理。

（2）ContinueButtonClick 事件。当单击 CreateUserWizard 控件上的【继续】按钮时会引发 ContinueButtonClick 事件。使用 ContinueButtonClick 事件可以提供单击【继续】按钮时的附加处理。

14.3.3 登录控件典型应用

实例位置：光盘\mingrisoft\dxyy\14\14_03

登录和用户注册是网站中必备的功能。在过去的网站开发中，这些功能都需要开发人员手动编写，在.NET Framework 3.5 版中新增了登录控件和用户注册控件，极大地方便了开发人员的编程。下面介绍如何使用 Login 控件和 CreateUserWizard 控件实现用户注册及登录。运行结果如图 14.35 和图 14.36 所示。

图 14.35 登录运行结果图

图 14.36 用户注册运行结果图

程序开发步骤如下。

(1) 新建一个网站，将其命名为 14_03，默认主页名为 LoginTest.aspx。

(2) LoginTest.aspx 页面主要使用了 Login 控件。

(3) 代码设计如下。

在 Login.aspx 页中，添加一个 Login 控件，并对其属性进行设置，代码如下：

```
<asp:Login ID="Login1" runat="server" BackColor="#E3EAEB" BorderColor="#E6E2D8"
    BorderPadding="4" BorderStyle="Solid" BorderWidth="1px" CreateUserText="注册新用户"
    CreateUserUrl="~/Register.aspx" DestinationPageUrl="~/welcome.aspx"
    Font-Names="Verdana" Font-Size="Small" ForeColor="#333333" Height="102px"
    Width="190px" TitleText="用户登录" UserNameLabelText="账号:">
    <TitleTextStyle BackColor="#1C5E55" Font-Bold="False" Font-Size="Small"
        ForeColor="White" />
    <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
    <TextBoxStyle Font-Size="0.8em" BorderStyle="Dotted" BorderWidth="1px"
        Width="100px" />
    <LoginButtonStyle BackColor="White" BorderColor="#C5BBAF" BorderStyle="Solid" BorderWidth="1px"
        Font-Names="Verdana" Font-Size="0.8em" ForeColor="#1C5E55" />
</asp:Login>
```

单击【注册新用户】按钮，将跳转到注册用户页面。在 Register.aspx 页中，添加一个 CreateUserWizard 控件，用于注册新用户并对其属性进行设置，代码如下：

```
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server" BackColor="#E3EAEB" BorderColor="#E6E2D8"
    BorderStyle="Solid" BorderWidth="1px"
    ContinueDestinationPageUrl="~/LoginTest.aspx" Font-Names="Verdana" Font-Size="Small"
    InvalidPasswordErrorMessage="该密码包括大写和小写字母以及标点，且长度至少为 8 个字符"
    AnswerLabelText="问题答案:" oncreateduser="CreateUserWizard1_CreatedUser"
    PasswordLabelText="登录密码:" QuestionLabelText="提示问题:" UserNameLabelText="登录账号:">
    <WizardSteps>
        <asp:CreateUserWizardStep runat="server">
        </asp:CreateUserWizardStep>
        <asp:CompleteWizardStep runat="server">
            <ContentTemplate>
                <table border="0" style="font-size: 100%; font-family: Verdana">
                    <tr>
                        <td align="center" colspan="2" style="font-weight: bold; color: white; height: 17px;
```

```
                                    background-color: #5d7b9d">
                                    完成</td>
                                </tr>
                                <tr>
                                    <td>
                                        已成功创建您的账户。</td>
                                </tr>
                                <tr>
                                    <td align="right" colspan="2">
                                        <asp:Button ID="ContinueButton" runat="server" BackColor="#FFFBFF"
BorderColor="#CCCCCC"
                                            BorderStyle="Solid" BorderWidth="1px" CausesValidation="False"
CommandName="Continue"
                                            Font-Names="Verdana" ForeColor="#284775" Text="继续"
ValidationGroup="CreateUserWizard1" />
                                    </td>
                                </tr>
                            </table>
                        </ContentTemplate>
                    </asp:CompleteWizardStep>
                </WizardSteps>
                <SideBarStyle BackColor="#1C5E55" Font-Size="0.9em" VerticalAlign="Top" />
                <TitleTextStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
                <SideBarButtonStyle ForeColor="White" />
                <NavigationButtonStyle BackColor="White" BorderColor="#C5BBAF" BorderStyle="Solid"
                    BorderWidth="1px" Font-Names="Verdana" ForeColor="#1C5E55" />
                <HeaderStyle BackColor="#666666" BorderColor="#E6E2D8" BorderStyle="Solid" BorderWidth="2px"
                    Font-Bold="True" Font-Size="0.9em" ForeColor="White" HorizontalAlign="Center" />
                <TextBoxStyle Width="100px" />
                <CreateUserButtonStyle BackColor="White" BorderColor="#C5BBAF" BorderStyle="Solid"
                    BorderWidth="1px" Font-Names="Verdana" ForeColor="#1C5E55" />
                <ContinueButtonStyle BackColor="White" BorderColor="#C5BBAF" BorderStyle="Solid"
                    BorderWidth="1px" Font-Names="Verdana" ForeColor="#1C5E55" />
                <StepStyle BorderWidth="0px" />
            </asp:CreateUserWizard>
```

第 15 章

Web 用户控件

- 用户控件的概述和用途
- 创建用户控件
- 用户控件的应用

15.1 用户控件的概述和用途

15.1.1 用户控件概述

用户控件是一种复合控件，可以向用户控件添加现有的 Web 服务器控件和标记，并定义控件的属性和方法，还可以将用户控件嵌入 ASP.NET3.5 网页中充当一个单元。

用户控件与 ASP.NET3.5 网页区别如下。

- 用户控件的文件扩展名为.ascx。
- 用户控件中没有@Page 指令，而是包含@Control 指令。
- 用户控件不能作为独立文件运行，而必须像处理控件一样，将其添加到 ASP.NET 3.5 页中。
- 用户控件中没有 html、body 或 form 元素。

15.1.2 用户控件的优点

用户控件可以用来重用现有的控件，同时可以方便地创建具有复杂用户界面元素的控件，用户控件提供了一个面向对象的编程模型，并且提供了比服务器端包含文件更多的功能。对于常用的内容或者控件，可以将其设计成为用户控件，然后便可以在多个网页中重复使用该用户控件，从而省去许多重复性的工作。例如，网页上的导航栏，几乎每个页都需要相同的导航栏，这时便可以将其设计为用户控件。当要改变网站的导航栏时，不需每个页都改动，只要改变原来的用户控件即可。这样，网页的设计与维护就变得相对容易。

15.2 创建用户控件

用户控件的创建很灵活，可以直接新建一个用户控件（.ascx 文件），也可以通过修改 Web 窗体页（.aspx 文件），将其转换为用户控件。

15.2.1 创建用户控件

用户控件创建步骤如下。

（1）在项目名称上单击鼠标右键，在右键菜单中选择“添加新项”命令，打开“添加新项”窗口，如图 15.1 所示。

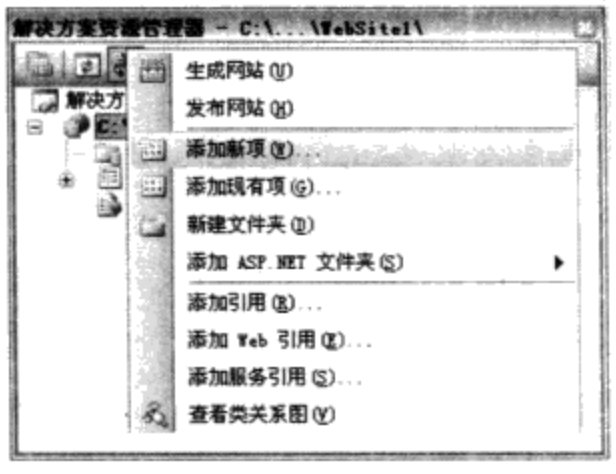


图 15.1 选择“添加新项”命令

（2）在“添加新项”窗口中选择“Web 用户控件”，单击【添加】按钮，创建用户控件，如图 15.2 所示。

注意：不能将用户控件放入网站的 App_Code 文件夹中，如果某个用户控件在 App_Code

文件夹中，则运行包含该控件的页面时将发生分析错误。

图 15.2 “添加新项”窗口

示例 15-01 使用用户自定义控件实现登录

本示例主要利用“代码隐藏模型”定义用户自定义控件（User_Login）实现登录验证功能，本例中定义了两个公有属性，以便引用页对其操作。其操作步骤如下。

（1）首先，新建一个项目，然后添加新项，选择 Web 用户控件。

（2）在用户控件窗体中添加一个表格用于控制布局。

（3）在用户控件窗体中添加两个按钮用于激发事件。

（4）在用户控件窗体中添加两个文本框用于获取信息，两个标签用于显示信息。

（5）实现的程序代码如下：

代码 15-01 光盘位置：光盘\mingrisoft\sl\15\15_01

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class WebUserControl : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.TextBox1.Text = strName;                    //设置用户名属性
        this.TextBox2.Text=strPwd;                       //设置密码属性
    }
     //声明用户名属性
    private string strName;
    public string str_Name
    {
        get { return strName; }
        set { strName = value; }
    }
    //声明密码属性
    private string strPwd;
    public string str_Pwd
```

```
    {
        get { return strPwd; }
        set { strPwd = value; }
    }
    protected void Button1_Click1(object sender, EventArgs e)
    {
        //用于验证引用页输入的信息
        if (this.TextBox1.Text == "mr" && this.TextBox2.Text == "mrsoft")
        {
            //如果输入的用户名和密码分别是 mr 和 mrsoft，则说明登录成功
            Response.Write("<script>alert('OK');</script>");
        }
        else//否则
        {
            //登录失败
            Response.Write("<script>alert('Error');</script>");
        }
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        //关闭当前页
        Response.Write("<script>window.close();</script>");
    }
}
```

控件创建完后，用户只需打开某个页面，在“解决方案资源管理器”中用鼠标选取新创建的用户控件并将其拖到页面中，这时页面将显示出用户控件，如图 15.3 所示。

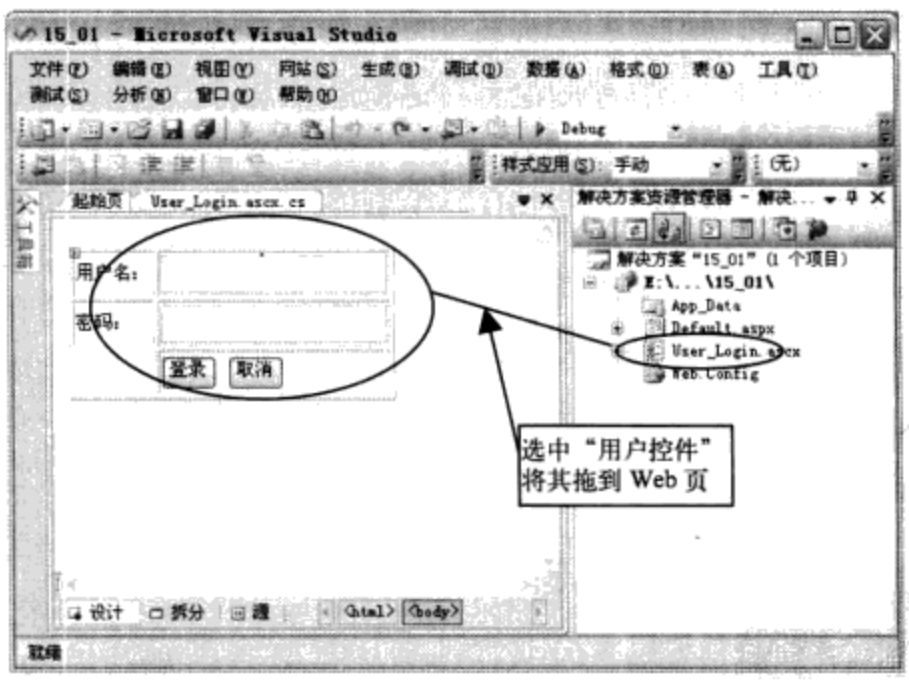


图 15.3　使用用户自定义控件

注意：不能将用户自定义控件设为起始页。

当拖曳完之后，在页面的 HTML 页中将会自动生成如下代码：

```
<%@ Register Src="User_Login.ascx" TagName="User_Login" TagPrefix="uc1" %>
```

在页面中使用自定义控件及其属性，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    User_Login1.str_Name = "mr";
    User_Login1.str_Pwd = "mrsoft";
}
```

设置自定义控件的属性，用户可以直接在“属性“编辑器中设置，通过属性窗口更改它的相应属性。属性窗口如图 15.4 所示。

用户自定义控件运行的效果如图 15.5、图 15.6 和图 15.7 所示。

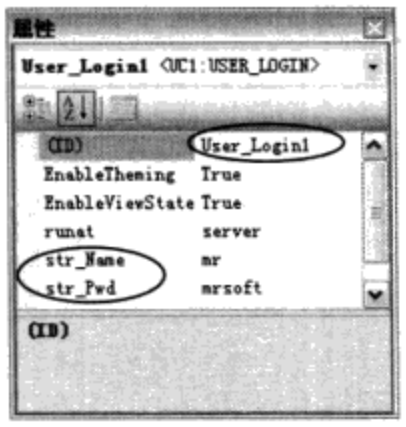


图 15.4　设置自定义窗体属性

图 15.5　登录窗体

图 15.6　提示对话框

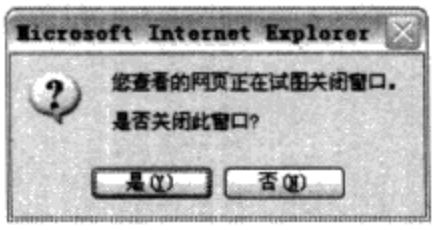


图 15.7　退出提示对话框

示例 15-02　使用用户自定义控件实现信息显示

本示例主要利用“单文件页模型”定义用户自定义控件，在用户自定义控件中，首先定义字符串数组（friend）作为数据源，然后根据输入的数据在数组中进行查找，如果数组中存在文本框中输入的数据，则输出相应的提示信息，示例运行结果如图 15.8 所示。

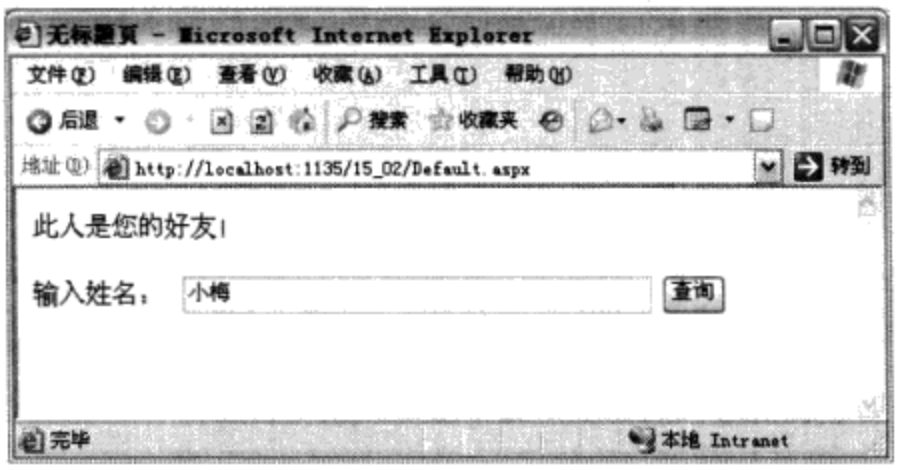


图 15.8　用户自定义控件中查询数组中是否存在输入的数据

用户自定义控件实现的具体步骤如下。

(1) 在 VS2008 工程中添加一个用户自定义窗体。

(2) 在窗体中添加一个文本框用于输入数据。

(3) 在窗体中添加一个按钮用于激发事件。

(4) 程序代码如下：

代码 15-02　光盘位置：光盘\mingrisoft\sl\15\15_02

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
```

```
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
public partial class WebUserControl : System.Web.UI.UserControl
{
    string[] friend = null;                                          //定义一个字符串数组
    protected void Page_Load(object sender, EventArgs e)
    {
        //设置数组中的项目
        friend = new string[] { "风笛","小梅","小王","小梁","小彬","小安","小欣","小玲"};
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        string name = txtName.Text.Trim();                           //获取输入的名称
        bool flag = false;                                           //声明一个标记
        for (int i = 0; i < friend.Length; i++)                      //遍历字符串数组
        {
            if (friend[i] == name)                                   //如果数组中的某一项等于输入的名称
            {
                flag = true;                                         //flag 为 true
                break;
            }
        }
        if (flag)                                                    //如果 flag 为 true
        {
            Response.Write("此人是您的好友！");                      //说明是好友
        }
        else                                                         //否则
        {
            Response.Write("查无此人！");                            //查无此人
        }
    }
}
```

15.2.2 将 Web 页转换为 ASP.NET 3.5 用户控件

当用户创建完一个 Web 页后，发现该 Web 页在很多地方都会用到，这时为了提高代码的重用性，最佳的选择方案就是将 Web 页转换为 ASP.NET 3.5 用户控件。ASP.NET 3.5 用户控件可以用来重用现有的控件，当某个页需要这一功能时便可以直接拖曳实现引用。下面将介绍这方面的内容。

1．单文件页

单文件页就是将标记、服务器端元素以及事件处理代码全都放在同一个.aspx 文件中。在对该页进行编译时，编译器将生成和编译一个从 Page 基类派生或从使用@Page 指令的 Inherits 属性定义的自定义基类派生的新类。例如，如果在应用程序的根目录中创建一个名为 PageFirst 的新 ASP.NET 3.5 网页，则随后将从 Page 类派生一个名为 PageFirst.aspx 的新类。对于应用程序子文件夹中的页，将使用子文件夹名称作为生成的类的一部分。生成的类中包含.aspx 页中的控件的声明以及用户的事件处理程序和其他自定义代码。

在生成页之后，生成的类将编译成程序集，并将该程序集加载到应用程序域，然后对该页类进行实例化并执行，将输出呈现到浏览器。如果对影响生成类的页进行更改（无论是添加控件还是修改代码），那么已编译的类代码将失效，并生成新的类。

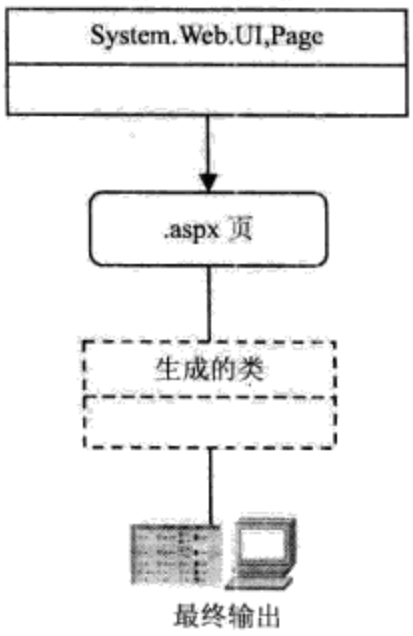


图 15.9　页类的继承模型

单文件 ASP.NET 3.5 网页中的页类的继承模型如图 15.9 所示。

2．单文件页的优点

通常，单文件模型适用于特定的页，在这些页中，代码主要由页中控件的事件处理程序组成。单文件页模型的优点包括以下几点。

（1）没有太多代码的页中，可以方便地将代码和标记保留在同一个文件中，这一点比代码页模型的其他优点都重要。

（2）因为只有一个文件，所以使用单文件模型编写的页更容易部署或发送给其他程序员。

（3）由于文件之间没有相关性，因此更容易对单文件页进行重命名。

（4）因为页自包含于单个文件中，所以在源代码管理系统中管理文件稍微简单一些。

3．将单文件 ASP.NET 3.5 网页转换为用户控件

（1）重命名.aspx 页使其文件扩展名为.ascx。

（2）从该页面中移除 html、body 和 form 元素。

源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="True" CodeFile="Login.aspx.cs"
Inherits="Login" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
void Button1_Click(Object sender, EventArgs e)
{
    Label1.Text = "当前时间为：  " + DateTime.Now.ToString();
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label id="Label1"
         runat="server" Text="">
       </asp:Label>

       <br/>
       <asp:Button id="Button1"
         runat="server"
         onclick="Button1_Click"
         Text="显示当前时间">
       </asp:Button>
    </div>
    </form>
</body>
</html>
```

（3）将@ Page 指令更改为@ Control 指令。

（4）移除@ Control 指令中除 Language、AutoEventWireup（如果存在）、CodeFile 和 Inhe 之外的所有属性。

更改后的代码如下：

```
<%@ Control   Language="C#"%>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "当前时间为：  " + DateTime.Now.ToString();
    }
</script>
<asp:Label id="Label1"
 runat="server" Text="">
</asp:Label>
```

```
<br/>
<asp:Button id="Button1"
 runat="server"
 onclick="Button1_Click"
 Text="显示当前时间">
</asp:Button>
```

15.3 用户控件的应用

为了让读者能够更好地理解并使用用户控件，下面通过站内搜索来进一步介绍用户控件的应用。

实例位置：光盘\mingrisoft\dxyy\15\15_01

本实例将向读者介绍如何利用用户控件来实现站内查询、搜索功能。实例运行结果如图 15.10 和图 15.11 所示。

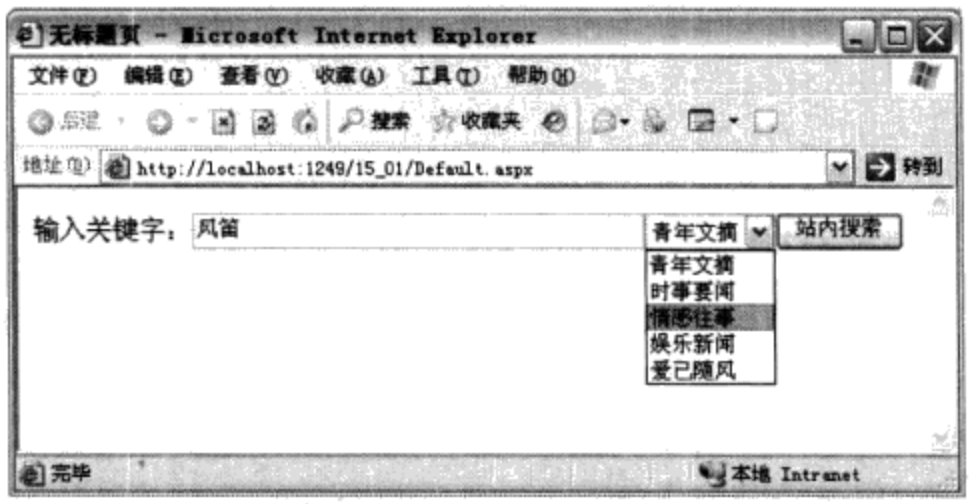


图 15.10 站内搜索用户控件

图 15.11 搜索页传参数

实现过程如下。

(1) 创建一个 Web 用户控件 search.ascx，用来实现搜索功能。

(2) 将创建 Web 用户控件拖到 Default.aspx 页的适当位置。

(3) 创建一个新页 search.aspx，在该页中获取通过地址栏传递的参数。

(4) 主要程序代码。

在 Web 用户控件 search.ascx 上，当用户单击【站内搜索】按钮时，将会触发 Button 控件的 Click 事件。将要查询的关键字传递到 Search 页，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string key = txtKey.Text.Trim();                          //获取输入的关键字
    string range = ddlRange.SelectedItem.Text;                //获取输入的查询范围
    //将关键字和查询范围通过参数传递到指定的页面
    Response.Redirect("search.aspx?key="+key+"&range="+range);
}
```

第 16 章

ASP.NET 3.5 内置对象

- Response 对象
- Request 对象
- Application 对象
- Session 对象
- Server 对象
- Cookie 对象

16.1 Response 对象

Response 对象用于将服务器的信息传递给客户端用户，响应客户端的请求，并将服务器的响应返回给用户。

16.1.1 Response 对象的属性

Response 对象的常用属性及说明如表 16.1 所示。

表 16.1　　Response 对象常用属性及说明

属　性	说　明
Buffer	是否缓冲输出，并在完成处理整个响应之后将其发送
BufferOutput	是否缓冲输出，并在完成处理整个页之后将其发送
Cache	获取 Web 页的缓存策略（过期时间、保密性、变化子句等）
CacheControl	将 Cache-Control HTTP 头设置为 Public 或 Private
Charset	设定或获取 HTTP 的输出字符编码
ContentEncoding	输出流的 HTTP 字符集
ContentType	输出流的 HTTP MIME 类型
Cookies	响应 Cookie 集合
Expires	浏览器上缓存的页过期之前的分钟数
ExpiresAbsolute	从缓存中移除缓存信息的绝对日期和时间
Filter	设置一个包装筛选器对象，该对象用于在传输之前修改 HTTP 实体主体
HeaderEncoding	设置一个 Encoding 对象，该对象表示当前标头输出流的编码
IsClientConnected	传回客户端是否仍然和 Server 连接
IsRequestBeingRedirected	客户端是否正在被传输到新的位置
Output	启用到输出 HTTP 响应流的文本输出
OutputStream	启用到输出 HTTP 内容主体的二进制输出
RedirectLocation	设置 HTTP“位置”标头的值
Status	返回到客户端的 Status 栏
StatusCode	返回给客户端的输出的 HTTP 状态代码
StatusDescription	返回给客户端的输出的 HTTP 状态字符串
SuppressContent	设定是否将 HTTP 的内容发送至客户端浏览器，若为 True，则网页将不会传至客户端

下面对比较重要的属性进行详细介绍。

（1）Charset 属性。用于设置输出流的 HTTP 字符编码。语法格式如下：

```
public string Charset { get; set; }
```

属性值：输出流的 HTTP 字符集。

例如，下面设置 Response 对象对输出流的编码字符集为 GB2312，代码如下：

```
Response.Charset = "GB2312";
```

（2）ContentEncoding 属性。用于获取或设置输出流的 HTTP 字符集。语法格式如下：

```
public Encoding ContentEncoding { get; set; }
```

属性值：包含有关当前响应的字符集信息的 Encoding 对象。

例如，下面获取输出流的 HTTP 字符集，代码如下：

```
string str = Response.ContentEncoding.EncodingName;
```

（3）Expires 属性。用于获取或设置在浏览器上的缓存页过期之前的分钟数。语法格式如下：

```
public int Expires { get; set; }
```

属性值：页过期之前的分钟数。

例如，如果用户设置一个页面的过期时间为 1min，那么在 1min 之内，如果要返回该页面，该页面将显示缓存的文本信息。设置页面的过期时间为 1min 的代码如下：

```
Response.Expires=1;
```

16.1.2 Response 对象的方法

Response 对象常用方法及说明如表 16.2 所示。

表 16.2　Response 对象常用方法及说明

方　法	说　明
AddHeader	将一个 HTTP 头添加到输出流
AppendCookie	将一个 HTTP Cookie 添加到内部 Cookie 集合
AppendHeader	将 HTTP 头添加到输出流
AppendToLog	将自定义日志信息添加到 IIS 日志文件
Clear	将缓冲区的内容清除
BinaryWrite	将一个二进制字符串写入 HTTP 输出流
ClearContent	清除缓冲区流中的所有内容输出
ClearHeaders	清除缓冲区流中的所有头
Close	关闭到客户端的套接字连接
End	将目前缓冲区中所有的内容发送至客户端然后关闭
Flush	将缓冲区中所有的数据发送至客户端
Redirect	将网页重新导向另一个地址
SetCookie	更新 Cookie 集合中的一个现有 Cookie
TransmitFile	将指定的文件直接写入 HTTP 响应输出流，而不在内存中缓冲该文件
Write	将数据输出到客户端
WriteFile	将指定的文件直接写入 HTTP 内容输出流
WriteSubstitution	允许将响应替换块插入响应，从而允许为缓存的输出响应动态生成指定的响应区域

下面对比较重要的方法进行详细介绍。

（1）Redirect 方法。用于将客户端重定向到新的 URL。语法格式如下：

```
public void Redirect (string url)
```

参数说明如下。

url：目标位置。

示例 16-01　使用 Redirect 方法实现页面跳转

本示例实现的是当程序运行时，在页面 Default.aspx 中单击【跳转页面】超级链接，页面跳转到 Default2.aspx 页面。本示例运行结果如图 16.1 和图 16.2 所示。

图 16.1　Default.aspx 页面

图 16.2　Default2.aspx 页面

程序代码如下。

代码 16-01　光盘位置：光盘\mingrisoft\sl\16\16_01

单击【链接新页面】超级链接，程序调用 Response 对象的 Redirect 方法实现页面跳转功能。

【链接新页面】超级链接的 Click 事件代码如下：

```
protected void LinkButton1_Click(object sender, EventArgs e)
{
    Response.Redirect("~/Default2.aspx");                    //转向指定页面
}
```

（2）Write 方法。用于将一个字符串写入 HTTP 响应输出流。语法格式如下：

```
public void Write (string s)
```

参数说明如下。

s：要写入 HTTP 输出流的字符串。

示例 16-02　使用 Write 方法输出字符串

当程序运行时，在页面 Default.aspx 中单击【Write 方法输出字符串】按钮，输出一个指定字符串。示例运行结果如图 16.3 所示。

图 16.3　使用 Write 方法输出字符串

程序代码如下。

代码 16-02　光盘位置：光盘\mingrisoft\sl\16\16_02

单击【Write 方法输出字符串】按钮，程序调用 Response 对象的 Write 方法在页面中输出指定字符串。【Write 方法输出字符串】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("风笛");                                  //输出指定字符串
}
```

16.1.3　Response 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_01

本节通过一个简单的实例讲解如何在 ASP.NET Web 页面中使用 Response 对象实现用户登录功能。实例运行结果如图 16.4 和图 16.5 所示。

图 16.4　登录界面

图 16.5　登录成功界面

程序开发步骤如下。

（1）新建一个网站，命名为 16_01，其主页默认为 Default.aspx。

（2）在 Default.aspx 页面中添加一个 Table 表格，用来布局页面，然后在该 Table 表格中添加两个 TextBox 控件和两个 Button 控件，分别用来输入用户名、密码和执行登录、重置功能。

（3）在该网站中添加一个 Web 页面，命名为 Default2.aspx，然后在该页中显示登录成功。

（4）程序主要代码如下。

Default.aspx 页面中，当用户单击【登录】按钮时，程序首先判断用户名和密码文本框是否为空，如果为空，调用 Response 对象的 Write 方法弹出信息提示框；否则，调用 Response 对象的 Redirect 方法进入到主页 Default2.aspx 页面中。【登录】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (txtName.Text == "" || txtPwd.Text == "")                    //如果用户名和密码为空
    {
        RegisterStartupScript("", "<script>alert('请输入用户名或密码')</script>");//弹出提示信息
    }
    else                                                            //否则
    {
        if (txtName.Text.Trim() == "风笛" && txtPwd.Text.Trim() == "19820112")//如果输入的用户名和密码正确
        {
            Response.Redirect("~/Default2.aspx");                   //转到 Default2.aspx
        }
        else                                                        //否则
        {
            //弹出提示信息
            RegisterStartupScript("", "<script>alert('用户名或密码错误，请核对后重新登录！')</script>");
        }
    }
}
```

16.2 Request 对象

Request 对象是 HttpRequest 类的一个实例，它提供对当前页请求的访问，其中包括标题、Cookie、客户端证书、查询字符串等，用户可以使用此对象来读取浏览器发送的内容。

16.2.1 Request 对象的属性

Request 对象常用属性及说明如表 16.3 所示。

表 16.3　　Request 对象常用属性及说明

属　性	说　明
AcceptTypes	获取客户端支持的 MIME 接收类型的字符串数组
AnonymousID	获取该用户的匿名标识符（如果存在）
ApplicationPath	获取服务器上 ASP.NET 应用程序虚拟应用程序的根目录路径
Browser	获取或设置有关正在请求的客户端浏览器的信息
ClientCertificate	获取当前请求的客户端安全证书
ContentEncoding	获取或设置实体主体的字符集
ContentLength	指定客户端发送的内容长度（以 KB 计）
ContentType	获取或设置传入请求的 MIME 内容类型
Cookies	获取客户端发送的 Cookie 集合
CurrentExecutionFilePath	获取当前请求的虚拟路径
FilePath	获取当前请求的虚拟路径
Files	获取采用多部分 MIME 格式的由客户端上传的文件集合
Filter	获取或设置在读取当前输入流时要使用的筛选器
Form	获取窗体变量集合
Headers	获取 HTTP 头集合
HttpMethod	获取客户端使用的 HTTP 数据传输方法（如 get、post 或 head）
InputStream	获取传入的 HTTP 实体主体的内容
IsAuthenticated	获取一个值，该值指示是否验证了请求
IsLocal	获取一个值，该值指示该请求是否来自本地计算机
IsSecureConnection	获取一个值，该值指示 HTTP 连接是否使用安全套接字（即 HTTPS）
Item	从 Cookies、Form、QueryString 或 ServerVariables 集合中获取指定的对象

续表

属　性	说　明
Params	获取 QueryString、Form、ServerVariables 和 Cookies 项的组合集合
Path	获取当前请求的虚拟路径
PathInfo	获取具有 URL 扩展名的资源的附加路径信息
PhysicalApplicationPath	获取当前正在执行的服务器应用程序的根目录的物理文件系统路径
PhysicalPath	获取与请求的 URL 相对应的物理文件系统路径
QueryString	获取 HTTP 查询字符串变量集合
RawUrl	获取当前请求的原始 URL
RequestType	获取或设置客户端使用的 HTTP 数据传输方法（如 get 或 post）
ServerVariables	获取 Web 服务器变量的集合
TotalBytes	获取当前输入流中的字节数
Url	获取有关当前请求的 URL 的信息
UrlReferrer	获取有关客户端上次请求的 URL 的信息及该请求链接到当前的 URL
UserAgent	获取客户端浏览器的原始用户代理信息
UserHostAddress	获取远程客户端 IP 主机地址
UserHostName	获取远程客户端 DNS 名称
UserLanguages	获取客户端语言首选项的排序字符串数组

下面对比较重要的属性进行详细介绍。

（1）Browser 属性。用于获取或设置有关正在请求的客户端浏览器的信息。语法格式如下：

```
public HttpBrowserCapabilities Browser { get; set; }
```

属性值：列出客户端浏览器的信息。

示例 16-03　使用 Browser 属性获得浏览器信息

本示例实现的是，当程序运行时，在页面 Default.aspx 中输出浏览器的一些相关信息。示例运行结果如图 16.6 所示。

图 16.6　使用 Browser 属性获得浏览器信息

程序代码如下。

代码 16-03　光盘位置：光盘\mingrisoft\sl\16\16_03

在 Default.aspx 页面的 Page_Load 事件中，调用 Request 对象的 Browser 属性输出与浏览器相关的一些信息，实现代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("浏览器使用的平台：" + Request.Browser.Platform + "<br>" + "浏览器类型：" + Request.
Browser.Type + "<br>" + "浏览器版本：" + Request.Browser.Version);
}
```

（2）QueryString 属性。用来获取客户端以 Get 方式提交的数据。语法格式如下：

```
public NameValueCollection QueryString { get; }
```

属性值：包含由客户端发送的查询字符串变量的集合。

示例 16-04　使用 QueryString 属性实现地址栏传值

当程序运行时，在页面 Default.aspx 的文本框中输入用户名和密码，然后单击【登录】按钮，页面通过 Get 方法获取通过地址栏传递的用户名和密码，示例运行结果如图 16.7 和图 16.8 所示。

图 16.7　Default.aspx 页面

图 16.8　Default2.aspx 页面

程序代码如下。

代码 16-04　光盘位置：光盘\mingrisoft\sl\16\16_04

Default.aspx 页面中，单击【登录】按钮，程序跳转到 Default2.aspx 页面，并使用标识 Name 和 Pwd 将输入的用户名和密码传递到 Default2.aspx 页面中。【登录】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (txtName.Text == "" || txtPwd.Text == "")                    //判断是否输入用户名和密码
    {
        RegisterStartupScript("", "<script>alert('请输入用户名和密码')</script>");//如果没有输入则弹出提示信息
    }
    else                                                            //否则
    {
        //将用户名和密码传递到 Default2.aspx 页面
        Response.Redirect("~/Default2.aspx?Name="+txtName.Text.Trim()+"&&Pwd="+txtPwd.Text.Trim());
    }
}
```

Default2.aspx 页面中，使用 Request 对象的 QueryString 属性接收 Default.aspx 页面中的值，实现代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    string name = Request.QueryString["Name"].ToString();           //获取用户名
    string pwd = Request.QueryString["Pwd"].ToString();             //获取密码
    Response.Write("**********登录用户信息**********<br/>");        //输出信息
    Response.Write("用户名："+name+"<br/>");                        //输出用户名
    Response.Write("密码："+pwd);                                   //输出密码
}
```

（3）UserHostAddress 属性。用于获取远程客户端的主机 IP 地址。语法格式如下：

```
public string UserHostAddress { get; }
```

属性值：远程客户端的 IP 地址。

例如，用一个字符串来记录远程客户端的主机 IP 地址，代码如下：

```
string str=Request.UserHostAddress;
```

16.2.2　Request 对象的方法

Request 对象常用方法及说明如表 16.4 所示。

表 16.4　　Request 对象常用方法及说明

方　法	说　明
BinaryRead	执行对当前输入流进行指定字节数的二进制读取
MapImageCoordinates	将传入图像字段窗体参数映射为适当的 *x* 坐标值和 *y* 坐标值
MapPath	为当前请求将请求的 URL 中的虚拟路径映射到服务器上的物理路径
SaveAs	将 HTTP 请求保存到硬盘
ValidateInput	验证由客户端浏览器提交的数据，如果存在具有潜在危险的数据，则引发一个异常

下面对比较重要的方法进行详细介绍。

（1）SaveAs 方法。用于将 HTTP 请求保存到硬盘。语法格式如下：

```
public void SaveAs (string filename,bool includeHeaders)
```

参数说明如下。

filename：要保存的文件路径。

includeHeaders：一个布尔值，该值指定是否应将 HTTP 头保存到硬盘。

示例 16-05　使用 SaveAs 方法保存 HTTP 请求

本示例实现的是，当程序运行时，将 HTTP 请求保存到一个 txt 文本文件中，文本文件中的内容如图 16.9 所示。

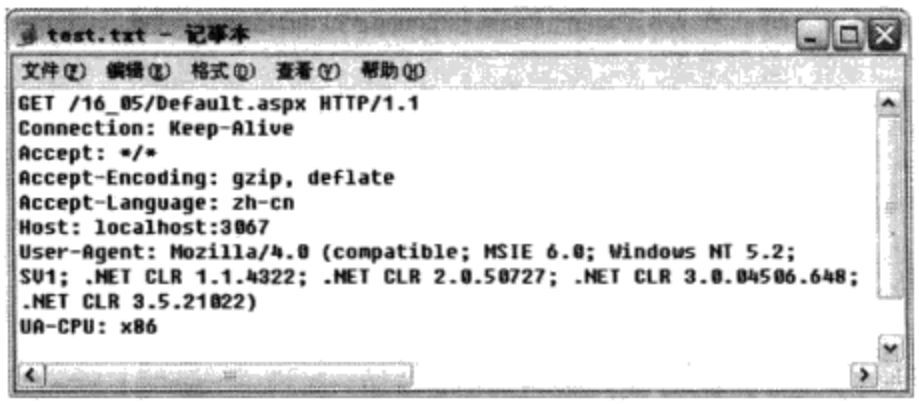


图 16.9　HTTP 请求信息

程序代码如下。

代码 16-05　光盘位置：光盘\mingrisoft\sl\16\16_05

程序运行时，调用 Request 对象的 SaveAs 方法，将 HTTP 请求保存到本网站虚拟目录下的 test.txt 文件中，关键代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Request.SaveAs(Server.MapPath("test.txt"), True);                    //SaveAs 方法保存 HTTP 请求
}
```

（2）MapPath 方法。用于将指定的虚拟路径映射到物理路径。语法格式如下：

```
public string MapPath (string virtualPath)
```

参数说明如下。

virtualPath：当前请求的虚拟路径（绝对路径或相对路径）。

返回值：由 virtualPath 指定的服务器物理路径。

例如，下面的代码用来取得页面 Default.aspx 的物理路径，关键代码如下：

```
string str = Request.MapPath("Default.aspx");
```


16.2.3　Request 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_02

本节通过一个简单的实例来介绍如何在 ASP.NET Web 页面中使用 Request 对象。实例运行

结果如图 16.10 所示。

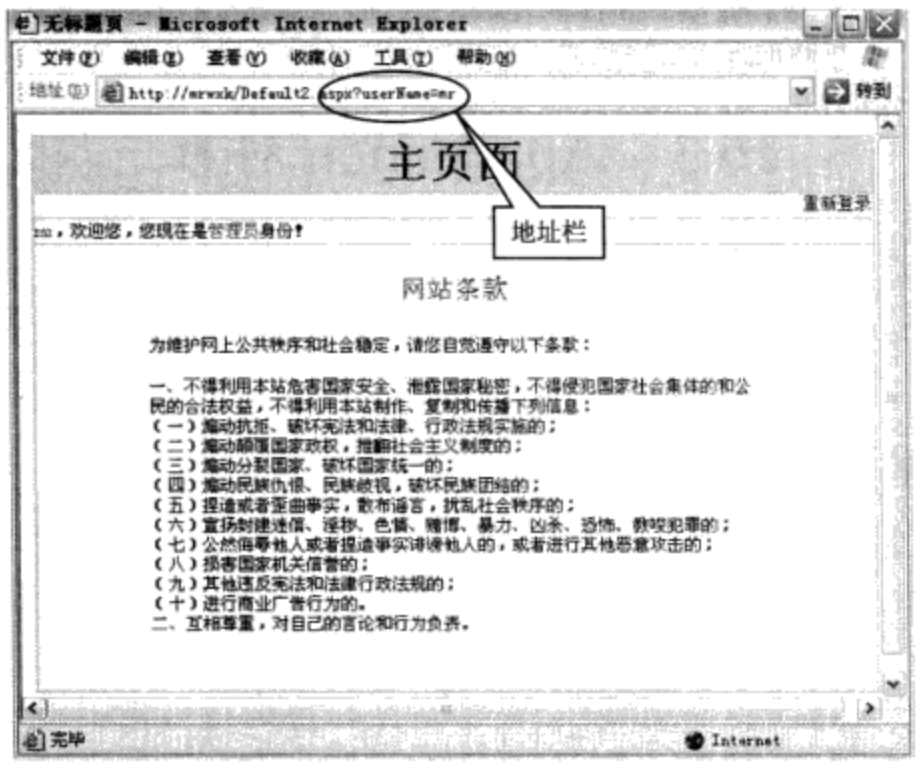


图 16.10　主页面

程序开发步骤如下。

（1）新建一个网站，命名为 16_02，其主页默认为 Default.aspx。

（2）在 Default.aspx 页面中添加一个 Table 表格，用来布局页面，然后在该 Table 表格中添加一个 TextBox 控件和两个 Button 控件，分别用来输入用户名、执行登录和退出功能。

（3）在该网站中添加一个 Web 页面，命名为 Default2.aspx，然后在该页中添加两个 Label 控件，分别用来接收传值和根据接收的值判断用户身份。

（4）程序主要代码如下。

Default.aspx 页面中，当用户单击【登录】按钮时，程序首先判断用户名文本框是否为空，如果为空，弹出信息提示框。否则，跳转到 Default2.aspx 页面，并使用标识 userName 将用户名传到 Default2.aspx 页面中。【登录】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (TextBox1.Text == "")                                        //如果没有输入用户名
    {
      Response.Write("<script>alert('用户名不能为空')</script>");    //输出提示信息
      TextBox1.Focus();                                             //使输入用户名的文本框获得焦点
    }
    else                                                            //否则
    {
      Response.Redirect("Default2.aspx?userName=" + TextBox1.Text.Trim());//转向 Default2.aspx 页面
    }
}
```

在 Default2.aspx 页面的 Page_Load 事件中，使用 Request 对象的 QueryString 属性接收用户名，并根据该用户名判断用户的身份。Default2.aspx 页面的 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    string str=Request.QueryString["userName"].ToString();          //获取传递过来的用户名
    Label1.Text = str;                                              //显示用户名
    if (str == "mr")                                                //如果登录用户名为 mr
    {
      Label2.Text = "管理员";                                        //则其登录身份为管理员
    }
    else                                                            //如果不是 mr
```

```
        {
            Label2.Text = "普通用户";                                    //则其登录身份为普通用户
        }
    }
```

16.3 Application 对象

Application 对象是 HttpApplicationState 类的一个实例，它可以生成一个所有 Web 应用程序都可以存取的变量，这个变量的使用范围涵盖全部使用者，只要正在使用这个网页的程序都可以存取这个变量。

16.3.1 Application 对象的属性

Application 对象变量的生命周期中止于关闭 IIS 或使用 Clear 方法后，Clear 方法是 Page 对象的成员，可以直接调用。Application 对象常用属性及说明如表 16.5 所示。

表 16.5　　Application 对象常用属性及说明

属　性	说　明
AllKeys	获取集合中的访问键
Count	获取 Application 对象变量的数量

下面对比较重要的属性进行详细介绍。

Count 属性。用于获取 Application 对象变量的数量。语法格式如下：

```
public override int Count { get; }
```

该属性为只读属性，默认对象数量为 0。

示例 16-06　使用 Count 属性获取 Application 对象数量

本示例实现的是，当程序运行时，在 Default.aspx 页面中输出 Application 对象的数量及每一项的值。本示例运行结果如图 16.11 所示。

图 16.11　使用 Count 属性获取 Application 对象数量

程序代码如下。

代码 16-06　光盘位置：光盘\mingrisoft\sl\16\16_06

Default.aspx 页面加载时，定义 3 个 Application 对象，然后使用 Application 对象的 Count 属性汇总其数量，并输出到页面上。Default.aspx 页面的 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Application["app1"] = "app1";                                    //定义第一个 Application 对象并赋值
    Application["app2"] = "app2";                                    //定义第二个 Application 对象并赋值
    Application["app3"] = "app3";                                    //定义第三个 Application 对象并赋值
    //获取/Application 对象数量以及每一项的值
    Response.Write("Application 对象数量为：" + Application.Count.ToString()
     + "个，分别为：<br>" + Application["app1"] + "，" + Application["app2"] + "和" + Application["app3"]);
}
```

16.3.2 Application 对象的方法

Application 对象常用方法及说明如表 16.6 所示。

表 16.6　　Application 对象常用方法及说明

方　法	说　明
Add	新增一个 Application 对象变量
Clear	清除全部 Application 对象变量
Get	通过名称或索引获取 Application 对象
GetEnumerator	返回循环访问 NameObjectCollectionBase 的枚举数
GetKey	通过索引获取 Application 对象名
Lock	锁定全部 Application 对象变量
Remove	使用变量名称移除一个 Application 对象变量
RemoveAll	移除全部 Application 对象变量
RemoveAt	按索引从集合中移除一个 Application 对象
Set	使用变量名称更新一个 Application 对象变量的内容
UnLock	解除锁定的 Application 对象变量

下面对比较重要的方法进行详细介绍。

（1）Add 方法。用于在 Application 对象中添加一个新的变量。语法格式如下：

```
public void Add (string name,Object value)
```

参数说明如下。

name：要添加到 Application 对象中的变量名。

value：变量的值。

例如，在 Application 对象中添加一个值为 test 的 app 变量，代码如下：

```
Application.Add("app", "test");
```

（2）Lock 和 UnLock 方法。Lock 方法用于锁定全部 Application 对象变量，以促进访问同步。语法格式如下：

```
public void Lock ()
```

UnLock 方法用于解除锁定的 Application 对象变量，以促进访问同步。语法格式如下：

```
public void UnLock ()
```

示例 16-07　使用 Lock 和 UnLock 方法实现在线人数统计

本示例实现当程序运行时，在页面中显示当前的在线人数。示例运行结果如图 16.12 所示。

图 16.12　使用 Lock 和 UnLock 方法实现在线人数统计

程序代码如下。

代码 16-07　光盘位置：光盘\mingrisoft\sl\16\16_07

添加一个 Global.asax 全局程序集文件，在该文件中的 Application_Start 事件中定义一个 Application 变量，并将其初始化值设置为 0，该变量用来记录网站的在线人数。Application_Start 事件代码如下：

```
void Application_Start(object sender, EventArgs e)
```

```
{
    Application["count"] = 0;                                    //初始化在线人数为 0
}
```

当一个新会话启动时，在 Session_Start 事件中锁定 Application 变量，以防止因为多个用户同时访问页面而造成的并行操作，同时将该变量的值加 1，然后调用 Application 对象的 UnLock 方法解除锁定的 Application 变量。Session_Start 事件代码如下：

```
void Session_Start(object sender, EventArgs e)
{
    Application.Lock();                                          //锁定 Application 变量
    Application["count"] = Convert.ToInt32(Application["count"]) + 1;   //使在线人数增加 1
    Application.UnLock();                                        //解除锁定
}
```

当一个会话结束时，在 Session_ End 事件中锁定 Application 变量，同时将该变量的值减 1，然后调用 Application 对象的 UnLock 方法解除锁定的 Application 变量。 Session_End 事件代码如下：

```
void Session_End(object sender, EventArgs e)
{
    Application.Lock();                                          //锁定 Application 变量
    Application["count"] = Convert.ToInt32(Application["count"]) - 1;   //如果会话结束使在线人数减 1
    Application.UnLock();                                        //解除锁定
}
```

Default.aspx 页面的 Page_Load 事件中，将定义的 Application 变量的值赋给 Label 控件，以在该页中显示当前的在线人数。Default 页面的 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = Application["count"].ToString();               //显示在线人数
}
```

（3）Remove 方法。用于从 Application 对象中移除指定名称的变量。语法格式如下：

```
public void Remove (string name)
```

参数说明如下。

name：要从 Application 对象中移除的变量名。

例如，在 Application 对象中移除名称为 app 的变量，代码如下：

```
Application.Remove("app");
```

16.3.3 Application 对象的事件

1．Application_Start 事件

Application_Start 事件在首次创建新的会话（即事件）之前发生，只有 Application 和 Server 内置对象可使用。在 Application_Start 事件中引用 Session、Request 或 Response 对象将导致错误。

Application 对象的 Start 事件肯定发生在 Session_Start 事件之前，不过，Application 对象不会像 Session 对象那样在一个新用户请求后触发，Application 对象只触发一次，即第一个用户的第一次请求。

Application_Start 事件触发惟一一个脚本程序，它存在于 Global.asax 文件中。

由于 Application 对象是多用户共享的，因此它与 Session 对象有着本质的区别，同时 Application 对象不会因为某一个用户甚至全部用户的离开而消失，一旦建立了 Application 对象，那么它就会一直存在直到网站关闭或者该 Application 对象被卸载，这通常需要很长时间。

由于 Application 对象创建之后不会自己注销，因此一定要特别小心的使用。另外，它会占用内存，要小心使用以免降低服务器对其他工作的响应速度。中止 Application 对象有 3 种方法，分别为服务被中止、Global.asax 文件被改变或者该 Application 对象被卸载。

2．Application_End 事件

Application_End 事件在应用程序退出时于 Session_End 事件之后发生，只有 Application 和

Server 内置对象可使用。

Application_End 事件只有在服务中止或者该 Application 对象卸载时才会触发，如果单独使用 Application 对象，该事件可以通过 Application 对象在利用 Unload 事件卸载时触发。一个 Application_End 事件肯定发生在 Session_End 事件之后。

Application_End 事件触发惟一一个脚本程序，它存在于 Global.asax 文件中。

16.3.4 Application 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_03

本节通过一个简单的实例来介绍如何使用 Application 对象制作一个简单的聊天室。聊天室主页运行结果如图 16.13 所示。

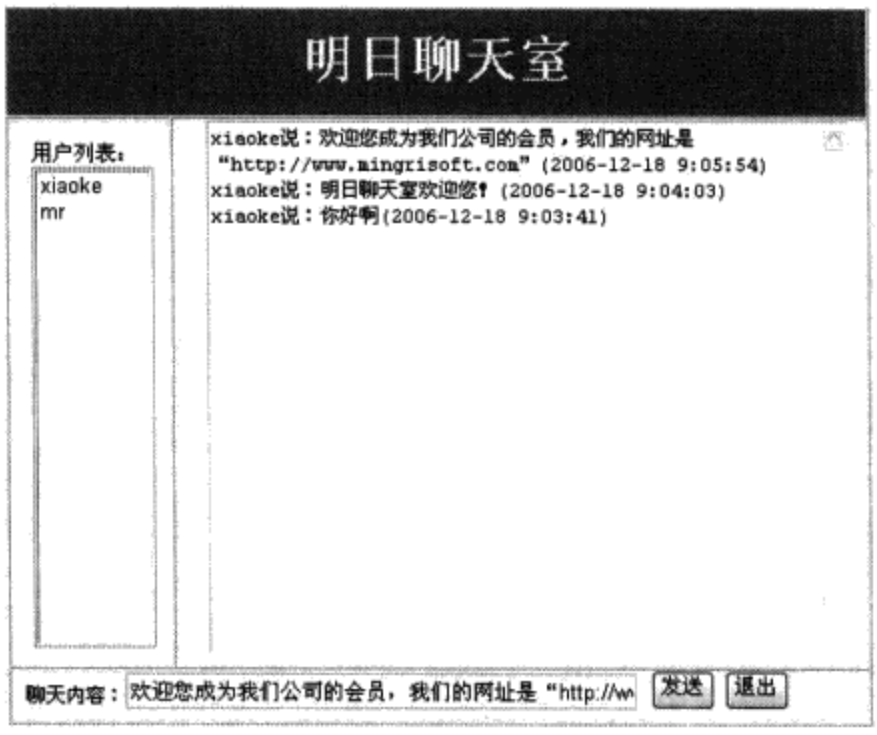


图 16.13 聊天室主页面

程序开发步骤如下。

（1）新建一个网站，命名为 16_03，其主页默认为 Default.aspx。

（2）在该网站中添加 3 个 Web 页面：Default2.aspx、Default3.aspx 和 Default4.aspx，其中 Default2.aspx 页面为聊天室的主页面，Default3.aspx 页面用来显示聊天信息，Default4.aspx 页面用来显示用户列表。

（3）在该网站中添加一个 Global.asax 全局程序集文件，用来初始化 Application 对象值。

（4）程序主要代码如下。

由于该聊天室是使用 Application 对象实现的，因此在应用程序启动时，应该将所有 Application 对象中的值设置为 0，其实现代码如下：

```
void Application_Start(object sender, EventArgs e)
{
    //在应用程序启动时运行的代码
    string user = "";                                   //建立用户列表
    Application["user"] = user;
    Application["userNum"] = 0;                         //人数
    string chats = "";                                  //聊天记录
    Application["chats"] = chats;
    Application["current"] = 0;                         //当前的聊天记录数
}
```

聊天室主页面中，单击【发送】按钮，程序调用 Application 对象的 Lock 方法对所有 Application 对象进行锁定，然后判断当前显示的信息记录数是否大于 20，如果大于，则将所有

记录清空，同时重新记录用户发送的信息；否则，在原有记录的基础上增加一条新信息。【发送】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    int P_int_current = int.Parse(Application["current"].ToString());          //获取聊天记录数
    Application.Lock();                                                        //锁定 Application 对象
    if (P_int_current == 0 || P_int_current > 20)                              //如果记录大于 20
    {
      P_int_current = 0;                                                       //清空记录数
      Application["chats"] = Session["userName"].ToString() + "说：" + TextBox1.Text.Trim() + "(" + DateTime.Now.
ToString() + ")";                                                              //重新记录用户发送的信息
    }
    else                                                                       //否则
    {
          Application["chats"] = Application["chats"].ToString() + "," + Session["userName"].ToString() + "说：" +
TextBox1. Text.Trim() + "(" + DateTime.Now.ToString() + ")";                    //在原有基础上增加新信息
    }
    P_int_current += 1;                                                        //记录数自增 1
    Application["current"] = P_int_current;                                    //设置 Application 对象中的记录数
    Application.UnLock();                                                      //解除锁定
}
```

显示聊天信息页面 Default3.aspx 加载时，从 Application 对象中读取保存的聊天信息，并将其显示在 TextBox 文本框中。Default3.aspx 页面的 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    int P_int_current = int.Parse(Application["current"].ToString());          //获取聊天记录数
    Application.Lock();                                                        //锁定
    string P_str_chats = Application["chats"].ToString();                      //获取聊天记录
    string[] P_str_chat = P_str_chats.Split(',');                              //拆分成字符串数组
    for (int i = P_str_chat.Length - 1; i >= 0; i--)                           //遍历字符串数组
    {
        if (P_int_current == 0)                                                //如果记录数为 0
        {
            TextBox2.Text = P_str_chat[i].ToString();                          //显示聊天记录
        }
        else                                                                   //否则
        {
            TextBox2.Text = TextBox2.Text + "\n" + P_str_chat[i].ToString();   //在原有基础上增加新信息
        }
    }
    Application.UnLock();                                                      //解除锁定
}
```

16.4 Session 对象

Session 对象是 HttpSessionState 类的一个实例，其功能和 Application 对象类似，都是用来存储跨网页程序的变量或者对象，但 Session 对象和 Application 对象的特性存在着差异。Session 对象只针对单一网页使用者，不同的客户端无法互相存取。Application 对象中止于停止 IIS 服务，而 Session 对象中止于联机机器离线时，也就是当网页使用者关掉浏览器或超过 Session 变量的有效时间时，Session 对象就会消失。

Session 对象和 Application 对象都是 Page 对象的成员，因此可直接在网页中使用。使用 Session 对象存放信息的语法如下：

```
Session["变量名"]= "内容";
```

从会话中读取信息的语法如下：

```
VariablesName=Session["变量名"];
```

16.4.1 Session 对象的属性

Session 对象常用属性及说明如表 16.7 所示。

表 16.7 **Session 对象常用属性及说明**

属　性	说　明
CodePage	获取或设置当前会话的字符集标识符
CookieMode	获取一个值，该值指示是否为无 Cookie 会话配置应用程序
Count	获取会话状态集合中的项数
IsCookieless	获取一个值，该值指示会话 ID 是嵌入在 URL 中还是存储在 HTTP Cookie 中
IsNewSession	获取一个值，该值指示会话是否是与当前请求一起创建的
IsReadOnly	获取一个值，该值指示会话是否为只读
IsSynchronized	获取一个值，该值指示对会话状态值的集合的访问是否是同步（线程安全）的
Item	获取或设置个别会话值
Keys	获取存储在会话状态集合中所有值的键集合
LCID	获取或设置当前会话的区域设置标识符（LCID）
Mode	获取当前会话状态模式
SessionID	获取会话的惟一标识符
SyncRoot	获取一个对象，该对象可用于同步对会话状态值的集合的访问
TimeOut	传回或设定 Session 对象的有效时间，当使用者超过有效时间没有动作，Session 对象就会失效。默认值为 20min

例如，可以在 Web.Config 文件中设置 Session 对象的有效时间为 30min，代码如下：

```
<sessionState mode="InProc"   timeout="30"/>
```

16.4.2 Session 对象的方法

Session 对象常用方法及说明如表 16.8 所示。

表 16.8 **Session 对象常用方法及说明**

方　法	说　明
Abandon	取消当前会话
Add	向 Session 对象集合中添加一个新项
Clear	清除全部的 Session 对象，但不结束会话
CopyTo	将会话状态值的集合复制到一维数组中（从数组的指定索引处开始）
GetEnumerator	返回一个枚举数，可用来读取当前会话中所有会话状态的变量名称
Remove	删除 Session 对象集合中的指定项
RemoveAll	从 Session 对象集合中移除所有项
RemoveAt	删除 Session 对象集合中指定索引处的项

下面对比较重要的方法进行详细介绍。

（1）Add 方法。用于向 Session 对象集合中添加一个新项。语法格式如下：

```
public void Add (string name,Object value)
```

参数说明如下。

name：要添加到 Session 对象集合中的项的名称。

value：要添加到 Session 对象集合中的项的值。

例如，在 Session 对象集合中添加两个项，通过两个按钮分别输出两个项目中的内容，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    Session.Add("s1","风笛");                          //在 Session 对象集合中添加新项
    Session.Add("s2","用一生下载你");                  //在 Session 对象集合中添加新项
}
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write(Session["s1"].ToString());          //输出 Session 对象集合中第一项的值
}
protected void Button2_Click(object sender, EventArgs e)
{
    Response.Write(Session["s2"].ToString());          //输出 Session 对象集合中第二项的值
}
```

（2）Remove 方法。用于删除 Session 对象集合中的指定项。语法格式如下：

```
public void Remove (string name)
```

参数说明如下。

name：要从 Session 对象集合中删除的项的名称。

例如，从 Session 对象集合中删除名称为“s1”的项，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //在 Session 对象集合中添加新项
    Session.Add("s1","风笛");
    Session.Add("s2","用一生下载你");
    Session.Remove("s1");                              // Remove 方法移除指定项
}
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write(Session["s1"].ToString());          //输出第一项的值
}
protected void Button2_Click(object sender, EventArgs e)
{
    Response.Write(Session["s2"].ToString());          //输出第二项的值
}
```

16.4.3 Session 对象的事件

1．Session_Start 事件

Session_Start 事件在服务器创建新会话（即事件）时发生，服务器在执行请求页之前先处理该脚本。Session_Start 事件是设置会话期变量的最佳时机，因为在访问任何页之前都会先调用该事件。Session_Start 事件和 Session_End 事件的脚本位于全局程序集文件 Global.asax 中。

2．Session_End 事件

Session_End 事件在会话（即事件）被放弃或超时时发生。在服务器内置对象中，只有 Application、Server 和 Session 对象可使用该事件。

16.4.4 Session 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_04

本节通过一个简单的实例来介绍如何使用 Session 对象来记录用户的登录名，并在其他页面中访问该 Session 对象。实例运行结果如图 16.14 和图 16.15 所示。

程序开发步骤如下。

（1）新建一个网站，命名为 16_04，其主页默认为 Default.aspx。

（2）在 Default.aspx 页面中添加一个 Table 表格，用来布局页面，然后在该 Table 表格中添加两个 TextBox 控件和两个 Button 控件，分别用来输入用户名、用户密码、执行登录和退出功能。

（3）在该网站中添加一个 Web 页面，命名为 Default2.aspx，然后在该页中添加两个 Label 控件，分别用来显示登录用户的姓名和本网站的总访问量。

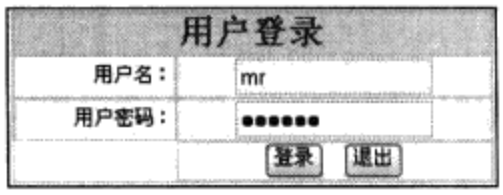


图 16.14　登录页面

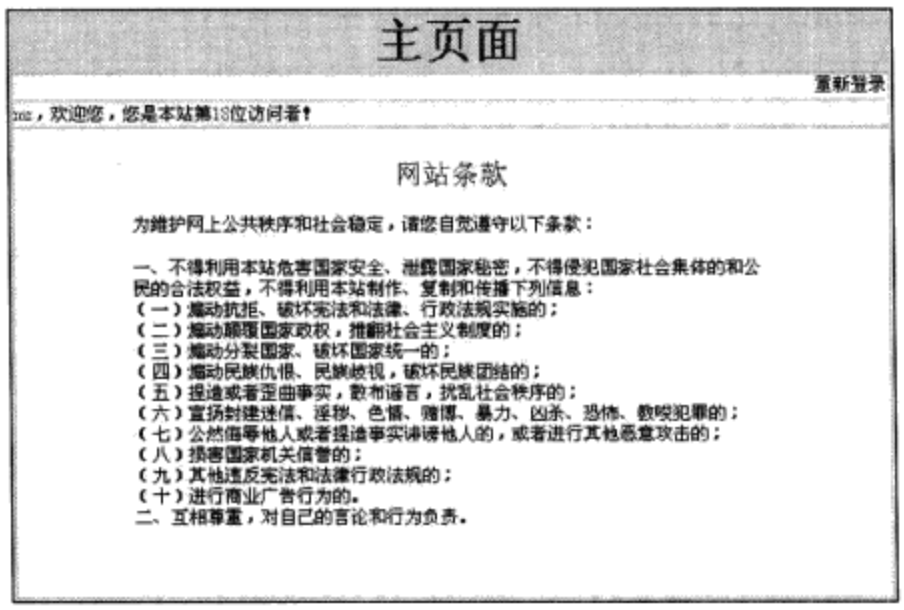


图 16.15　主页面

（4）程序主要代码如下。

Default.aspx 页面中，当用户单击【登录】按钮时，程序首先判断用户名文本框是否为空，如果为空，弹出信息提示框；否则，将页面跳转到 Default2.aspx 页面，同时定义一个 Session 对象来记录用户的登录名。【登录】按钮的 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (TextBox1.Text == "")                                    //如果没有输入用户名
    {
        Response.Write("<script>alert('用户名不能为空')</script>");   //弹出提示信息
        TextBox1.Focus();                                       //使输入用户名的文本框获得焦点
    }
    else                                                        //否则
    {
        Session["userName"] = TextBox1.Text.Trim();             //创建 Session 对象记录登录用户名
        Response.Redirect("Default2.aspx");                     //转向 Default2.aspx 页
    }
}
```

Default2.aspx 页面的 Page_Load 事件中，使用 Label 控件显示 Session 对象中记录的用户名，其实现代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
        Label1.Text = Session["userName"].ToString();           //显示登录用户名
        Label2.Text = Application["counter"].ToString();        //显示访问者数量
}
```

16.5　Server 对象

Server 对象是 HttpServerUtility 类的一个实例，它定义了一个与 Web 服务器相关的类，用来提供对服务器上方法和属性的访问。

16.5.1　Server 对象的属性

Server 对象常用属性及说明如表 16.9 所示。

表 16.9　　Server 对象常用属性及说明

属　性	说　明
MachineName	获取服务器的计算机名称
ScriptTimeout	获取和设置请求超时值（以 s 计）

下面对比较重要的属性进行详细介绍。

MachineName 属性。用来获取服务器的计算机名称。语法格式如下：

```
public string MachineName { get; }
```

属性值：本地计算机的名称。

示例 16-08　使用 MachineName 属性获取服务器名称

本示例实现的是，当程序运行时，在页面 Default.aspx 中显示服务器名称。示例运行结果如图 16.16 所示。

图 16.16　使用 MachineName 属性获取服务器名称

程序代码如下。

代码 16-08　光盘位置：光盘\mingrisoft\sl\16\16_08

在 Default.aspx 页面的 Page_Load 事件中，调用 Server 对象的 MachineName 属性获取服务器名称，然后使用 ToLower 方法将得到的服务器名称转换为小写，并在 Label 控件中显示出来，实现代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = Server.MachineName.ToLower();                    //显示获取的服务器名称
}
```

16.5.2　Server 对象的方法

Server 对象常用方法及说明如表 16.10 所示。

表 16.10　Server 对象常用方法及说明

方　法	说　明
ClearError	清除前一个异常
Execute	在当前请求的上下文中执行指定资源的处理程序，然后将控制返回给该处理程序
GetLastError	返回前一个异常
HtmlDecode	对已被编码以消除无效 HTML 字符的字符串进行解码
HtmlEncode	对要在浏览器中显示的字符串进行编码
MapPath	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径
Transfer	终止当前页的执行，并为当前请求开始执行新页
UrlDecode	对字符串进行解码，该字符串为了进行 HTTP 传输而进行编码并在 URL 中发送到服务器
UrlEncode	编码字符串，以便通过 URL 从 Web 服务器到客户端进行可靠的 HTTP 传输
UrlPathEncode	对 URL 字符串的路径部分进行 URL 编码，并返回已编码的字符串

下面对比较重要的方法进行详细介绍。

（1）HtmlDecode 和 HtmlEncode 方法。HtmlDecode 方法用于对 HTML 编码的字符串进行解码，并返回已解码的字符串。语法格式如下：

```
public string HtmlDecode (string s)
```

参数说明如下。

s：要解码的 HTML 字符串。

返回值：已解码的文本。

HtmlEncode 方法用于对字符串进行 HTML 编码，并返回已编码的字符串。语法格式如下：

```
public string HtmlEncode (string s)
```

参数说明如下。

s：要编码的文本字符串。

返回值：已使用 HTML 编码的文本。

示例 16-09　使用 HtmlDecode 和 HtmlEncode 方法编码和解码相同字符串

本示例实现的是当程序运行时，在页面 Default.aspx 中分别输出使用 Server 对象的 HtmlEncode 方法和 HtmlDecode 方法解码和编码的相同字符串。示例运行结果如图 16.17 所示。

图 16.17　使用 HtmlDecode 和 HtmlEncode 方法编码相同字符串

程序代码如下。

代码 16-09　光盘位置：光盘\mingrisoft\sl\16\16_09

在 Default.aspx 页面的 Page_Load 事件中，分别调用 Server 对象的 HtmlEncode 方法和 HtmlDecode 方法对相同的字符串进行编码和解码，然后在页面中输出，实现代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("使用 HtmlEncode 方法为指定字符串编码：" + Server.HtmlEncode("<B>浏览器中字符串</B>") + "<br>"
        + "使用 HtmlDecode 方法为指定字符串解码：" + Server.HtmlDecode("<B>浏览器中字符串</B>"));
}
```

（2）MapPath 方法。用于返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径。语法格式如下：

```
public string MapPath (string path)
```

参数说明如下。

path：Web 服务器上的虚拟路径。

返回值：与 path 相对应的物理文件路径。

例如，获取 Default.aspx 文件的物理路径，代码如下：

```
Server.MapPath("Default.aspx");
```

16.5.3　Server 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_05

本节通过一个简单的实例来介绍如何使用 Server 对象获得客户端和服务器的相关信息。实例运行结果如图 16.18 所示。

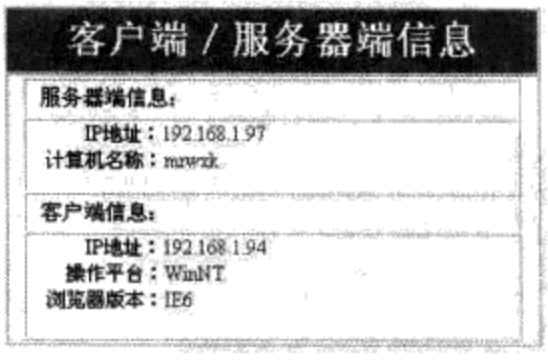


图 16.18　Server 对象的应用

程序开发步骤如下。

（1）新建一个网站，命名为 16_05，其主页默认为 Default.aspx。

（2）在 Default.aspx 页面中添加一个 Table 表格，用来布局页面，然后在该 Table 表格中添加 5 个 Label 控件，分别用来显示客户端和服务器端的相关信息。

（3）程序主要代码如下。

Default.aspx 页面的 Page_Load 事件中，分别调用 Server 对象和 Request 对象的相关属性获取客户端和服务器端信息，并将这些信息显示在相应的 Label 控件中。Default.aspx 页面的 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
        Label2.Text = Request.Browser.Platform;                    //获取客户端操作平台名称
        Label3.Text = Request.Browser.Type;                        //客户端浏览器版本
        Label5.Text = Server.MachineName.ToLower();                //服务端计算机名称
        Label1.Text = Request.UserHostAddress;                     //客户端 IP 地址
        string hostName = Dns.GetHostName();                       //获取本地计算机名称
        IPAddress[] hostip;
        hostip = Dns.GetHostAddresses(hostName);
        foreach (IPAddress ip in hostip)
        {
            Label4.Text = ip.ToString();                           //显示服务端 IP 地址
        }
    }
```

16.6 Cookie 对象

Cookie 对象是 HttpCookieCollection 类的一个实例，它用于保存客户端浏览器请求的服务器页面，也可存放非敏感性的用户信息，信息保存的时间可以根据用户的需要进行设置。如果没有设置 Cookie 失效日期，那么它们仅保存到关闭浏览器程序为止；如果将 Cookie 对象的 Expires 属性设置为 MinValue，则表示 Cookie 永远不会过期。

Cookie 存储的数据量有限制，大多数浏览器支持的最大容量为 4 096 字节，因此，一般不要用 Cookie 对象来保存数据集或其他大量数据。并非所有的浏览器都支持 Cookie，并且数据信息是以明文文本的形式保存在客户端计算机中，因此最好不要保存敏感的、未加密的数据，否则会影响网络的安全性。要存储一个 Cookie 变量，可以通过 Response 对象的 Cookies 集合，其使用语法如下：

```
Response. Cookies[Name].Value="资料";
```

要取回 Cookie，使用 Request 对象的 Cookies 集合，并将指定的 Cookie 集合返回，其使用语法如下：

```
变量名=Request. Cookies[Name].Value;
```

16.6.1 Cookie 对象的属性

Cookie 对象常用属性及说明如表 16.11 所示。

表 16.11　　Cookie 对象常用属性及说明

属　性	说　明
Comment	获取或设置服务器可添加到 Cookie 中的注释
CommentUri	获取或设置服务器可通过 Cookie 来提供的 URI 注释
Discard	获取或设置由服务器设置的丢弃标志
Domain	获取或设置 Cookie 对其有效的 URI
Expired	获取或设置 Cookie 的当前状态
Expires	设定 Cookie 变量的有效时间，默认为 1000min，若设为 0，则可以实时删除 Cookie 变量
HttpOnly	确定页脚本或其他活动内容是否可访问此 Cookie
Name	获取或设置 Cookie 变量的名称
Path	获取或设置 Cookie 适用于的 URI
Port	获取或设置 Cookie 适用于的 TCP 端口的列表
Secure	获取或设置 Cookie 的安全级别
TimeStamp	获取 Cookie 作为 DateTime 发出的时间
Value	获取或设置 Cookie 变量的内容值
Version	获取或设置 Cookie 符合的 HTTP 状态维护版本

下面对比较重要的属性进行详细介绍。

（1）Value 属性。用来获取或设置 Cookie 的值。语法格式如下：

```
public string Value { get; set; }
```

属性值：Cookie 的 Value。

示例 16-10 通过 Value 属性设置 Cookie 值

本示例实现的是，当程序运行时，单击页面 Default.aspx 中的【写入 Cookie】按钮，将数据写入 Cookie 中，然后单击【读取 Cookie】按钮，读取指定名称的 Cookie 值，示例运行结果如图 16.19 所示。

图 16.19 使用 Name 属性和 Value 属性获取定义的 Cookie 的名称和值

程序代码如下。

代码 16-10 光盘位置：光盘\mingrisoft\sl\16\16_10

```
protected void Button1_Click(object sender, EventArgs e)
{
    HttpCookie mycookie = new HttpCookie("userName");          //创建一个 Cookie 对象
    mycookie.Value = "风笛";                                    //向 Cookie 中写入数据
    Response.Write("正在写入 Cookie...");                       //输出信息
    Response.Cookies.Add(mycookie);                            //将创建的 Cookie 对象添加到集合中
    Response.Write("<p>Cookie 写入完毕");                       //写入完毕输出提示信息
}
protected void Button2_Click(object sender, EventArgs e)
{
    Response.Write("正在读取 Cookie...");                       //输出提示信息
    string cookie = Request.Cookies["userName"].Value.ToString();  //获取写入的 Cookie 值
    Response.Write("<p>Cookie 值为："+cookie);                  //输出获取的 Cookie 值
}
```

（2）Expires 属性。设置 Cookie 的有效期限。语法格式如下：

```
public DateTime Expires { get; set; }
```

属性值：Cookie 过期的日期和时间。

例如，通过设置 Cookie 对象的 Expires 属性，实现 Cookie 的有效期限是 3 小时，代码如下：

```
Response.Cookies["userName"].Expires = DateTime.Now.AddHours(2);
```

16.6.2 Cookie 对象的应用

实例位置：光盘\mingrisoft\dxyy\16\16_06

本节通过一个简单的实例来介绍如何使用 Cookie 对象实现设置登录有效期的功能，实例运行结果如图 16.20 和图 16.21 所示。

程序开发步骤如下。

（1）新建一个网站，命名为 16_06，默认主页名为 Default.aspx。

（2）在 Default.aspx 页面中添加一个 Table 表格，用来布局页面，然后，在该 Table 表格中添加两个 DropDownList 控件和 3 个 TextBox 控件，分别用来选择安全问题、选择 Cookie 有效期、输入用户名、密码和问题答案。

（3）在该网站中添加一个 Web 页面，命名为 Default2.aspx，该页面主要用来显示登录信息。

图 16.20 用户登录页面

图 16.21 用户登录成功界面

（4）程序主要代码。

在 Default.aspx 页面中，可以选择保存 Cookie 信息的有效期，这样，当下一次打开此页面登录系统时不用输入用户名和密码，直接便可以登录。输入用户名和密码后，单击【登录】按钮将登录信息保存到 Cookie 中，并转到登录成功界面，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (txtName.Text == "" || txtPwd.Text == "")                          //如果用户名和密码为空
    {
        RegisterStartupScript( "", "<script>alert('输入用户名或密码！')</script>");   //弹出提示信息
    }
    else                                                                  //否则
    {
        string name = HttpUtility.UrlEncode(txtName.Text.Trim());         //获取登录用户名
        string pwd = HttpUtility.UrlEncode(txtPwd.Text.Trim());           //获取登录密码
        string question = ddlQ.SelectedItem.Text;                         //获取安全问题
        string answer = txtA.Text.Trim();                                 //获取安全问题的答案
        Session["q"] = question;                                          //将问题写入 Session 对象
        Session["a"] = answer;                                            //将答案写入 Session 对象
        Session["validperiod"] = ddlvalidperiod.SelectedItem.Text;        //获取 Cookie 有效期
        int index = ddlvalidperiod.SelectedIndex;                         //获取有效期的索引
        switch (index)
        {
            case 0:                                                       //如果选择第一项
                //登录用户名和密码的有效期为 1 小时
                Response.Cookies["userName"].Expires = DateTime.Now.AddHours(1);
                Response.Cookies["userPwd"].Expires = DateTime.Now.AddHours(1);
                break;
            case 1:                                                       //如果选择第二项
                //登录用户名和密码的有效期为 1 天
                Response.Cookies["userName"].Expires = DateTime.Now.AddDays(1);
                Response.Cookies["userPwd"].Expires = DateTime.Now.AddDays(1);
                break;
            case 2:                                                       //如果选择第三项
                //登录用户名和密码的有效期为 1 个月
                Response.Cookies["userName"].Expires = DateTime.Now.AddMonths(1);
                Response.Cookies["userPwd"].Expires = DateTime.Now.AddMonths(1);
                break;
            case 3:                                                       //如果选择第四项
                //登录用户名和密码的有效期为永不过期
                Response.Cookies["userName"].Expires = DateTime.MinValue;
                Response.Cookies["userPwd"].Expires = DateTime.MinValue;
                break;
        }
        Response.Cookies["userName"].Value = name;                        //将用户名写入 Cookie 对象
        Response.Cookies["userPwd"].Value = pwd;                          //将密码写入 Cookie 对象
        Response.Redirect("~/Default2.aspx");                             //转向 Default2.aspx 页
}
```

第 17 章

ASP.NET 3.5 母版页

- 母版页的概述
- 创建母版页
- 创建内容页
- 嵌套母版页
- 访问母版页的控件和属性

为了给访问者一致的感觉，每个网站都需要具有统一的风格和布局。例如，整个网站具有相同的网页头尾、导航栏、功能条以及广告区等。对于这一点，在不同的技术发展阶段有着不同的实现方法。

在 ASP.NET 1.x 中，开发人员使用用户控件将网站中公用部分整合为一个控件，当需要实现时，在页面的适当位置引入该用户控件。这种方法可以实现一致性的要求，并且也能够提高创建页面的工作效率、缩短页面的维护时间，但是这种方法不是开发技术的内建方法，也就是说，需要开发人员自己套用，才能享受利用模板设计网站的好处。

在 ASP.NET 3.5 中提出了一个新功能—— 母版页。设计母版页的目的就是要在从内部建立支持网页模板的功能，实现网站一致性要求。本章内容将重点介绍母版页的优点，并使用母版页功能快速创建风格一致的应用程序。

17.1 母版页的概述

ASP.NET 3.5 提供的母版页为提高工作效率、降低开发和维护强度提供了有力支持。本节将从以下几个方面来介绍母版页。

1．母版页和内容页的概念

母版页能够为 ASP.NET 3.5 应用程序创建统一的用户界面和样式，在实现网站一致性的过程中，必须包含两种文件：一种是母版页，另一种是内容页。母版页后缀名是.master，用来封装页面中的公共元素。内容页实际是普通的.aspx 文件，它包含除母版页之外的其他非公共内容。在运行过程中，ASP.NET 3.5 引擎将两种页面内容合并执行，最后将结果发送给客户端浏览器。

2．母版页运行机制

单独的母版页是不能被用户访问的。没有内容页支持，母版页仅仅是一个页面模板，没有使用价值。同样道理，单独的内容页没有母版页支持，也不能够应用。由此可见，母版页与内容页关系密切，是不可分割的两个部分。只有同时正确创建和使用母版页以及内容页，才能发挥它们的强大功能。下面从代码结构和母版页的运行过程两个方面说明母版页和内容页的相互关系。

● 从代码结构方面，对母版页和内容页的相互关系进行深入说明。

母版页以页面的公共部分为主，包括代码头、ContentPlaceHolder 控件以及其他常见 Web 元素。新创建的母版页源代码如下：

```
<%@ Master Language="C#" AutoEventWireup="True" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/ TR/xhtml1/DTD/
xhtml1- transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
        </asp:contentplaceholder>
    </div>
    </form>
</body>
</html>
```

母版页中的占位符

说明：

① 母版页在页面定义部分把.aspx 页面中的@Page 替换成了@Master。

② 代码中的<asp:contentplaceholder></asp:ContentPlaceHolder>部分称为占位符，在母版页中，可以根据需要，设置多个占位符。具体在页面上显示什么内容，则由内容页来决定。

内容页主要包含页面非公共部分，是由代码头和 Content 控件组成的。Content 控件中包含着页面的非公共内容。与母版页相关联的内容页源代码如下：

```
<%@ Page Language="C#" MasterPageFile="～/MasterPage.master" AutoEventWireup="True" CodeFile="Default.aspx.cs" Inherits="_Default" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
```

说明：

① 内容页的声明部分指明要调用的母版页，并且去掉页面的顶级元素，如 html、body 等元素。

② 必须包含 Content 服务器控件，通过该控件的 ContentPlaceHolderID 属性来与母版页中的 ContentPlaceHolder 服务器控件相关联。

③ 内容页中可以有多个 Content 服务器控件，其中内容页里的 Content 服务器控件的 ContentPlaceHolderID 属性值必须与母版页中的 ContentPlaceHolderID 服务器控件的 ID 属性匹配，否则会出错。

● 从母版页的运行过程说明两者的相互关系。

当客户端浏览器向服务器发出请求，要求浏览页面时，ASP.NET 3.5 执行引擎将执行内容页和母版页的代码，并将最终结果发送给客户端浏览器，其运行过程可以概括为以下 5 个步骤。

① 用户通过键入内容页的 URL 来请求某页。

② 获取内容页后，读取@Page 指令。如果该指令引用一个母版页，则也读取该母版页。如果是第一次请求这两个页，则两个页都要进行编译。

③ 母版页合并到内容页的控件树中。

④ 各个 Content 控件的内容合并到母版页中相应的 ContentPlaceHolder 控件中。

⑤ 呈现得到的结果页。

整个过程具有很强的逻辑性，并且母版页和内容页配合得非常巧妙。从用户角度来看，合并后的母版页和内容页是一个完整的页面，并且其 URL 访问路径与内容页的路径相同。从开发人员的角度来看，控件的巧妙应用和配合是实现的关键。

注意：在运行时，母版页成为了内容页的一部分。

3．母版页和内容页事件的顺序

母版页和内容页中的事件顺序对于网站开发人员并不重要。但是，如果所创建的事件处理程序取决于某些事件的可用性，那么了解母版页和内容页中的事件顺序就变得非常重要。

在某些情况下，内容页和母版页中会引发相同的事件。例如，两者都引发 Init、Load 和 PreRender 事件。引发事件的一般规则是，初始化 Init 事件从最里面的控件（母版页）向最外面的控件（Content 控件即内容页）引发，所有其他事件则从最外面的控件向最里面的控件引发。需要牢记，母版页会合并到内容页中，并被视为内容页中的一个控件，这一点十分重要。母版页与内容页合并后事件的发生顺序如下。

（1）母版页中控件的 Init 事件。

（2）内容页中 Content 控件的 Init 事件。

（3）母版页 Init 事件。

（4）内容页 Init 事件。

（5）内容页 Load 事件。

（6）母版页 Load 事件。

（7）内容页中 Content 控件的 Load 事件。

（8）内容页 PreRender 事件。

（9）母版页 PreRender 事件。

（10）母版页控件的 PreRender 事件。

（11）内容控件的 PreRender 事件。

在创建应用程序时，必须注意事件的发生顺序。例如，当在内容页中访问母版页的属性或者服务器控件时，如果按照过去的处理思路，可能会在内容页的 Page_Load 事件处理程序中加以实现。由母版页与内容页合并后事件的发生顺序可知，在母版页 Load 事件引发之前，内容页 Load 事件已经引发，那么过去的思路显然是不正确的。

交叉链接：如何才能在内容页中访问母版页包含对象，请参见本章的 17.5 节。

4．母版页的优点

使用母版页，可以为 ASP.NET 3.5 应用程序页面创建一个通用的外观。开发人员可以利用母版页创建一个单页布局，然后将其应用到多个内容页中。母版页具有下面的优点。

- 由于公共内容被集中在母版页中，因此，只要修改母版页，就可以快速完成网站的修改和维护任务，这在很大程度上提高了工作效率。
- 使用母版页可以方便地创建一组公共控件和代码，并将其应用于网站中所有引用该母版页的网页。例如，可以在母版页上使用控件来创建一个应用于所有页的功能菜单。
- 可以通过控制母版页中的占位符 ContentPlaceHolder 对网页进行布局。
- 由内容页和母版页组成的对象模型，能够为应用程序提供一种高效、易用的实现方式，并且这种对象模型的执行效率比以前的处理方式有了很大的提高。

17.2 创建母版页

本节主要介绍如何使用 Visual Stuido 2008 创建母版页。母版页中包含的是页面的公共部分，因此，在创建母版页之前，必须判断哪些内容是页面的公共部分。图 17.1 所示为网上商城购物系统的前台首页 Index.aspx，该网页是由 4 部分组成的，即页头、页尾、功能菜单栏和内容页，其中页头、页尾和功能菜单栏是网上商城购物系统中页面的公共部分，内容 1 是网上商城购物系统的非公共部分，是 Index.aspx 页面所独有的。结合母版页和内容页的相关知识可知，如果使用母版页和内容页创建 Index.aspx 页面，那么必须创建一个母版页 MasterPage.master 和一个内容页 Index.aspx。其中母版页包含页头、页尾和功能菜单栏，内容页则包含内容。

在网上商城购物系统 NetShop 网站中，创建母版页的具体步骤如下。

（1）在 NetShop 网站的解决方案下，右键单击网站名称，在弹出的快捷菜单中选择“添加新项”命令。

（2）打开“添加新项”对话框，如图 17.2 所示。在图中选择母版页并为其命名，单击【添加】按钮就可以创建一个新的母版页。

注意：

① 在图 17.2 中，有一个“将代码放在单独的文件中”复选框，默认情况下，该复选框处于选中状态，表示 Visual Studio 2008 将会为 MasterPage.master 文件应用代码隐藏模型，即在创建 MasterPage.master 文件的基础上，自动创建一个与该文件相关的 MasterPage.master.cs 文件。如果不选中该项，那么只会创建一个 MasterPage.master 文件而已。建议读者选取该项。

② 母版页都是以.master 扩展名结尾，以防止浏览器直接打开。

交叉链接：新创建的母版页代码及其说明请参见 17.1 节。

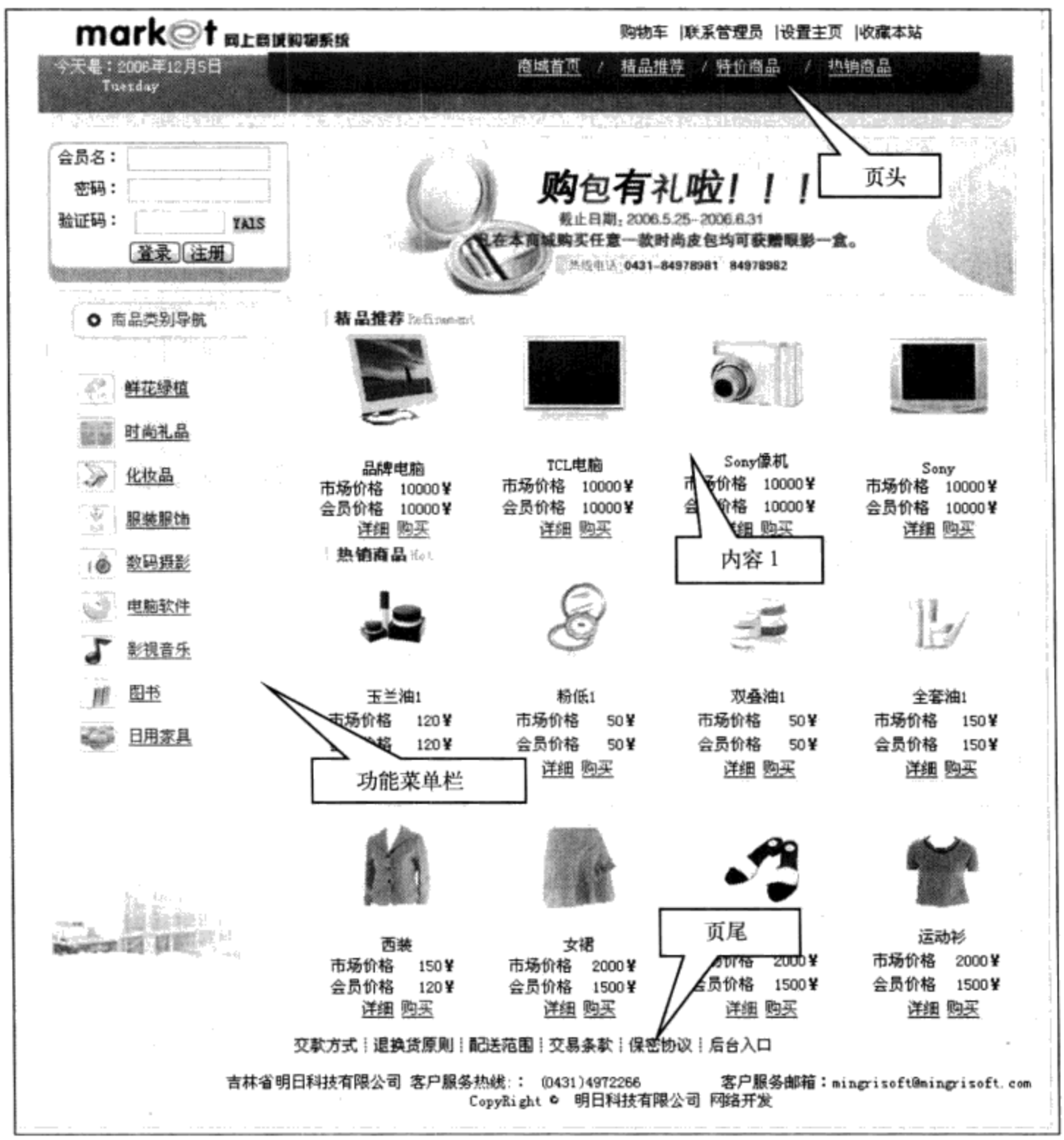


图 17.1 网上商城购物系统前台首页

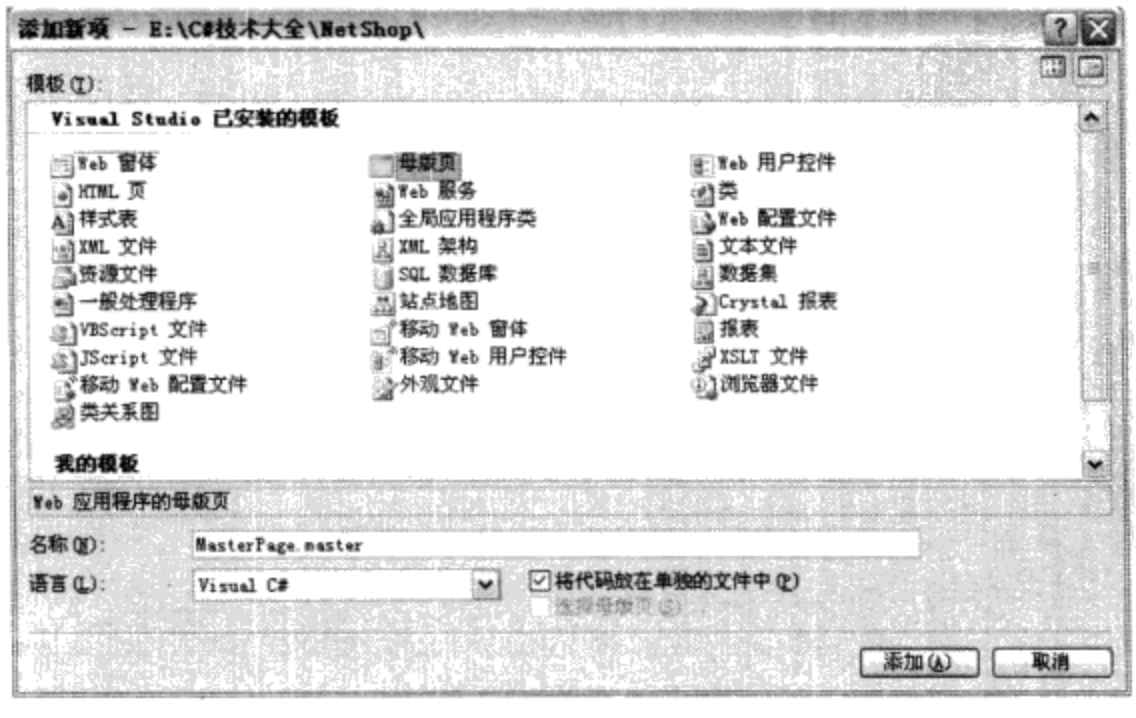


图 17.2 创建母版页

17.3 创建内容页

内容页的创建步骤如下。

（1）在 NetShop 网站的解决方案下，右键单击网站名称，在弹出的快捷菜单中选择“添加新项”命令。

（2）打开“添加新项”对话框，如图 17.3 所示。在图中选择 Web 窗体并为其命名，同时选中“将代码放在单独的文件中”和“选择母版页”复选框，单击【添加】按钮，弹出如图 17.4 所示的“选择母版页”对话框，在该窗体中，选择一个母版页，单击【添加】按钮，就可以创建一个新的内容页。

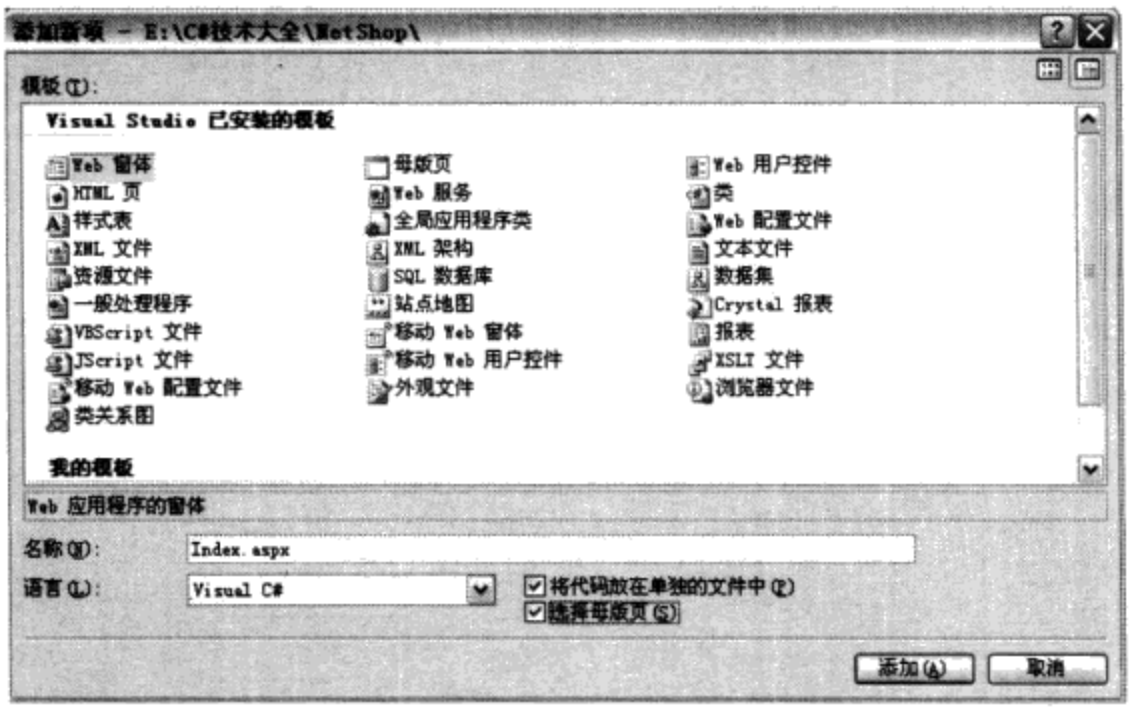


图 17.3 创建内容页

注意：

① 内容页与普通.aspx 页面的扩展名相同。

② 选择“将代码放在单独的文件中”和“选择母版页”复选框是为了将内容页与母版页绑定。

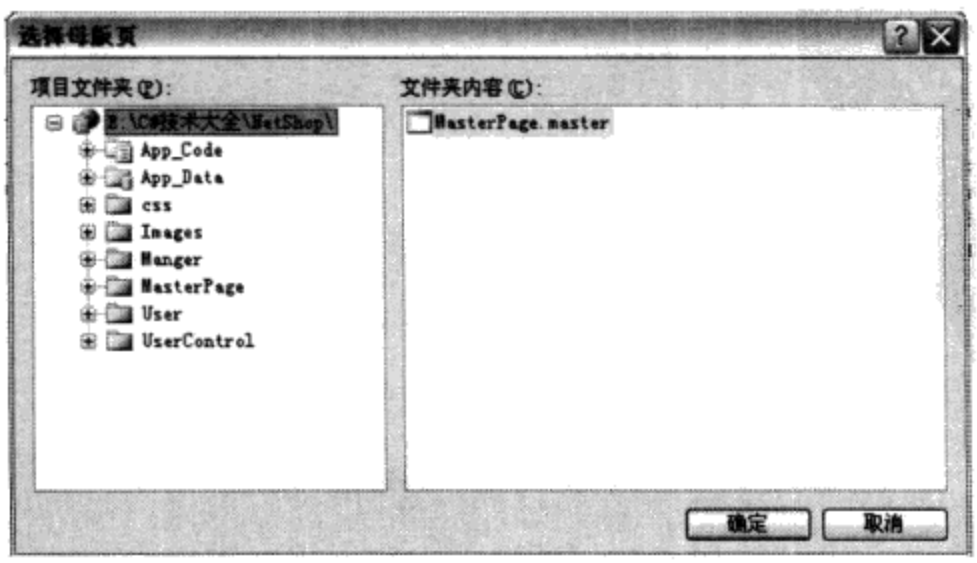


图 17.4 选择母版页

17.4 嵌套母版页

母版页可以嵌套，可以创建让一个母版页引用另外一个页作为其母版页。利用嵌套的母版页可以创建组件化的母版页。例如，大型网站可能包含一个用于定义站点外观的总体母版页，然后，不同的网站内容合作伙伴又可以定义各自的子母版页，这些子母版页引用网站母版页，并相应定义合作伙伴的内容外观。嵌套母版页的示意图如图 17.5 所示。

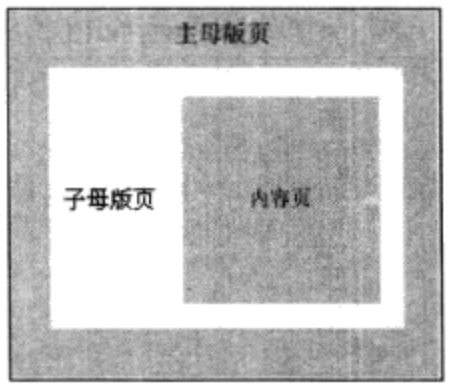


图 17.5 嵌套母版页的示意图

注意：无论母版页如何嵌套构建页面，必须包含一个内容页，原因是客户端浏览器不能访问扩展名为.master 的母版页。

与任何母版页一样，子母版页的扩展名为.master。它通常会包含一些 Content 控件，这些控件将映射到主母版页的 ContentPlaceHolder 控件上，因此，子母版页具有一定占位功能。就这方面而言，子母版页的布局方式与所有内容页类似。但是，子母版页有自己的 ContentPlaceHolder 控件，包含在 Content 控件中，可用于显示它所绑定的内容页提供的内容。

示例 17-01 创建一个简单的嵌套母版页

本示例将通过一个简单的嵌套母版页示例来加深读者对嵌套母版页的理解。程序开发步骤如下。

（1）新建一个网站，将其命名为 17_01。

（2）在该网站的解决方案下，右键单击网站名称，在弹出的快捷菜单中选择“添加新项”，打开“添加新项”对话框，首先添加两个母版页，分别命名为 MainMasterPage（主母版页）和 SubMasterPage（子母版页），然后再添加一个 Web 窗体，命名为 Default.aspx，并将其作为 SubMasterPage（子母版页）的内容页。

示例运行结果如图 17.6 所示。

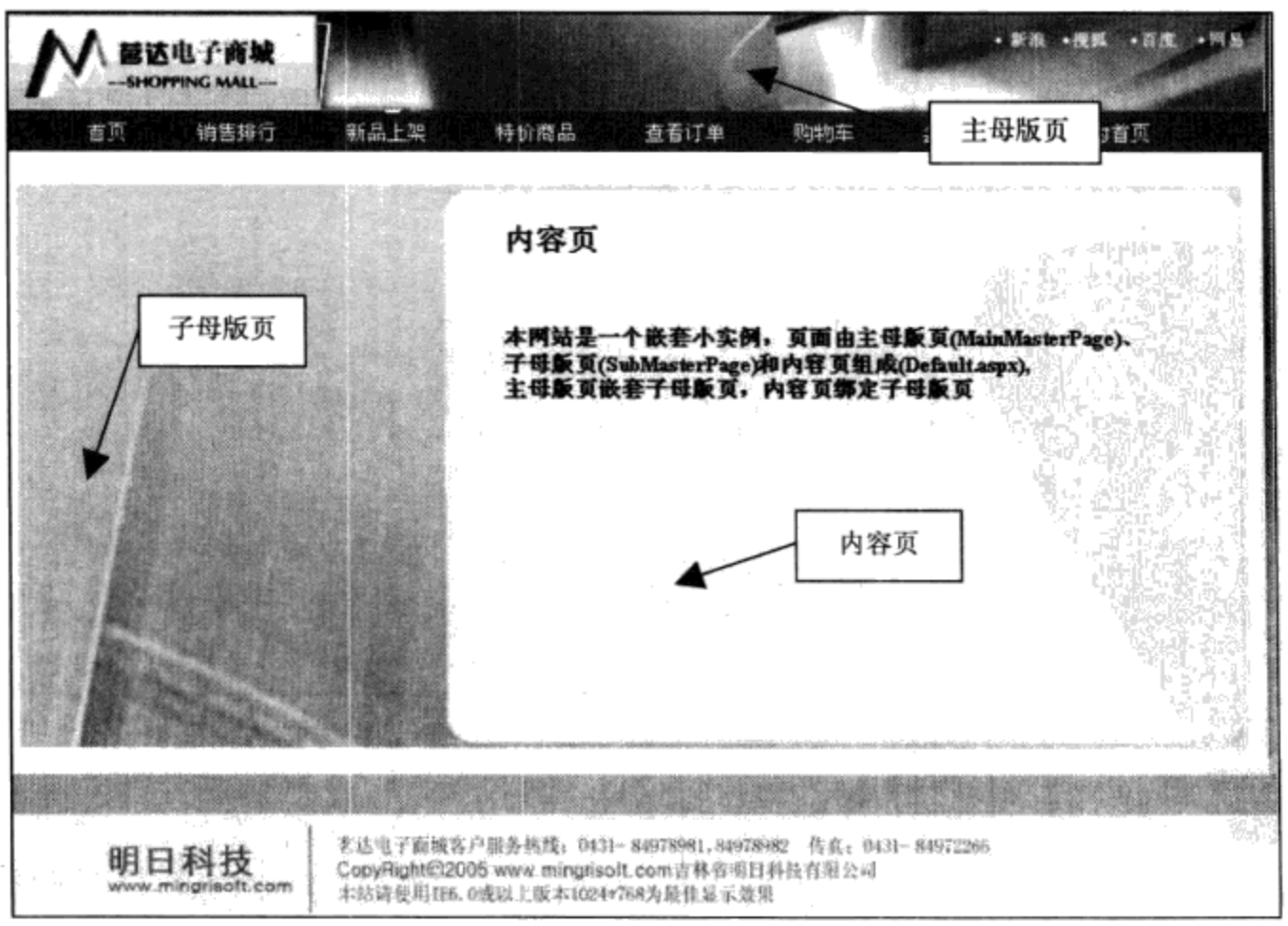


图 17.6 嵌套母版页示例图

图 17.6 所示的页面是由主母版页（MainMasterPage）、子母版页（SubMasterPage）和内容页（Default.aspx）组成的，主母版页包含的内容主要是页面的公共部分，主母版页嵌套子母版页，内容页绑定子母版页。

主母版页 MainMasterPage.master 源代码如下：

代码 17-01　光盘位置：光盘\mingrisoft\sl\17\17_01

```
<%@ Master Language="C#" AutoEventWireup="True" CodeFile="MainMasterPage.master.cs" Inherits="MainMaster
Page" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1- transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>主母版页</title>
    <style >
   body{ margin-top:0px;}
    </style>
</head>
<body >
    <form id="form1" runat="server">
        <table   style=" width :762px; height :601px;" border =0   background ="Images/框架结构图.jpg"   align =center >
            <tr>
                <td   style="height: 80px; width: 641px;" align =center >
                <h1></h1>
                </td>
            </tr>
            <tr>
                <td align =center valign =top>
                <asp:ContentPlaceHolder ID ="MainContent" runat =server >
                </asp:ContentPlaceHolder>

                </td>
            </tr>
            <tr>
            <td style="height: 41px; width: 641px;"   align =center   valign =top >
            </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

母版页中的占位符

注意：主母版页的构建方法与普通母版页使用的方法一致。由于主母版页嵌套一个子母版页，因此必须在适当的位置设置一个 ContentPlaceHolder 控件实现占位功能。

子母版页 SubMasterPage.master 源代码如下：

```
<%@ Master Language="C#" AutoEventWireup="True" CodeFile="SubMasterPage.master.cs"
Inherits="SubMasterPage"   MasterPageFile ="~/MainMasterPage.master" %>
<asp:Content id="Content1" ContentPlaceholderID="MainContent" runat="server">
<table style=" width :730px; height :307px;"   border =0 align =center     background ="Images/框架结构图 2.jpg" >
<tr valign =middle>
<td>
 <h2> </h2>
</td>
<td>
 <asp:ContentPlaceHolder ID="SubContent" runat="server" >
 </asp:ContentPlaceHolder>
</td>
</tr>
</table>
</asp:Content>
```

子母版页中的占位符

说明：

① 子母版页以.master 为扩展名，其代码包括两个部分，即代码头声明和 Content 控件。

② 子母版页与普通母版页相比，子母版页中不包括<html>、<body>等 Web 元素。

③ 子母版页的代码头添加了一个 MasterPageFile 属性，以设置嵌套子母版页的主母版页路径，通过设置这个属性，实现主母版页和子母版页之间的嵌套。

④ 子母版页中的 Content 控件中声明的 ContentPlaceHolder 控件用于为内容页实现占位。

内容页 Default.aspx 源代码如下：

```
<%@ Page Language="C#" MasterPageFile="~/SubMasterPage.master" AutoEventWireup="True"
 CodeFile=" Default. aspx. aspx.cs" Inherits="Index" Title="ChildPage" %>
<asp:Content id="Content1" ContentPlaceholderID="SubContent" runat="server">
<table   border =0 align =right      style ="width :474px; height :331px;"   background ="Images/框架结构图 3.jpg">
<tr>
<td   align =left valign =top   >
<br />
  <h3 >内容页</h3><br />
  <h5>本网站是一个嵌套小实例，页面由主母版页(MainMasterPage)、<br />
  子母版页(SubMasterPage)和内容页组成(Index.aspx),<br />
  主母版页嵌套子母版页，内容页绑定子母版页</h5>
</td>
</tr>
</table>
</asp:Content>
```

引用子母版页

说明：

① 内容页的构建方法与普通内容页的一致。

② 内容页的代码包括两部分，即代码头声明和 Content 控件。由于内容页绑定子母版页，所以代码头中的属性 MasterPageFile 必须设置为子母版页的路径。

注意：创建嵌套母版页过程中，Visual Studio 2008 只支持对主母版页的可编辑，而不支持子母版页和内容页。所以，必须在代码模式下创建子母版页和内容页。

17.5 访问母版页的控件和属性

母版页和内容页都可以有各自的控件和属性，而且在各自的内部引用是比较容易的。如果从内容页中引用母版页的控件和属性，那将会变得有些复杂。本节将向读者介绍在内容页中如何编写代码访问母版页的控件和属性。

1．使用 Master.FindControl()方法访问母版中的控件

在内容页中，对象 Page 具有一个公共属性 Master，该属性能够实现对相关母版页基类 MasterPage 的引用。母版页中的 MasterPage 相当于普通 ASP.NET 页面中的 Page 对象，因此，可以使用 Masterpage 对象，实现对母版页中各个子对象的访问，但由于母版页中的控件是受保护的，不能直接访问，因此必须使用 MasterPage 对象的 FindControl 方法实现。

示例 17-02　使用 FindControl 方法获取母版页的控件

本示例将向读者介绍如何使用 FindControl 方法，获取母版页中用于显示系统时间的 Label 控件。示例运行结果如图 17.7 所示。

程序开发步骤如下。

（1）新建一个网站，命名为 17_02。

（2）在该网站的解决方案下，右键单击网站名称，然后单击“添加新项”，打开“添加新项”对话框，首先添加一个母版页，命名为 MainMasterPage.master；再添加一个 Web 窗体，命名为 Default.aspx，作为母版页的内容页。

（3）在母版页 MainMasterPage.master 上添加一个 Label 控件，用于显示母版页中的系统日期；再在内容页 Default.aspx 中添加一个 Label 控件，获取母版页中的 Label 控件值，并将其显示出来。

图 17.7 所示的 Web 页是由母版页和内容页组成的，母版页包括页头和页尾，内容页是中间的非公共部分。在母版页中定义了一个 Label 控件，使其显示系统当前的日期，在内容页中编写代码获取母版页中的 Label 控件的 Text 值，赋给内容页中的 Label 控件，并将其显示出来。

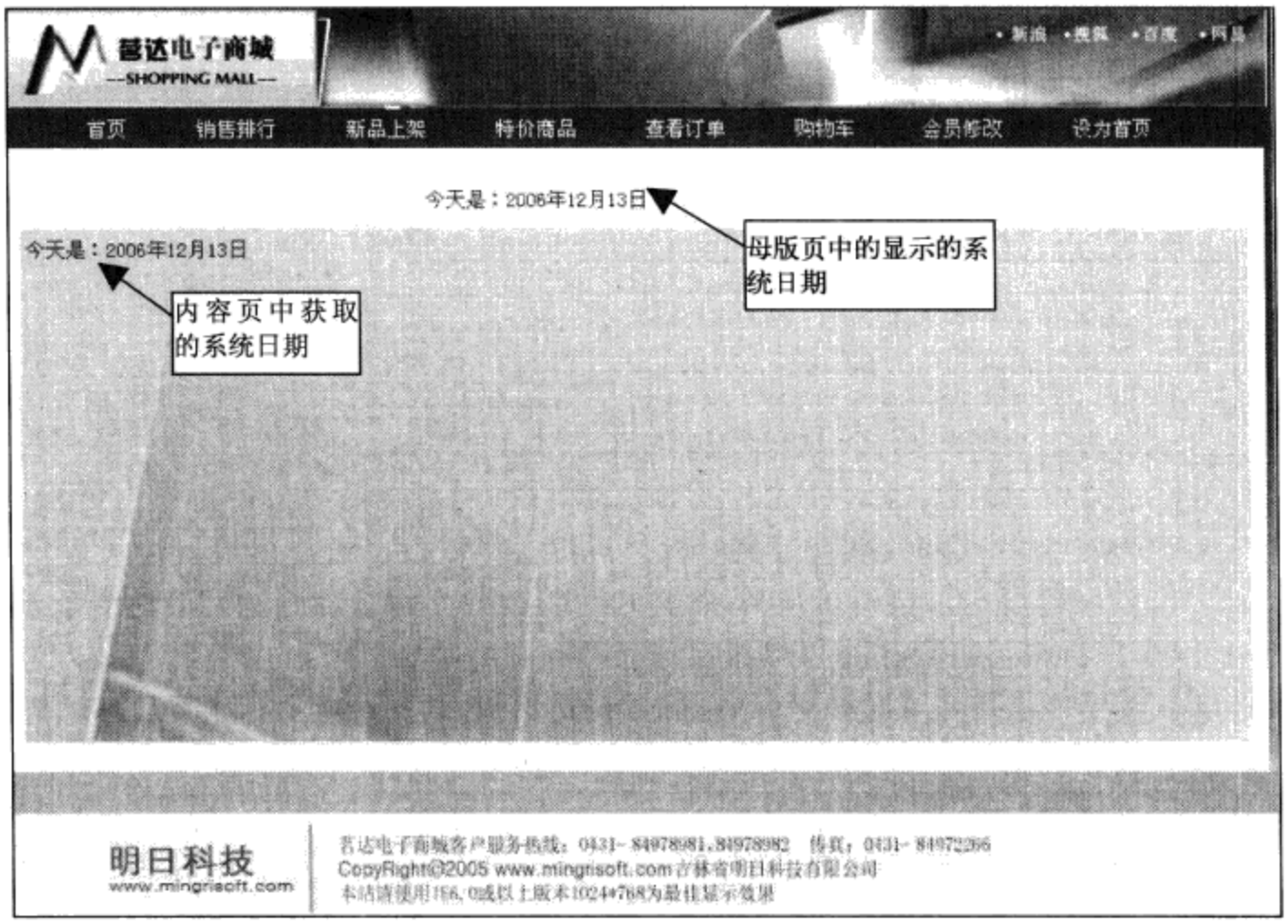


图 17.7　获取母版页中控件的示例

母版页 MainMasterPage.master 源代码如下：

代码 17-02　光盘位置：光盘\mingrisoft\sl\17\17_02

```
<%@ Master Language="C#" AutoEventWireup="True" CodeFile="MainMasterPage.master.cs" Inherits=" MainMasterPage" %>
<script runat =server   >
    protected void Page_Load(object sender, EventArgs e)
    {
        MasterLable.Text = "今天是：" + System.DateTime.Now.ToLongDateString();
    }
</script>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/ TR/xhtml1/DTD/
xhtml1- transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>访问母版页的成员</title>
    <style >
   body{ margin-top:0px;}
    </style>
</head>
<body >
    <form id="form1" runat="server">
        <table   style=" width :762px; height :601px;" border =0   background ="Images/框架结构图.jpg"   align =center >
            <tr>
                <td   style="height: 43px; width: 641px;" align =center >
                </td>
            </tr>
            <tr style="width: 641px; height: 17px">
```

母版页中的 Label 控件显示系统时间

```
            <td>
            </td>
            </tr>
            <tr>
                <td align="center" style="width: 641px; height: 17px">
                    <font color=red style="font-size: 9pt; font-family: 宋体" >
                     <asp:Label ID="MasterLable" runat="server"
    Text="Label"></asp:Label></font></td>
            </tr>
            <tr>
                <td align =center valign =top>
                <asp:ContentPlaceHolder ID ="MainContent" runat =server >
                 </asp:ContentPlaceHolder>

                 </td>
            </tr>
            <tr>
            <td style="height: 41px; width: 641px;"   align =center   valign =top >
            </td>
            </tr>
        </table>
    </form>
  </body>
  </html>
```

在母版页中添加 Label 控件

母版页中的占位符

说明：将母版页中 Label 控件命名为 MasterLable，并在 Page_Load 事件下，将系统日期赋给 Label 控件的 Text 值。

注意：为了方便读者的理解，笔者将 Page_Load 事件放在了源代码中，实际应用中，也可以将 Page_Load 事件放在后台代码中。

内容页 Default.aspx 源代码如下：

```
    <%@ Page Language="C#" MasterPageFile="~/MainMasterPage.master" AutoEventWireup="True" CodeFile=
"Default.aspx.cs" Inherits="_Default" Title="Untitled Page" %>
    <script runat =server >
     protected void ContentLabel_PreRender(object sender, EventArgs e)
       {
          Label MLable1 = (Label)this.Master.FindControl("MasterLable");
          ContentLabel.Text = MLable1.Text;
       }
    </script>
    <asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
    <table background ="Images/框架结构图 2.jpg" style=" width :730px; height :307px;"   border =0 align =center>
    <tr align =left valign =top   >
    <td >
        <font color=red style="font-size: 9pt; font-family: 宋体" ><asp:Label ID="ContentLabel" runat="server"
Text="Label" OnPreRender="ContentLabel_PreRender"></asp:Label></font>
    </td>
    </tr>
    </table>
    </asp:Content>
```

使用 Master.FindControl 方法获取母版页中的 Label 控件值

说明：由母版页和内容页的运行顺序可知，在母版页的 Page_Load 事件引发之前，内容页 Page_Load 事件已经引发。在这种情况下，从内容页中访问母版页中的控件比较困难。所以，本实例在 Label 控件的 PreRender 事件下利用 FindControl()方法来获取母版页的控件，当然还可以在 ASP.NET 3.5 新增的 Page.LoadComplete 事件下完成此功能，其中 Page.LoadComplete 事件是在生命周期内、网页加载结束时触发。

2．使用 MasterType 指令引用母版页的公共属性

在内容页中可以引用母版页中的属性和方法，但这种引用有一定的限制，母版页上的属性和方法必须被声明为公共成员，才可以引用它们。当然，引用母版页中的属性和方法，还需要在内容页中使用 MasterType 指令，将内容页的 Master 属性强类型化，即通过 MasterType 指令创建与内容页相关的母版页的强类型引用。另外，在设置 MasterType 指令时，必须设置

VirtualPath 属性以便指定与内容页相关的母版页存储地址。

示例 17-03　MasterType 指令引用母版页的公共属性

交叉链接：本示例是在示例 17-02 的基础上的扩展，将向读者介绍如何在内容页中使用 MasterType 指令引用母版页的公共属性，并将“『明日科技欢迎您！』”字样赋给母版页的公共属性。程序开发步骤请参见示例 17-02。

示例运行结果如图 17.8 所示。

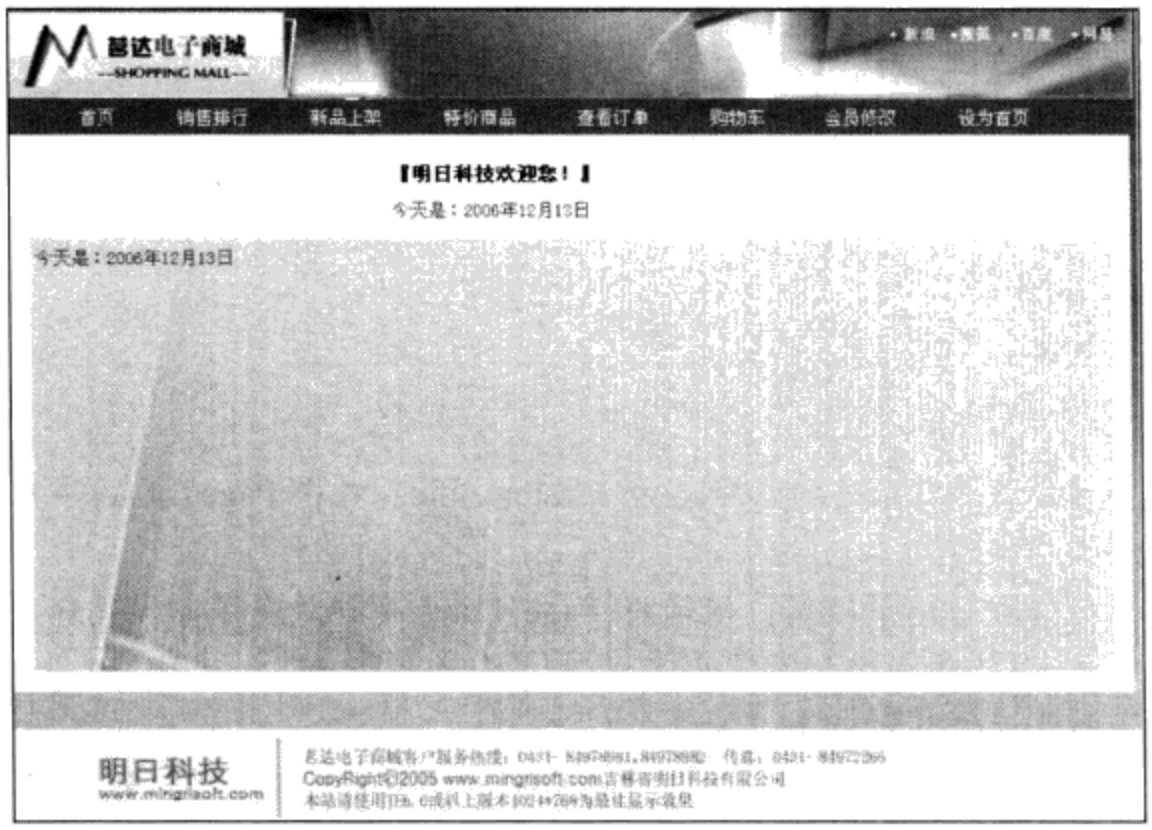


图 17.8　使用 MasterType 指令获取母版页的公共属性

母版页 MainMasterPage.master 源代码如下：

代码 17-03　光盘位置：光盘\mingrisoft\sl\17\17_03

```
<%@ Master Language="C#" AutoEventWireup="True" CodeFile="MainMasterPage.master.cs" Inherits="Main
MasterPage" %>
<script runat =server   >
    string _MVar = "";
    public string MVar
    {
        get
        {
            return _MVar;
        }
        set
        {
            _MVar = value;
        }
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        MasterLable.Text = "今天是：" + System.DateTime.Now.ToLongDateString();
    }
</script>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/ TR/xhtml1/
DTD/xhtml1-   transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>访问母版页的成员</title>
    <style >
   body{ margin-top:0px;}
```

定义了一个 String 类型的公共属性 Mvar

```
        </style>
    </head>
    <body >
        <form id="form1" runat="server">
            <table   style=" width :762px; height :601px;" border =0   background ="Images/框架结构图.jpg"   align
=center >
                <tr>
                    <td   style="height: 43px; width: 641px;" align =center >
                    </td>
                </tr>
                <tr style="width: 641px; height: 17px">
                <td>

                </td>
                </tr>
                <tr>
                    <td align="center" style="width: 641px; height: 17px">
                      <%=MVar%>
                      <font color=red style="font-size: 9pt; font-family: 宋体" >
                        <asp:Label ID="MasterLable" runat="server" Text="Label"></asp:Label></font></td>
                </tr>
                <tr>
                    <td align =center valign =top>
                    <asp:ContentPlaceHolder ID ="MainContent" runat =server >
                     </asp:ContentPlaceHolder>

                    </td>
                </tr>
                <tr>
                <td style="height: 41px; width: 641px;"   align =center   valign =top >
                </td>
                </tr>
            </table>
        </form>
    </body>
    </html>
```

说明：与普通定义属性的方法相同，源代码中定义了一个 String 类型的属性 Mvar。Mvar 属性是一个可读写的简单属性，主要用于显示“『明日科技欢迎您！』”字样，并且通过<%=MVar%>显示在母版面中。

内容页 Default.aspx 源代码如下：

```
    <%@ Page Language="C#" MasterPageFile="~/MainMasterPage.master" AutoEventWireup="True" CodeFile="
Default.aspx.cs" Inherits="_Default" Title="Untitled Page" %>
    <%@ MasterType VirtualPath ="~/MainMasterPage.master" %>
    <script runat =server >
        protected void Page_Load(object sender, EventArgs e)
        {
            Master.MVar = "<h5>『明日科技欢迎您！』</h5>";
        }
        protected void ContentLabel_PreRender(object sender, EventArgs e)
        {
            Label MLable1 = (Label)this.Master.FindControl("MasterLable");
            ContentLabel.Text = MLable1.Text;
        }
    </script>
    <asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
    <table background ="Images/框架结构图 2.jpg" style=" width :730px; height :307px;"   border =0 align =center>
    <tr align =left valign =top   >
    <td >
        <font color=red style="font-size: 9pt; font-family: 宋体" ><asp:Label ID="ContentLabel" runat="server"
Text="Label" OnPreRender="ContentLabel_PreRender"></asp:Label></font>
    </td>
    </tr>
    </table>
    </asp:Content>
```

利用 Master 对象引用母版页中的公共变量 Mvar，并将“『明日科技欢迎您！』”字样赋给母版中的公共变量 Mvar

说明：

① 在内容页代码头的设置中，增加了<%@MasterType%>，并在其中设置了 VirtualPath 属性，用于设置被强类型化的母版页的 URL 地址。

② 在 Page_Load 事件下，利用 Master 对象引用母版页中的公共变量 Mvar，并将“『明日科技欢迎您！』”字样赋给母版中的公共变量 Mvar。

注意：在<%@Page%>指令中，MasterPageFile 属性用于设置内容页绑定的母版页的 URL 地址，而在<%@MasterType%>指令中，VirtualPath 属性用于设置被强类型化的母版页的 URL 地址。这两者是完全不同的，不可缺少任何一个。

第 18 章

ASP.NET 3.5 主题

- 主题的概述
- 创建主题
- 应用主题

通常在开发网站过程中都会为一个网站定义一种“风格”，如文字颜色、大小、行间距、边框样式等。在 ASP.NET 3.5 出现之前，网站开发人员经常使用级联样式表（CSS）来实现。ASP.NET 3.5 引入了功能强大的样式、外观和主题等特性，允许网站开发人员更加方便快捷的来定义网站风格，而且能够动态实现不同外观的切换。本章将详细介绍主题方面的有关知识。

18.1 主题的概述

网站外观主要与页面和控件的样式属性有关。几乎所有控件都支持 Style 对象模型，Style 对象模型用于设置字体、边框、背景色、高度和宽度等样式属性。同时控件还完全支持将样式设置与控件属性分离的样式表（CSS），在实现过程中，为了统一网站的开发风格，开发人员需要设置控件样式属性，这种做法不仅繁琐，而且不容易保持站点外观的一致性和独立性。ASP.NET 3.5 提供的“主题”功能可以为控件定义一次样式属性就能方便的将该样式应用到站点的所有页面中。本节主要向读者介绍主题的组成元素、文件存储和组织方式等方面的内容。

18.1.1 组成元素

主题是页面和控件外观属性设置的集合。利用主题功能，不仅可以设置页面和控件的外观，还可以在所有的 Web 应用程序、单个 Web 应用程序的所有页面或单个 Web 页面中，快速一致地应用所定义的外观。主题是由外观、级联样式表（CSS）、图像和其他资源组成的。其中，外观是主题中的必要元素。

1．外观

外观文件是主题的核心内容，它主要用于定义页面中服务器控件的外观。在主题中可以包含一个或者多个外观文件。外观文件的扩展名为.skin，其中包含对各种服务器控件（例如，Button、Label、TextBox 或 Calendar 控件）的属性设置。例如，下面是 Button 控件的外观设置代码：

```
<asp:button runat="server" BackColor="lightblue" ForeColor="black" />
```

说明：控件外观的代码设置与控件声明代码类似。在控件外观设置中，只能包含可作为主题的属性定义。这部分属性可能是样式属性、集合属性、模板属性或数据绑定表达式等。

2．级联样式表（CSS）

主题中可以包含一个或者多个 CSS 文件（扩展名为.css）。CSS 文件在主题功能出现之前，已经得到了广泛应用。主题中的 CSS 文件和在主题功能出现之前的 CSS 文件的比较如表 18.1 所示。

表 18.1　主题中的 CSS 文件和在主题功能出现之前的 CSS 文件的比较

主题出现之前的 CSS 文件	主题中的 CSS 文件
首先，在 CSS 文件中定义 body、table 等样式属性；其次，在需要应用的页面中引用 CSS 文件链接；最后，设置相关的 class（多针对于 HTML 控件）或者 CssClass（多针对于 Web 服务器控件）属性值	主题中的 CSS 文件与过去的 CSS 文件没有本质区别。不同之处如下。 （1）应用方法不需要在页面中指定 CSS 文件链接，而是设置页面或者整个网站所使用的主题。当主题得到应用时，主题中 CSS 文件将自动应用到页面上（但必须确保定义了<head runat="server">控件） （2）定义内容方面。过去的 CSS 文件中可能会定义应用于服务器控件的样式属性，然后，通过服务器控件的 CssClass 属性应用。主题中的 CSS 文件都包含普通 HTML 控件和页面的样式属性，而对于服务器控件的样式属性则都设置在外观文件中

3．图像和其他资源

主题中还可以包含图形和其他资源，例如，脚本文件或声音文件。通常，主题的资源文件与该主题的外观文件位于同一个文件夹中，但它们也可以存储在 Web 应用程序中的其他文件夹中，例如，主题目录的某个子文件夹中或者是主题目录以外的位置。

18.1.2 文件存储和组织方式

在 Web 应用程序中，主题文件必须存储在根目录的 App_Themes 文件夹下（除全局主题之外），开发人员可以手动或者使用 Visual Studio 2008 在网站的根目录下创建该文件夹。下面给出 App_Themes 文件夹的示意图（如图 18.1 所示），以加深读者的理解。

在 App_Themes 文件夹中包括两个文件夹“主题一”和“主题二”。每个主题文件夹中都可以包含外观文件、CSS 文件和图像文件等。

外观文件是主题的核心部分，每个主题文件夹下都可以包含一个或者多个外观文件，开发人员在开发过程中，需要根据实际情况对外观文件进行有效的管理，下面给出 3 种常见外观文件的组织方式。

图 18.1 App_Themes 文件夹的示意图

1．根据 SkinID 组织

在对控件外观设置时，相同的 SkinID 放在同一个外观文件中，这种方式适用于网站页面较多、设置内容复杂的情况。

2．根据控件类型

组织外观文件时，以控件类型进行分类，这种方式适用于页面中包含控件较少的情况。

3．根据文件组织

组织外观文件时，以网站中的页面进行分类，这种方式适用于网站中页面较少的情况。

18.2 创 建 主 题

本节将向读者介绍如何创建主题，包括创建外观文件、为主题添加 CSS 样式。

18.2.1 创建外观文件

外观文件是主题的核心内容，用于定义页面中服务器控件的外观，即“默认外观”和“已命名外观”。

默认外观是在设置控件外观时，没有 SkinID 属性，当向页面应用主题时，默认外观自动应用于同一类型的所有控件。例如，如果为 GridView 控件创建一个默认外观，则该控件外观适用于使用本主题的页面上的所有 GridView 控件。另外，默认外观严格按控件类型来匹配，因此 Button 控件外观适用于所有 Button 控件，但不适用于 LinkButton 控件或从 Button 对象派生的控件。

已命名外观是设置了 SkinID 属性的控件外观。已命名外观不会自动按类型应用于控件，通过设置控件的 SkinID 属性将已命名外观显式应用于控件。通过创建已命名外观，可以为应用程序中同一控件的不同实例设置不同的外观。

控件外观设置的属性可以是简单属性，也可以是复杂属性。简单属性是控件外观设置中最常见的类型，例如，控件背景颜色（BackColor）、控件的宽度（Width）等。复杂属性主要包括集合属性、模板属性、数据绑定表达式（仅限于<%#Eval%>或<%#Bind%>）等类型。下面创建一个简单的外观示例，加深读者对创建外观文件的理解。

示例 18-01 创建一个简单的外观示例

该示例中包括 2 个 Button 控件和 2 个 TextBox 控件，其中上端的 TextBox 控件和 Button 控件设置了默认外观，另外的 TextBox 控件和 Button 控件设置了已命名外观，示例运行结果

如图 18.2 所示。

主题必须保存在网站根目录下的 App_Themes 文件夹下，并且主题本身就是一个文件夹。所以，创建主题时可以手动添加该文件夹，也可以通过系统自动生成，具体操作步骤如下。

（1）在 Visual Studio 2008 的“解决方案资源管理器”中，右键单击项目名称，选择右键菜单中的“添加 ASP.NET 文件夹”选项。

（2）在该选项的子菜单中选择“主题”命令，系统就会自动在根目录下创建一个名为“App_Themes”的文件夹，并在该文件夹中创建一个主题文件，如图 18.3 所示。

图 18.2　创建外观文件示例图

图 18.3　添加主题

（3）可以根据需要设置该主题文件的名称，例如，本示例中将主题文件命名为“blue”，然后邮件单击该主题，选择右键菜单中的“添加新项”命令，弹出“添加新项”对话框，在对话框中选择“外观文件”，如图 18.4 所示。

（4）设置主题文件名称及开发语言后，单击“添加”按钮，就会为主题添加一个皮肤文件，然后就可以编辑添加的皮肤文件了，如图 18.5 所示。

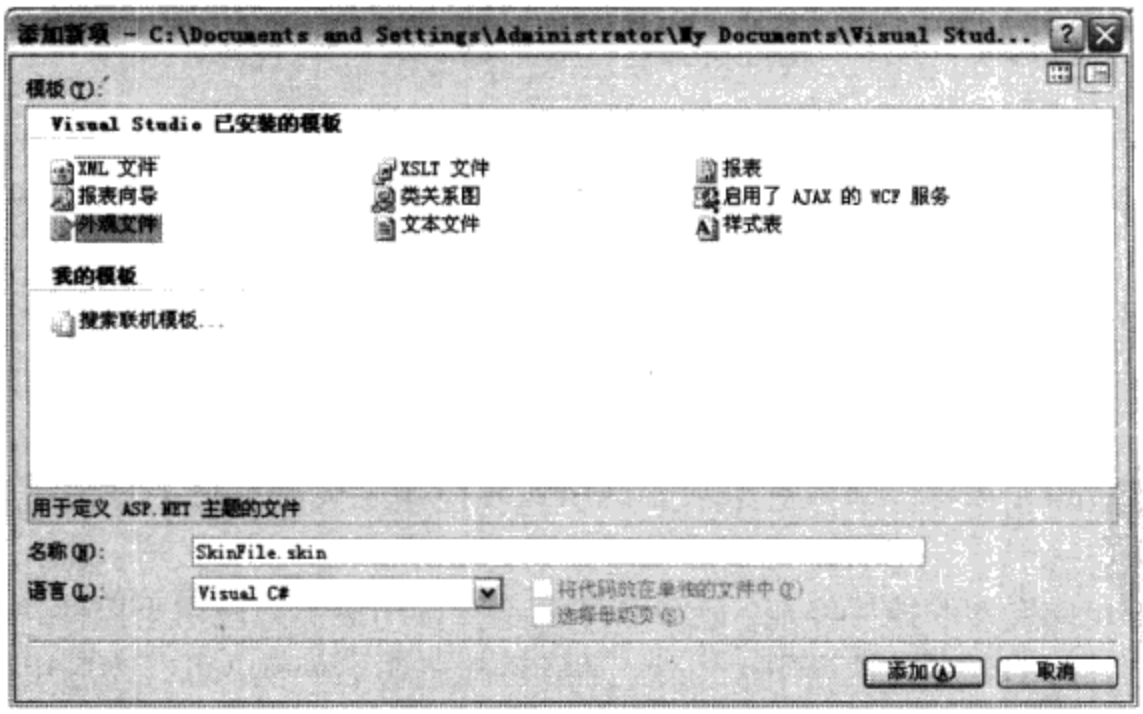


图 18.4　“添加新项”对话框

图 18.5　添加皮肤文件

程序主要代码如下：

代码 18-01　光盘位置：光盘\mingrisoft\sl\18\18_01

在主题文件中设置文本框和按钮的样式，首先设置没有添加 SkinID 属性的文本框和按钮的样式，然后再设置添加 SkinID 属性的文本框和按钮的样式，前者也被称为默认主题样式。默认主题样式对整个页面的所有控件都起作用，皮肤文件的代码如下：

```
//默认的主题样式
<asp:TextBox  runat="server"  BorderStyle="Dotted"
    BorderWidth="1px" Height="17px" Width="223px"></asp:TextBox>
```

```
<asp:Button   runat="server" BorderColor="#CCCCCC"
    BorderStyle="Solid" BorderWidth="1px" Text="Button" />
//设置了 SkinId 的主题样式
<asp:TextBox   runat="server" SkinId="ls" BorderStyle="Dashed" Width="223px"
    BorderWidth="2px"></asp:TextBox>
<asp:Button   runat="server" SkinId="ls" BorderStyle="Dashed"   Text="Button" />
```

技巧：在创建控件外观时，一个简单的方法是将控件添加到.aspx 页面中，然后，利用 Visual Studio 2008 的属性窗口及可视化设计功能对控件进行设置，最后，再将控件代码复制到外观文件中并做适当的修改。

注意：在外观文件中应移除控件的 ID 属性。

创建一个 Default.aspx 页面，在该页面上添加 2 个 Button 控件和 2 个 TextBox 控件，并将其中的一个 TextBox 控件和 Button 控件的 SkinID 属性设为皮肤文件中自定义的 SkinID，该页面的源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true"   CodeFile="Default.aspx.cs" Inherits="_Default" Theme="blue" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Button" />
        <br />
        <asp:TextBox ID="TextBox2" runat="server" SkinID="ls"></asp:TextBox>
        <asp:Button ID="Button2" runat="server" SkinID="ls" Text="Button" />
    </div>
    </form>
</body>
</html>
```

应用主题

说明：

① 应用主题的方法是在<%@ Page%>标签中设置一个 Theme 属性，该属性是主题文件的名称，例如，本示例中主题文件名为 blue。

② 如果为控件设置默认外观，则不用设置控件的 SkinID 属性；如果为控件设置了已命名外观，则需要设置控件的 SkinID 属性。

③ 如果在控件代码中添加了与控件外观相同的属性，则页面最终显示的是控件外观的设置效果。

18.2.2 为主题添加 CSS 样式

主题还可以包含级联样式表（.css 文件）。将.css 文件放在主题目录中时，样式表自动作为主题的一部分应用。.css 样式在主题文件夹中主要用于设置页面和普通 HTML 控件的外观样式，例如，设置页面的背景颜色和普通文本的字体等。下面通过一个简单示例说明为主题添加 CSS 文件的方法。

示例 18-02 为主题添加 CSS 文件

本示例实现的内容包括超链接文字的颜色、鼠标经过超链接文字时是否显示下划线、页面普通文字的字体和颜色等样式，示例运行结果如图 18.6 所示。

图 18.6 为主题添加 CSS 文件

CSS 文件的源代码如下：

代码 18-02　光盘位置：光盘\mingrisoft\sl\18\18_02

```
body
{
    background-color:Transparent;                    //设置背景颜色
    font-family:楷体_GB2312;                         //设置字体
    font-size:14px;                                  //设置字体大小
    color:Black;                                     //设置字体颜色
}
a{font-size:16px;font-family:宋体;}
a:visited
{
    color:Teal;                                      //被访问过的超链接字体的颜色
    text-decoration:none;                            //设置超链接字体没有下划线
}
a:hover
{
    text-decoration:underline;                       //鼠标经过超链接字体时显示下划线

}
a:active
{
    color:Blue;                                      //单击超链接字体时字体的颜色
    text-decoration:none;                            //设置超链接字体没有下划线
}
```

注意：

① 主题中的 CSS 文件与普通的 CSS 文件没有任何区别，但主题中包含的 CSS 文件主要针对页面和普通的 HTML 控件进行设置。

② 主题中的 CSS 文件必须保存在主题文件夹中。

在 Default.aspx 页面中，应用主题中的 CSS 文件样式的源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="_Default"
Theme="fengdi"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>无标题页</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <a href=Default2.aspx>用一生下载你</a><br />
        <br />
        十里平湖霜满天，<br />
        寸寸青丝愁华年。<br />
        对月形单影相互，<br />
        只羡鸳鸯不羡仙。</div>
    </form>
</body>
</html>
```

在主题中应用 CSS 文件必须保证在页面头部定义<head runat="server">

注意：

① 要应用主题中的 CSS 样式，需要在<%@ Page%>标签中设置 Theme 属性值为主题名。

② 必须保证在页面头部定义<head runat="server">，否则，主题中的 CSS 文件将无法正常应用。

18.3 应用主题

目前大多数网站都有切换网站风格的功能，这个功能主要是通过主题技术来实现的，也就是建立多个主题，然后根据选择将主题应用到网站中，本节通过实例演示如何为页面创建多种风格，根据不同需求切换网站风格。

实例位置：光盘\mingrisoft\dxyy\18\18_01

本实例通过下拉框选择不同的主题，单击【确定】按钮更换网页主题。根据选择的主题不同，网页的背景颜色及控件样式也不同，实例运行结果如图 18.7 和图 18.8 所示。

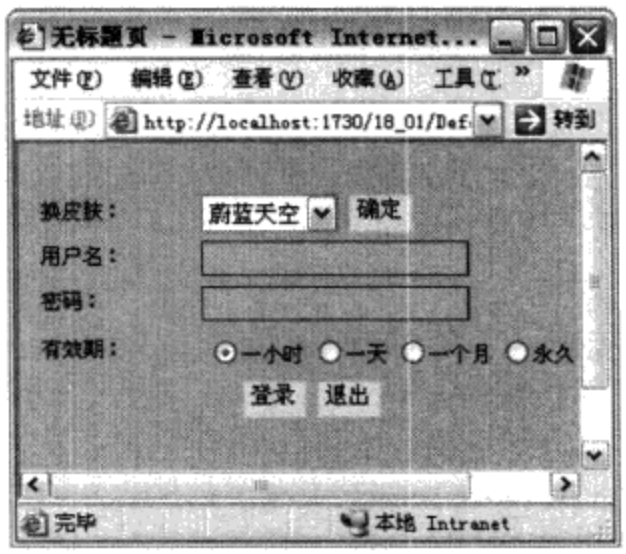


图 18.7 主题一

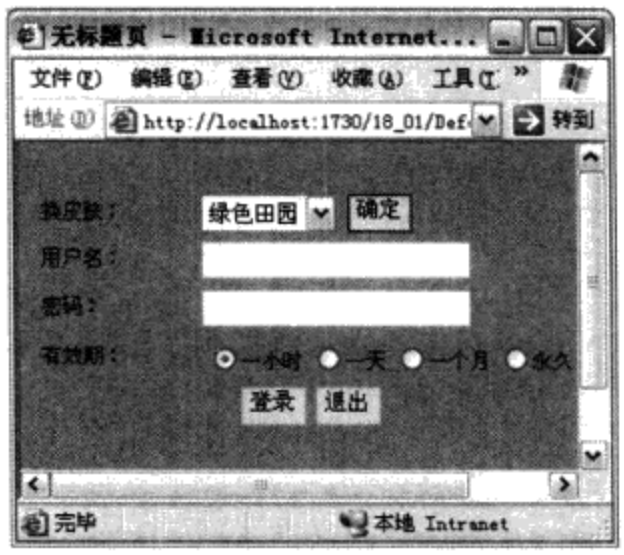


图 18.8 主题二

程序开发步骤如下。

（1）新建一个网站，命名为 18_01，其主页默认为 Default.aspx。

（2）在项目中添加“App_Themes”文件夹，然后在该文件夹中添加两个主题，分别在两个主题中添加一个皮肤文件和一个 CSS 样式文件，用于实现两个不同的网页主题。

（3）程序主要代码如下。

在本实例中两个主题的皮肤文件都是用来设置页面中文本框和按钮样式的，主题一皮肤文件的代码如下：

```
<asp:TextBox runat="server" BackColor="#7BD0F9" BorderColor="#3333FF" Width="150px" BorderWidth="1px"
></asp:TextBox>
<asp:Button runat="server" BorderColor="#CCCCCC" BorderStyle="Solid" BorderWidth="1px" />
```

主题二皮肤文件的代码如下：

```
<asp:TextBox runat="server" Width="150px" BorderStyle="dotted"></asp:TextBox>
<asp:Button runat="server" BorderColor="#CCCCCC" BorderStyle="Solid" BorderWidth="1px" />
```

在两个主题的 CSS 样式文件中使用了相同的代码，用于控制页面中文字的大小、字体、颜色和网页背景颜色，代码如下：

```
body
{
    font-size:12px;                          //设置字体大小
    font-family:宋体;                        //设置字体
    color:Black;                             //设置字体颜色
    background-color:#5ed0f5;                //设置背景颜色
}
```

在 Default.aspx 的后台代码中，首先要声明 Page_PreInit 事件，该事件在开始时进行主题的加载。在该事件中，首先判断传递过来的 themes 值是否为空，该值代表主题的名称，如果不为空则将其赋给 Page 对象的 Theme 属性更改网页主题，代码如下：

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (Request.QueryString["themes"] != null)                    //如果传递过来的 themes 值不为空
    {
        Page.Theme = Request.QueryString["themes"].ToString();    //设置网页的主题
    }
}
```

在 Default.aspx 页面中选择主题，单击【确定】按钮后，会根据选择的主题将相应的主题名传递给 Default.aspx 页面，然后在 Page_PreInit 事件中接收主题名并更改网页主题，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (DropDownList1.SelectedIndex == 0)                         //获取选择的主题的索引
    {
        Response.Redirect("Default.aspx?themes=Themes1");         //选择第一项则传递参数 themes 为 Themes1
    }
    else                                                          //否则
    {
        Response.Redirect("Default.aspx?themes=Themes2");         //选择第二项则传递参数 themes 为 Themes2
    }
}+
```

第 19 章

站点导航和导航控件

- 站点地图概述
- SiteMapPath 控件
- TreeView 控件
- Menu 控件
- 导航控件的应用

站点导航对于每个网站都是必不可少的，它相当于城市公路中的路标，可以指引浏览者找到自己想要的内容。在 ASP.NET 3.5 中提供了 3 个标准的站点导航服务器控件，即 SiteMapPath、TreeView 和 Menu。本章将对这 3 个标准的站点导航服务器控件进行详细介绍。

19.1 站点地图概述

站点地图是一种以.sitemap 为扩展名的标准的 XML 文件，主要为站点导航控件提供站点层次结构信息，例如，站点分为多少个模块，每个模块中包含多少个子模块，以及每个模块的入口页面 URL 地址等。

默认情况下，站点地图文件被命名为 Web.sitemap，该文件必须位于应用程序的根目录中。如果站点地图文件不是以.sitemap 为扩展名，则需要将该文件放置在不能被用户下载（.sitemap 扩展名默认不能被浏览者下载）的目录下，App_Data 如下。

站点地图创建的过程如下。

（1）在“解决方案资源管理器”的 Web 站点中，单击鼠标右键，在弹出的快捷菜单中选择“添加新项”选项。

（2）弹出“添加新项”对话框，如图 19.1 所示。在“添加新项”对话框中选择“站点地图”，默认名为 Web.sitemap。单击“添加”按钮就可以创建一个新的 Web.sitemap 文件。

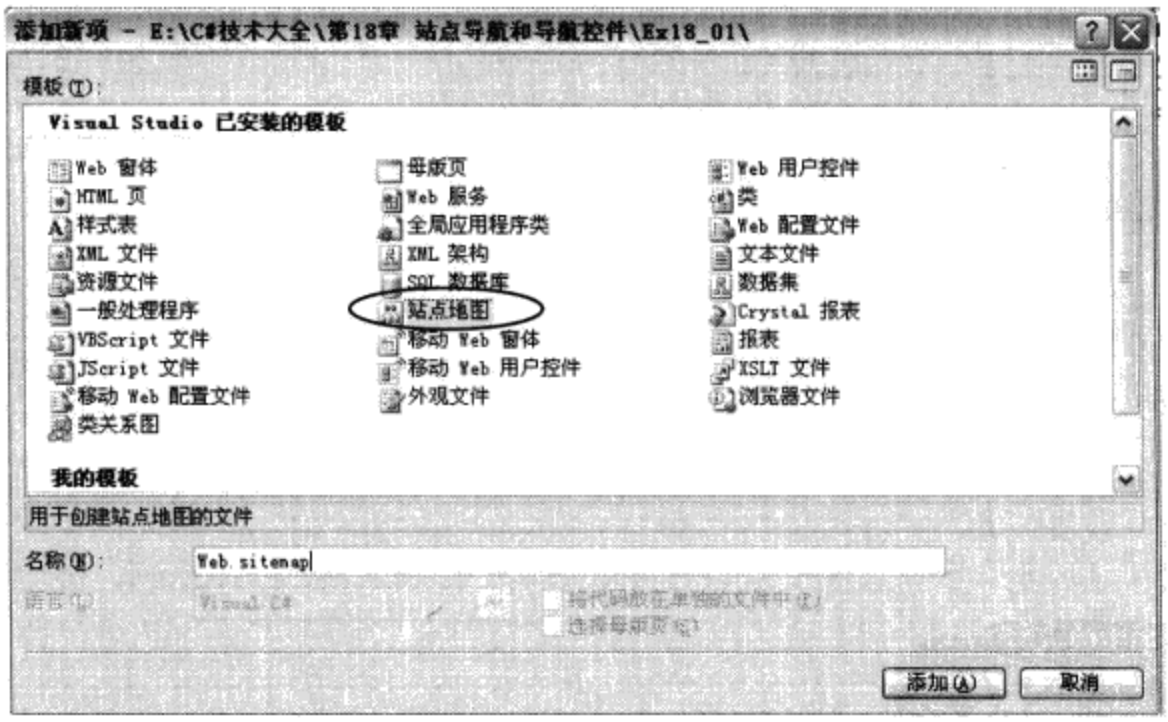


图 19.1 “添加新项”对话框

新创建的 Web.siteMap 文件的源代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title=""   description="">
        <siteMapNode url="" title=""   description="" />
        <siteMapNode url="" title=""   description="" />
    </siteMapNode>
</siteMap>
```

从上述代码可知，Web.sitemap 文件内容严格遵循 XML 文件格式。该文件中包括一个根节点 siteMap，在根节点下包括多个 siteMapNode 子节点，其中设置了 url、title 和 description 属性。siteMapNode 节点常用属性及说明如表 19.1 所示。

表 19.1　　siteMapNode 节点常用属性及说明

属性名称	说　明
url	设置用于节点导航的 URL 地址。在整个站点地图文件中，该属性必须惟一
title	设置节点名称
description	设置节点说明文字
key	定义表示当前节点的关键字
roles	定义允许查看该站点地图文件的角色集合。多个角色可使用（;）和（,）进行分隔
Provider	定义处理其他站点地图文件的站点导航提供程序名称，默认值为 XmlSiteMapProvider
siteMapFile	设置包含其他相关 SiteMapNode 元素的站点地图文件

19.2　SiteMapPath 控件

19.2.1　功能

SiteMapPath 服务器控件能够自动从站点地图中获取导航的数据，准确定位当前页面在整个 Web 站点的位置，同时，使用层次化表示方法，将位置信息显示为有序的静态文本或者超级链接。另外，通过调整相关属性，可以自定义位置信息的外观以及其他内容，从而适应站点的总体设计风格。

下面通过一个简单示例，加深读者对 SiteMapPath 服务器控件功能的理解。

示例 19-01　SiteMapPath 服务器控件的使用

本示例通过 SiteMapPath 服务器控件实现页面的导航功能，通过该控件可以很直观的了解当前所处的位置，示例运行结果如图 19.2 所示。

图 19.2　将 SiteMapPath 控件拖放到 Web 窗体上

该控件将根据默认站点地图文件（Web.sitemap 文件）中的数据自动显示导航信息。根据前面内容可知，在使用 SiteMapPath 服务器控件前必须创建 Web.sitemap 文件。在本示例中，站点地图文件 Web.sitemap 的源代码如下：

代码 19-01　实例位置：光盘\mingrisoft\sl\19\19_01

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode title="您的位置">
    <siteMapNode url="Default.aspx" title="首页", description="风笛新闻网">
      <siteMapNode url="news.aspx" title="新闻快递" description="新闻快递"/>
      <siteMapNode url="gnnews.aspx" title="国内新闻" description="国内新闻"/>
      <siteMapNode url="gjnews.aspx" title="国际新闻" description="国际新闻"/>
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

注意：SiteMapPath 控件可以直接使用网站的站点地图数据，并将站点地图数据在客户端中显示出来。如果将 SiteMapPath 控件用在尚未在站点地图中表示的页面上，则该控件将不会向客户端显示任何信息。

19.2.2　属性

虽然 SiteMapPath 服务器控件可以简单地实现页面导航功能，但是，默认情况下，它生成

的最终结果还是比较简单的。该控件的一些属性可以帮助程序开发人员定义显示的样式，从而适应站点的总体设计风格。

SiteMapPath 控件的常用属性及说明如表 19.2 所示。

表 19.2　　SiteMapPath 控件的常用属性及说明

属　性	说　明
CurrentNodeStyle	用于获取 SiteMapPath 控件当前节点所使用的样式
CurrentNodeTemplate	获取或者设置 SiteMapPath 控件中当前节点的自定义模板
NodeStyle	用于获取 SiteMapPath 控件中所有节点所使用的样式
NodeTemplate	获取或设置 SiteMapPath 控件中所有节点的自定义模板
ParentLevelsDisplayed	获取或者设置相对于当前节点，所显示父节点的级数
PathDirection	获取或者设置节点显示的方向
PathSeparator	获取或者设置节点之间显示的分隔符
PathSeparatorStyle	用于获取 PathSeparator 属性中定义的任意文字外观
PathSeparatorTemplate	获取或者设置 SiteMapPath 控件中，节点间分隔符的自定义模板
RenderCurrentNodeAsLink	获取或者设置是否将表示当前页的节点，显示为指向自身的超链接
RootNodeStyle	用于获取 SiteMapPath 控件中根节点所使用的样式
RootNodeTemplate	获取或者设置 SiteMapPath 控件中根节点的自定义模板
SiteMapProvider	获取或者设置用于处理当前相关 SiteMap 文件的站点导航提供程序名称

下面对比较重要的属性进行详细介绍。

（1）ParentLevelsDisplayed 属性。ParentLevelsDisplayed 属性用于获取或设置 SiteMapPath 控件显示相对于当前显示节点的父节点级别数。例如，将 ParentLevelsDisplayed 属性设置为 3，则表示在当前节点之前，还要显示 3 级父节点，语法格式如下：

```
public virtual int ParentLevelsDisplayed { get; set; }
```

属性值：指定相对于当前节点显示的父节点的级别数。默认值为–1，表示将所有节点完全展开。

例如，设置 SiteMapPath 控件在当前节点之前还要显示 3 级父节点，代码如下：

```
SiteMapPath1. ParentLevelsDisplayed=3;
```

（2）PathDirection 属性。PathDirection 用来获取或设置节点显示的方向。有两种显示方向可供选择：CurrentToRoot 和 RootToCurrent。默认值为 RootToCurrent。例如，当设置 PathDirection 属性值为 RootToCurrent 时，显示方式为从最顶部的节点到当前节点（例如，主页>明日科技图书>C#技术大全），当设置 PathDirection 值为 CurrentToRoot 时，显示方式为从当前节点到最顶部节点（例如，C#技术大全>明日科技图书>主页）。语法格式如下：

```
public virtual PathDirection PathDirection { get; set; }
```

属性值：PathDirection 表示导航节点显示的分层顺序。默认值为 RootToCurrent，表示节点显示的方式为从最顶部的节点到当前节点。

例如，节点显示的方式为从最顶部的节点到当前节点，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    SiteMapPath1.PathDirection = PathDirection.RootToCurrent;
}
```


19.2.3　事件

SiteMapPath 控件常用事件及说明如表 19.3 所示。

表 19.3 **SiteMapPath 控件常用事件及说明**

名　称	说　明
DataBinding	当前服务器控件绑定到数据源时发生
Disposed	当从内存释放服务器控件时发生
Init	当服务器控件初始化时发生
ItemCreated	当 SiteMapPath 控件创建一个 SiteMapNodeItem 对象，将其与 SiteMapNode 关联时发生
ItemDataBound	当 SiteMapNodeItem 绑定到 SiteMapNode 包含的站点地图数据时发生
Load	当服务器控件加载到 Page 对象中时发生
PreRender	在加载页面之前发生
Unload	当服务器控件从内存中卸载时发生

在 SiteMapPath 控件中，值得关注的事件是 ItemCreated 和 ItemDataBound。前者涉及创建节点过程，后者涉及数据绑定过程。

19.3 TreeView 控件

19.3.1 功能

TreeView 服务器控件用于以树形结构显示分层数据，如目录或文件目录，它支持如下功能。

- 支持数据绑定。即允许通过数据绑定方式，使得控件节点与 XML、表格、关系型数据等结构化数据建立紧密联系。
- 支持站点导航功能。即通过集成 SiteMapDataSource 控件，实现站点导航功能。
- 节点文字可显示为普通文本或超级链接文本。
- 可自定义树形和节点的样式、主题等外观特征。
- 支持动态功能。通过编程访问 TreeView 对象模型，完成动态创建树形结构、构造节点和设置属性等任务。
- 在客户端浏览器支持的情况下，支持由客户端构建节点，减少到服务端的回送。
- TreeView 服务器控件呈现的树状结构可以兼容不同类型的浏览器。
- 具有在节点显示 CheckBox 控件的功能。

19.3.2 属性

TreeView 服务器控件常用属性及说明如表 19.4 所示。

表 19.4 **TreeView 服务器控件常用属性及说明**

名　称	说　明
AutoGenerateDataBindings	获取或设置 TreeView 服务器控件是否自动生成树节点绑定
CheckedNodes	用于获取 TreeView 控件中被用户选中 CheckBox 的节点集合
CollapseImageToolTip	获取或设置可折叠节点的指示符所显示图像的提示文字
CollapseImageUrl	获取或设置节点在折叠状态下，所显示图像的 URL 地址
DataSource	获取或设置绑定到 TreeView 服务器控件的数据源对象

续表

名　称	说　明
DataSourceID	获取或设置绑定到 TreeView 服务器控件的数据源控件的 ID
EnableClientScript	获取或设置 TreeView 服务器控件是否呈现客户端脚本以处理展开和折叠事件
ExpandDepth	获取或设置默认情况下 TreeView 服务器控件展开层次数
ExpandImageToolTip	获取或设置可展开节点的指示符所显示图像的提示文字
ExpandImageUrl	获取或设置用作可展开节点的指示符的自定义图像的 URL
ImageSet	获取或设置 TreeView 服务器控件的图像组，是 TreeViewImageSet 枚举值之一
LineImagesFolder	获取或设置用于链接子节点和父节点的线条图像的文件夹的路径
MaxDataBindDepth	获取或设置要绑定到 TreeView 服务器控件的最大树级别数
NodeIndent	获取或设置 TreeView 服务器控件的子节点的缩进量，单位是像素
Nodes	用于获取 TreeView 控件中的 TreeNode 对象集合。可通过特定方法，对树型结构中的节点进行添加、删除、修改等操作
NodeWrap	获取或设置空间不足时节点中的文本是否换行
NoExpandImageUrl	获取或设置不可展开节点的指示符的自定义图像的 URL
PathSeparator	获取或设置用于分隔由 ValuePath 属性指定的节点值的字符，为防止冲突和得到错误的数据，节点的 Value 属性中不应当包含分隔符字符
PopulateNodesFromClient	获取或设置是否启用由客户端构建节点的功能
SelectedNode	获取 TreeView 服务器控件中选定节点的 TreeNode 对象
SelectedValue	获取 TreeView 服务器控件中选定节点的值
ShowCheckBoxes	获取或设置哪些节点类型将在 TreeView 控件中显示复选框
ShowExpandCollapse	获取或设置是否显示展开节点指示符
ShowLines	获取或设置是否显示链接子节点和父节点的线条
Target	获取或设置单击节点时网页内容的目标窗口或框架名字

下面对比较重要的属性进行详细介绍。

（1）ExpandDepth 属性。获取或设置默认情况下 TreeView 服务器控件展开层次数。例如，若将该属性设置为 2，则将展开根节点及根节点下方紧邻的所有父节点。默认值为-1，表示将所有节点完全展开。语法格式如下：

语法：

```
public int ExpandDepth { get; set; }
```

属性值：最初显示 TreeView 控件时要显示的深度。

例如，设置 TreeView 控件展开 3 层，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    TreeView1.ExpandDepth = 3;
}
```

（2）Nodes 属性。使用 Nodes 属性可以获取一个包含树中所有根节点的 TreeNodeCollection 对象。Nodes 属性通常用于快速循环访问所有根节点，或者访问树中的某个特定根节点，同时还可以使用 Nodes 属性以编程方式管理树中的根节点，即可以在集合中添加、插入、移除和检索 TreeNode 对象。语法格式如下：

语法：

```
public TreeNodeCollection Nodes { get; }
```

属性值：TreeView 控件中的根节点。

例如，在使用 Nodes 属性遍历树时，判断根节点数，代码如下：

```
if (TreeView1.Nodes.Count > 0)
{
        for (int i = 0; i < TreeView1.Nodes.Count; i++)
{
                ……其他操作
        }
}
```

（3）SelectedNode 属性。SelectedNode 属性用于获取用户选中节点的 TreeNode 对象。当节点显示为超链接文本时，该属性返回值为 null，不可用。语法格式如下：

```
public TreeNode SelectedNode { get; }
```

属性值：TreeView 控件中的选定节点。

例如，从 TreeView 控件中将选择的节点值赋给 Label 控件，代码如下：

```
Label1.Text += "<li>被选择的节点为："+TreeView1.SelectedNode.Text;
```

（4）ShowCheckBoxes 属性。ShowCheckBoxes 属性是用来获取或设置一个值，该值指示哪些节点类型将在 TreeView 控件中显示复选框。ShowCheckBoxes 属性是 System.Web.UI.Web Controls.TreeNodeTypes 的枚举值组合，System.Web.UI.WebControls.Tree NodeTypes 的枚举值类型及说明如表 19.5 所示。

表 19.5　　System.Web.UI.WebControls.TreeNodeTypes 的枚举值类型及说明

节点类型	说　明
TreeNodeType.All	为所有节点显示复选框
TreeNodeType.Leaf	为所有叶节点显示复选框
TreeNodeType.None	不显示复选框
TreeNodeType.Parent	为所有父节点显示复选框
TreeNodeType.Root	为所有根节点显示复选框

19.3.3 事件

TreeView 服务器控件常用事件及说明如表 19.6 所示。

表 19.6　　TreeView 服务器控件常用事件及说明

名　称	说　明
SelectedNodeChanged	在 TreeView 控件中选定某个节点时发生
TreeNodeCheckChanged	当 TreeView 服务器控件的复选框在向服务器的两次发送过程之间状态有所更改时发生
TreeNodeExpanded	当展开 TreeView 服务器控件中的节点时发生
TreeNodeCollapsed	当折叠 TreeView 服务器控件中的节点时发生
TreeNodePopulate	当其 PopulateOnDemand 属性设置为 True 的节点在 TreeView 服务器控件中展开时发生
TreeNodeDataBound	当数据项绑定到 TreeView 服务器控件中的节点时发生

下面主要介绍 SelectedNodeChanged 事件。SelectedNodeChanged 事件在 TreeView 控件中选定某个节点时发生。TreeView 服务器控件的节点文字有两种模式：选择模式和导航模式。默认情况下，节点文字处于选择模式，如果节点的 NavigateUrl 属性设置不为空，则该节点处于导航模式。TreeView 服务器控件处于选择模式时，当用户单击 TreeView 服务器控件的不同节点的文字，将触发 Selected NodeChanged 事件，在该事件下可以获得所选择节点对象。

示例 19-02　TreeView 控件的 SelectedNodeChanged 事件的应用

本示例实现的是，当程序运行时，在页面 Default.aspx 中单击 TreeView 控件上的任意节点

时，都会在界面上将已选择的节点值显示出来。示例运行结果如图 19.3 所示。

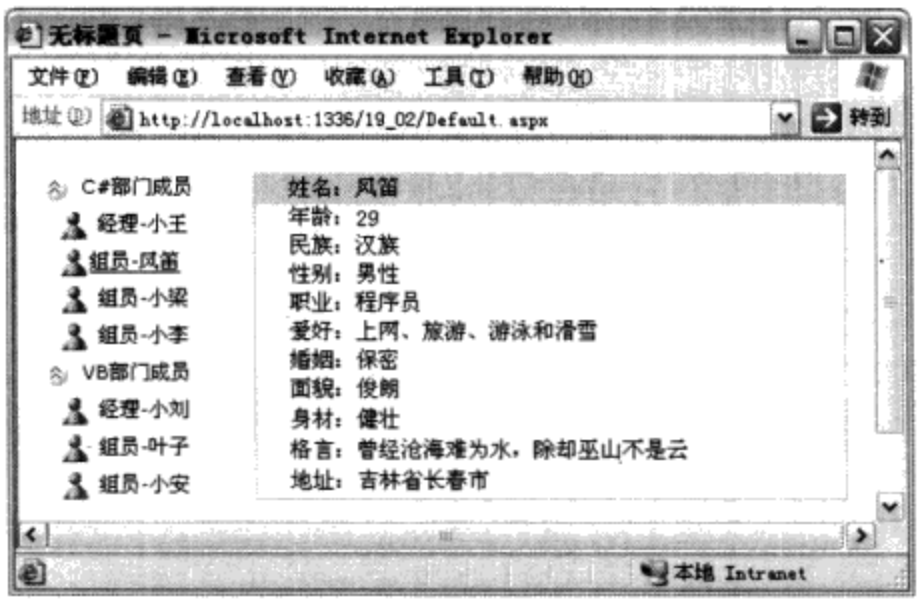


图 19.3　TreeView 控件的 SelectedNodeChanged 事件的应用

程序代码如下。

代码 19-02　光盘位置：光盘\mingrisoft\sl\19\19_02

单击 TreeView 控件的任一节点，都会触发 TreeView 控件的 SelectedNodeChanged 事件，在该事件下，获取选择的节点值，然后将选择组员的相关信息显示出来，代码如下：

```
protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
    if (TreeView1.SelectedValue == "组员-风笛")                    //如果选择“组员-风笛”节点
    {
        Label1.Text = "风笛";                                      //显示名称
        Label2.Text = "29";                                        //显示年龄
        Label3.Text = "汉族";                                      //显示民族
        Label4.Text = "男性";                                      //显示性别
        Label5.Text = "程序员";                                    //显示职位
        Label6.Text = "上网、旅游、游泳和滑雪";                    //显示爱好
        Label7.Text = "保密";                                      //显示婚姻状况
        Label8.Text = "俊朗";                                      //显示面貌
        Label9.Text = "健壮";                                      //显示身材
        Label10.Text = "曾经沧海难为水，除却巫山不是云";           //显示格言
        Label11.Text = "吉林省长春市";                             //显示地址
    }
}
```

19.4　Menu 控件

19.4.1　功能

Menu 服务器控件用来在页面上生成下拉菜单，其显示类型如下。

（1）根据菜单项显示方式，Menu 控件可分为两种类型，即静态菜单和动态菜单。

静态菜单是指菜单选项始终静态显示。默认情况下，位于 Menu 控件的根部的根菜单项就是一种典型的静态菜单，通过属性设置可以控制静态菜单的显示级数。动态菜单是指当用户鼠标位于父菜单项上方时，在其水平或者垂直方向将会自动显示子菜单项。子菜单项在显示短暂时间后，将自动消失。通过设置属性值可以调整子菜单的延迟显示时间。

（2）根据菜单项的排列方向，Menu 控件分为水平菜单和垂直菜单。

Menu 控件的功能如下。

- 支持数据绑定。即允许通过数据绑定方式，使得菜单选项与 XML 和表格等结构化数据建立紧密联系。
- 支持站点导航功能。即通过集成 SiteMapDataSource 控件，实现站点导航功能。
- 可使用主题、样式属性等，实现自定义外观特征。
- 支持动态功能。通过编程访问 Menu 对象模型，完成动态创建菜单、构造菜单选项和设置属性等任务。

19.4.2 属性

Menu 控件常用属性及说明如表 19.7 所示。

表 19.7　　Menu 控件常用属性及说明

名　称	说　明
DataSource	获取或设置绑定到 Menu 控件的数据源
DataSourceID	获取或设置绑定到 Menu 服务器控件的数据源 ID
DisappearAfter	获取或设置当鼠标离开 Meun 控件后菜单的延迟显示时间
DynamicHorizontalOffset	获取或设置动态菜单相对于其父菜单项的水平移动像素数，默认值为 0。如果将此属性设置为负值即可反方向移动动态菜单。如果显示的是垂直菜单，负值则会导致动态菜单与其父菜单项重叠
DynamicHoverStyle	设置鼠标指针置于动态菜单项上时的菜单项外观
DynamicMenuItemStyle	设置动态菜单中的菜单项的外观
DynamicMenuStyle	设置动态菜单的外观
DynamicSelectedStyle	设置用户所选动态菜单项的外观
DynamicVerticalOffset	获取或设置动态菜单相对于其父菜单项的垂直移动像素数，默认值为 0
Items	获取或者设置 Menu 控件中所有菜单选项的集合。通过该属性，可循环访问所有菜单选项或者访问特定选项。另外，还可通过特定方法，对菜单选项进行添加、删除、修改等操作，所有操作将会自动更新菜单
ItemWrap	获取或设置如果文本空间不足，是否自动拆分在下一行继续显示。默认值为 False
MaximumDynamicDisplayLevels	获取或设置动态菜单中显示的最大级数，默认值为 3。当该属性值设置为 0 时，将不会显示任何动态菜单
Orientation	获取或设置 Menu 控件的显示方向。有两个选项：水平（Horizontal）和垂直（Vertical）
SelectedItem	获取选定的菜单项。当用户选中某菜单选项时，将激发 MenuItemClick 事件，可自行定义事件处理程序
SelectedValue	获取选定菜单项的值，默认值为空，表示当前无选项被选中。当需要获取菜单选项 Text 属性值时，可使用该属性；如果需要访问其他属性，则必须访问 SelectedItem 属性
StaticDisplayLevels	获取或设置静态菜单的显示级数，默认值是 1。在该级以下的菜单选项，均显示为动态菜单
StaticHoverStyle	设置鼠标指针置于静态菜单项上时的菜单项外观
StaticMenuItemStyle	设置静态菜单中的菜单项的外观
StaticMenuStyle	设置静态菜单的外观
StaticSelectedStyle	设置用户在静态菜单中选择的菜单项的外观
StaticSubMenuIndent	获取或设置静态菜单中子菜单的缩进间距（以像素为单位）

下面对比较重要的属性进行详细介绍。

（1）DisappearAfter 属性。该属性用来获取或设置当鼠标离开 Meun 控件后，菜单的延迟显

示时间，默认值为 500，单位是 ms。在默认情况下，当鼠标离开 Menu 控件后，菜单将在一定时间内自动消失。如果希望菜单立刻消失，可单击 Meun 控件以外的空白区域。当设置该属性值为-1 时，菜单将不会自动消失，在这种情况下，用户只有在菜单外部单击时，动态菜单项才会消失。语法格式如下：

语法：

```
public int DisappearAfter { get; set; }
```

属性值：鼠标指针不再置于菜单上后，显示动态菜单的时间量（以 ms 为单位）。

示例 19-03　DisappearAfter 属性的应用

本示例通过设置 Meun 控件的 DisappearAfter 属性，实现将鼠标指针从动态菜单移开 2s 后消失的功能。示例运行结果如图 19.4 所示，当鼠标指针移向图 19.4 所示的动态菜单后，其运行结果转为如图 19.5 所示；当鼠标指针从图 19.5 中移开 2s 后，其运行结果又转为图 19.4 所示的状态。

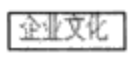


图 19.4　动态菜单

图 19.5　鼠标指针指向动态菜单

程序代码如下。

代码 19-03　光盘位置：光盘\mingrisoft\sl\19\19_03

设置 Meun 控件的 DisappearAfter 属性，实现用户将鼠标指针从动态菜单移开 2s 后消失，实现该功能的前台主要代码如下：

```
<asp:Menu ID="Menu1" runat="server" BackColor="#F7F6F3" DisappearAfter="2000"
    DynamicHorizontalOffset="2" Font-Names="Verdana" Font-Size="Small"
    ForeColor="#7C6F57" Orientation="Horizontal"
    StaticEnableDefaultPopOutImage="False" StaticSubMenuIndent="10px">
    <StaticSelectedStyle BackColor="#5D7B9D" />
    <StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
    <DynamicHoverStyle BackColor="#7C6F57" ForeColor="White" />
    <DynamicMenuStyle BackColor="#F7F6F3" />
    <DynamicSelectedStyle BackColor="#5D7B9D" />
    <DynamicMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
    <StaticHoverStyle BackColor="#7C6F57" ForeColor="White" />
    <Items>
        <asp:MenuItem Text="企业文化" Value="新建项">
            <asp:MenuItem Text="企业结构" Value="企业结构"></asp:MenuItem>
            <asp:MenuItem Text="总裁致词" Value="总裁致词"></asp:MenuItem>
            <asp:MenuItem Text="企业体系" Value="企业体系"></asp:MenuItem>
            <asp:MenuItem Text="企业宗旨" Value="企业宗旨"></asp:MenuItem>
        </asp:MenuItem>
    </Items>
</asp:Menu>
```

鼠标移开 2s 后消失

（2）MaximumDynamicDisplayLevels 属性。获取或设置动态菜单中显示的最大级数。默认值为 3。当该属性值设置为 0 时，将不会显示任何动态菜单。语法格式如下：

```
public int MaximumDynamicDisplayLevels { get; set; }
```

属性值：动态菜单显示的级别数。

例如，如果 Menu 控件的 StaticDisplayLevels 属性取默认值 1 不变，将该控件的 MaximumDynamicDisplayLevels 属性值设置为 1，则动态菜单将显示两个菜单级别，其他更高

菜单的级别将会被丢弃。运行结果如图 19.6 所示。

（3）Orientation 属性。使用 Orientation 属性指定 Menu 控件的显示方向，如果 Orientation 的属性值为 Horizontal，则水平显示 Menu 控件；如果 Orientation 的属性值为 Vertical，则垂直显示 Menu 控件。语法格式如下：

语法：

```
public Orientation Orientation { get; set; }
```

属性值：Orientation 枚举值之一。

例如，将 Menu 控件的 Orientation 属性值设置为 Horizontal，则运行结果如图 19.7 所示。

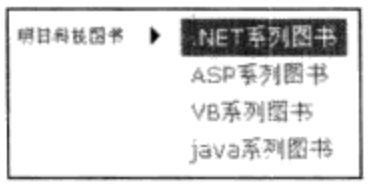


图 19.6　获取或设置动态菜单中显示的最大级数

图 19.7　水平显示动态菜单

19.4.3　事件

Menu 控件常用事件及说明如表 19.8 所示。

表 19.8　　Menu 控件常用事件及说明

名　称	说　明
MenuItemClick	在单击 Menu 控件中某个菜单项时激发
MenuItemDataBound	Menu 控件中的某个菜单选项绑定数据时激发

下面主要介绍 Menu 控件的 MenuItemClick 事件，当单击 Menu 控件中某个菜单项时激发此事件。

例如，单击 Menu 控件中某个菜单项，并显示该菜单的文本，代码如下：

```
protected void Menu1_MenuItemClick(object sender, MenuEventArgs e)
{
    if (Menu1.SelectedItem.Text == "企业体系")          //判断是否选择“企业体系”
    {
        Response.Write("企业体系");                     //如果选择则显示选择的内容
    }
}
```

19.5　导航控件的应用

19.5.1　TreeView 菜单导航的应用

对于一个导航文字很多，并且可以对导航内容进行分类的网站来说，可以将页面中的导航文字以树状图的形式显示，这样不仅可以有效地节约页面，而且可以方便用户查看。在本节中，将通过几个示例向读者介绍 TreeView 菜单导航的应用。

（1）动态绑定 TreeView 控件。

示例 19-04　使用 TreeView 控件显示员工架构

本示例动态将数据库中的数据绑定到 TreeView 控件中，当单击某个节点后，会根据该节点检索数据库，将与节点相关的数据检索出来绑定到 GridView 控件中，示例运行结果如图 19.8 所示。

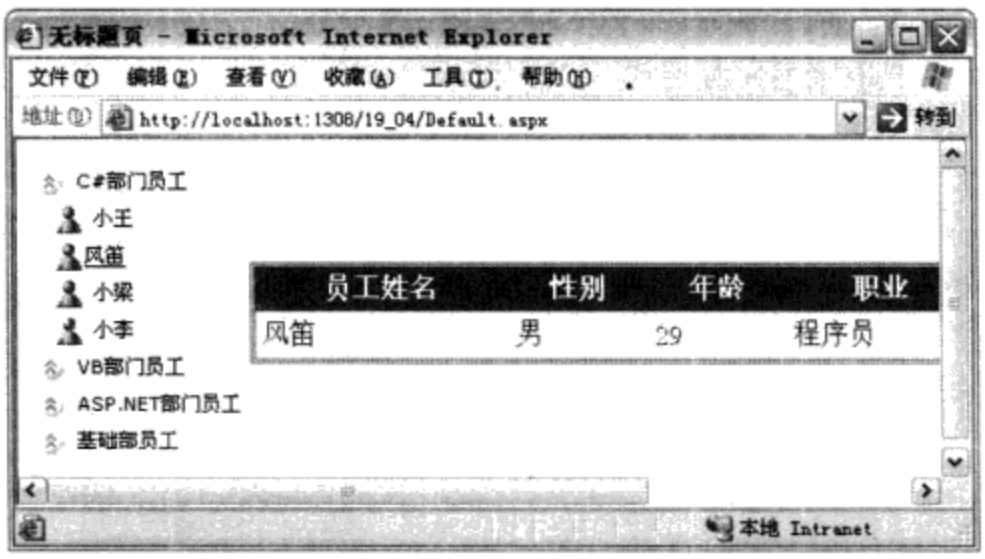


图 19.8 使用 TreeView 控件显示员工架构

程序主要代码如下：

代码 19-04 光盘位置：光盘\mingrisoft\sl\19\19_04

自定义的 BindData 方法用于将数据库中部门信息表和员工信息表中的数据动态绑定到 TreeView 控件中，代码如下：

```
private void BindData()
{
    SqlConnection conn = getCon();                                  //连接数据库
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_ParentTree", conn);
    DataSet ds = new DataSet();                                     //声明一个 DataSet 对象
    sda.Fill(ds, "tb_ParentTree");                                  //填充对象
    TreeView1.ExpandAll();                                          //展开所有节点
    if (ds.Tables.Count > 0)                                        //如果存在数据
    {
        for (int i = 0; i < ds.Tables[0].Rows.Count; i++)           //遍历表中所有的行
        {
            string depName = ds.Tables[0].Rows[i][1].ToString();    //获取部门名称
            TreeNode TN = new TreeNode(depName);                    //为每个部门建立节点
            TreeView1.Nodes.Add(TN);                                //将部门节点添加到 TreeView1 控件中
            SqlDataAdapter sda1 = new SqlDataAdapter("select * from tb_ChildTree where ID=" + ds.Tables[0].Rows[i]
[0].ToString(), conn);                                              //获取每个部门下的所有员工信息
            DataSet ds1 = new DataSet();                            //声明 DataSet 对象
            sda1.Fill(ds1, "tb_ChildTree");                         //填充该对象
            for (int j = 0; j < ds1.Tables[0].Rows.Count; j++)      //遍历员工表格中的所有行
            {
                string userName = ds1.Tables[0].Rows[j][1].ToString();  //员工名称
                TreeNode TN1 = new TreeNode(userName);              //创建员工节点
                TN.ChildNodes.Add(TN1);                             //将节点添加到控件中
            }
        }
    }
}
```

当单击某个节点后，会激发 TreeView 控件的 SelectedNodeChanged 事件，在此事件中编写代码，实现根据选择的节点，检索出相关的信息绑定到 GridView 控件中显示出来，代码如下：

```
protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
    string name = TreeView1.SelectedValue;                          //获取选择的节点名称
    SqlConnection conn = getCon();                                  //连接数据库
    SqlDataAdapter sda = new SqlDataAdapter("select ChildTreeNode as '员工姓名',sex as '性别',age as '年龄',job as '
职业',addr as '地址' from tb_ChildTree where ChildTreeNode='" + name + "'", conn);//从数据库中检索数据
    DataSet ds = new DataSet();                                     //声明一个 DataSet 对象
    sda.Fill(ds, "tb_ParentTree");                                  //填充该对象
    GridView1.DataSource = ds.Tables[0];                            //设置 GridView1 数据源
```

```
        GridView1.DataBind();                                    //将检索出来的数据绑定到控件上
    }
```

（2）TreeView 控件绑定 Web.sitemap 文件。

下面通过示例介绍 TreeView 控件绑定 Web.sitemap 文件的方法。

示例 19-05　使用 TreeView 控件设计供求网站导航

本示例使用一个简单 Web.sitemap 文件与 TreeView 控件相结合，在 SiteMapDataSource 数据源控件的支持下，实现供求网站导航。示例运行结果如图 19.9 所示。

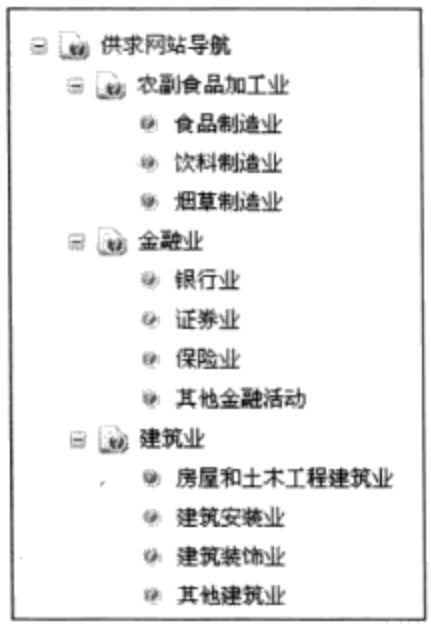


图 19.9　使用 Web.sitemap 文件与 TreeView 控件相结合实现供求网站导航

图 19.9 中显示的是一个由 TreeView 控件构建的树形结构，实现本示例，主要包括两个关键步骤：一是创建树型结构的数据基础 Web.sitemap 文件，二是设置 TreeView 控件属性。

在本示例中用到的 Web.sitemap 文件源代码如下：

代码 19-05　光盘位置：光盘\mingrisoft\sl\19\19_05

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode title="供求网站导航" url="">
    <siteMapNode title="农副食品加工业">
      <siteMapNode title="食品制造业" url=""/>
      <siteMapNode title="饮料制造业" url=""/>
      <siteMapNode title="烟草制造业" url=""/>
    </siteMapNode>
    <siteMapNode title="金融业" >
      <siteMapNode title="银行业" url=""/>
      <siteMapNode title="证券业" url=""/>
      <siteMapNode title="保险业" url=""/>
      <siteMapNode title="其他金融活动" url=""/>
    </siteMapNode>
    <siteMapNode title="建筑业" >
      <siteMapNode title="房屋和土木工程建筑业" url=""/>
      <siteMapNode title="建筑安装业" url=""/>
      <siteMapNode title="建筑装饰业" url=""/>
      <siteMapNode title="其他建筑业" url=""/>
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

这个结构简单的 Web.sitemap 文件，存储站点导航信息，其数据采用普通 XML 格式，将站点逻辑结构层次化的列出。在该文件中，包含一个根节点和多个嵌套节点，并且还为每个节点设置了 title 和 url 属性。在 TreeView 控件中，title 属性值将显示为节点名称，url 属性值是节点的超级链接地址。

注意：url 属性值可以根据实际需要设置相关的 Web 页面，以便单击节点时，导航到该页面中，本示例只做演示，在此设为空值（“”）。

定义完 Web.sitemap 文件后，需要设置 TreeView 控件的相关属性。

首先，将 TreeView 控件从工具箱中拖放到 Web 窗体的适当位置，同时将数据源控件 SiteMapDataSource 拖放到 Web 窗体中，用来支持 TreeView 控件，以便正确显示存储在 Web.sitemap 文件中的数据。

其次，设置 TreeView 控件与 SiteMapDataSource 数据源集成，即设置 TreeView 控件的属性，

使得 TreeView 控件的 DataSourceID 属性值等于 SiteMapDataSource 控件的 ID 属性值。实现方法有如下 3 种。

方法一：利用 Visual Studio 2005 的属性窗口，将 TreeView 1 控件的 DataSourceID 属性值设置为 SiteMapDataSource 1 控件的 ID 属性值，如图 19.10 所示。

方法二：在 TreeView 控件的任务列表中，单击“选择数据源”下拉菜单，选择 SiteMapDataSource 1 控件的 ID 属性值 SiteMapDataSource1，如图 19.11 所示。

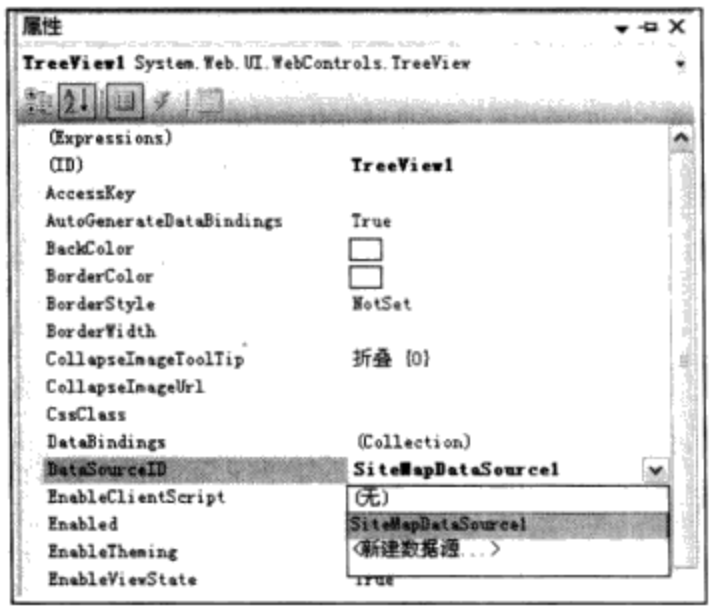


图 19.10　在属性窗口中设置 TreeView 控件的 DataSourceID 属性值

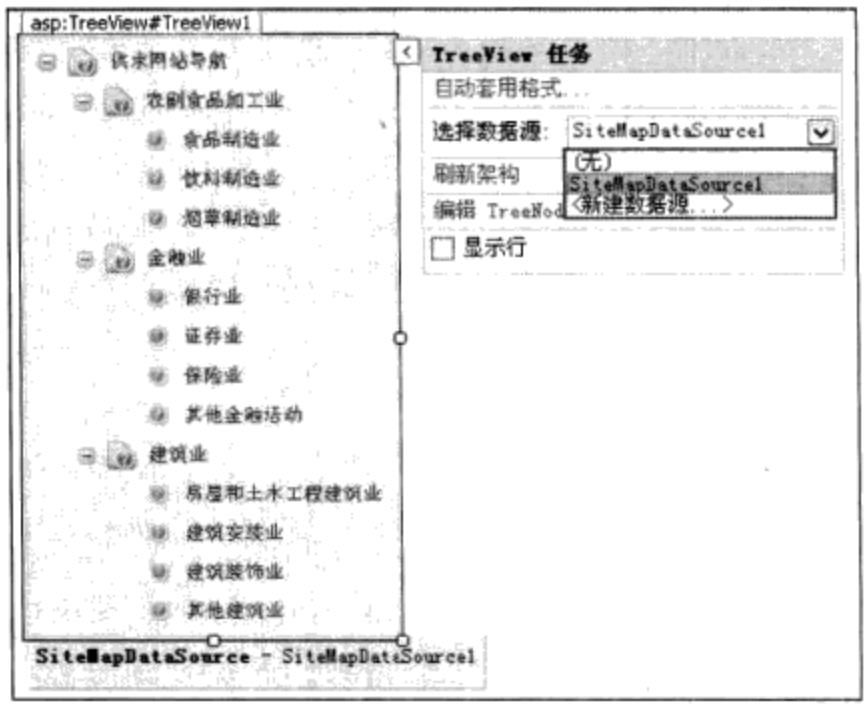


图 19.11　在任务列表中设置 TreeView 控件的数据源

方法三：在页面的 HTML 文件中，添加如下代码，实现 TreeView 控件的 DataSourceID 属性值等于 SiteMapDataSource 1 控件的 ID 属性值。

```
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1" >
        …:…其他代码
    </asp:TreeView>
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

说明：SiteMapDataSource 数据源控件默认处理文件为 Web.sitemap，因此，无需设置 SiteMapDataSource 控件与 Web.sitemap 文件关联。

（3）TreeView 控件绑定 XML 文件。

在程序开发中，某些信息会存储在 XML 文件中，下面就介绍 TreeView 控件如何绑定到 XML 文件。

示例 19-06　使用 TreeView 控件绑定 XML 文件

本示例将 XML 文件绑定到 TreeView 控件上。执行程序，示例运行结果如图 19.12 所示。

程序开发步骤如下。

（1）新建一个网站，默认主页为 Default.aspx。在 Default.aspx 页上添加一个 TreeView 控件和一个 XmlDataSource 控件。

（2）设置 XmlDataSource 控件的数据源：在 XmlDataSource 控件的“XmlDataSource 任务”快捷菜单中选择“配置数据源”，然后，通过单击“浏览”按钮指定 XML 文件的名称为“XMLFile.xml”，最后，单击“确定”按钮完成设置。如图 19.13 所示。

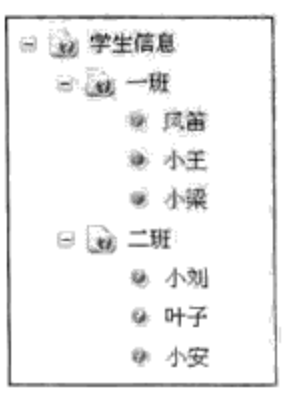


图 19.12　TreeView 控件绑定 XML 文件

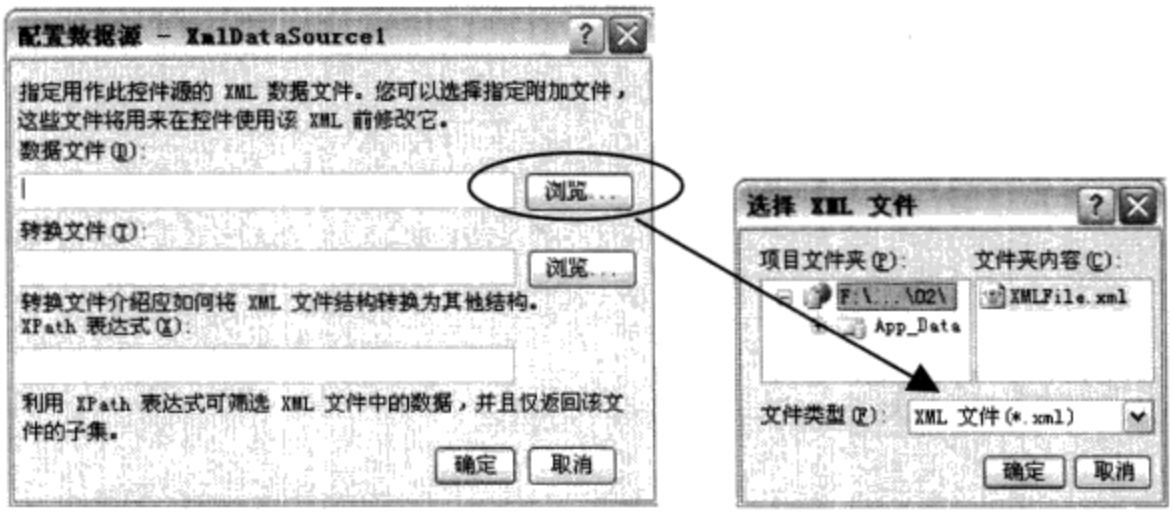


图 19.13　指定 XMLFile.xml 文件

（3）为 TreeView 控件指定数据源：将 TreeView 控件的 DataSourceID 属性设为“XmlDataSource1”，完成后如图 19.14 所示。从图中可以看出 TreeView 控件中显示的不是实际学生的信息，而是只显示节点，这是因还没有设置 XML 节点对应的字段。

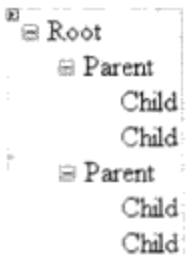


图 19.14　指定数据源后的显示结果

（4）设置 XML 节点对应的字段：在“TreeView 任务”快捷菜单上选择“编辑 TreeNode 数据绑定”，打开“TreeView DataBindings 编辑器”，添加 Root、Parent 和 Child 3 个节点，然后，分别选取 Root、Parent 和 Child 节点，并在属性窗口中设置相关对应字段：NavigateUrlField 属性设置为“url”，TextField 属性设置为“name”，ValueField 属性设置为“describe”，Parent 和 Child 节点设置同上，如图 19.15 所示。单击“确定”按钮结束，这时 TreeView 控件就已经绑定了 XML 文件。

图 19.15　设置 XML 节点对应的字段

（5）程序主要代码如下。

代码 19-06　实例位置：光盘\mingrisoft\sl\19\19_06

XMLFile.xml 文件的源代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<Root url="Default.aspx" name="学生信息" describe="studentinfo">
  <Parent url="class1.aspx" name="一班" describe="classone">
```

```
    <Child url="stu11.aspx" name="风笛" describe="fengdi"></Child>
    <Child url="stu12.aspx" name="小王" describe="xiaowang"></Child>
    <Child url="stu13.aspx" name="小梁" describe="xiaoliang"></Child>
  </Parent>
  <Parent url="class2.aspx" name="二班" describe="classtwo">
    <Child url="stu121.aspx" name="小刘" describe="xiaoliu"></Child>
    <Child url="stu122.aspx" name="叶子" describe="yezi"></Child>
    <Child url="stu123.aspx" name="小安" describe="xiaoan"></Child>
  </Parent>
</Root>
```

19.5.2 Menu、SiteMapPath 联合制作站点导航

在开发一些大型的商业网站或门户类网站时，可以利用服务器控件 Menu 实现动态的导航条，方便用户在页面之间的跳转，同时还可以利用服务器控件 SiteMapPath 在每个网页中定位，将当前浏览页在网站中的位置显示给用户，使用户轻松地浏览网页。

示例 19-07　使用服务器控件 Menu 和 SiteMapPath 联合制作 BBS 导航

下面通过一个简单的示例，使用服务器控件 Menu 和 SiteMapPath 联合制作 BBS 导航。示例运行结果如图 19.16 所示，用来显示网站导航信息，在该图中单击“论坛专题”按钮，将会弹出如图 19.17 所示的界面，用来显示“论坛专题”页面在网站中所处的位置。

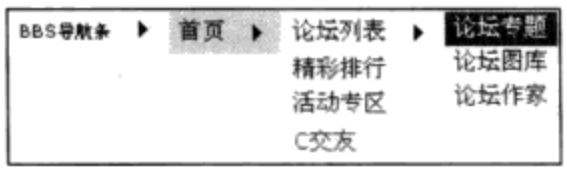


图 19.16　使用 Menu 控件制作 BBS 导航

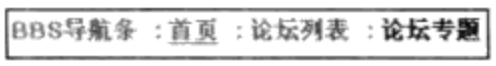


图 19.17　使用 SiteMapPath 控件定位网页在网站的位置

程序开发步骤如下。

（1）新建一个网站，命名为 19_06，其主页默认为 Default.aspx。

（2）在该页上添加一个 Menu 控件和一个数据源控件 SiteMapDataSource，并将 Menu 控件的 DataSourceID 属性值设置为数据源控件 SiteMapDataSource 的 ID 值。其中 Menu 控件需要在数据源控件 SiteMapDataSource 的支持下绑定 Web.sitemap 文件，显示整个网站的导航信息。

（3）在该网站中，再添加 6 个 Web 页面：1.aspx、2.aspx、3.aspx、4.aspx、5.aspx 和 6.aspx，并在各个网页中添加 SiteMapPath 导航控件，用来显示每个页在网站中所处的位置。

（4）程序主要代码如下。

在使用 Menu 和 SiteMapPath 服务器控件前，需要先创建 Web.sitemap 文件，为站点导航控件提供站点层次结构信息，站点地图文件 Web.sitemap 的源代码如下：

代码 19-07　实例位置：光盘\mingrisoft\sl\19\19_07

```
<?xml version="1.0" encoding="utf-8" ?>
  <siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
    <siteMapNode url="" title="BBS 导航条"  description="BBS 导航条"  >
      <siteMapNode url="～/Default.aspx" title="首页"  description="首页" >
      <siteMapNode title="论坛列表"  description="论坛列表" >
        <siteMapNode  title="论坛专题"  description="论坛专题" url="～/1.aspx"/>
        <siteMapNode title="论坛图库"  description="论坛图库" url="～/2.aspx"/>
        <siteMapNode  title="论坛作家"  description="论坛作家" url="～/3.aspx"/>
      </siteMapNode>
      <siteMapNode  title="精彩排行"  description="精彩排行" url="～/4.aspx"/>
      <siteMapNode  title="活动专区"  description="活动专区" url="～/5.aspx"/>
    <siteMapNode  title="C 交友" description="C 交友"  url="～/6.aspx"/>
    </siteMapNode></siteMapNode>
  </siteMap>
```

第 20 章

ASP.NET 3.5 缓存技术

- ASP.NET 3.5 缓存概述
- 页面输出缓存
- 页面部分缓存
- 页面数据缓存

20.1 ASP.NET 3.5 缓存概述

缓存是 ASP.NET 中非常重要的一个特性，可以生成高效能的 Web 应用程序。生成高性能的 Web 应用程序最重要的因素之一，就是将那些频繁访问，而且不需要经常更新的数据存储在内存中，当客户端再一次访问这些数据时，可以避免重复获取满足先前请求的信息，实现快速显示请求的 Web 页面。

ASP.NET 3.5 中有 3 种 Web 应用程序可以使用缓存技术：页面输出缓存、页面部分缓存和页面数据缓存。ASP.NET 3.5 的缓存功能具有以下优点。

（1）支持更为广泛和灵活的可开发特征。

ASP.NET 3.5 包含一些新增的缓存控件和 API。例如，自定义缓存依赖、Substitution 控件、页面输出缓存 API 等，这些特征能够明显改善开发人员对于缓存功能的控制。

（2）增强可管理性。

使用 ASP.NET 3.5 提供的配置和管理功能，可以更加轻松地管理缓存功能。

（3）提供更高的性能和可伸缩性。

ASP.NET 3.5 提供了一些新的功能，例如，SQL 数据缓存依赖等，这些功能将帮助开发人员创建高性能、伸缩性强的 Web 应用程序。

注意：缓存功能也有其自身的不足。例如，显示的内容可能不是最新、最准确的，为此，必须设置合适的缓存策略。又如，缓存增加了系统的复杂性并使其难于测试和调试，因此建议在没有缓存的情况下开发和测试应用程序，然后在性能优化阶段启用缓存选项。

20.2 页面输出缓存

页面输出缓存是最简单的缓存机制，该机制将整个 ASP.NET 3.5 页面内容保存在服务器内存中。当用户请求该页面时，系统从内存中输出相关数据，直到缓存数据过期。在这个过程中，缓存内容直接发送给用户，而不必再次经过页面处理生命周期。通常情况下，页面输出缓存对于那些包含不需要经常修改内容，但需要大量处理才能编译完成的页面特别有用。另外，页面输出缓存是将页面全部内容都保存在内存中，并用于完成客户端请求。

页面输出缓存需要利用有效期来对缓存区中的页面进行管理。@OutputCache 指令的格式如下：

```
<%@ OutputCache Duration="#ofseconds"
Location="Any | Client | Downstream | Server | None | ServerAndClient "
Shared="True | False"
VaryByControl="controlname"
VaryByCustom="browser | customstring"
VaryByHeader="headers"
VaryByParam="parametername"
%>
```

下面对@OutputCache 指令中的各个属性进行详细介绍。

（1）Duration 属性。

设置 Web 页或用户控件进行缓存的时间（以 s 计），在 Web 页或用户控件上设置 Duration 属性，为请求对象的 HTTP 响应建立一个过期策略，并将自动缓存页或用户控件输出。

注意：Duration 属性为必选属性。HttpCachePolicy 类的 SetExpires 方法用来确定有效期的时间，它的参数是一个 DateTime 类的实例，表示过期的绝对时间。

（2）Location 属性。

OutputCacheLocation 枚举值之一，默认值为 Any。OutputCacheLocation 枚举成员如表 20.1 所示。

表 20.1　　OutputCacheLocation 枚举成员列表

成员名称	说　明
Any	输出缓存可位于产生请求的浏览器客户端、参与请求的代理服务器（或任何其他服务器）或处理请求的服务器上。此值对应于 HttpCacheability.Public 枚举值
Client	输出缓存位于产生请求的浏览器客户端上。此值对应于 HttpCacheability.Private 枚举值
Downstream	输出缓存可存储在任何 HTTP 1.1 可缓存设备中，源服务器除外。这包括代理服务器和发出请求的客户端
None	对于请求的页，禁用输出缓存。此值对应于 HttpCacheability.NoCache 枚举值
Server	输出缓存位于处理请求的 Web 服务器上。此值对应于 HttpCacheability.Server 枚举值
ServerAndClient	输出缓存只能存储在源服务器或发出请求的客户端中。代理服务器不能缓存响应。此值对应于 HttpCacheability.Private 和 HttpCacheability.Server 枚举值的组合

注意：包含在用户控件（.ascx 文件）中的@ OutputCache 指令不支持 Location 属性。

（3）CacheProfile 属性。

与当前页关联的缓存设置的名称。这是可选属性，默认值为空字符（""）。

注意：包含在用户控件（.ascx 文件）中的@ OutputCache 指令不支持 CacheProfile 属性。在页中指定 CacheProfile 属性时，属性值必须与 outputCacheSettings 节下面的 outputCacheProfiles 元素中的一个可用项的名称匹配。如果此名称与配置文件项不匹配，将引发异常。

（4）NoStore 属性。

设置一个布尔值，它决定了是否阻止敏感信息的二级存储。

注意：包含在用户控件中的@ OutputCache 指令不支持此属性。将此属性设置为 True 等效于在请求期间执行以下代码：Response.Cache.SetNoStore()。

（5）Shared 属性。

设置一个布尔值，确定用户控件输出是否可以由多个页共享。默认值为 False。

注意：包含在 ASP.NET 3.5 页（.aspx 文件）中的@ OutputCache 指令不支持此属性。

（6）SqlDependency 属性。

标识一组数据库和数据表名称的字符串值，页或控件的输出缓存依赖于这些数据库名称和数据表名称。

注意：SqlDependency 属性的 CommandNotification 值仅在网页（.aspx）中有效。

（7）VaryByCustom 属性。

表示自定义输出缓存要求的任意文本。如果该属性的值为 browser，缓存将随浏览器名称和主要版本信息的不同而异。如果输入自定义字符串，则必须在应用程序的 Global.asax 文件中重写 GetVaryByCustomString 方法。

（8）VaryByHeader 属性。

分号分隔的 HTTP 标头列表，用于使输出缓存发生变化。将该属性设为多标头时，对于每个指定标头组合，输出缓存都包含一个不同版本的请求文档。

注意：设置 VaryByHeader 属性将启用在所有 HTTP 1.1 版缓存中的缓存项，而不仅仅在 ASP.NET 3.5 缓存中进行缓存。用户控件中的@ OutputCache 指令不支持此属性。

（9）VaryByParam 属性。

分号分隔的字符串列表，用于使输出缓存发生变化。默认情况下，这些字符串与随 GET

方法属性发送的查询字符串值对应，或与使用 POST 方法发送的参数对应。将该属性设置为多个参数时，对于每个指定参数组合，输出缓存都包含一个不同版本的请求文档。该属性可能的值包括 none、星号（*）以及任何有效的查询字符串或 POST 参数名称。

注意：在 ASP.NET 3.5 页和用户控件上使用@ OutputCache 指令时，需要该属性或 VaryByControl 属性。如果没有包含它，则发生分析器错误。如果不希望通过指定参数来改变缓存内容，请将值设置为 none。如果希望通过所有的参数值改变输出缓存，请将属性设置为星号（*）。

（10）VaryByControl 属性。

一个分号分隔的字符串列表，用于更改用户控件的输出缓存。这些字符串代表用户控件中声明的 ASP.NET 服务器控件的 ID 属性值。

注意：在 ASP.NET 3.5 页和用户控件上使用@ OutputCache 指令时，需要该属性或 Vary ByParam 属性。

示例 20-01　设置页面缓存的过期时间为当前时间加 30s

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(30));
```

等同于@OutputCache 指令格式：

```
<%@ OutputCache Duration ="30" VaryByParam ="none"%>
```

本示例演示的是，当程序运行时，将 Web 页面的缓存时间设置为 30s，在 30s 内刷新页面时，页面数据不发生变化，30s 后则刷新页面后数据发生改变，如图 20.1 和图 20.2 所示。

图 20.1　30s 以内的页面缓存

图 20.2　30s 后的页面缓存

程序代码如下：

代码 20-01　光盘位置：光盘\mingrisoft\sl\20\20_01

```
<%@ OutputCache Duration="30" VaryByParam="none" %>                    //设置缓存时间
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
/xhtml1- transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>页面输出缓存</title>
</head>
<script language="C#" runat="server">
void Page_Load(object sender, EventArgs e)                             //页面加载事件
{
    //输出页面缓存设置的时间和当前时间
    Response.Write("页面缓存设置示例：<br> 设置缓存时间为 30s，当前时间为：" + DateTime.Now.ToString());
}
</script>
<body>
</body>
</html>
```

20.3 页面部分缓存

通常情况下，缓存整个页是不合理的，因为页的某些部分可能在每一次请求时都进行更改，这种情况下，只能缓存页的一部分即页面部分缓存。页面部分缓存是将页面部分内容保存在内存中以便响应用户请求，而页面其他部分内容则为动态内容。

页面部分缓存的实现包括两种方式：控件缓存和替换后缓存。前者也可称为片段缓存，这种方式允许将需要缓存的信息包含在一个用户控件内，然后，将该用户控件标记为可缓存的，以此来缓存页面输出的部分内容。这一方式缓存了页面中的特定内容，而没有缓存整个页面，因此，每次都需重新创建整个页。例如，如果要创建一个显示大量动态内容（如论坛帖子信息）的页，其中有些部分为静态内容（如主题信息），这时可以将静态部分放在用户控件中，并允许缓存这些内容。缓存后替换与用户控件缓存正好相反。这种方式可以缓存整个页，但页中的各段都是动态的。

20.3.1 使用@ OutputCache 指令

用户控件缓存与页面输出缓存的@ OutputCache 指令设置方法基本相同，都在文件顶部设置@ OutputCache 指令。两者不同点包括如下两个方面。

（1）用户控件缓存的@ OutputCache 指令设置在用户控件文件中，而页面输出缓存的@ OutputCache 指令设置在普通 ASP.NET 文件中。

（2）用户控件缓存的@ OutputCache 指令只能设置 6 个属性，Duration、Shared、SqlDependency、VaryByControl、VaryByCustom 和 VaryByParam。而在页面输出缓存的@ OutputCache 指令字符串中设置的属性多达 10 个。

用户控件中的@ OutputCache 指令设置源代码如下：

```
<%@ OutputCache Duration="60" VaryByParam="none" VaryByControl="ControlID" %>
```

以上代码为用户控件中的服务器控件设置缓存，其中缓存时间为 60s，ControlID 是服务器控件 ID 属性值。

注意：ASP.NET 3.5 页面和其中包含的用户控件都通过@ OutputCache 指令设置了缓存，应注意以下 3 点。

① ASP.NET 3.5 允许在页面和页面的用户控件中同时使用@ OutputCache 指令设置缓存，并且允许设置不同的缓存过期时间值。

② 如果页面输出缓存过期时间长于用户控件输出缓存过期时间，则页面的输出缓存持续时间优先。例如，如果页面输出缓存设置为 100s，而用户控件的输出缓存设置为 50s，则包括用户控件在内的整个页将在输出缓存中存储 100s，而与用户控件较短的时间设置无关。

③ 如果页面输出缓存过期时间比用户控件的输出缓存过期时间短，则即使已为某个请求重新生成该页面的其余部分，也将一直缓存用户控件直到其过期时间到期为止。例如，如果页面输出缓存设置为 50s，而用户控件输出缓存设置为 100s，则页面其余部分每到期两次，用户控件才到期一次。

20.3.2 使用 PartialCachingAttribute 类

使用 PartialCachingAttribute 类定义 Web 窗体用户控件，用于设置有关控件缓存的配置内容。本节主要讲解 PartialCachingAttribute 类的 6 个常用属性和 4 种类构造函数。6 个常用属性是 Duration、Shared、SqlDependency、VaryByControl、VaryByCustom 和 VaryByParam。与 20.2 节中的@ OutputCache 指令设置的 6 个属性完全相同，但所使用的方式不同。PartialCaching

Attribute 类中的构造函数和常用属性及说明如表 20.2 和表 20.3 所示。

表 20.2　　PartialCachingAttribute 类的构造函数及说明

构造函数	说　明
PartialCachingAttribute（nt32）	使用分配给要缓存的用户控件的指定持续时间初始化 PartialCachingAttribute 类的新实例
PartialCachingAttribute （nt32、String、String、String）	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名和用于改变缓存的自定义输出缓存要求
PartialCachingAttribute（Int32、String、String、String、Boolean）	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名、用于改变缓存的自定义输出缓存要求以及用户控件输出是否可在多页间共享
PartialCachingAttribute （Int32、String、String、String、String、Boolean）	初始化 PartialCachingAttribute 类的新实例，指定缓存持续时间、所有 GET 和 POST 值、控件名、用于改变缓存的自定义输出缓存要求、数据库依赖项以及用户控件输出是否可在多页间共享

表 20.3　　PartialCachingAttribute 类的常用属性及说明

属　性	说　明
Duration	获取缓存项保留在输出缓存中的时间（以 s 为单位）
Shared	获取一个值，该值指示用户控件输出是否可在多页间共享
SqlDependency	获取一个分隔字符串，该字符串标识用户控件所依赖的一个或多个数据库和表的名称对
TypeId	当在派生类中实现时，获取该 Attribute 的惟一标识符
VaryByControls	获取输出缓存用于使用户控件发生变化的用户控件属性列表
VaryByCustom	获取输出缓存将用于改变用户控件的自定义字符串列表
VaryByParams	获取输出缓存将用于改变用户控件的查询字符串或窗体 POST 参数的列表

以上介绍了 PartialCachingAttribute 类的 6 个属性和 4 种构造函数。下面通过一个典型示例说明该类的具体应用方法。

本示例使用 PartialCachingAttribute 类设置用户控件（WebUserControl.ascx 文件）的缓存有效期时间是 20s。

使用 PartialCachingAttribute 类实现设置用户控件缓存，必须在用户控件类声明前设置“[PartialCaching（20）]”，代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
[PartialCaching(20)]                    //设置用户控件的缓存时间为 20s
public partial class WebUserControl : System.Web.UI.UserControl
{
}
```

以上代码设置了缓存有效时间为 20s。这与在 WebUserControl.ascx 文件顶部设置@ Output Cache 指令的 Duration 属性值为 20 是完全一致的。

20.3.3　使用 ControlCachePolicy 类

ControlCachePolicy 是.NET Framework 2.0 中新增的类，主要用于提供对用户控件的输出缓存设置的编程访问。ControlCachePolicy 类与 20.3.2 小节 HttpCachePolicy 类实现的作用类似，

只是二者所访问的对象不同。HttpCachePolicy 类用于访问页面输出缓存，而 ControlCachePolicy 类用于访问用户缓存。

ControlCachePolicy 类包含 6 个属性，分别是 Cached、Dependency、Duration、Supports Caching、VaryByControl 和 VaryByParams，如表 20.4 所示。

表 20.4　　ControlCachePolicy 类的常用属性及说明

属　性	说　明
Cached	获取或设置一个值，该值指示是否为用户控件启用片段缓存
Dependency	获取或设置与缓存的用户控件输出关联的 CacheDependency 类的实例
Duration	获取或设置缓存的项将在输出缓存中保留的时间
SupportsCaching	获取一个值，该值指示用户控件是否支持缓存
VaryByControl	获取或设置要用来改变缓存输出的控件标识符列表
VaryByParams	获取或设置要用来改变缓存输出的 GET 或 POST 参数名称列表

示例 20-02　使用 ControlCachePolicy 类实现设置用户控件缓存

下面示例主要通过使用 PartialCachingAttribute 类设置用户控件(WebUserControl.ascx 文件)的缓存有效期时间是 20s。执行程序，示例运行结果如图 20.3 所示，当刷新页面后，用户控件中的系统时间在 20s 内是不变化的，而当前页面中的时间是随着页面的刷新而变化的，运行结果如图 20.4 所示。

图 20.3　示例运行初期

图 20.4　刷新页面后用户控件中的时间与系统时间的差异

程序实现的主要步骤如下。

(1) 新建一个网站，默认主页为 Default.aspx。并在该页中添加一个 Label 控件用于显示当前系统时间。

(2) 在该网站中，添加一个用户控件，默认名为 WebUserControl.ascx。在该用户控件中，添加一个 Label 控件用于显示当前系统时间。

(3) 为了使用 PartialCachingAttribute 类设置用户控件（WebUserControl.ascx 文件）的缓存有效期时间为 20s，必须在用户控件类声明前设置“[PartialCaching（20）]”。

程序代码如下。

代码 20-02　光盘位置：光盘\mingrisoft\sl\20\20_02

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
```

```
using System.Web.UI.HtmlControls;
[PartialCaching(20)]                    //设置用户控件的缓存时间为 20 秒
public partial class WebUserControl : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            this.Label1.Text = "用户控件中的系统时间：" + DateTime.Now.ToString();
        }
    }
}
```

20.4 页面数据缓存

页面数据缓存也即应用程序数据缓存，它提供了一种编程方式，可通过键/值将任意数据存储在内存中。使用应用程序缓存与使用应用程序状态类似。但是，与应用程序状态不同的是，应用程序数据缓存中的数据是容易丢失的，即数据并不是在整个应用程序生命周期中都存储在内存中。应用程序数据缓存的优点是由 ASP.NET 管理缓存，它会在项过期、无效或内存不足时移除缓存中的项，还可以配置应用程序缓存，以便在移除项时通知应用程序。

ASP.NET 中提供了类似于 Session 的缓存机制，即页面数据缓存。利用数据缓存，可以在内存中存储各种与应用程序相关的对象。对于各个应用程序来说，数据缓存只是在应用程序内共享，并不能在应用程序间进行共享。Cache 类用于实现 Web 应用程序的缓存，在 Cache 中存储数据的最简单的方法如下：

```
Cache["Key"]=Value;
```

从缓存中取数据时，需要先判断缓存中是否有内容，其方法如下：

```
Value=(string)Cache["key"];
If(Value!=null)
{
    //do something
}
```

注意：从 Cache 中得到的对象是一个 object 类型的对象，因此在通常情况下，需要进行强制类型转换。

Cache 类有两个很重要的方法，Add 和 Insert 方法，其语法格式如下：

```
public Object Add[Insert] (
    string key,
    Object value,
    CacheDependency dependencies,
    DateTime absoluteExpiration,
    TimeSpan slidingExpiration,
    CacheItemPriority priority,
    CacheItemRemovedCallback onRemoveCallback
)
```

参数说明如下。

key ：用于引用该项的缓存键。

value：要添加到缓存的项。

dependencies：该项的文件依赖项或缓存键依赖项，当任何依赖项更改时，该对象即无效，并从缓存中移除。如果没有依赖项，则此参数可以设置为 null。

absoluteExpiration：过期的绝对时间。

slidingExpiration：最后一次访问所添加对象时与该对象过期时之间的时间间隔。

priority：缓存的优先级，由 CacheItemPriority 枚举表示。缓存的优先级共有 6 种，从高到低依次是 NotRemoveable、High、AboveNormal、Normal、BelowNormal 和 Low。

OnRemoveCallback：在从缓存中移除对象时所调用的委托（如果没有，可以设置为 null）。当从缓存中删除应用程序的对象时，它将会被调用。

Insert 方法声明与 Add 方法类似，但 Insert 方法为可重载方法。其结构如表 20.5 所示。

表 20.5　　Insert 重载方法列表

重 载 方 法	说　明
Cache.Insert (String, Object)	向 Cache 对象插入项，该项带有一个缓存键引用其位置，并使用 CacheItemPriority 枚举提供的默认值
Cache.Insert (String, Object, CacheDependency)	向 Cache 中插入具有文件依赖项或键依赖项的对象
Cache.Insert (String, Object, CacheDependency, DateTime, TimeSpan)	向 Cache 中插入具有依赖项和过期策略的对象
Cache.Insert (String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, Cache ItemRemovedCallback)	向 Cache 对象中插入对象，后者具有依赖项、过期和优先级策略以及一个委托（可用于在从 Cache 移除插入项时通知应用程序）

Insert 方法中，CacheDependency 是指依赖关系，DateTime 是有效时间，TimeSpan 是创建对象的时间间隔。

下面通过示例来讲解 Insert 方法的使用。

例如，将文件中的 xml 数据插入缓存，在以后请求时无需从文件读取该数据。Cache Dependency 的作用是确保缓存在文件更改后立即到期，以便可以从文件中提取最新数据，重新进行缓存。如果缓存的数据来自若干个文件，还可以指定一个文件名的数组。代码如下：

```
Cache.Insert("key", myXMLFileData, new
System.Web.Caching.CacheDependency(Server.MapPath("users.xml")));
```

例如，插入键值为 key 的第二个数据块（取决于是否存在第一个数据块）。如果缓存中不存在名为 key 的键，或者与该键相关联的项已到期或被更新，那么 dependentkey 的缓存条目将到期。代码如下：

```
Cache.Insert("dependentkey", myDependentData, new
System.Web.Caching.CacheDependency(new string[] {}, new string[]
{"key"}));
```

下面是一个绝对到期的示例，此示例将对受时间影响的数据缓存 1min，1min 过后，缓存将到期。

注意：绝对到期和滑动到期不能一起使用。

程序代码如下：

```
Cache.Insert("key", myTimeSensitiveData, null,
DateTime.Now.AddMinutes(1), TimeSpan.Zero);
```

下面是一个绝对滑动到期的示例，此示例将缓存一些频繁使用的数据。数据将在缓存中一直保留，除非数据未被引用的时间达到了 1min。代码如下：

```
Cache.Insert("key", myFrequentlyAccessedData, null,
System.Web.Caching.Cache.NoAbsoluteExpiration,
TimeSpan.FromMinutes(1));
```

第 4 篇

数据库开发篇

C#

第 21 章

数据库的安装与创建

- Access 数据库
- SQL Server 数据库
- 配置 ODBC 数据源
- 数据库的连接
- 数据库备份与恢复
- 附加和分离数据库

本章将介绍如何安装、创建 Access 及 SQL Server 数据库。

21.1 Access 数据库

21.1.1 安装 Access 数据库

下面分别介绍 Access 2000 和 Access 2003 数据库的安装。

1．Access 2000 数据库

在安装 Access 2000 之前，首先需要准备好 Office 2000 安装盘。具体操作步骤如下。

（1）将 Office 2000 安装盘放入光驱后，系统会自动启动程序，如果禁用了光驱的自动运行功能，那么打开“我的电脑”/“CD 驱动器（或 DVD 驱动器）”，然后双击“SETUP.EXE”安装文件，根据安装向导对话框，进行安装，如图 21.1 所示。

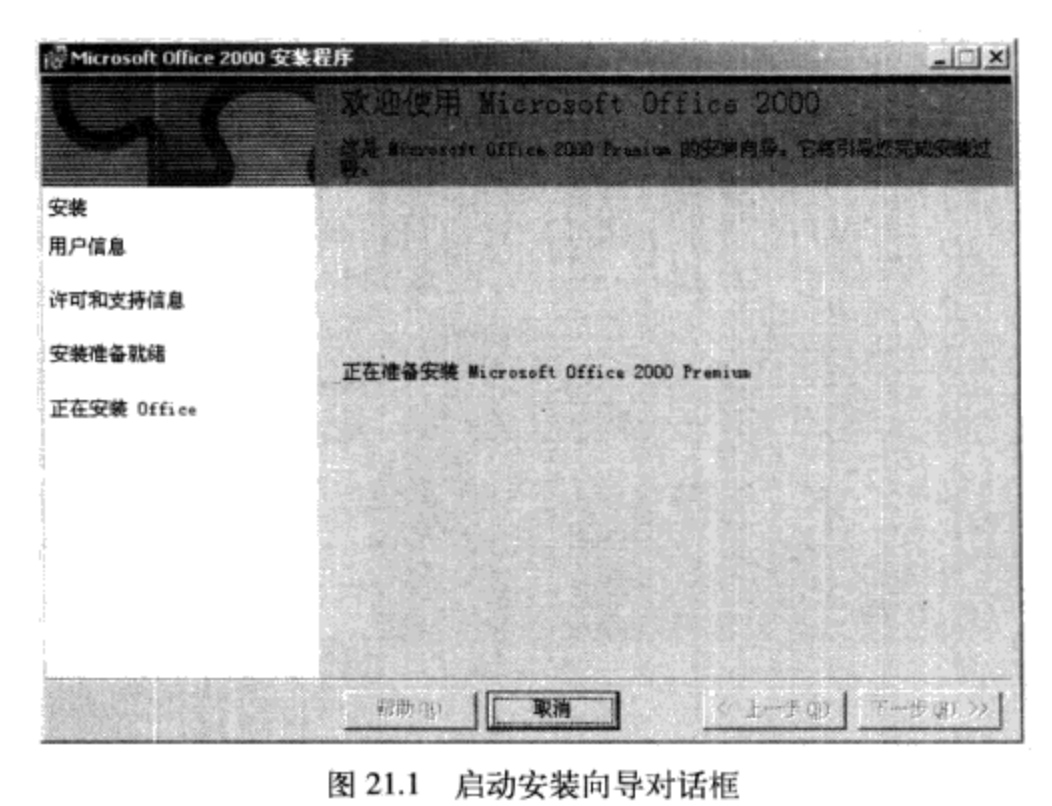


图 21.1　启动安装向导对话框

（2）启动安装向导后，在安装向导对话框中单击“下一步”按钮，会打开如图 21.2 所示的安装注册对话框，输入相关的用户信息，然后输入正确的安装注册码。

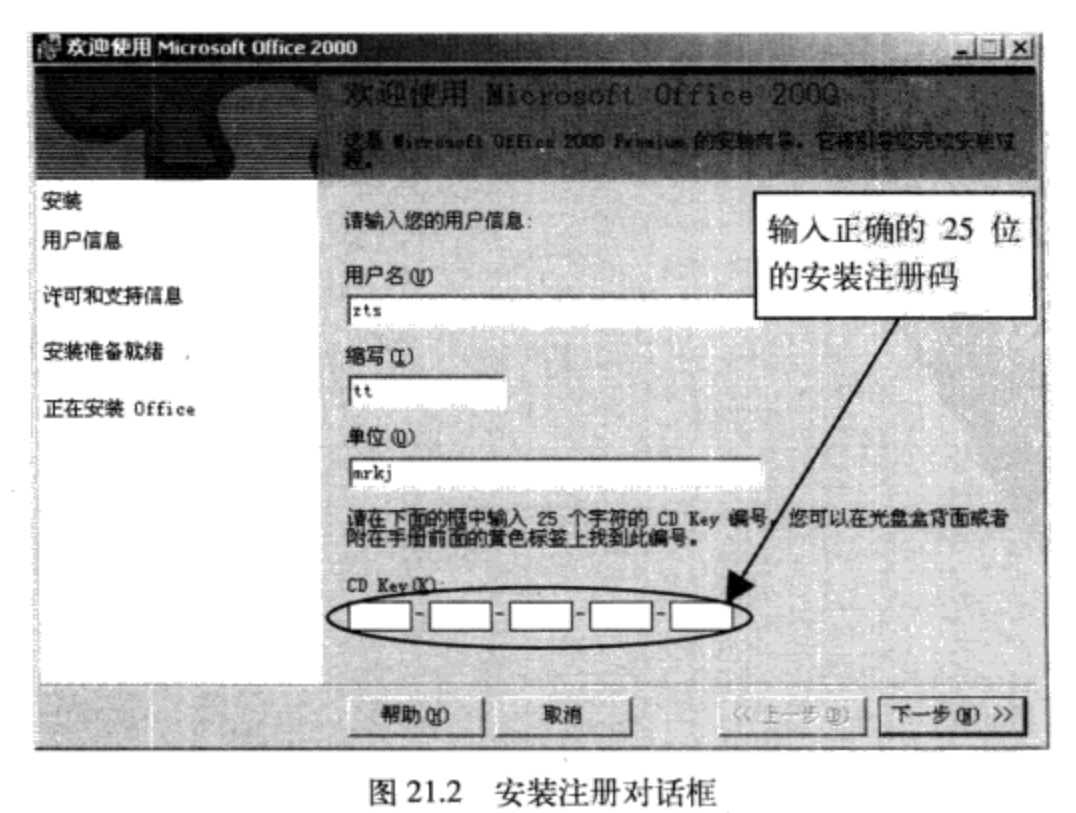


图 21.2　安装注册对话框

(3) 单击“下一步”按钮，打开如图 21.3 所示的对话框，用户必须选中“我接受《许可协议》中的条款(A)”复选框。

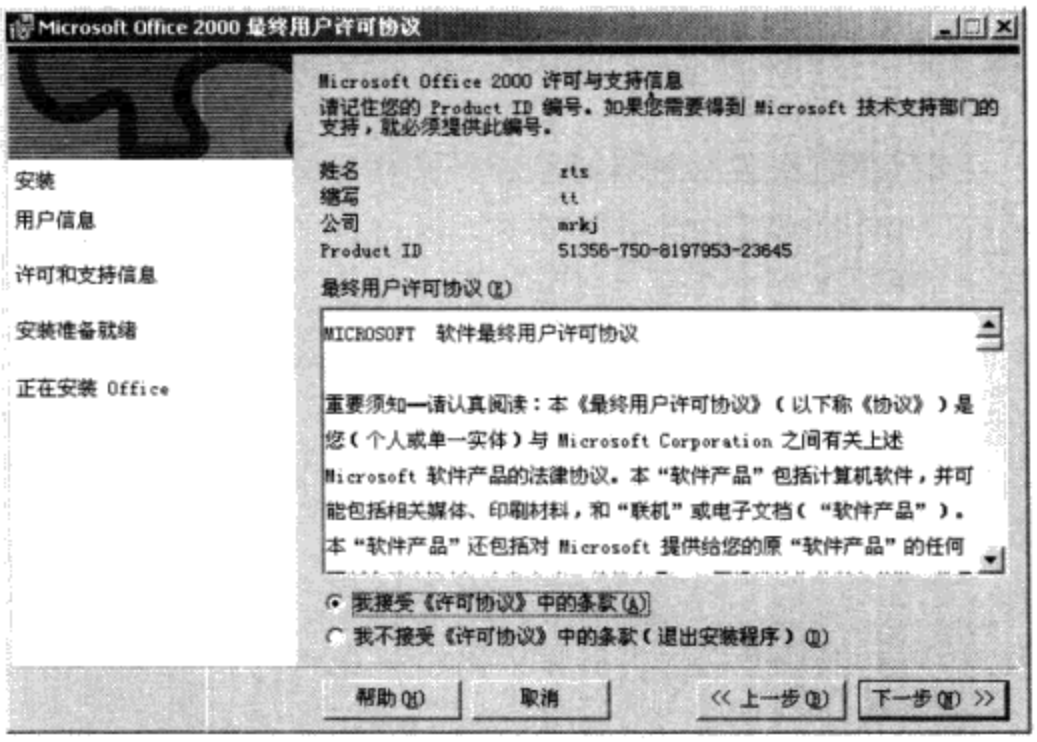


图 21.3 最终用户许可协议对话框

(4) 单击“下一步”按钮，打开如图 21.4 所示的对话框，选择安装的类型，可选择“开始安装”（本例选择“开始安装”），或“自定义”安装。

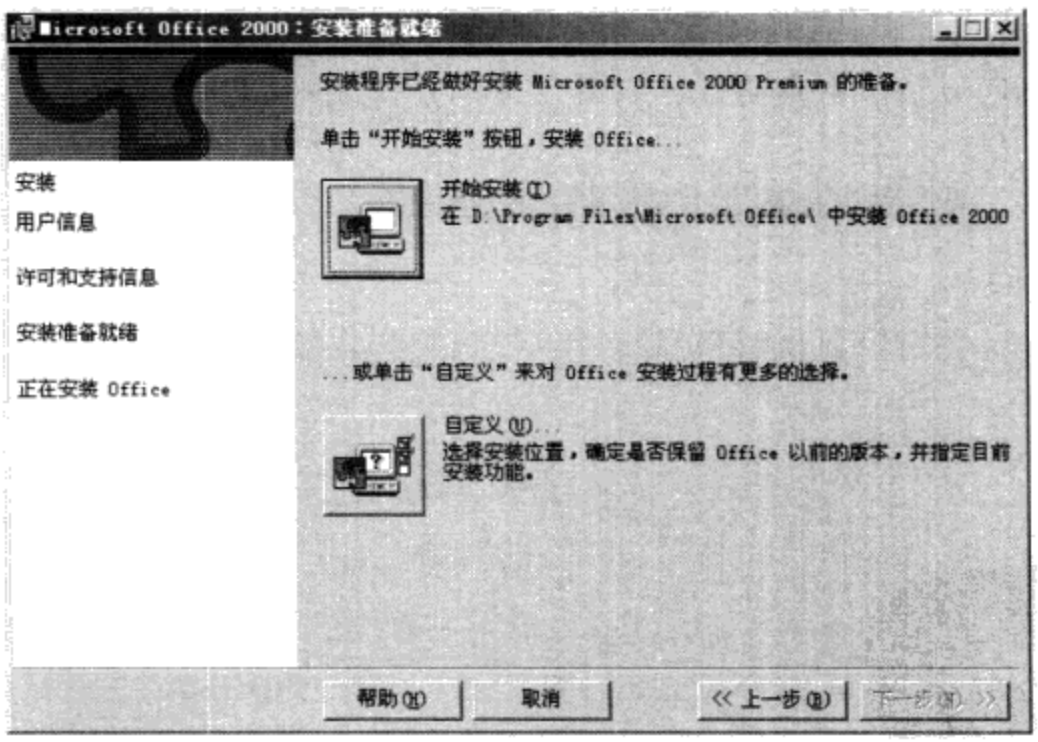


图 21.4 安装对话框

(5) 单击“开始安装”按钮，打开如图 21.5 所示的安装进程对话框。

(6) 安装进程结束后，将弹出如图 21.6 所示的安装成功对话框。单击“确定”按钮。至此，Access 2000 安装完成，用户可以在开始菜单中找到并启动 Access 2000。

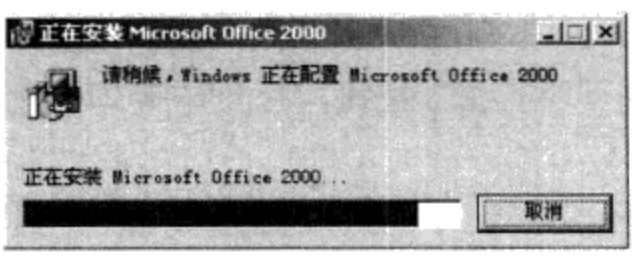


图 21.5 安装进程对话框

图 21.6 安装成功对话框

2．Access 2003 数据库

在安装 Access 2003 之前，首先需要准备好 Office 2003 安装盘。具体操作步骤如下。

（1）将 Office 2003 安装盘放入光驱后，系统会自动启动程序或者直接打开安装盘目录双击“SETUP.EXE”安装文件，打开如图 21.7 所示的启动安装向导对话框。

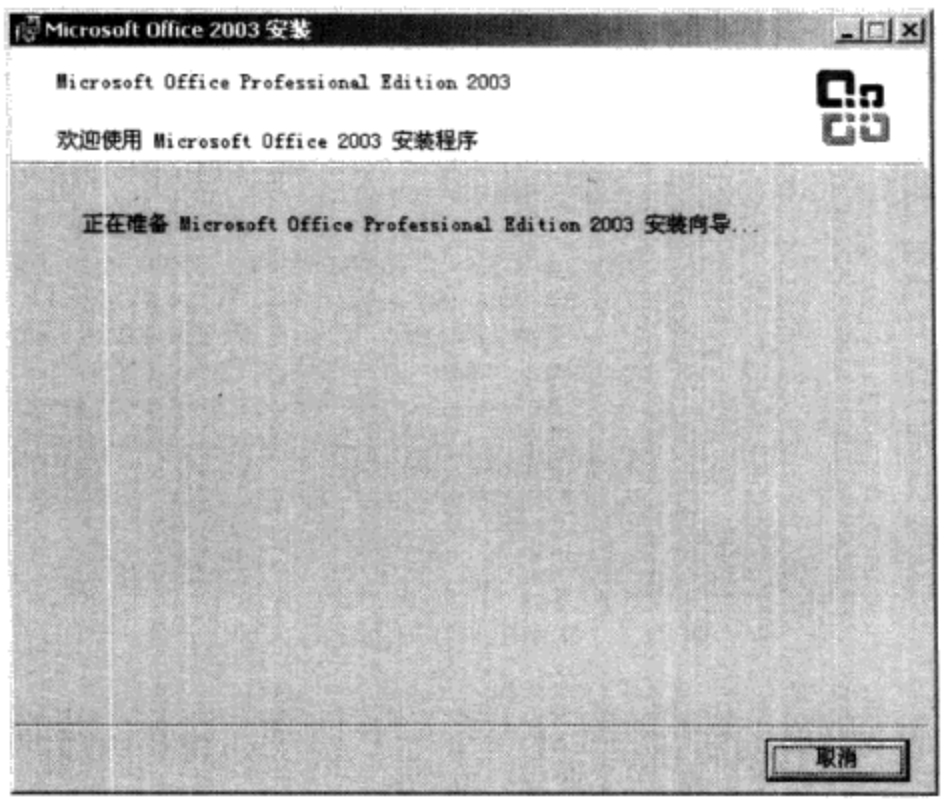


图 21.7　启动安装向导对话框

（2）启动安装向导后，将自动打开如图 21.8 所示安装对话框。

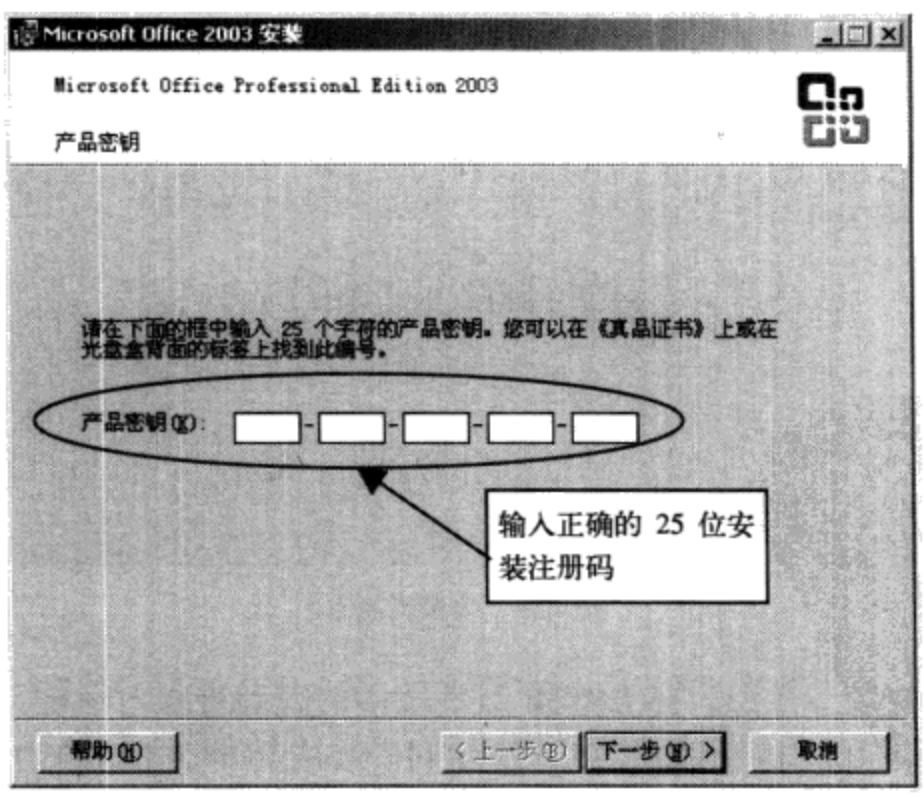


图 21.8　安装对话框

（3）用户输入正确的 25 位安装注册码后，单击“下一步”按钮，打开如图 21.9 所示的用户信息对话框。

（4）输入相关的用户信息后，单击“下一步”按钮，打开如图 21.10 所示的最终用户许可协议对话框。

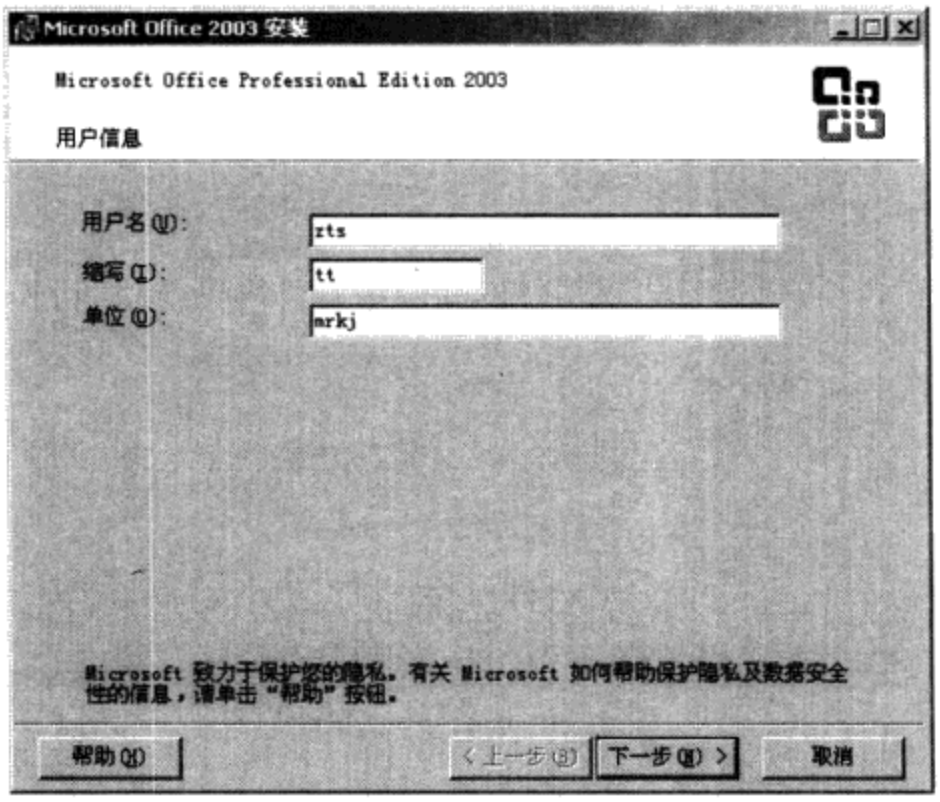


图 21.9　用户信息对话框

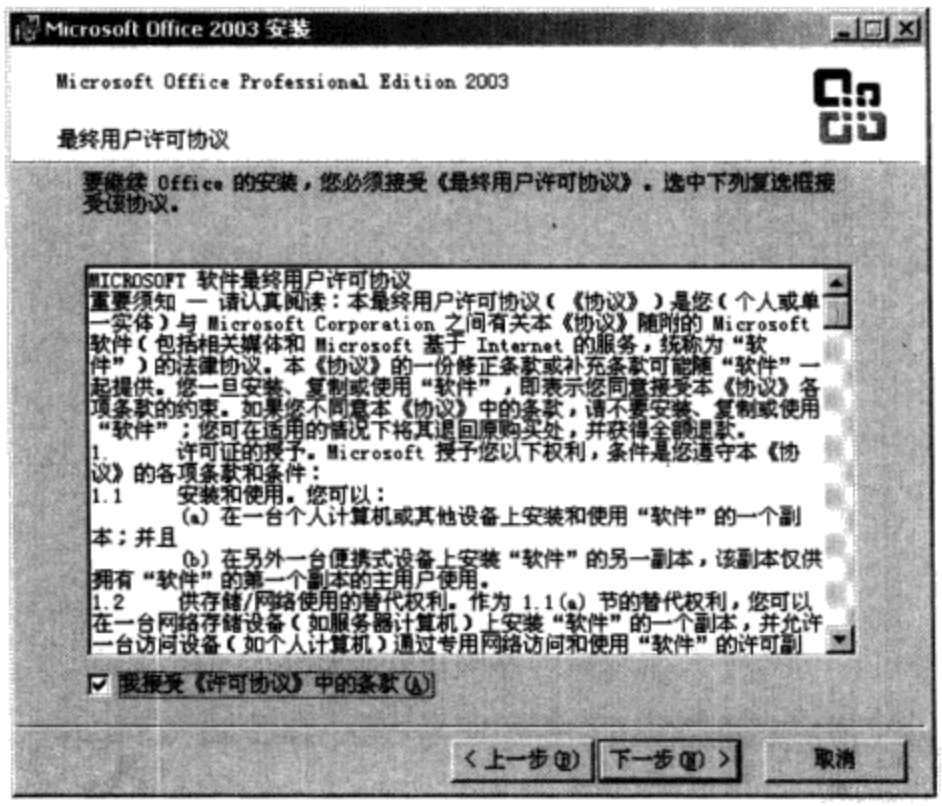


图 21.10　最终用户许可协议对话框

(5) 选中“我接受《许可协议》中的条款(A)”复选框，然后单击“下一步”按钮，打开如图 21.11 所示的安装类型对话框。

(6) 在安装类型对话框中，用户需要选择安装的类型，因为 office 是一个套件，里面包含了如 Word、Access、Excel 等多种软件，如果每种都需要，则选择“典型安装”（本例选择典型安装），否则选择“自定义”安装。另外，还需要设置安装路径，单击“浏览”按钮，从打开的对话框中选择安装文件的路径。

(7) 单击“下一步”按钮，打开如图 21.12 所示的安装任务对话框。

(8) 单击“安装”按钮，打开如图 21.13 所示的安装进度对话框。

(9) 安装进程结束后，将自动打开如图 21.14 所示的安装成功对话框。单击“完成”按钮，至此 Access 2003 安装完成，用户可以在开始菜单中找到并启动 Access 2003。

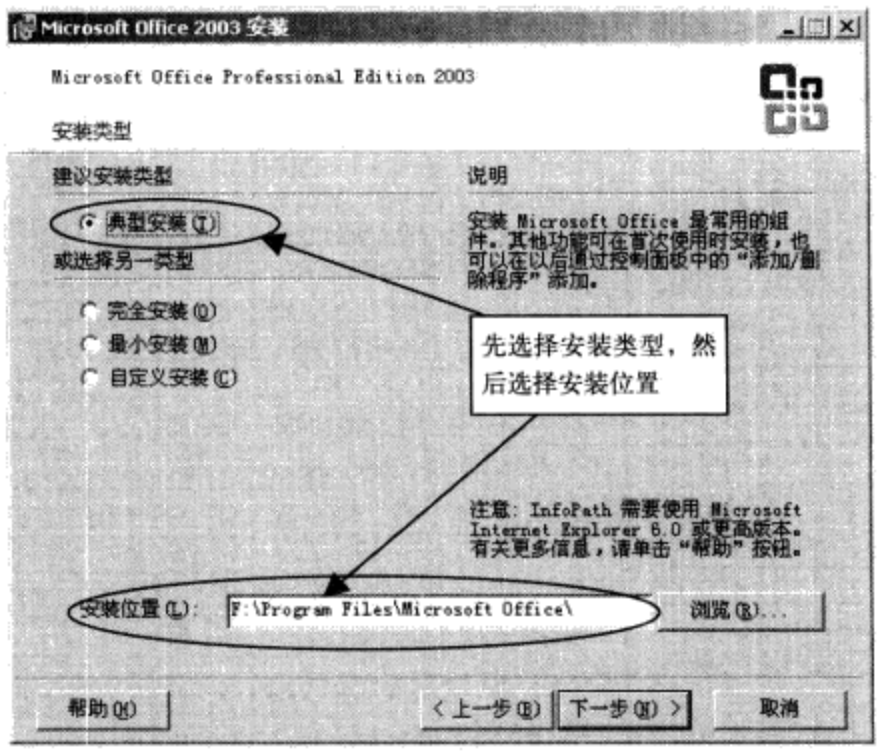


图 21.11　安装类型对话框

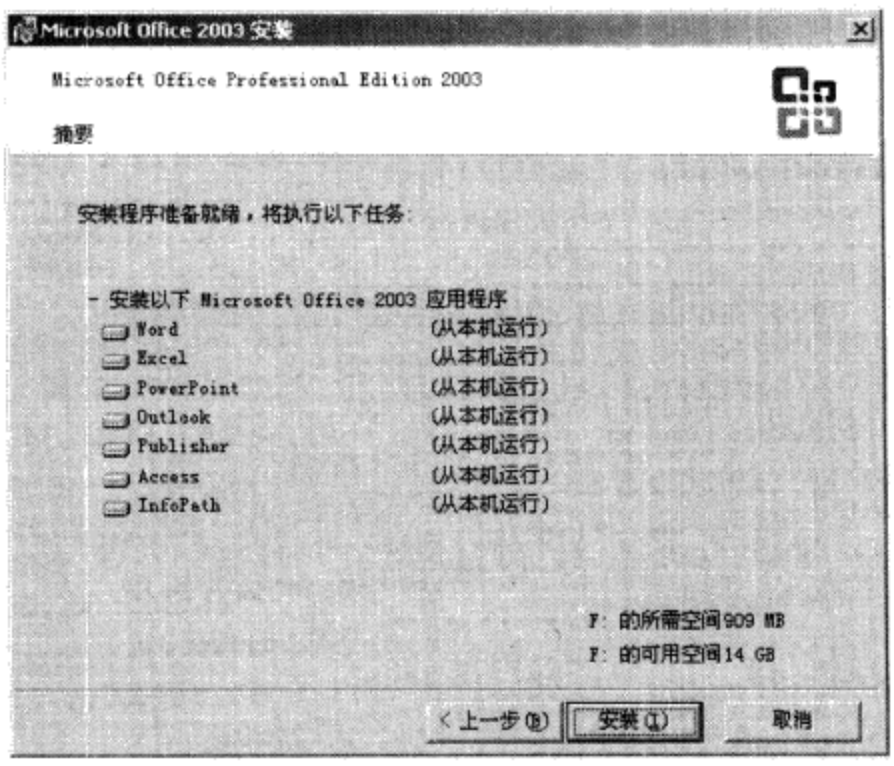


图 21.12　安装任务对话框

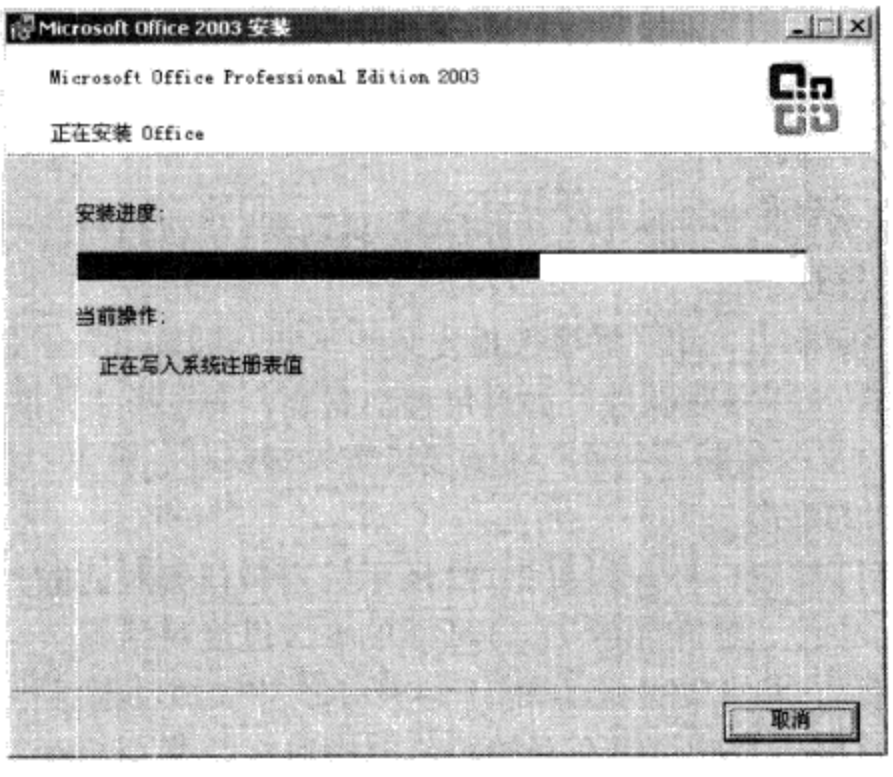


图 21.13　安装进度对话框

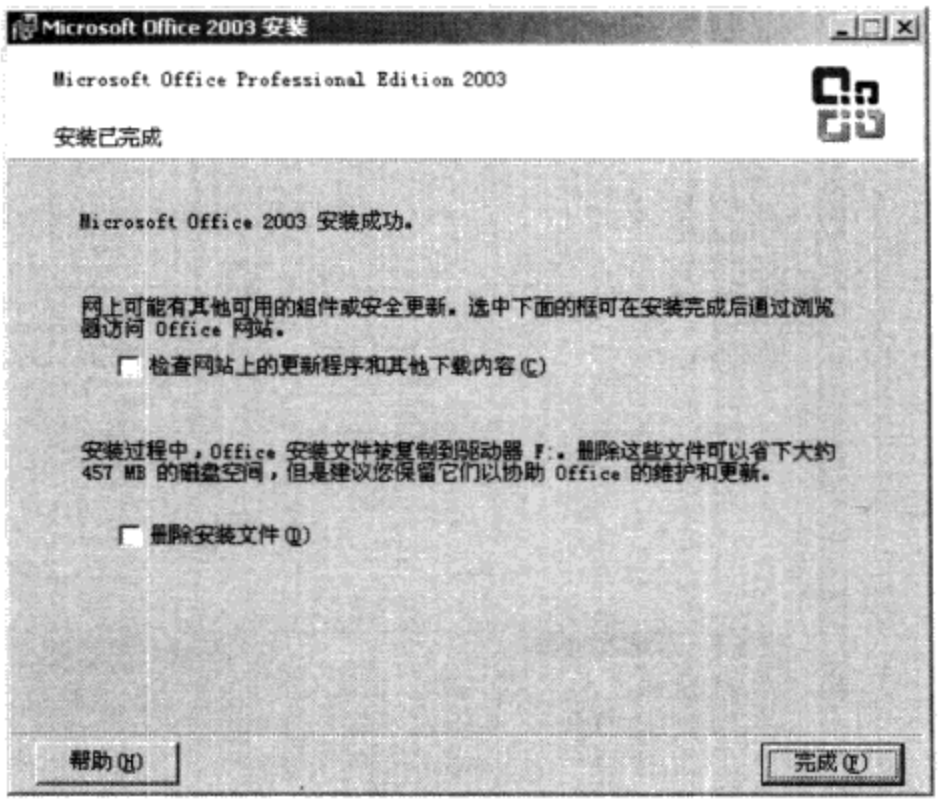


图 21.14　安装成功对话框

21.1.2　创建 Access 数据库

创建 Access 数据库有两种方法：使用向导和用户自定义。下面分别介绍这两种创建数据库的方法（数据库的创建过程以 Access 2000 版本为例）。

1．使用数据库向导创建数据库

（1）在启动 Access 时，系统会打开如图 21.15 所示的对话框，选择“新建数据库”中的“Access 数据库向导、数据页和项目（A）”选项，然后单击“确定”按钮，弹出如图 21.16 所示的新建数据库对话框。

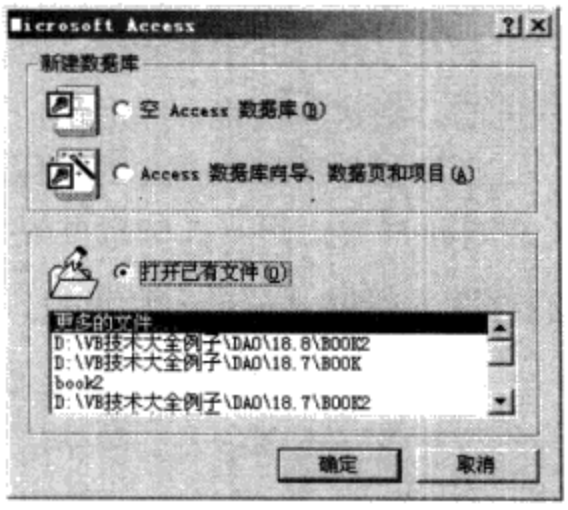


图 21.15　数据库对话框

如果已经打开了数据库，或者在 Access 启动时的对话框已经关闭，可以单击常用工具栏上的【新建】按钮，在弹出的“新建”对话框中选择“数据库”选项卡。

（2）在“数据库”选项卡上，双击要创建的数据库类型的图标。

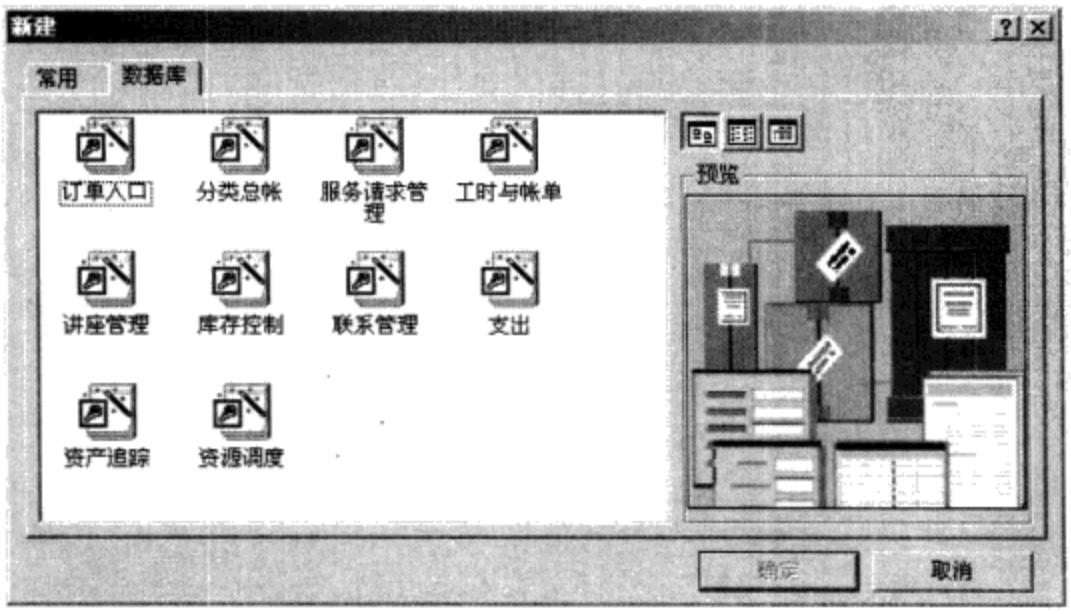


图 21.16　新建数据库对话框

（3）这时系统将打开如图 21.17 所示的文件新建数据库对话框，用户可以指定数据库的名

称和位置，也可以采用系统默认的名称和位置。

图 21.17　文件新建数据库对话框

(4) 单击“创建”按钮，通过“数据库向导”建立新的数据库即可完成。

2．自定义创建数据库

(1) 在第一次启动 Access 时，系统将打开一个对话框，在“新建数据库”中，选择“空 Access 数据库”选项，然后单击“确定”按钮。

如果已经打开了数据库，或者当 Access 打开时显示的对话框已经关闭，可以单击常用工具栏上的“新建”按钮，然后双击“常用”选项卡的“空数据库”图标。

(2) 指定新建数据库的名称及位置，并单击“创建”按钮，将建立新的数据库。

注意：用户在给新建的数据库命名时，系统会自动在该名称后加上“.MDB”后缀。

21.1.3　创建 Access 数据表

使用设计器创建 Access 数据表的步骤如下(数据表的创建过程以 Access 2000 版本为例)。

(1) 打开要创建表的数据库，出现该数据库的窗体，在“使用设计器”图标上单击鼠标右键，弹出如图 21.18 所示的窗口。

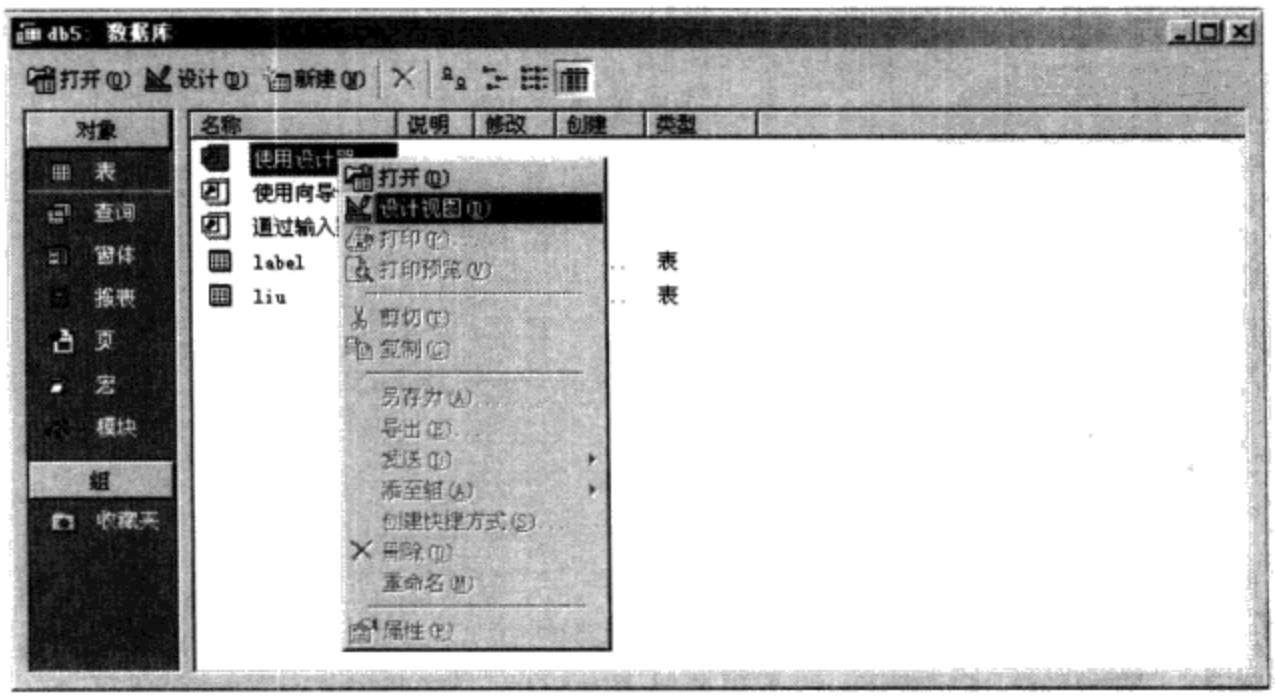


图 21.18　数据库窗体

(2) 在快捷菜单中选择“设计视图”，在表设计器窗口中输入字段名称、选择数据类型和输

入相应的字段说明。表设计器窗口如图 21.19 所示。

(3) 关闭表设计器窗口时，将出现是否保存的信息提示框，单击“是”按钮，弹出如图 21.20 所示的另存为对话框。

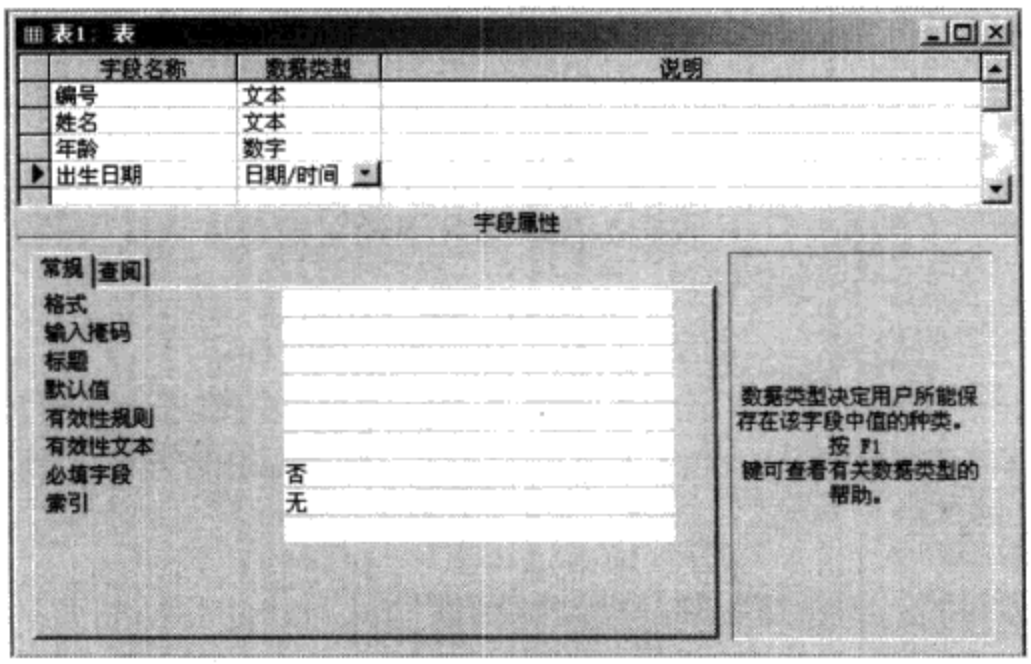


图 21.19 表设计器

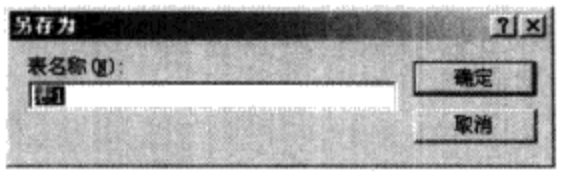


图 21.20 “另存为”对话框

(4) 输入表的名称，单击“确定”按钮，新建的表就可以出现在数据库设计器中了。

21.2 SQL Server 数据库

21.2.1 安装 SQL Server

1．安装 SQL Server 2000

SQL Server 2000 有多个版本，由于篇幅关系，本书不能详细介绍每个版本的安装过程，下面仅以安装“SQL Server 2000 个人版”为例介绍安装 SQL Server 2000 的方法。

(1) 将 SQL Server 2000 个人版安装光盘放入光驱中，系统自动启动安装程序，如图 21.21 所示。

图 21.21 启动对话框

注意：如果系统禁用了光驱的自动运行功能，使用“我的电脑”或者 Windows 资源管理器，浏览光盘内容，双击 Autorun 文件启动安装程序。

（2）单击“安装 SQL Server 2000 组件”选项，打开“安装组件”对话框。

（3）在该对话框中单击“安装数据库服务器” 选项，进入“欢迎”对话框（安装向导欢迎界面）。

（4）单击“下一步”按钮，打开“计算机名”对话框。在“计算机名”对话框中选择“本地计算机”选项。

（5）单击“下一步”按钮，打开“安装选择”对话框，在“安装选择”窗口中选择安装方式。这里选择“创建新的 SQL Server 实例，或安装客户端工具”选项。

（6）单击“下一步”按钮，打开“用户信息”对话框，输入姓名和公司名称。

（7）单击“下一步”按钮，打开“软件许可证协议”对话框。选择“是”按钮进入“安装定义”对话框。在该对话框中提供了 3 种安装类型：“仅客户端工具”、“服务器和客户端工具”和“仅连接”，如图 21.22 所示。

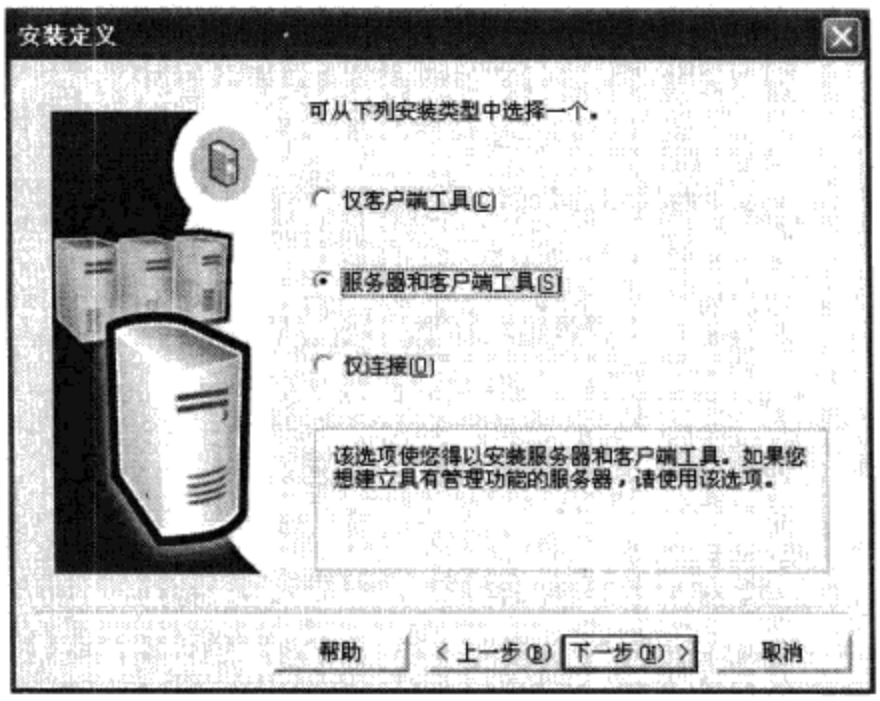


图 21.22 “安装定义”对话框

（8）选择“服务器和客户端工具”选项，单击“下一步”按钮，打开“实例名”对话框。在“实例名”窗口中设置安装 SQL Server 的实例名称。用户可以使用默认的实例名，即选中“默认”复选框即可。如果用户要自行设置实例名，首先取消“默认”复选框，然后在“实例名”文本框中输入相应的实例名。

（9）选择默认的实例名。单击“下一步”按钮，打开“安装类型”对话框。这里选择“典型”选项，即进行典型安装，如果用户要更改默认的安装路径，可以单击“浏览”按钮重新指定安装路径，这里选择默认设置，单击“下一步”按钮，打开“服务账户”窗口。

（10）选择“对每个服务使用同一账户。自动启动 SQL Server 服务”，在“服务设置”中选择“使用本地系统账户”选项。

（11） 单击“下一步”按钮，进入“身份验证模式”对话框，如图 21.23 所示。

SQL Server 2000 提供了两种身份验证模式：Windows 身份验证模式和混合模式。如果选择 Windows 身份验证模式，SQL Server 2000 使用 Windows 操作系统中的信息验证账户名和密码；选择混合模式，SQL Server 2000 允许用户使用 Windows 身份验证或 SQL Server 身份验证进行连接。

（12）选择“混合模式（Windows 身份验证和 SQL Server 身份验证）”选项，并设置 sa（默

认系统管理员账户）登录密码，这里选择空密码，如图 21.23 所示。单击“下一步”按钮，打开“开始复制文件”对话框。

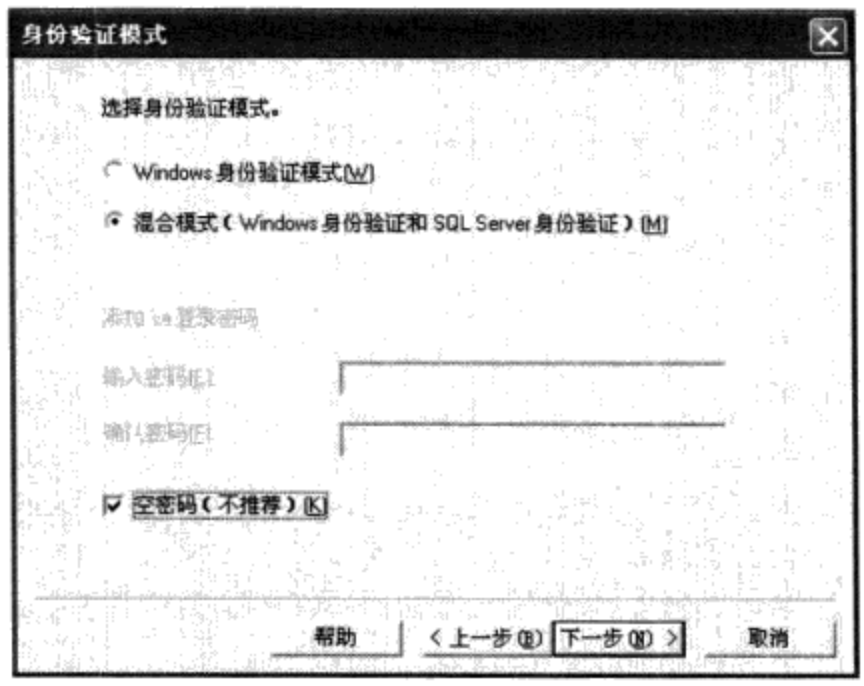


图 21.23 “身份验证模式”对话框

（13）单击“下一步”按钮，启动安装进程，开始复制文件，安装成功后，系统将弹出“安装完毕”窗口。单击“完成”按钮，完成 SQL Server 2000 的安装。

安装完毕后，重新启动计算机，系统的任务栏右下角会出现 SQL Server 2000 服务图标，表示 SQL Server 2000 服务已经启动。

2．安装 SQL Server 2005

下面以安装 SQL Server 2005 企业版为例介绍如何安装 SQL Server 2005，具体步骤如下。

（1）将 SQL Server 2005 安装盘放入光驱内，光驱将会自动运行，并打开开始界面。

（2）在开始界面中选择“服务器组件、工具、联机丛书和示例（C）”命令，执行安装程序，打开“最终用户许可协议”窗口，如图 21.24 所示。

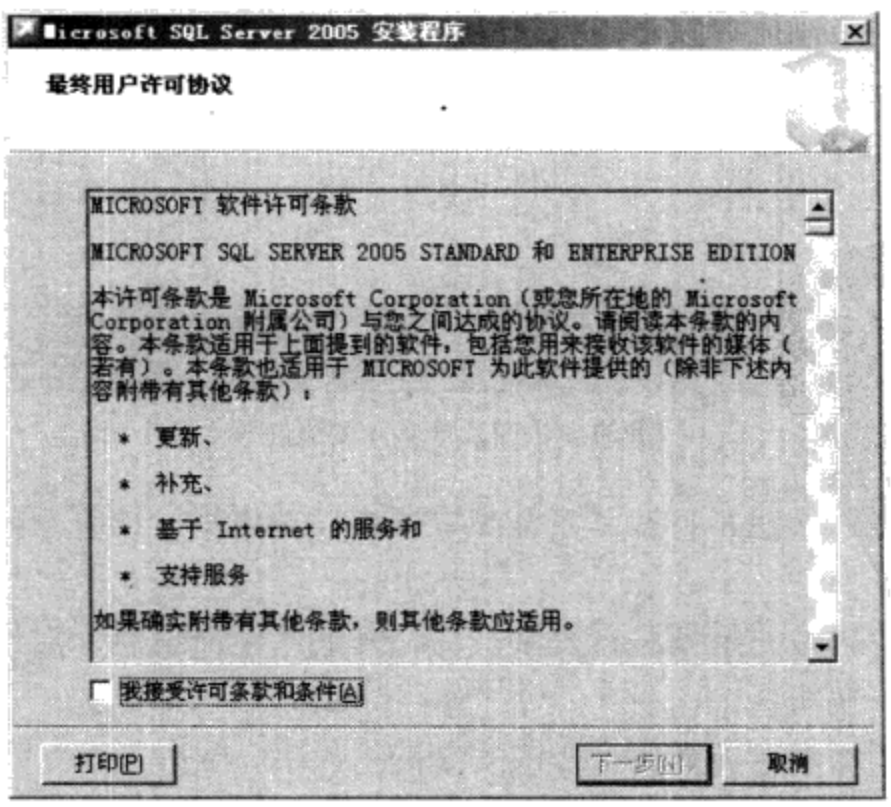


图 21.24 “最终用户许可协议”对话框

（3）在“最终用户许可协议”对话框中选择“我接受许可条款和条件（A）”复选框，单击“下一步”按钮，打开“安装必备组件”对话框，如图 21.25 所示。

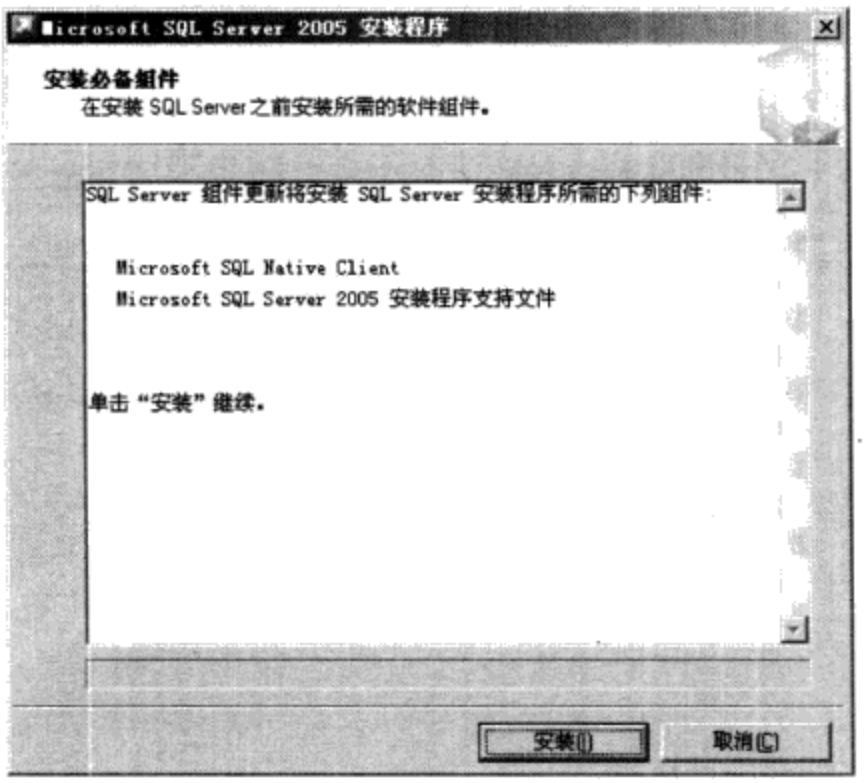


图 21.25 “安装必备组件”对话框

(4) 单击“安装”按钮，自动安装 SQL Server 2005 必备组件。单击“下一步”按钮，开始 SQL Server 2005 安装向导，如图 21.26 所示。

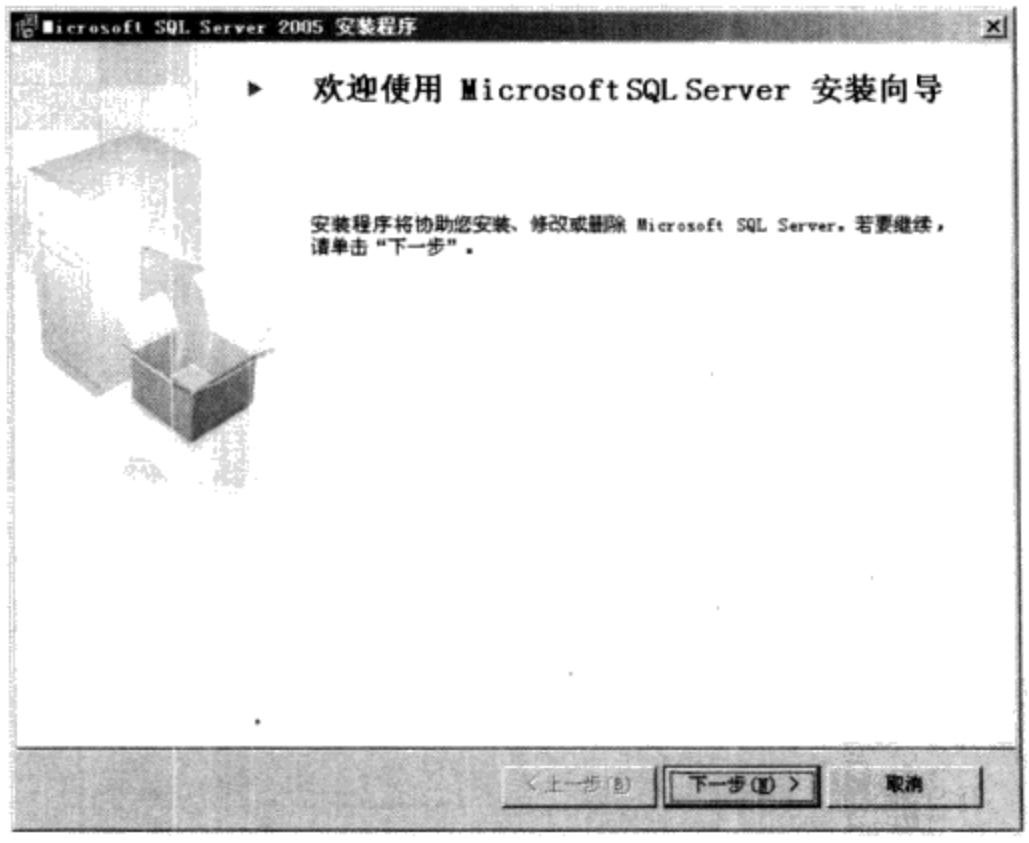


图 21.26 SQL Server 安装向导

(5) 单击“下一步”按钮，打开“系统配置检查”对话框，检查系统中是否存在潜在的安装问题。

(6) 单击“下一步”按钮继续进行安装。在打开的“注册信息”对话框中，输入相关注册信息，单击“下一步”按钮。

(7) 在打开的对话框中选择需要升级或安装的组件后，单击“下一步”按钮继续进行安装。

(8) 在“实例名”对话框中选择实例的命名方式（这里选择“默认实例”单选框），单击“下一步”按钮继续进行安装，如图 21.27 所示。

(9) 在“服务账户”对话框中选择登录时使用的账户。选择“为每个服务账户进行自定义”复选框，可以对不同的服务定义使用不同的账户。服务账户选项包括了“使用内置系统账户”

（本地系统账户、网络服务账户或本地服务账户）和“使用域用户账户”（一个使用 Windows 身份验证的域用户账户）。选择相应的 SQL Server 服务器安装结束启动服务，单击“下一步”按钮继续进行安装，如图 21.28 所示。

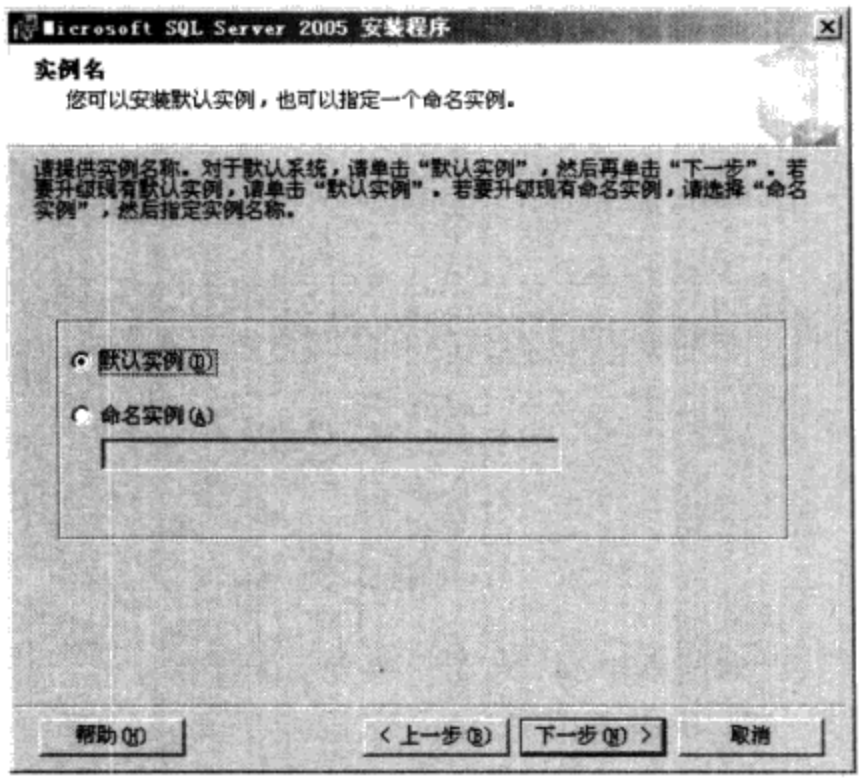


图 21.27 “实例名”对话框

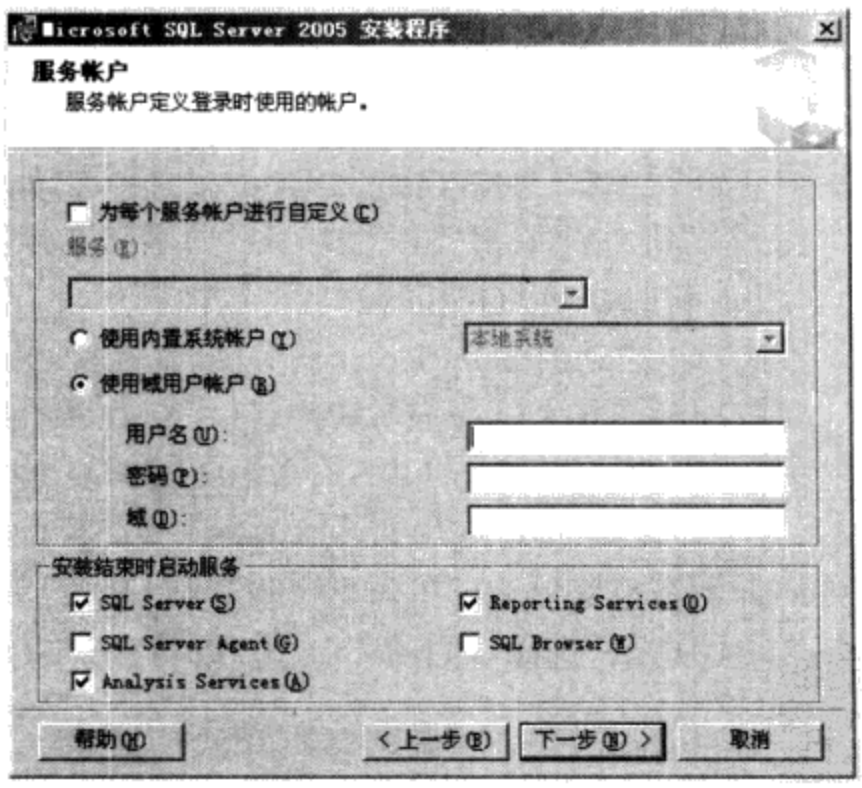


图 21.28 “服务账户” 对话框

(10) 在“身份验证模式”对话框中选择客户端和服务器连接时身份验证时所用的安全模式。如果选择“混合模式”，则可以输入并确认 SQL Server 系统管理员（sa）密码。选择相应的验证方式后，单击“下一步”按钮继续进行安装，如图 21.29 所示。

(11)“排序规则设置”对话框用于设置默认排序规则。选择排序规则后，单击“下一步”按钮继续进行安装。

(12) 在“报表服务器安装选项”对话框中选择报表服务器安装设置。单击“下一步”按钮继续进行安装。

(13) 在“错误和使用情况报告设置”对话框中，选择传输报告的内容，单击“下一步”按钮继续进行安装。

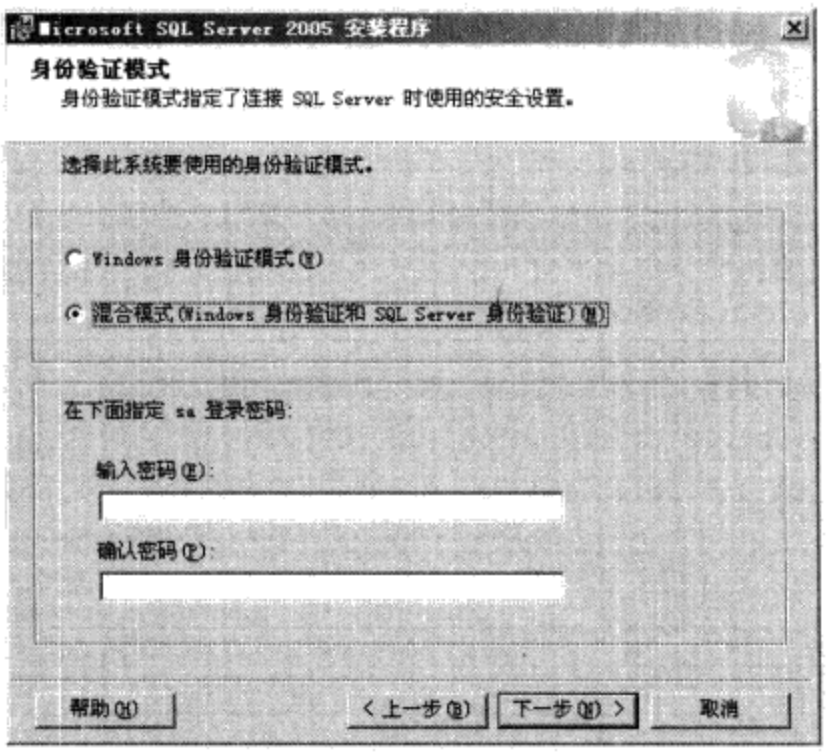


图 21.29 “身份验证模式”对话框

（14）单击“准备安装”对话框中的“安装”按钮开始安装。

（15）当“安装进度”对话框中所用产品安装完成后，单击“下一步”按钮，打开“完成 Microsoft SQL Server 2005 安装”窗口，单击“完成”按钮即可完成 SQL Server 2005 的安装。

21.2.2 启动 SQL Server 服务管理器

启动 SQL Server 服务管理器有如下 3 种方法。

（1）依次打开“开始→所有程序→Microsoft SQL Server”中选择“服务管理器”，打开 SQL Server 服务管理器，单击“开始／继续”按钮启动 SQL Server 服务，如图 21.30 所示。

（2）在系统任务栏的右下角右键单击 SQL Server 服务图标，在快捷菜单中选择“MSSQLServer-启动”的命令，可以启动 SQL Server 服务管理器。

（3）在企业管理器中，选择待启动的 SQL Server 服务的服务器，单击鼠标右键，在快捷菜单中选择“启动”命令，可以启动 SQL Server 服务管理器。

注意：如果用户安装了 SQL Server 数据库，计算机启动后会自动启动 SQL Server 服务管理器。

例如，当前的 SQL Server 服务已经停止，在企业管理器中，选择服务器名 ly，单击鼠标右键，打开快捷菜单，如图 21.31 所示，选择“启动”命令，即可启动 SQL Server 服务。

图 21.30 SQL Server 服务管理器

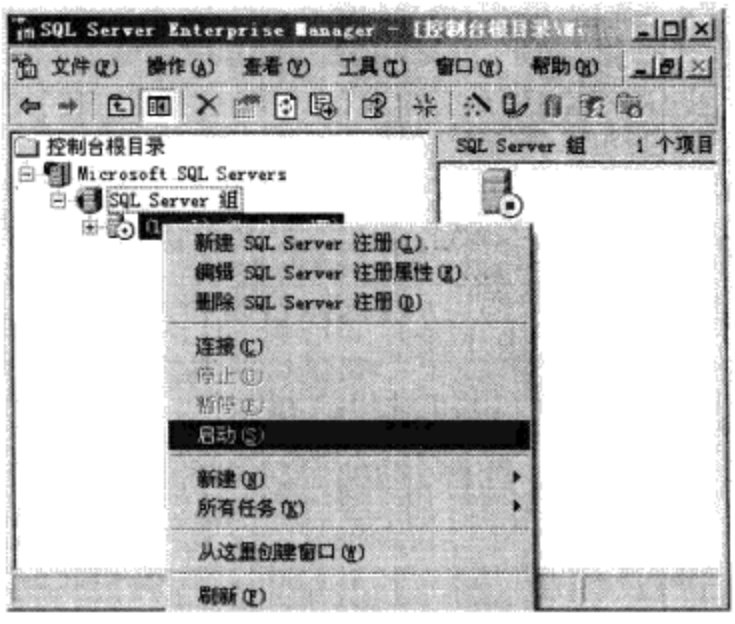


图 21.31 企业管理器中启动服务器对话框

21.2.3 创建 SQL Server 数据库

在 SQL Server 2000 中，创建数据库有多种方法。这里只介绍使用企业管理器创建数据库，这种方法简单易学，应用比较广泛。

在 SQL Server 2000 的企业管理器中，可以将所有步骤涉及的窗口聚集在一个对话框中，在这个对话框中就可以设置数据库的所有属性，创建数据库。下面以创建 mrsoft 数据库为例介绍这种方法。

(1) 在企业管理器中，选择要在其中创建数据库的 SQL Server 服务器，展开目录，选择“数据库”选项，单击鼠标右键打开快捷菜单，如图 21.32 所示。

(2) 选择“新建数据库”命令，打开“数据库属性”对话框，如图 21.33 所示，在该对话框中设置待创建数据库的属性。一共有 3 个选项卡：“常规”、“数据文件”和“事务日志”。

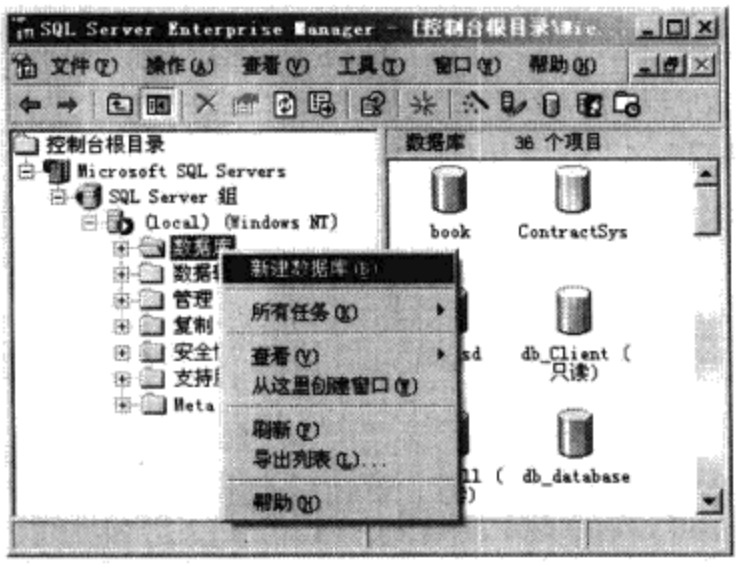


图 21.32 企业管理器中创建数据库的对话框

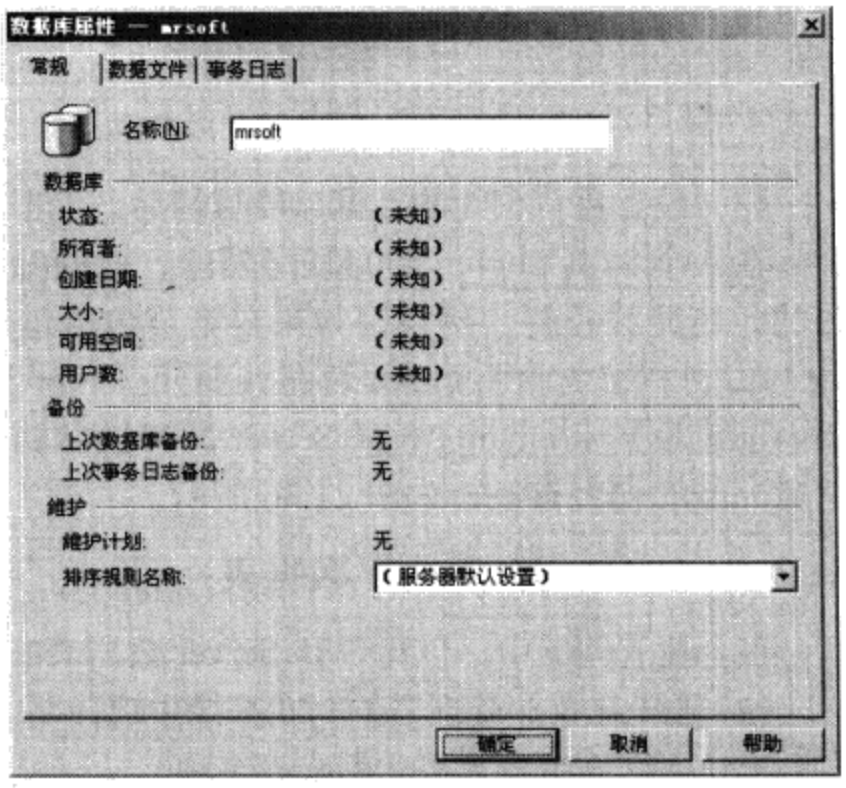


图 21.33 数据库属性对话框

“常规”选项卡主要用来设置数据库的名称，这也是在对话框中设置待创建数据库属性的第一步。本例中，在文本框中输入 mrsoft。

“数据文件”选项卡用来设置数据文件的属性。在“常规”选项卡中设置了数据库的名称 mrsoft，SQL Server 2000 系统会默认产生数据文件 mrsoft_Data.MDF，该选项卡中显示了数据库和文件属性的默认设置，可以自行修改这些默认设置。

注意：可以在此选项卡中添加数据文件。这个数据文件是次数据文件，因为一个数据库有且仅有一个主数据文件。单击“位置”列的按钮，可以设置数据文件的存放位置，如图 21.34 所示。系统会自动显示该次数据文件名及其默认存放位置，文件名是以.mdf 为扩展名的。

“事务日志”选项卡用来设置日志文件的属性。如图 21.35 所示，在“常规”选项卡中设置了数据库的名称 mrsoft，SQL Server 2000 系统会默认产生日志文件 mrsoft_Log.LDF，“事务日志”选择卡中显示了日志文件属性的默认设置。可以自行修改这些默认设置，也可以添加多个日志文件。

图 21.34　存放次数据文件的对话框

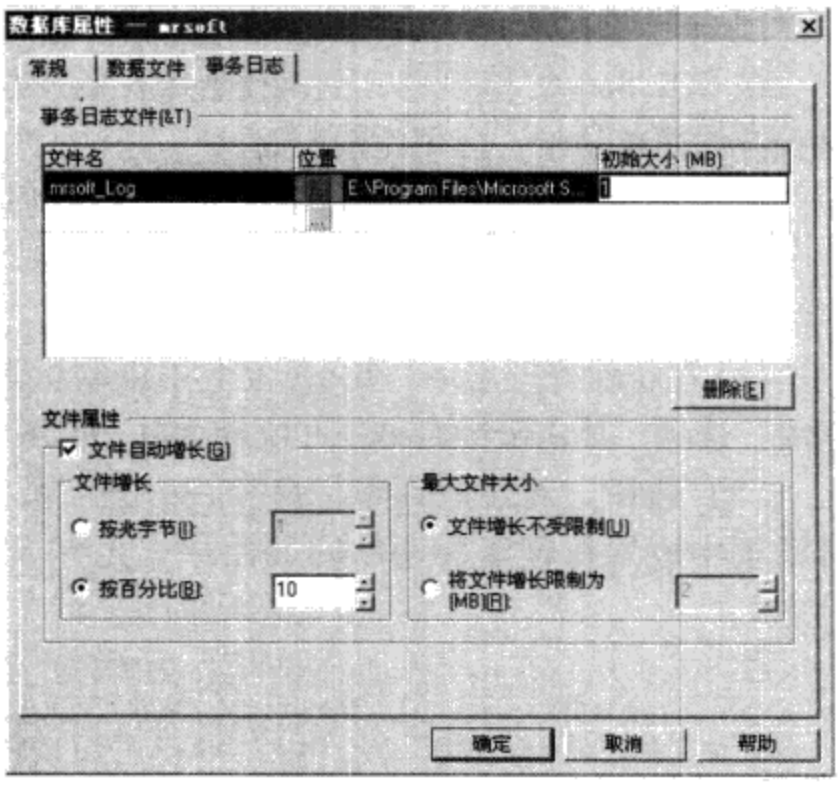


图 21.35　新建数据库的事务日志选项卡

(3) 设置完待创建的数据库的属性后，单击“确定”按钮，创建数据库。

在一个 SQL Server 2000 服务器中，数据库名称是惟一的，不允许重复。例如，如果在上节中创建了数据库 mrsoft，现利用企业管理器要再在该 SQL Server 2000 服务器中创建数据库 mrsoft，SQL Server 2000 会报错，提示数据库 mrsoft 已经存在，如图 21.36 所示。

图 21.36　数据库重名的错误信息提示框

21.2.4　创建 SQL Server 数据表

在企业管理器中，创建 SQL Server 数据表的步骤如下。

(1) 首先在企业管理器的树状目录中选择要创建数据表的数据库，单击该数据库下边的表，右侧窗口出现系统表名称和用户表名称。

(2) 在右侧窗口上空白处单击鼠标右键，出现如图 21.37 所示的窗口。

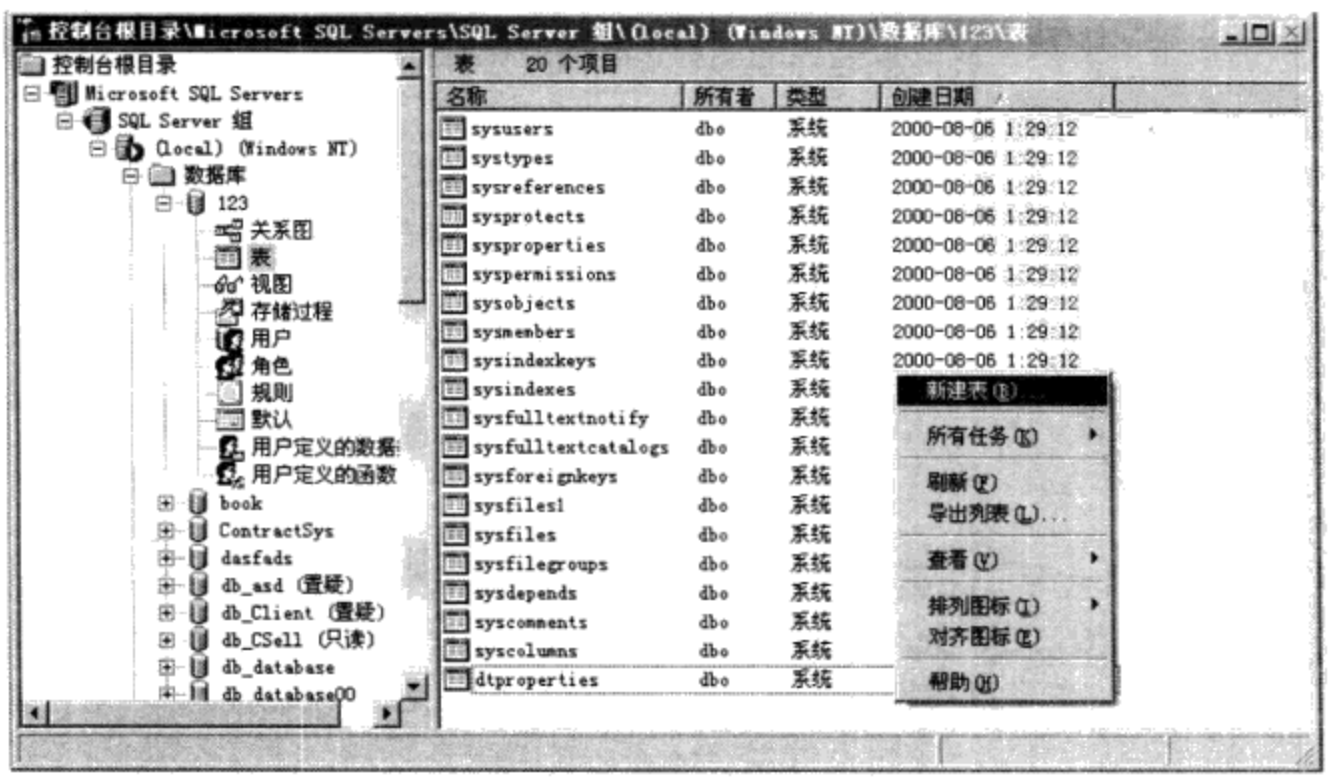


图 21.37　企业管理器窗口

（3）在弹出的快捷菜单中选择“新建表”命令，显示创建表结构的设计器，输入字段名、选择字段类型，输入字段最大的长度，选择是否为空。新建表的设计器如图 21.38 所示。

（4）输入完所有的字段后，关闭设计器，弹出是否保存信息提示框，单击“是”按钮，出现如图 21.39 所示对话框，输入数据表的名称，单击“确定”按钮，新建的数据表将出现在企业管理器的右侧窗口中。

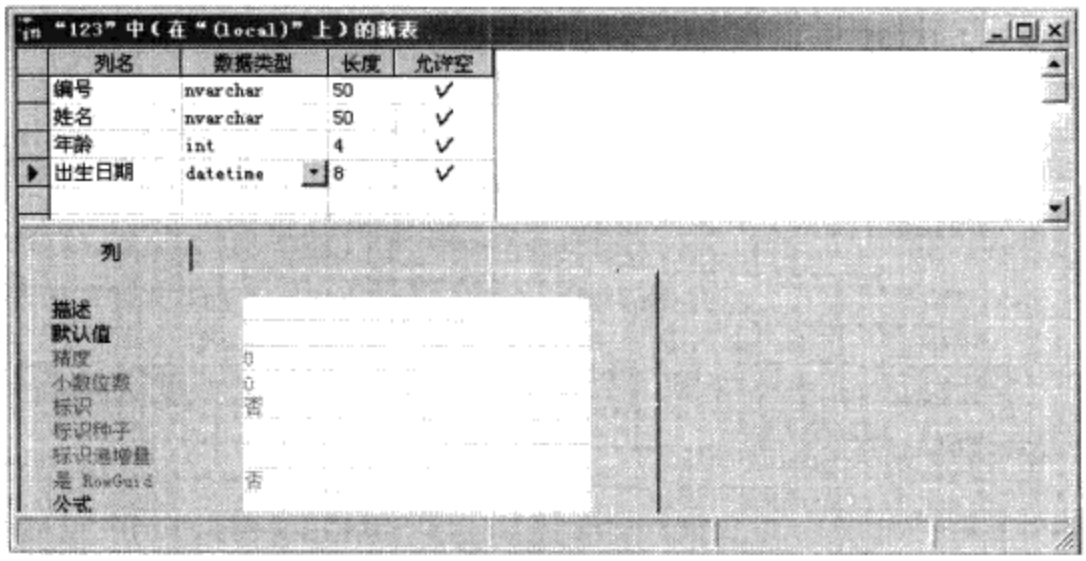


图 21.38　表设计器

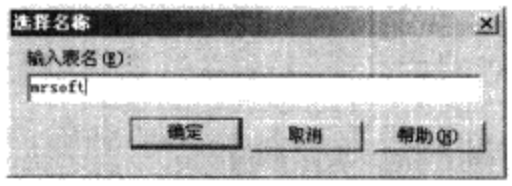


图 21.39　输入表名称对话框

21.3　配置 ODBC 数据源

为了使客户应用程序能够方便地使用 ODBC，必须在客户端配置 ODBC 数据源。ODBC 数据源又叫 DSN，它把客户应用程序所要使用的驱动程序、数据库、用户名和口令等信息组合起来，供客户端程序使用。

ODBC 数据源有以下 3 种类型。

用户 DSN：只能由配置该 DSN 的用户使用或只能在当前的计算机上使用。

系统 DSN：可以被任何使用用户计算机的人使用。另外，如果用户要建立 Web 数据库应用程序，应使用此数据源。

文件 DSN：除了能够被用户在其他计算机上使用之外，其他与系统 DSN 相似。

21.3.1　配置 SQL Server 的 ODBC 数据源

配置 ODBC 数据源，可以利用 Windows 控制面板中的 ODBC 数据源管理器来完成，添加数据源的具体步骤如下。

（1）单击“控制面板”→“管理工具”，打开 ODBC 数据库源管理器。

（2）切换到“用户 DSN”选项卡，单击“添加”按钮，选择 ODBC 驱动程序 SQL Server，如图 21.40 所示。

（3）单击“完成”按钮，进入添加数据来源的设置，如图 21.41 所示。

在这个窗口中，必须输入“数据源名称”，这个名称是自定义的，用户可以根据实际情况来设置这个名称，但是需要注意的是这个名称应与程序中使用的名称一致。另外，在“描述”文本框中，可以对这个 ODBC 数据来源做一点描述，以便日后维护。接下来应在“服务器”下拉列表框中选择或输入 SQL 服务器名称，然后单击“下一步”按钮。

（4）在图 21.42 所示的窗口中输入用户名称和密码。这里有两个选项，一个是使用 Windows NT 认证，另一个是使用 SQL 服务器上设置的账号作为认证，采用哪种验证方式，由系统设置而定，通常使用 SQL 服务器的认证。

图 21.40　创建新数据源

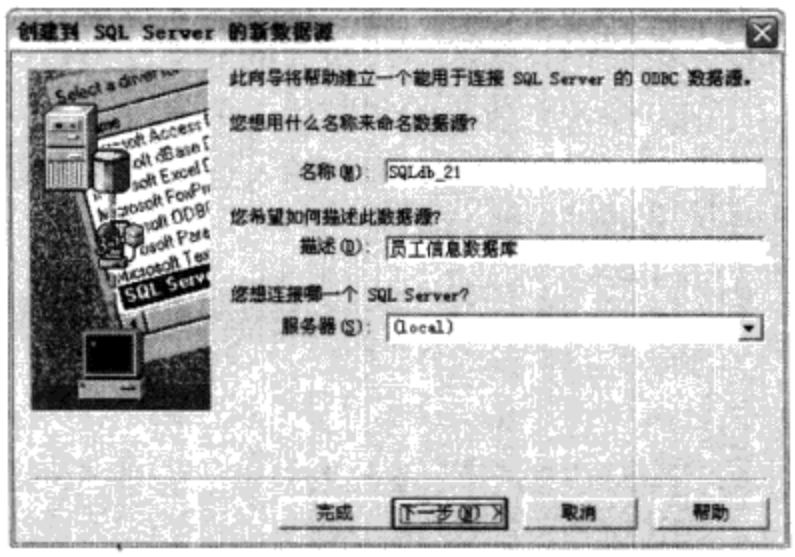


图 21.41　创建到 SQL Server 的新数据源

（5）输入完成后，单击“下一步”按钮，进入图 21.43 所示的窗口，在这里可以选择需要使用的数据库名称，或直接使用系统默认的数据库。

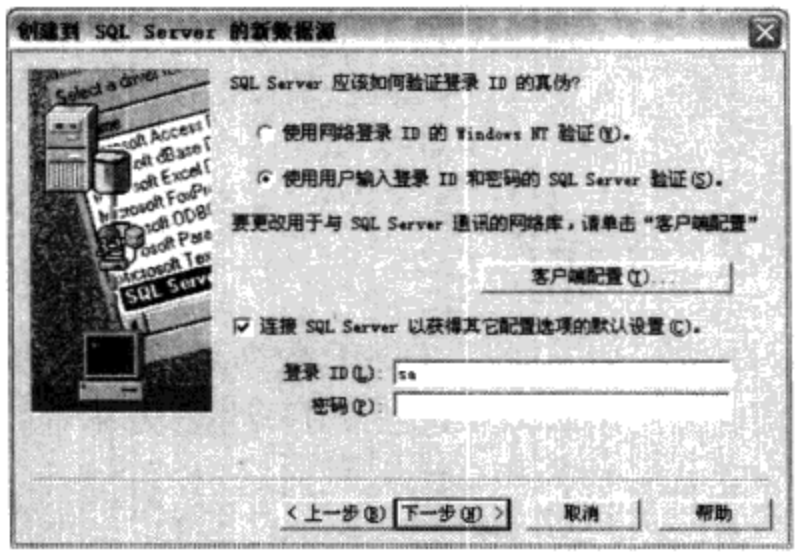


图 21.42　SQL 服务器登录设置

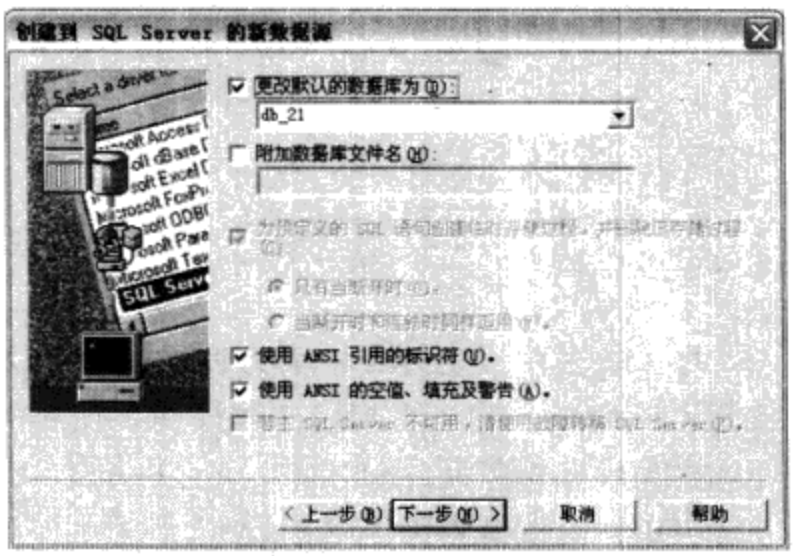


图 21.43　选择数据库名称

（6）接下来将显示图 21.44 所示的窗口，在此窗口只需单击“下一步”按钮即可，不需要做任何的改变。

（7）单击“完成”按钮，出现图 21.45 所示的窗口。如果要测试数据源连接是否成功，可以单击“测试数据源”按钮，如果数据源连接成功，将弹出设置成功的画面。

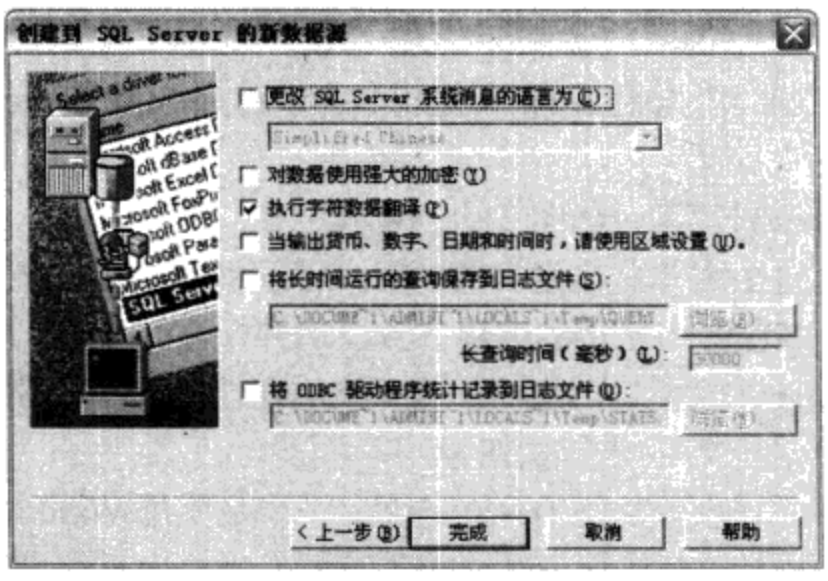


图 21.44　语言及 LOG 路径设置

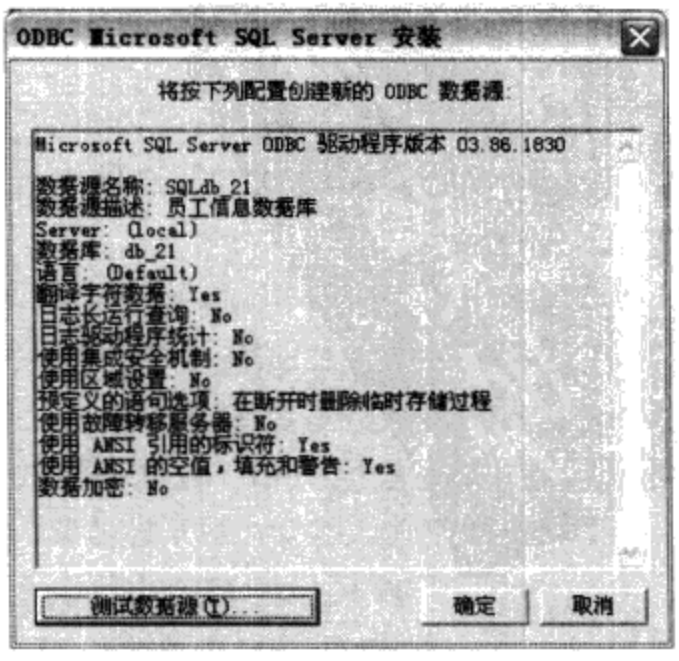


图 21.45　ODBC 设置完成

21.3.2 配置 Access 的 ODBC 数据源

添加 Access 数据库数据源的具体步骤如下。

（1）单击“控制面板”→“管理工具”，打开 ODBC 数据库源管理器。

（2）切换到“用户 DSN”选项卡，单击“添加”按钮，选择 ODBC 驱动程序 SQL Server，如图 21.46 所示。

（3）单击“完成”按钮，进入添加数据来源的设置，如图 21.47 所示。

图 21.46 创建新数据源

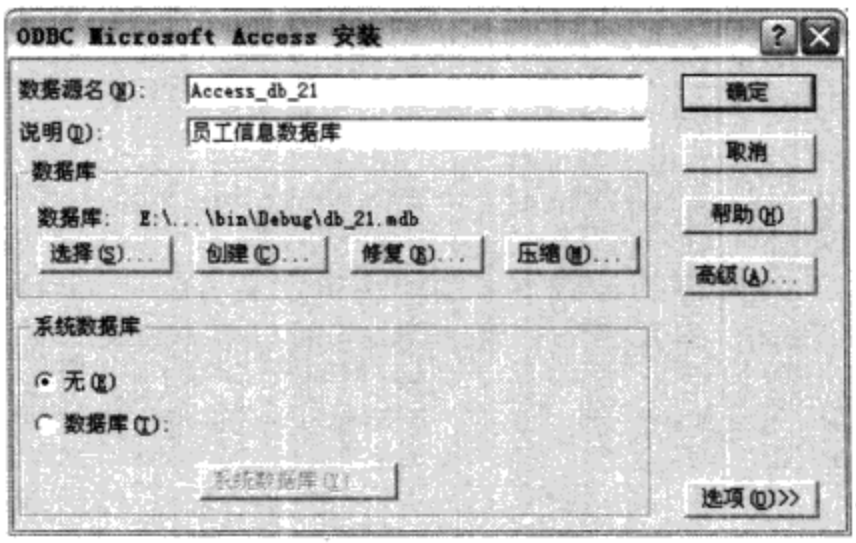


图 21.47 创建到 Access 的新数据源

在这个窗口中，必须输入“数据源名称”，这个名称是自定义的，用户可以根据实际情况来设置这个名称，但是需要注意的是这个名称应与程序中使用的名称一致。另外，在“描述”文本框中，可以对 ODBC 数据源进行描述，以便日后维护。单击“选择”按钮，在弹出的“选择数据库”对话框中选择要设置数据库源的 Access 数据库。最后单击“确定”按钮，完成数据源的配置。

说明：Excel 数据库的数据源配置与 Access 数据库基本相同，在这里不做讲解。

21.4 数据库的连接

在开发应用程序时连接数据库是必不可少的，本节主要讲解 Access 数据库、SQL Server 数据库和 Excel 数据库的连接，这 3 种数据库都可以使用 OLE DB、ODBC DSN 和 ADO 进行连接，唯一不同的是连接数据库的连接字符串。下面分别用这 3 种方法对 Access、SQL Server 和 Excel 数据库的连接进行讲解。

21.4.1 连接 Access 数据库

连接 Access 数据库操作最重要的部分是连接数据库的连接字符串。代码如下：

```
string ConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data source=PathofAccessDataBase;"
```

下面对连接字符串中的各属性进行说明。

（1）Provider 属性指定使用的数据库引擎为 Microsoft.Jet.OLEDB.4.0。

（2）Data source 属性指定数据库文件位于计算机中的物理位置，也就是 Access 数据库所在路径。

示例 21-01 连接 Access 数据库

主要用 OLE DB 连接 Access 数据库，并在 DataGridView 控件中显示数据库中指定数据表的内容。程序代码如下：

代码 21-01　光盘位置：光盘\mingrisoft\sl\21\21_01

自定义方法 AccessDataComponent 用于连接 Access 数据库，并显示数据库中指定数据表的内容。参数 strPath 表示数据库的所在路径，参数 tableN 表示要显示的表名。代码如下：

```
public void AccessDataComponent(string strPath,string tableN)
{
    string ConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data source=" + strPath;  //设置连接字符串
    OleDbConnection oleCon = new OleDbConnection(ConStr);                       //打开数据源的连接
    OleDbDataAdapter oleDap = new OleDbDataAdapter("select * from " + tableN, oleCon);   //连接数据库
    DataSet ds = new DataSet();                                                 //实例化 DataSet 对象
    oleDap.Fill(ds, tableN);                                                    //将数据添加到 DataSet 对象中
    this.dataGridView1.DataSource = ds.Tables[0].DefaultView;                   //显示数据表中的数据
    ds.Dispose();                                                               //释放对象中的资源
    oleDap.Dispose();                                                           //释放对象中的资源
    oleCon.Dispose();                                                           //释放对象中的资源
}
```

在窗体的加载事件中调用自定义方法 AccessDataComponent。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    AccessDataComponent(Application.StartupPath + "\\db_21.mdb", "tb_EmployeeInfo");
}
```

按<Ctrl+F5>键运行程序，运行结果如图 21.48 所示。

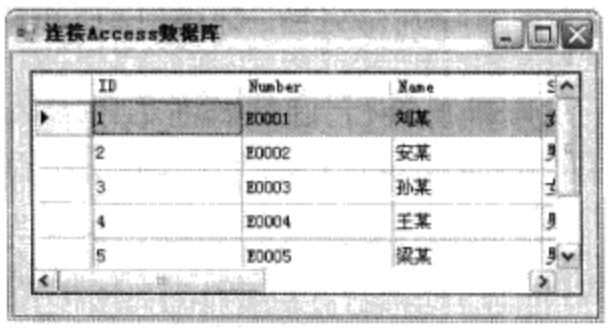


图 21.48　连接 Access 数据库

注意：在.NET 中应用 OLEDB 数据提供程序连接数据库时，需要引用 using System.Data.OleDb 命名空间。

21.4.2　连接 SQL Server 数据库

1．连接 SQL Server 数据库

连接 SQL Server 数据库操作最重要的部分是连接数据库的连接字符串。代码如下：

```
string ConStr = "driver=SQL Server;server=.;UID=sa;PWD=;database=databaseName;"
```

下面对连接字符串中的各属性进行说明。

（1）Driver 属性为数据库类型。

（2）UID 属性为用户身份。

（3）PWD 属性为密码。

（4）SERVER 属性为服务器。

（5）DATABASE 属性为数据库。

说明：在“示例 21-01 连接 Access 数据库”中，只要将 Access 连接数据库的连接字符串改成 SQL Server 的连接字符串，就可以完成 SQL Server 的连接。

交叉链接：关于 ADO.NET 中 SqlConnection 对象连接 SQL Server 数据库，请参见本书 22.2.2 节。

2．用 ODBC 连接 SQL Server 数据库

当开发者用 ODBC 数据源连接数据库时，用 System.Data.Odbc 命名空间下的对象来实现，

连接数据库的连接字符串代码如下：

```
string ConODBCStr ="DSN=ODBCName";uid=sa;pwd="
```

下面对连接字符串中的各属性进行说明。

（1）DSN 属性为数据源名称。

（2）UID 属性为用户身份。

（3）PWD 属性为密码。

示例 21-02　用 ODBC 连接 SQL Server 数据库

示例主要用 ODBC DSN 连接 SQL Server 数据库，并在 DataGridView 控件中显示数据库中指定数据表的内容。程序代码如下：

代码 21-02　光盘位置：光盘\mingrisoft\sl\21\21_02

自定义方法 SQLServerDataODBCConnect 用于连接 ODBC 数据源，并显示数据源中指定数据表的内容。参数 ODBCName 表示数据源名称，参数 TableN 表示数据表的名称。代码如下：

```
public void SQLServerDataODBCConnect(string ODBCName,string TableN)
{
    //连接 ODBC 数据源
    OdbcConnection odbcCon = new OdbcConnection("DSN=" + ODBCName + ";uid=sa;pwd=");
    try
    {
        //将命令集连接到数据源中
        OdbcDataAdapter odbcDat = new OdbcDataAdapter("SELECT * from " + TableN, odbcCon);
        DataTable dt = new DataTable(TableN);                    //实例化 DataTable 对象
        odbcDat.Fill(dt);                                        //将数据添加到 DataSet 对象中
        this.dataGridView1.DataSource = dt.DefaultView;          //显示数据表中的数据
        dt.Dispose();                                            //释放对象中的资源
        odbcDat.Dispose();                                       //释放对象中的资源
        odbcCon.Dispose();                                       //释放对象中的资源
    }
    catch (Exception ey)                                         //发生异常
    {
        MessageBox.Show(ey.Message);                             //提示异常信息
    }
}
```

在窗体的加载事件中调用自定义方法 SQLServerDataODBCConnect。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    SQLServerDataODBCConnect("SQLdb_21", "tb_EmployeeInfo");
}
```

按<Ctrl+F5>键运行程序，运行结果如图 21.48 所示。

注意：本示例不但可以连接 SQL Server 数据源，还可以连接 Access 和 Excel 数据源。

21.4.3　连接 Excel 数据库

连接 Excel 数据库操作最重要的部分是连接数据库的连接字符串。代码如下：

```
string ConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Persist Security Info=False;Data Source= PathofAccessDataBase;
Extended Properties=Excel 8.0"
```

下面对连接字符串中的各属性进行说明。

（1）Provider 属性指定使用的数据库引擎为 Microsoft.Jet.OLEDB.4.0。

（2）Persist Security Info 属性表示是否保存安全信息，默认值为 false。

（3）Data source 属性指定数据库文件位于计算机中的物理位置，也就是 Excel 数据库所在路径。

（4）Extended Properties 属性是一个扩展属性。

示例 21-03　连接 Excel 数据库

示例主要用 OLE DB 连接 Excel 数据库，并在 DataGridView 控件中显示数据库中指定数据表的内容。程序代码如下：

代码 21-03　光盘位置：光盘\mingrisoft\sl\21\21_03

自定义方法 ExcelDataConnect 用于连接 Excel 数据库，并显示数据库中指定数据表的内容。参数 ExcelPath 表示数据库的所在路径。代码如下：

```
public void ExcelDataConnect(string ExcelPath)
{
    try
    {
        string StrCon = @"Provider=Microsoft.Jet.OLEDB.4.0;Persist Security Info=False;Data Source=" +
ExcelPath + ";Extended Properties=Excel 8.0";                    //设置连接字符串
        OleDbConnection OleDB = new OleDbConnection(StrCon);     //打开数据源的连接
        //设置命令集连接到数据库中
        OleDbDataAdapter OleDat = new OleDbDataAdapter("select * from [Sheet1$]", OleDB);
        DataTable dt = new DataTable();                          //实例化 DataTable 对象
        OleDat.Fill(dt);                                         //将数据添加到 DataSet 对象中
        this.dataGridView1.DataSource = dt.DefaultView;          //显示数据表中的数据
        dt.Dispose();                                            //释放对象资源
        OleDat.Dispose();                                        //释放对象资源
        OleDB.Dispose();                                         //释放对象资源
    }
    catch (Exception ey)                                         //发生异常
    {
        MessageBox.Show(ey.Message);                             //提示异常信息
    }
}
```

在窗体的加载事件中调用自定义方法 ExcelDataConnect。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    ExcelDataConnect(Application.StartupPath + "\\db_21.xls");
}
```

按<Ctrl+F5>键运行程序，运行结果如图 21.48 所示。

21.5　数据库备份与恢复

备份数据库是指对数据库或事务日志进行复制，当系统、磁盘或数据库文件损坏时，可以使用备份文件进行恢复，防止数据丢失。

还原数据库是使用数据库的备份文件对数据库进行还原操作。由于病毒的破坏、磁盘损坏或操作员操作失误等原因会导致数据丢失、不完整或数据错误，此时，需要对数据库进行还原，将数据还原到某一天，前提是当天必须进行了数据备份。

本节主要学习数据库的备份、恢复操作。通过本节的学习，读者可以快速掌握备份和恢复 SQL Server 2000 数据库的方法。

21.5.1　数据库备份

数据库的备份可以用两种方法实现，一是通过数据库本身进行备份，二是通过代码进行备份。下面分别对这两种方法进行讲解。

1．通过数据库本身进行备份

（1）打开企业管理器，展开“数据库”节点，选中要备份的数据库，单击鼠标右键，弹出一个快捷菜单，在快捷菜单中选择“所有任务”/“备份数据库”选项，如图 21.49 所示。

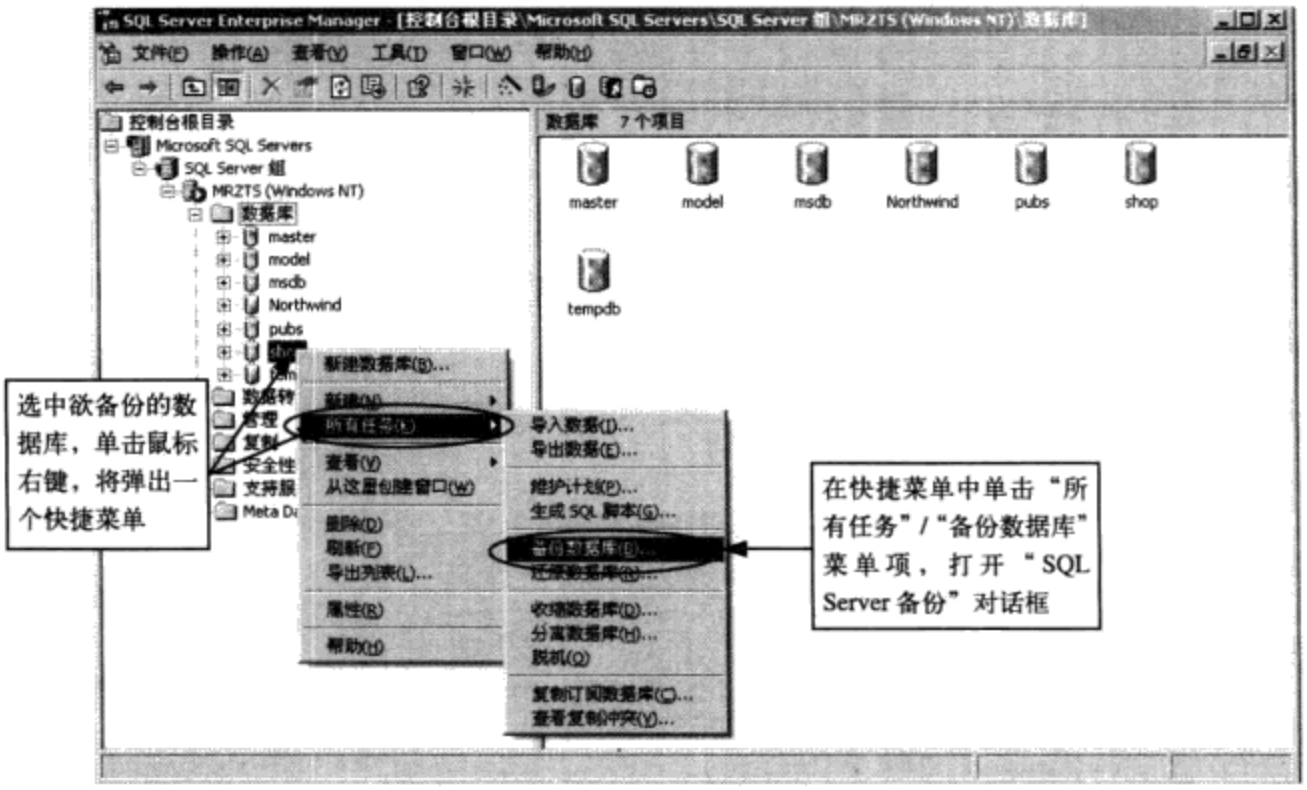


图 21.49　选择要备份的数据库

（2）在“SQL Server 备份”对话框中，单击“添加”按钮，如图 21.50 所示。

（3）在“选择备份目的”对话框中，为数据库指定备份文件。选中“文件名”单选按钮，单击文本框后面的按钮为数据库指定备份文件所在路径，如图 21.51 所示。

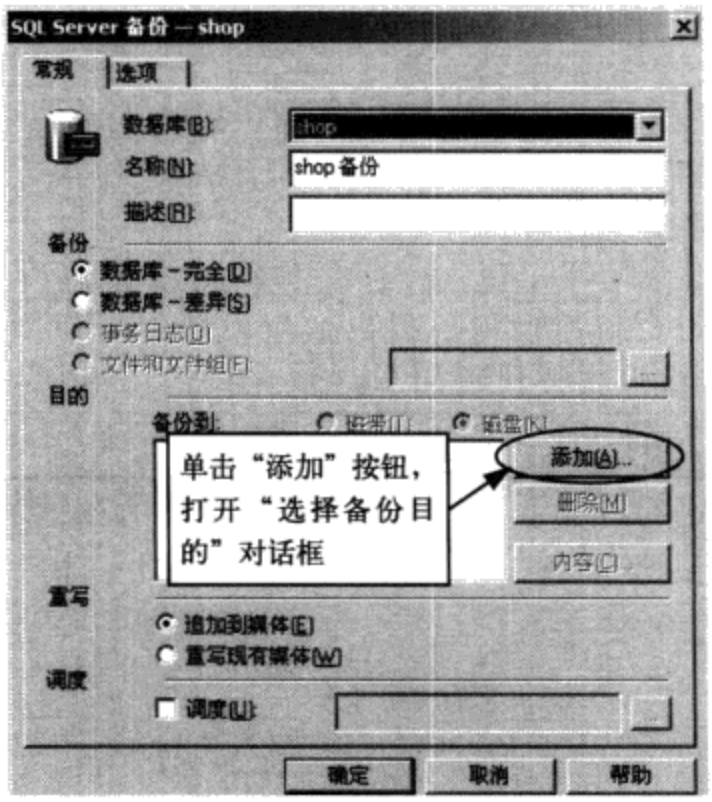


图 21.50　打开数据库备份对话框

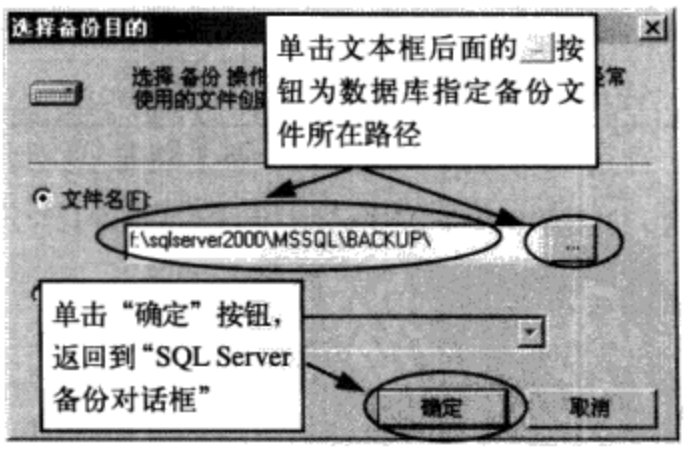


图 21.51　选择备份目的

（4）单击“确定”按钮，返回到“SQL Server 备份”对话框，单击“确定”按钮，开始备份数据库，如图 21.52 所示。

2．通过代码进行备份

备份 SQL Server 数据库需要使用 SQL 语句中的 BACKUP DATABASE。

语法：

```
BACKUP DATABASE { database_name | @database_name_var } TO < backup_device > [ ,...n ]
```

参数说明如下：

DATABASE：指定一个完整的数据库备份。

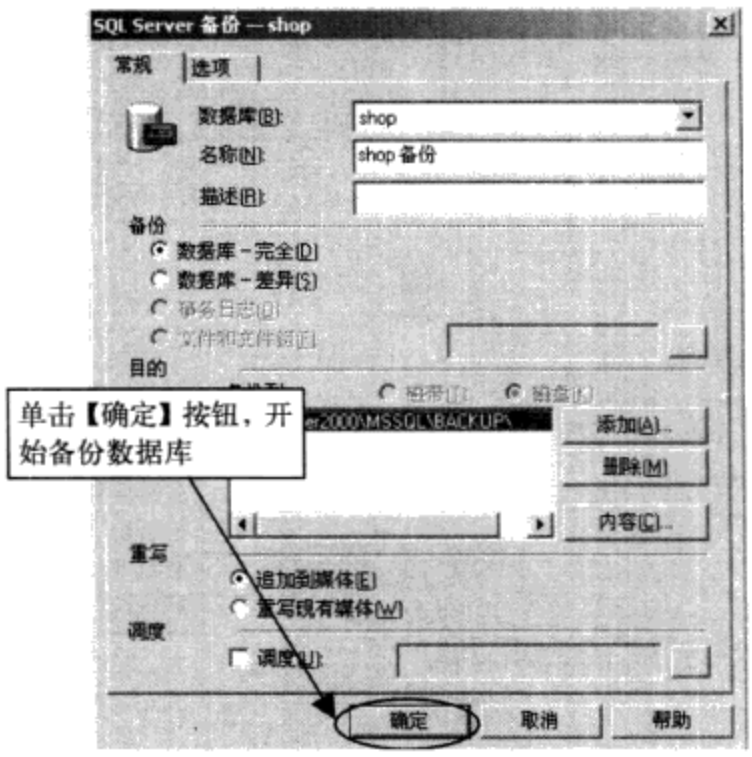


图 21.52　备份数据库

{ database_name | @database_name_var }：指定了一个数据库，从该数据库中对事务日志、部分数据库或完整的数据库进行备份。

< backup_device >：指定备份操作时要使用的逻辑或物理备份设备。

说明：如果设置备份文件为一个相对路径名，备份文件将存储到默认的备份目录中，也就是存储在注册表 KEY_LOCAL_MACHINE\Software\Microsoft\MSSQLServer\MSSQLServer 目录下的 BackupDirectory 键值中。

例如，在用该语句备份数据库时，要为其指定一个数据库名和备份位置。SQL 语句如下：

```
BACKUP DATABASE MyData to Disk = 'E:\MyData.bak '
```

以上 SQL 语句将数据库 MyData 备份到 E:\MyData.bak 中。

示例 21-04　备份数据库

示例主要对 SQL Server 2000 数据库进行备份。程序主要代码如下：

代码 21-04　光盘位置：光盘\mingrisoft\sl\21\21_04

自定义方法 beifenInfo 用于将数据库备份到指定的目录下，并返回备份文件的所在目录，参数 DataName 表示要备份的数据库名称。代码如下：

```
public void beifenInfo(string DataName)
{
    string BackupPath = null;                                    //记录备份的路径
    SaveFileDialog saveFile = new SaveFileDialog();              //实例化 SaveFileDialog 对象
    saveFile.InitialDirectory = Application.StartupPath + "\\";  //默认路径为可执行文件的目录中
    saveFile.FilterIndex = 1;                                    //设置文件对话框当前选中的为第一个
    saveFile.RestoreDirectory = true;                            //重新定位保存路径
    saveFile.Filter = "备份文件 (*.bak)|*.bak|所有文件 (*.*)|*.*";  //筛选器，定义文件类型
    if (saveFile.ShowDialog() == DialogResult.OK)                //打开“保存”对话框
    {
        if (!File.Exists(saveFile.FileName.ToString()))          //如果保存的文件不存在
        {
            SqlConnection con = new SqlConnection();             //实例化 SqlConnection 对象
            //设置连接数据源的字符串
            con.ConnectionString = "server=.;uid=sa;pwd=;database='" + DataName + "'";
            con.Open();                                          //打开数据源的连接
            SqlCommand com = new SqlCommand();                   //实例化 SqlCommand 对象
            BackupPath = saveFile.FileName.ToString();           //记录保存的文件路径
```

```
                com.CommandText = "BACKUP DATABASE " + this.comboBox1.Text + " TO DISK = '" +
saveFile.FileName.ToString() + "'";                                    //设置 SQL 语句
                com.Connection = con;                                  //执行语句连接数据源
                com.ExecuteNonQuery();                                 //执行备份语句
                com.Dispose();                                         //释放对象资源
                con.Close();                                           //释放对象资源
                con.Dispose();                                         //释放对象资源
                MessageBox.Show("数据备份成功！");
            }
        }
        saveFile.Dispose();                                            //释放对象资源
        return BackupPath;
    }
```

“备份”按钮的单击事件主要用于调用自定义方法 beifenInfo，并根据返回值判断是否备份成功。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string ReturnValue = beifenInfo(this.comboBox1.Text);              //调用 beifenInfo 方法
    if (ReturnValue == null)                                           //如果返回值为空
        MessageBox.Show("备份失败");
    else
    {
        textBox1.Text = ReturnValue;                                   //显示备份文件的路径
        MessageBox.Show("备份成功");
    }
}
```

按<Ctrl+F5>键运行程序，运行结果如图 21.53 所示。

图 21.53　备份数据库

注意：在命名空间中引用 using System.Data.SqlClient 和 using System.IO 命名空间。在开发程序时，如果想备份数据库，必须在备份数据库前关闭当前程序与数据库的连接。

21.5.2　数据库还原

数据库的还原可以用两种方法实现，一是通过数据库本身进行还原，二是通过代码进行还原。下面分别对这两种方法进行讲解。

1．通过数据库本身进行还原

（1）打开企业管理器，展开“数据库”节点，选中要还原的数据库，单击鼠标右键，将弹出一个快捷菜单，选择“所有任务”/“还原数据库”选项，如图 21.54 所示。

（2）在打开的“还原数据库”对话框中，在表格中选择备份媒体，如图 21.55 所示，单击“确定”按钮开始还原数据库。

2．通过代码进行还原

还原 SQL Server 数据库需要使用 SQL 语句中的 RESTORE DATABASE 进行还原。

语法：

```
RESTORE DATABASE { database_name | @database_name_var } [ FROM < backup_device > [ ,...n ] ]
```

参数说明如下：

DATABASE：指定一个要还原的数据库。

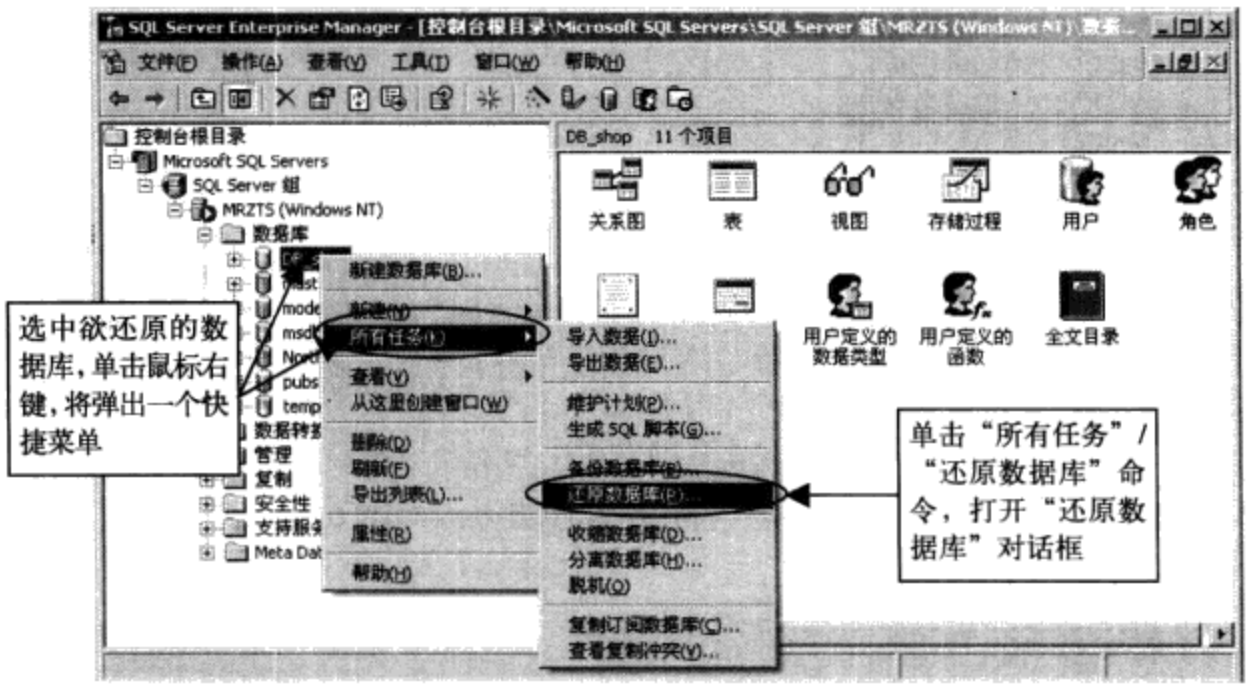


图 21.54　选择要还原的数据库

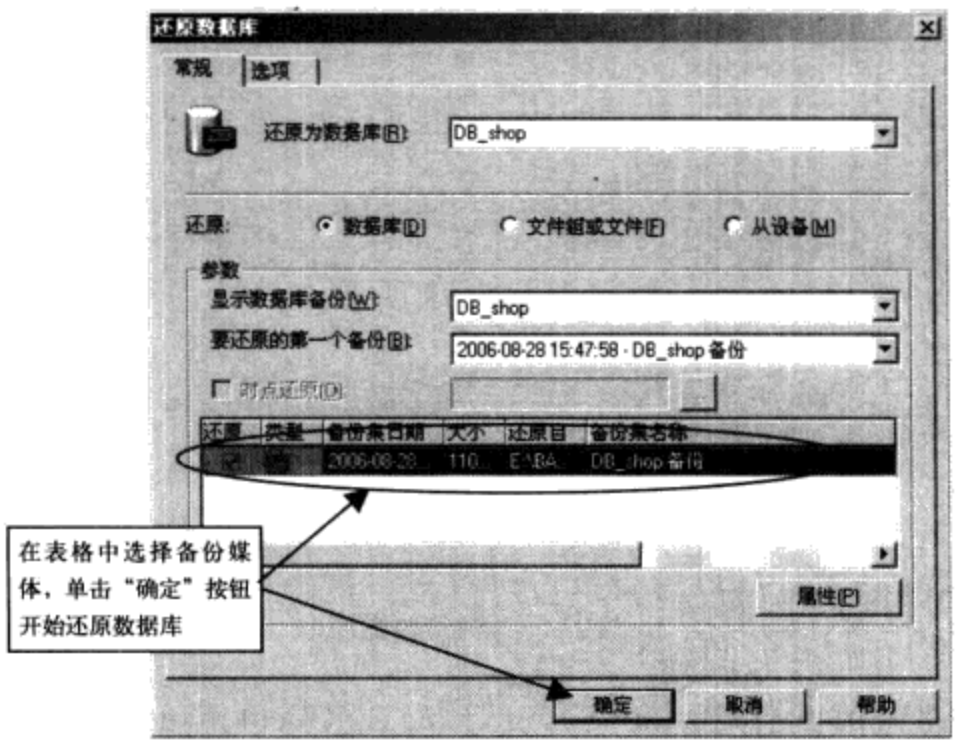


图 21.55　还原数据库

{ database_name | @database_name_var }：指定了一个数据库，从该数据库中对事务日志、部分数据库或完整的数据库进行还原。

< backup_device >：指定还原操作时要使用的逻辑或物理备份设备。

例如，在用该语句还原数据库时，必须为其指定要还原的数据库名和备份文件。SQL 语句如下：

```
RESTORE DATABASE MyData to Disk = 'E:\MyData.bak ';
```

以上 SQL 语句将 E:\MyData.bak 备份文件恢备到 MyData 数据库中。

在对数据库进行还原时，为了完全中断与数据库的连接，用 SINGLE_USER 语句每次只允许一个用户连接到数据库，用 ROLLBACK IMMEDIATE 语句立即中断不合格的连接，并且所有未完成的事务都将回滚。SQL 语句如下：

```
alter database DataName set single_user with rollback immediate;
```

以上代码中 DataName 参数为数据库名称。

示例 21-05　还原数据库

示例主要对 SQL Server 2000 数据库进行还原。程序主要代码如下：

代码 21-05　光盘位置：光盘\mingrisoft\sl\21\21_05

自定义方法 restoreInfo 用于将指定的备份文件还原到数据库备，参数 DataName 表示要还原的数据库，参数 RestorePath 表示备份文件的路径。代码如下：

```
private void restoreInfo(string DataName, string RestorePath)
{
    string SqlStr1 = "Server=(local);database='" + DataName + "';Uid=sa;Pwd=";   //设置连接数据库的字符串
    //设置还原的 SQL 语句
    string SqlStr2 = "alter database " + DataName + " set single_user with rollback immediate use master restore
database " + DataName + " from disk='" + RestorePath + "'";
    using (SqlConnection con = new SqlConnection(SqlStr1))                      //连接数据源
    {
        con.Open();                                                             //打开数据源
        try
        {
            SqlCommand cmd = new SqlCommand(SqlStr2,con);                       //连接执行语句和数据源
            cmd.Connection = con;                                               //设置连接的数据库
            cmd.ExecuteNonQuery();                                              //执行 SQL 语句
            MessageBox.Show("还原数据成功");
            cmd.Dispose();                                                      //释放对象资源
        }
        catch (Exception error)                                                 //发生异常
        {
            MessageBox.Show("还原失败，请确保还原项与库对应");
        }
        finally
        {
            con.Close();                                                        //释放对象资源
        }
    }
}
```

“还原”按钮的单击事件主要用于调用自定义方法 restoreInfo 还原指定的数据库。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (comboBox1.Text.Length > 0 && textBox1.Text.Length > 0)          //如果选择了还原数据库和还原文件
        restoreInfo(comboBox1.Text, textBox1.Text);                     //调用自定义方法
}
```

按<Ctrl+F5>键运行程序，运行结果如图 21.56 所示。

图 21.56　还原数据库

注意：在开发程序时，如果想还原数据库，必须在还原数据库前关闭当前程序与数据库的连接。

21.6　附加和分离数据库

本节主要讲解数据库的附加和分离操作。通过本节的学习，读者可以掌握附加和分离 SQL Server 2000 数据库的方法。

21.6.1　附加数据库

附加数据库可以用两种方法实现，一是通过数据库本身进行附加，二是通过代码进行附加。

下面分对这两种方法进行讲解。

1．通过数据库本身进行附加

通过附加方式可以向服务器中添加数据库，前提是需要存在数据库文件和数据库日志文件。下面以“shop”数据库为例介绍如何附加数据库。

（1）打开企业管理器，鼠标右键单击“数据库”选项，将弹出一个快捷菜单，依次选择“所有任务”/“附加数据库”选项，如图 21.57 所示。

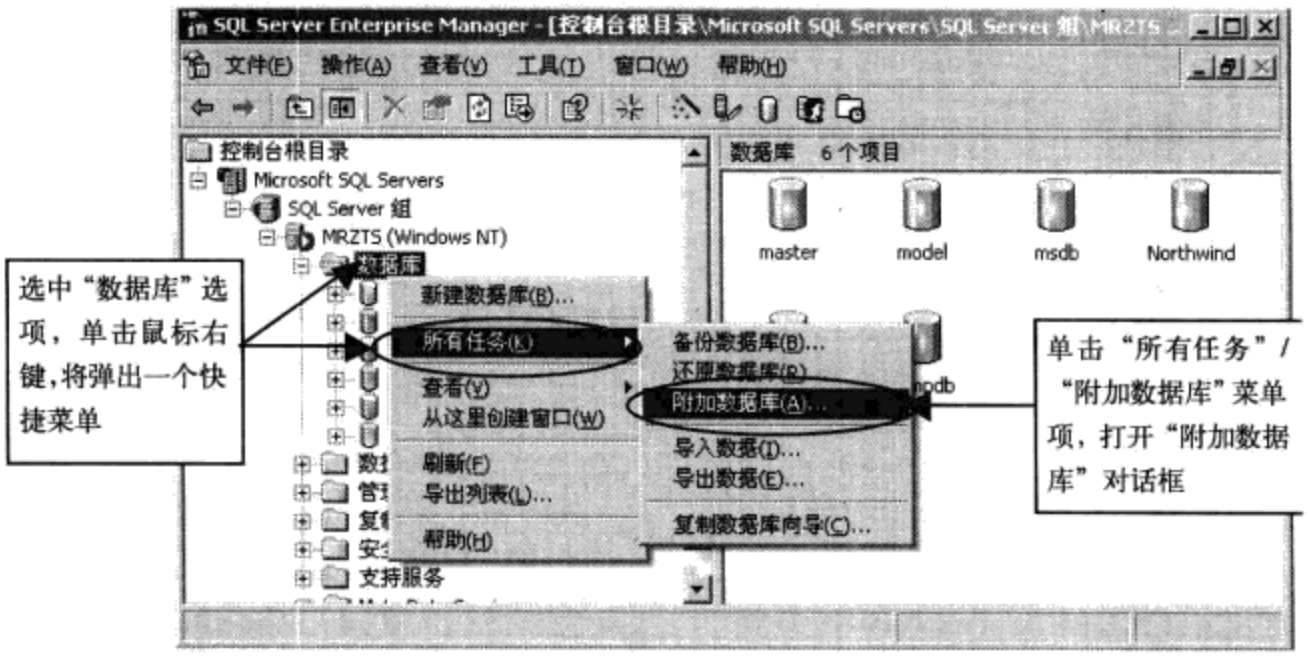


图 21.57　选择要附加的数据库

（2）在“附加数据库”对话框中，单击“要附加数据库的.MDF 文件”文本框后面的 ... 按钮设置.MDF 文件所在路径，如图 21.58 所示，单击“确定”按钮，完成附加数据库操作。

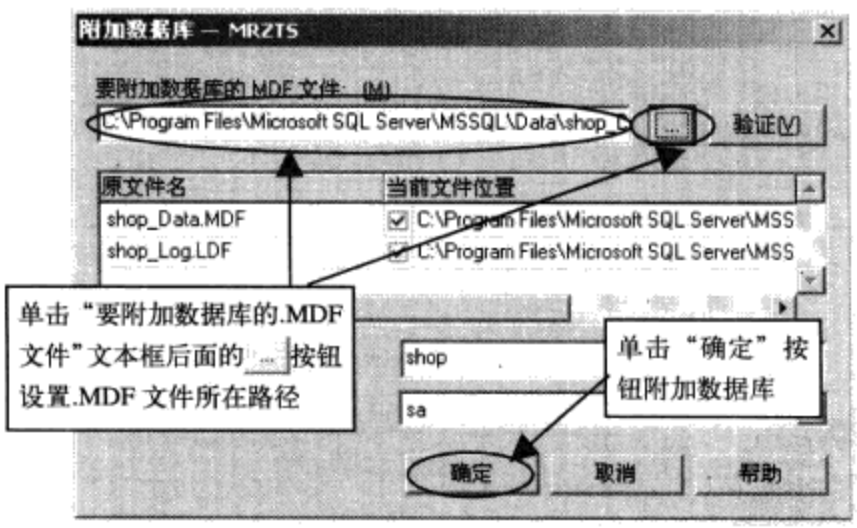


图 21.58　附加数据库

2．通过代码进行附加

附加 SQL Server 数据库可以用 SQL 语句中的 sp_attach_db 和 sp_attach_single_file_db 来实现。

（1）sp_attach_db 语句，主要用于将数据库附加到服务器上。

语法：

```
sp_attach_db [ @dbname = ] 'dbname' , [ @filename1 = ] 'filename_n' [ ,...16 ]
```

参数说明如下。

[@dbname =] 'dbname'：要附加到服务器的数据库的名称。该名称必须是惟一的。

[@filename1 =] 'filename_n'：数据库文件的物理名称，包括路径。

返回值：0（成功）或 1（失败）。

例如，将 db_21 中的两个文件附加到当前服务器上。SQL 语句如下：

```
EXEC sp_attach_db @dbname = N'db_21', @filename1 = N'c:\\db_21.mdf', @filename2 = N'c:\\db_21_log.ldf'
```

（2）sp_attach_single_file_db 语句，将只有一个数据文件的数据库附加到当前服务器。

语法：

```
sp_attach_single_file_db [ @dbname = ] 'dbname' , [ @physname = ] 'physical_name'
```

参数说明如下。

[@dbname =] 'dbname'：要附加到服务器的数据库的名称。dbname 的数据类型为 sysname，默认值为 NULL。

[@physname =] 'phsyical_name'：数据库文件的物理名称，包括数据路径。physical_name 的数据类型为 nvarchar(260)，默认值为 NULL。

返回值：0 表示成功；1 表示失败。

注意：只有 ysadmin 和 dbcreator 固定服务器角色的成员才能执行本过程。

例如，将只有一个文件的 db_21 附加到当前服务器上。SQL 语句如下：

```
EXEC sp_detach_db @dbname = 'db_21'
EXEC sp_attach_single_file_db @dbname = 'db_21', @physname = 'c:\db_21.mdf'
```

示例 21-06　附加数据库

示例主要附加 SQL Server 2000 中的数据库，附加的数据库可以是单文件（mdf 文件），也可以是双文件（mdf 文件和 ldf 文件）。程序主要代码如下：

代码 21-06　光盘位置：光盘\mingrisoft\sl\21\21_06

```
private void Addons(string mdfPath)
{
    FileInfo fileInfo = new FileInfo(mdfPath);                                  //获取文件的基本信息
    if (fileInfo.Extension.ToLower() != ".mdf")                                 //断判文件是否为 mdf 文件
        return;
    string DataName = fileInfo.Name.Substring(0, fileInfo.Name.Length - 9);     //获取数据库名
    string DataLdf = fileInfo.DirectoryName + "\\" + DataName + "_Log.LDF";     //设置 LDF 文件的路径
    try
    {
        SqlConnection con = new SqlConnection("server=.;pwd=;uid=sa;database=master");   //连接数据源
        SqlCommand cmd = new SqlCommand();                                      //实例化 SqlCommand 对象
        if (System.IO.File.Exists(DataLdf))                                     //判断 LDF 文件是否存在
        {
            con.Open();                                                         //打开数据源的连接
            cmd.Connection = con;                                               //SQL 语句与数据库的连接
            StringBuilder sb = new StringBuilder();                             //实例化一个可变字符串
            sb.Append("sp_attach_db @dbname='" + DataName + "',");              //向字符串中添加 SQL 语句
            sb.Append("@filename1='" + mdfPath + "'");
            sb.Append(",@filename2='" + DataLdf + "'");
            cmd.CommandText = sb.ToString();                                    //设置 SQL 语句
            cmd.ExecuteNonQuery();                                              //执行 SQL 语句
        }
        else                                                                    //单文件附加
        {
            con.Open();                                                         //打开数据源的连接
            cmd.Connection = con;                                               //SQL 语句与数据库的连接
            StringBuilder sb = new StringBuilder();                             //实例化一个可变字符串
            sb.Append("sp_attach_single_file_db @dbname='" + DataName + "',");  //向字符串中添加 SQL 语句
            sb.Append("@physname='" + mdfPath + "'");
            cmd.CommandText = sb.ToString();                                    //设置 SQL 语句
            cmd.ExecuteNonQuery();                                              //执行 SQL 语句
        }
        MessageBox.Show("附加成功");
    }
    catch (Exception ety)                                                       //发生异常
```

```
        {
            MessageBox.Show(ety.Message);                                    //提示异常信息
        }
    }
```

注意：在命名空间中引用 using System.Data.SqlClient 和 using System.IO 命名空间。

按<Ctrl+F5>键运行程序，运行结果如图 21.59 所示。

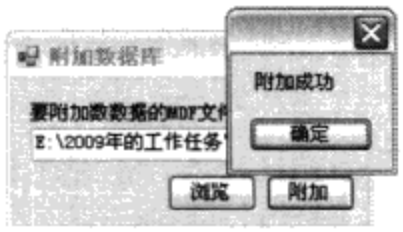


图 21.59　附加数据库

21.6.2　分离数据库

分离数据库可以用两种方法实现，一是通过数据库本身进行分离，二是通过代码进行分离。下面分对这两种方法进行讲解。

1．通过数据库本身进行分离

分离数据库是将数据库从服务器中分离出去，但并没有删除数据库，数据库文件依然存在，如果在需要使用数据库时，可以通过附加的方式将数据库附加到服务器中。

在 SQL Server 2000 中分离数据库非常简单，方法如下。

（1）打开企业管理器，展开“数据库”节点，选中要分离的数据库，单击鼠标右键，将弹出一个快捷菜单，依次选择“所有任务”/“分离数据库”选项，如图 21.60 所示。

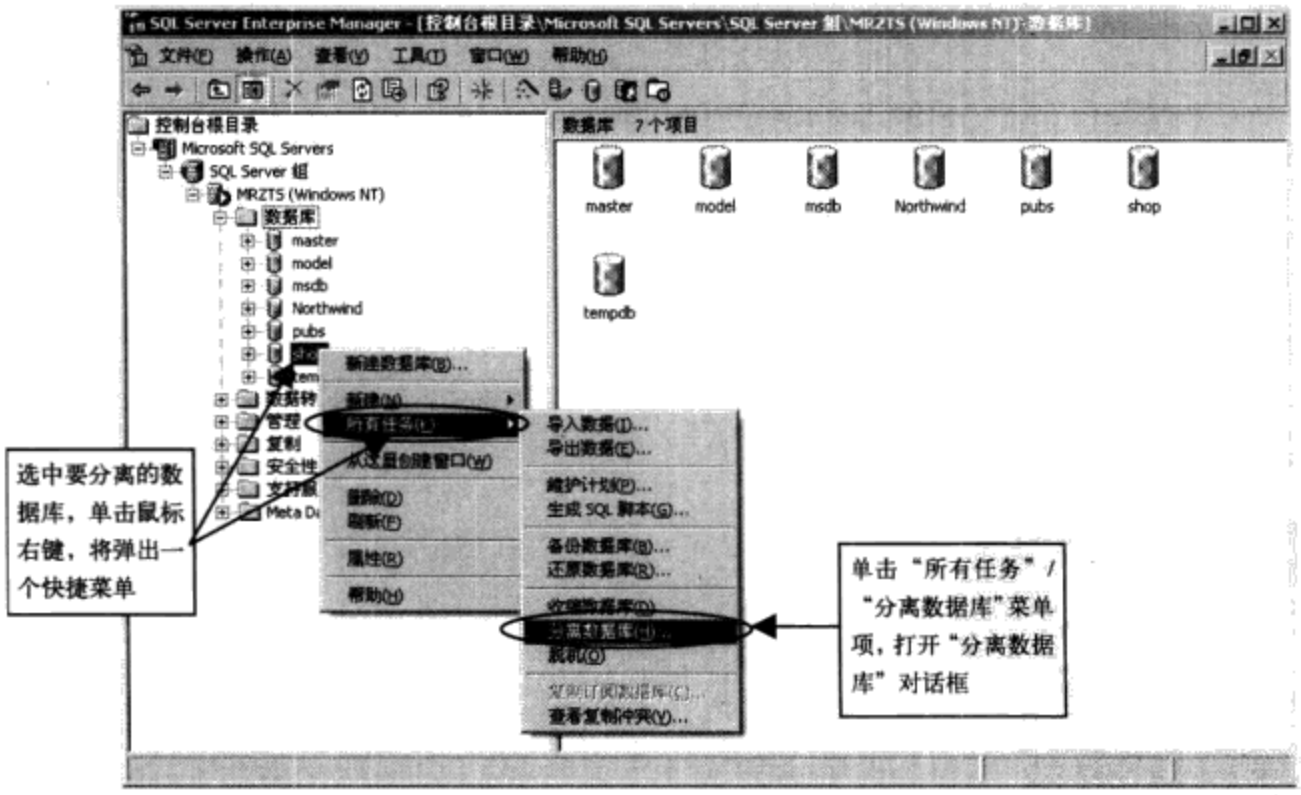


图 21.60　选择要分离的数据库

（2）打开如图 21.61 所示的“分离数据库”对话框，单击“确定”按钮，开始分离数据库。

2．通过代码进行分离

分离 SQL Server 数据库可以用 SQL 语句中的 sp_detach_db 来实现。

sp_detach_db 语句主要用于从服务器分离数据库，并可以选择在分离前在所有的表上运行 UPDATE STATISTICS。

语法：

```
sp_detach_db [ @dbname = ] 'dbname' [ , [ @skipchecks = ] 'skipchecks' ]
```

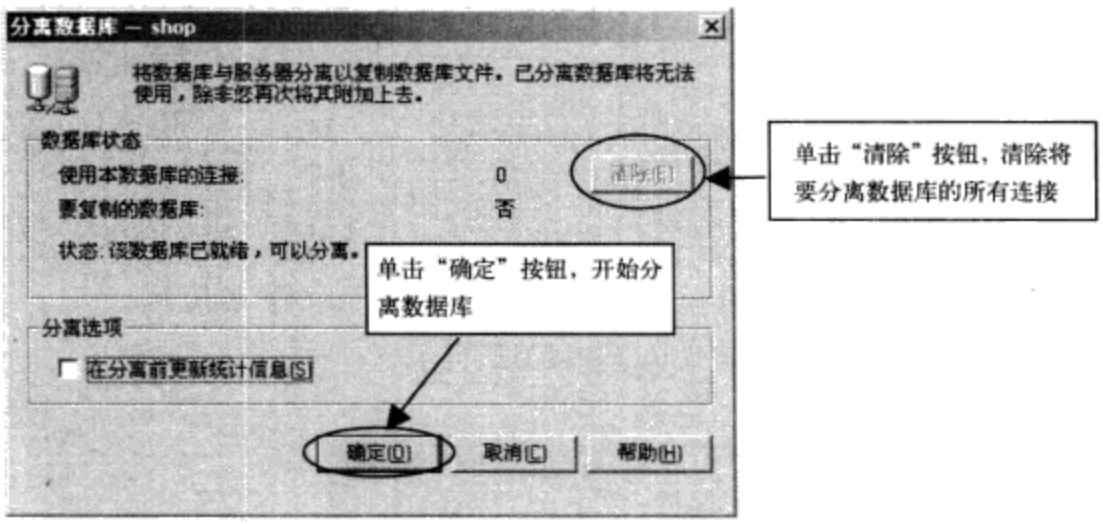


图 21.61　分离数据库

参数说明如下。

[@dbname =] 'dbname'：要分离的数据库名称。

[@skipchecks =] 'skipchecks'：skipchecks 的数据类型为 nvarchar(10)，默认值为 NULL。如果为 true，则跳过 UPDATE STATISTICS。如果为 false，则运行 UPDATE STATISTICS。对于要移动到只读媒体上的数据库，此选项很有用。

返回值：0（成功）或 1（失败）。

例如，分离数据库的中的 db_21 数据库。SQL 语句如下：

```
EXEC sp_detach_db 'pubs'
```

示例 21-07　分离数据库

示例主要分离 SQL Server 2000 中的指定数据库。程序代码如下：

代码 21-07　光盘位置：光盘\mingrisoft\sl\21\21_07

自定义方法 biandingiInfo 可以在窗体加载事件（Load）或第一次显示事件（Shown）中调用，主要用于将 SQL Server 2000 中的数据库名称添加到 comboBox1 控件中。代码如下：

```
private void biandingiInfo()
{
    using (SqlConnection con = new SqlConnection("server=.;pwd=;uid=sa;database=master"))  //连接数据源
    {
        DataTable dt = new DataTable();                                          //实例化一个数据表
        //设置 SQL 语句与数据源的连接
        SqlDataAdapter da = new SqlDataAdapter("select name from sysdatabases", con);
        da.Fill(dt);                                                             //将刷新的数据添加到数据表中
        this.comboBox1.DataSource = dt.DefaultView;                              //连接数据源
        this.comboBox1.DisplayMember = "name";                                   //设置字段名
        this.comboBox1.ValueMember = "name";                                     //设置字段值
    }
}
```

“分离数据库”按钮的单击事件用于将指定的数据库进行分离。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    using (SqlConnection con = new SqlConnection("server=.;pwd=;uid=sa;database=master"))  //连接数据源
    {
        try
        {
            SqlCommand cmd = new SqlCommand();                                   //实例化 SqlCommand 对象
            con.Open();                                                          //打开数据源的连接
            cmd.Connection = con;                                                //连接数据源
            cmd.CommandText = "sp_detach_db @dbname='" + this.comboBox1.Text+"'";    //添加 SQL 语句
            cmd.ExecuteNonQuery();                                               //执行 SQL 语句
```

```
            MessageBox.Show("分离成功");
        }
        catch(Exception ey)                                    //发生异常
        {
            MessageBox.Show(ey.Message);                       //提示异常信息
        }
    }
}
```

注意：在命名空间中引用 using System.Data.SqlClient 命名空间。

按<Ctrl+F5>键运行程序，运行结果如图 21.62 所示。

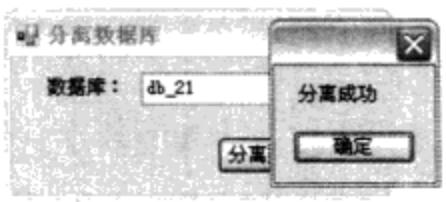


图 21.62　分离数据库

第 22 章

ADO.NET 技术

- ADO.NET 概述
- ADO.NET 对象模型
- 数据库开发基本操作

ADO.NET 是.NET Framework 提供的一组专门用于存取数据的类，其中包含的一些对象可以配合 SQL 语句对数据库进行操作。

下面将介绍 ADO.NET、ADO.NET 中的各个对象及其在数据库开发中的具体应用。

22.1 ADO.NET 概述

依据对数据处理的功能，把 ADO.NET 分解为多个可以单独使用的组件。ADO.NET 包含用于连接数据库、执行命令和检索结果的.NET Framework 提供程序。ADO.NET 类在 System.Data.dll 中，并且与 System.Xml.dll 中的 XML 类集成。

ADO.NET 对象被分为两大类，一类是与数据库直接连接的联机对象（称为.NET Data Provider），其中包含了 Command 对象、DataReader 对象以及 DataAdapter 对象等，通过该类对象，可以在应用程序内完成连接数据源以及数据维护等相关操作；另一类则是与数据源无关的断线对象，例如，DataSet 对象以及 DataRelation 对象等。用户可以利用连接对象取得数据源所需的原始数据，并且利用断线对象，一次返回给前端用户，前端用户在处理变动数据的过程中，并不需要保持与数据库连接，当对所有数据完成变动操作之后，则再一次通过连接对象将数据返回到数据库。因为不需要时时保持与数据库的连接，所以能够极大的降低系统资源的消耗。

ADO.NET 对象的组织结构如图 22.1 所示。

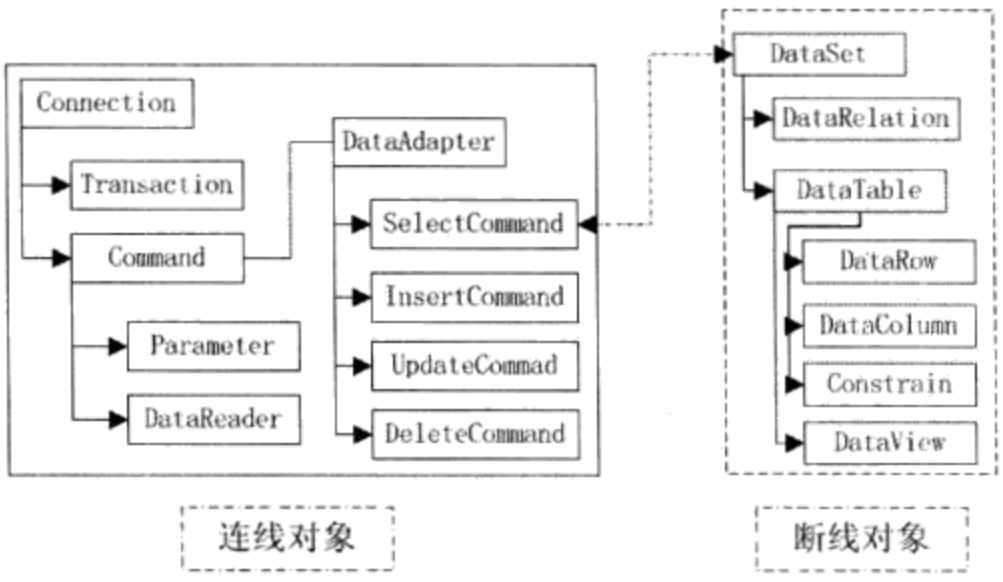


图 22.1 ADO.NET 对象的组织结构图

22.2 ADO.NET 对象模型

22.2.1 Connection 对象

Connection 对象也称为连接对象，在进行任何与数据库的操作之前，首先需要创建与目标数据库的连接，ADO.NET 提供了与数据源连接的对象，主要包括 OleDbConnection 对象、SqlConnection 对象和 OdbcConnection 对象，其命名空间如表 22.1 所示。

表 22.1 连接对象的命名空间

命 名 空 间	连 接 对 象
System.Data.OleDb	OleDbConnection
System.Data.SqlClient	SqlConnection
System.Data.Odbc	OdbcConnection

微软公司开发的 Access 以及 SQL Server，甲骨文公司开发的 Oracle 是目前使用最频繁的数据库。下面分别详细介绍。

1．Access

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=dataPath;
```

例如，在 ASP.NET 中，连接一个无密码的 Access 数据库“AccessBase.mdb”的连接字符串如下：

```
string ConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Data source=" + Server.MapPath("AccessBase.mdb");
```

例如，在 ASP.NET 中，连接一个有密码的 Access 数据库“AccessBase.mdb”的连接字符串如下：

```
string ConStr = "Provider=Microsoft.Jet.OLEDB.4.0;Jet OLEDB:DataBase Password="+TxtPwd.Text+";User Id=admin;
Data source=" + Server.MapPath("AccessBase.mdb");
```

在以上连接字符串中，Provider 属性指定使用的数据库引擎 Microsoft.Jet.OLEDB.4.0。Data source 属性指定数据库文件位于计算机中的物理位置，这里应用 Server 对象的 MapPath 方法将虚拟路径转换为物理路径。另外，使用 C#开发的 Windows 应用程序，也可以同样使用该方法连接 Access 数据库。

注意：在应用 OLEDB 数据提供程序连接数据库时，需要引用 using System.Data. OleDb 命名空间。

2．SQL Server

```
Provider=SQLOLEDB;Data Source=mysqlServer;Integrated Security=SSPI;
```

例如，通过信任连接模式与本地 SQL Server 数据库连接，连接字符串如下：

```
Provider=SQLOLEDB;Data Source=mysqlServer;Integrated Security=SSPI;
```

例如，连接带密码的 SQL Server 数据库，连接字符串如下：

```
Server=(Local);pwd=password;uid=user id;database=mydatabase;
```

读者应该按照需要设置 Server、Database、User id 和 Pwd 数据连接属性。数据连接属性及说明如表 22.2 所示。

表 22.2　连接属性说明

参　数	描　述	举　例
Server	在 IIS 服务器上建立的访问 SQL Server 服务器的别名	Local（. 或 IP 地址）
Database	访问数据库的名称	Northwind
User id	使用的用户名称	Sa
Pwd	使用的用户口令	空

说明：local 代表 SQL Server 在本地计算机上，如果要连接远程计算机将它换成远程计算机的 IP 地址或计算机名称即可，例如，Server=192.168.1.98。

3．Oracle

```
Provider=MSDAORA;Data Source=ORACLR8i;User ID=OLEDB;Password=OLEDB;
```

例如，连接 Oracle 9i 数据库，连接字符串如下：

```
string OrlCon = "Data Source=Oracle9i;Integrated Securite=yes";
```

示例 22-01　建立 SQL Server 数据库连接

下面举例说明如何连接 SQL Server 数据库，步骤如下。

（1）创建 Connection 对象。

（2）把连接字符串赋值给 Connection 对象的 ConnectionString 属性。

（3）调用 Connection 对象的 Open 方法打开连接。

（4）连接使用完毕后，应调用 Connection 对象的 Close 方法关闭连接。

（5）调用 Connection 对象的 Dispose 方法释放其所占用的资源。

示例通过自定义方法 ConDB 使用 ADO.NET 的 Connection 对象连接 SQL Server 中的“db_22”数据库。程序代码如下：

代码 22-01　光盘位置：光盘\mingrisoft\sl\22\22_01

```
private SqlConnection con;//实例化 SqlConnection 对象
public SqlConnection ConDB()
{
    con = new SqlConnection("server=.;uid=sa;pwd=;database=db_22");    //连接数据源
    if (con.State == ConnectionState.Closed)                           //如果关闭数据库的连接
    {
        con.Open();                                                    //打开连接
    }
    return con;
}
```

22.2.2　Command 对象

使用 Connection 对象与数据源建立连接之后，可以使用 Command 对象来对数据源执行查询、添加、删除和修改等各种操作。操作实现的方式可以使用 SQL 语句，也可以使用存储过程。

Command 对象常用属性及说明如表 22.3 所示。

表 22.3　Command 对象常用属性及说明

属　性	说　明
CommandText	获取或设置要对数据源执行的 Transact-SQL 语句或存储过程
CommandTimeout	获取或设置在终止执行命令的尝试并生成错误之前的等待时间
CommandType	获取或设置一个值，该值指示如何解释 CommandText 属性
Connection	获取或设置 SqlCommand 类的此实例使用的 SqlConnection 对象
DesignTimeVisible	获取或设置一个值，该值指示命令对象是否应在 Windows 窗体设计器控件中可见
Notification	获取或设置一个指定与此命令绑定的 SqlNotificationRequest 对象的值
NotificationAutoEnlist	获取或设置一个值，该值指示应用程序是否应自动接收来自公共 SqlDependency 对象的查询通知
Parameters	获取 SqlParameterCollection 对象
Transaction	获取或设置将在其中执行 SqlCommand 类的 SqlTransaction 对象
UpdatedRowSource	已重写。获取或设置命令结果在由 DbDataAdapter 类的“Update”方法使用时，如何应用于 DataRow 对象

下面介绍几个主要的属性。

（1）CommandText 属性。获取或设置要对数据源执行的 Transact-SQL 语句或存储过程。

语法：

```
public override string CommandText { get; set; }
```

属性值：要执行的 Transact-SQL 语句或存储过程。默认为空字符串。

通过 Command 对象执行 SQL 语句或存储过程。

例如，通过 Command 对象来执行一条 SQL 语句显示表中信息。代码如下：

```
SqlCommand cmd = new SqlCommand("select * from TableName")
```

例如，通过 Command 对象来执行存储过程（Proc_Name）。代码如下：

```
SqlCommand cmd = new SqlCommand("Proc_Name")
```

（2）CommandText 属性。获取或设置一个值，该值指示如何解释 CommandText 属性。

语法：

```
public override CommandType CommandType { get; set; }
```

属性值：CommandType 值（StoredProcedure 存储过程的名称、TableDirect 表的名称、Text SQL 文本命令）之一。默认值为 Text。

例如，创建一个 SqlCommand 类的对象 command，令 command 对象执行文本命令。代码

如下：

```
SqlCommand command = new SqlCommand();
command.CommandType = CommandType.Text;
```

说明：①SQL 语句是指将指定的 SQL 语句传达到连接对象所连接的数据库中执行，若是此段 SQL 语句返回搜索数据的内容，则利用 DataSet 对象将数据返回，另外，也可以传入动态查询 SQL 表达式，得到动态数据内容。②存储过程：数据库系统通常会扩充标准的 SQL 语法，例如，SQL Server 的 T-SQL，ORACLE 专用的 PL/SQL，使用这些专门的 SQL 语法，编写具备操作数据功能的专门程序，存放在数据库系统中，由前端程序调用执行。Command 对象可以调用这些存放于数据库的程序，执行数据的操作与维护工作，甚至返回搜索结果。

（3）Parameters 属性。获取 SqlParameterCollection，也就是获取与 SqlCommand 相关联的参数的集合以及各个参数到 DataSet 中列的映射。

语法：

```
public SqlParameterCollection Parameters { get; }
```

属性值：Transact-SQL 语句或存储过程的参数。默认为空集合。

例如，用 Parameters 属性将一组数赋值给与 SqlCommand 相关联的参数。代码如下：

```
string strCon = " Server =(local);Database=db_22;User id=sa;PWD=";          //设置连接数据库的字符串
SqlConnection sqlconn = new SqlConnection(strCon);                          //连接数据源
sqlconn.Open();                                                             //打开数数源连接
SqlCommand command = new SqlCommand("select * from tb_Logon where Name=@Name and Pass=@Pass",
sqlconn);                                                                   //将 SQL 语句与数据库相连接
command.Parameters.Add("@Name", SqlDbType.VarChar);                         //添加参数类型
command.Parameters["@Name"].Value = "mr";                                   //添加参数值
command.Parameters.Add("@Pass", SqlDbType.VarChar);
command.Parameters["@Pass"].Value = "mrsoft";
SqlDataReader My_read = command.ExecuteReader();                            //执行 SQL 语句
if (My_read.Read())                                                         //读取查找后的记录
    MessageBox.Show(My_read.GetString(1));                                  //获取查找后第一条记录的信息
else
    MessageBox.Show("查找失败");
```

注意：在用参数进行赋值时，有时会因为字段的类型太过复杂而增加代码量，这时，可以将下面的代码：

```
command.Parameters.Add("@Name", SqlDbType.VarChar);                         //添加参数类型
command.Parameters["@Name"].Value = "mr";                                   //添加参数值
```

更改为：

```
command.Parameters.AddWithValue("@Name", "mr" );
```

（4）Transaction 属性。获取或设置将在其中执行 SqlCommand 类的 SqlTransaction 对象。也就是对数据库中的事务进行处理。

语法：

```
public SqlTransaction Transaction { get; set; }
```

属性值：SqlTransaction 对象，默认值为空引用。

例如，通过 SqlConnection 类的 BeginTransaction 方法实例化一个 SqlTransaction 对象，然后将其赋给 command 对象的 Transaction 属性，完成事务的操作（开启事务、提交事务、回滚事务），程序代码如下：

```
SqlTransaction sqlTran = connection.BeginTransaction();          //开始事务
SqlCommand command = connection.CreateCommand();                 //创建 SqlCommand 对象
command.Transaction = sqlTran;                                   //将 SqlCommand 与 SqlTransaction 关联起来
sqlTran.Commit();                                                //提交事务
sqlTran.Rollback();                                              //回滚事务
```

说明：SqlTransaction 类中，Commit 方法用于提交数据库事务，Rollback 方法用于挂起状态回滚事务，Save 方法则用于在事务中创建保存点（它可用于回滚事务的一部分），并指定

保存点名称。

交叉链接：事务的具体操作可参见本章 22.3.6 节。

Command 对象常用方法及说明如表 22.4 所示。

表 22.4　　Command 对象常用方法及说明

方　法	说　明
BeginExecuteNonQuery	启动此 SqlCommand 对象描述的 Transact-SQL 语句或存储过程的异步执行
BeginExecuteReader	启动此 SqlCommand 对象描述的 Transact-SQL 语句或存储过程的异步执行，并从服务器中检索一个或多个结果集
BeginExecuteXmlReader	启动此 SqlCommand 对象描述的 Transact-SQL 语句或存储过程的异步执行，并将结果作为 XmlReader 对象返回
Cancel	尝试取消 SqlCommand 对象的执行
Clone	创建作为当前实例副本的新 SqlCommand 对象
CreateParameter	创建 SqlParameter 对象的新实例
Dispose	释放由 Component 类占用的资源
EndExecuteNonQuery	完成 Transact-SQL 语句的异步执行
EndExecuteReader	完成 Transact-SQL 语句的异步执行，返回请求的 SqlDataReader 对象
EndExecuteXmlReader	完成 Transact-SQL 语句的异步执行，将请求的数据数据以 XML 形式返回
ExecuteNonQuery	对连接执行 Transact-SQL 语句并返回受影响的行数
ExecuteReader	将 CommandText 对象发送到 Connection 类，并生成一个 SqlDataReader 对象
ExecuteScalar	执行查询，并返回查询所返回的结果集中第一行的第一列。忽略其他列或行
ExecuteXmlReader	将 CommandText 对象发送到 Connection 类，并生成一个 XmlReader 对象
Prepare	在 SQL Server 的实例上创建命令的一个准备版本
ResetCommandTimeout	将 CommandTimeout 属性重置为默认值

下面介绍几个主要的方法。

（1）ExecuteNonQuery 方法。对连接执行 Transact-SQL 语句并返回受影响的行数。

语法：

```
public override int ExecuteNonQuery()
```

返回值：受影响的行数。

注意：使用 ExecuteNonQuery 方法来执行目录操作（例如，查询数据库的结构或创建数据库对象），或执行 UPDATE 语句、INSERT 语句或 DELETE 语句。对于 UPDATE 语句、INSERT 语句和 DELETE 语句，返回值为该命令所影响的行数。对于所有其他类型的语句，返回值为-1。如果发生回滚，返回值也为-1。

例如，创建一个 SqlCommand 类的 command 对象，然后使用 command 对象的 ExecuteNonQuery 方法执行，并返回所影响的行数。程序代码如下：

```
SqlCommand command = new SqlCommand("delete from tb_Logon where Name='Blove'");//设置要执行的 SQL 语句
command.Connection = new SqlConnection("Server =(local);Database=db_22;User id=sa;PWD=");    //连接数据源
command.Connection.Open();                                                                    //打开连接的数据源
int n= command.ExecuteNonQuery();                                                             //执行 SQL 语句
MessageBox.Show(n.ToString());                                                                //显示影响的行数
```

（2）ExecuteScalar 方法。执行查询，并返回查询所返回的结果集中第一行的第一列。忽略其他列或行。

语法：

```
public override Object ExecuteScalar ()
```

返回值：结果集中第一行的第一列或空引用（如果结果集为空）。

注意：使用 ExecuteScalar 方法从数据库中检索单个值（一般为一个聚合值）。

例如，在用户登录时，用于检查用户是否合法，程序代码如下：

```
//设置检查用户的 SQL 语句
command.CommandText = "SELECT COUNT(*) FROM Login where Name='"+strName+"' and Pass='"+strPass+"'";
Int32 count = (Int32) command.ExecuteScalar();                    //获取检查的行数
If(count>0) MessageBox.show("OK");                                //如果行数大于 0 则表中有该用户
```

（3）ExecuteReader 方法。将 CommandText 属性发送到 Connection 对象并生成一个 SqlDataReader 结果集。

语法：

```
public SqlDataReader ExecuteReader ()
```

返回值：一个 SqlDataReader 对象。

注意：当 CommandType 属性设置为 StoredProcedure 值时，CommandText 属性应设置为存储过程的名称。调用 ExecuteReader 方法时，该命令将执行此存储过程。

例如，对 User 表进行查询操作，通过 ExecuteReader 方法把信息返回给 SqlDataReader 对象，程序代码如下：

```
private static SqlDataReader sqlDR()
{
    SqlCommand command = new SqlCommand("select * from User");                    //设置 SQL 语句
    command.Connection = new SqlConnection("Server =(local);Database=db_22;User id=sa;PWD=");//连接数据源
    SqlDataReader dr = cmd.ExecuteReader();                                       //执行 SQL 语句
    SqlCommand.Dispose();                                                         //释放对象资源
    return dr;                                                                    //返回 SqlDataReader 对象
}
```

Command 对象必须与数据库保持连接并进行沟通，Command 对象返回的数据集可以通过 DataSet 类对象获取。在 Command 对象中较为重要的属性有 CommandText 属性、CommandType 属性和 Connetion 属性，其中 CommandText 属性是用于获取或设置数据源运行的文本命令；CommandType 属性用于指定 Command 对象操作数据的方式，是使用 SQL 语句，还是使用存储过程。

示例 21-02　通过 SqlCommand 对象调用存储过程

本示例主要通过 SqlCommand 对象来调用存储过程，以实现数据的删除。程序代码如下：

代码 21-02　光盘位置：光盘\mingrisoft\sl\21\21_02

自定义方法 delete_Proc 用于调用带参数的存储过程，删除指定编号的数据。代码如下：

```
public bool delete_Proc(string strID)
{
    try
    {
        //实例化 SqlCommand 对象，设置存储过程与数据源的连接
        SqlCommand cmd = new SqlCommand("Proc_deleteOfPelple", ConDB());
        cmd.CommandType = CommandType.StoredProcedure;          //指定 SqlCommand 类型为存储过程
        cmd.Parameters.AddWithValue("@tb_PID", strID);          //添加参数
        int i = cmd.ExecuteNonQuery();                          //执行存储过程
        if (i != 0)
            return true;
        else
            return false;
    }
    catch
    {
        return false;
    }
}
```

在 SQL Server 中存储过程 Proc_deleteOfPelple 的设置代码如下：

```
create PROCEDURE Proc_deleteOfPelple
@tb_PID varchar(20)                                    --定义参数
```

```
AS
begin
delete from tb_People where tb_PID=@tb_PID                    --执行 SQL 语句
end
GO
```

按<Ctrl+F5>键运行程序，运行结果如图 22.2 所示。

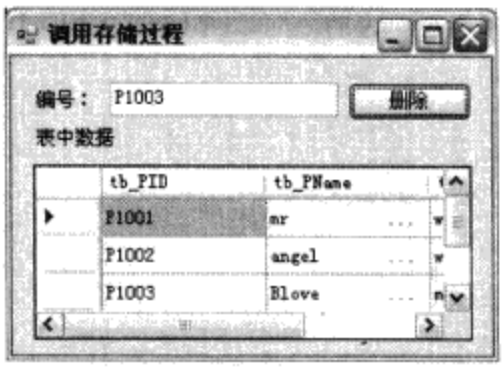


图 22.2　通过 SqlCommand 对象调用存储过程

22.2.3　DataReader 对象

DataReader 对象是一个简单的数据集，用于从数据源中检索只读数据集，常用于检索大量数据。DataReader 对象只允许以只读、从上而下不可逆的方式查看其中所存储的数据，提供一个非常有效率的数据查看模式，同时 DataReader 对象还是一种非常节省资源的数据对象。

注意：每次打开一个新对象，必须关闭前一个 DataReader 对象，因为它是以独占式与数据库交互的，否则，会接收到其产生的异常。

DataReader 对象可通过 Command 对象的 ExecuteReader 方法从数据源中检索数据来创建。下面介绍该对象的常用属性和方法。

DataReader 对象常用属性及说明如表 22.5 所示。

表 22.5　　DataReader 对象常用属性及说明

属　性	说　明
Connection	获取与 SqlDataReader 关联的 SqlConnection
IsClosed	检索一个布尔值，该值指示是否已关闭指定的 SqlDataReader 实例
FieldCount	获取当前行的列数
RecordsAffected	获取执行 SQL 语句所更改、添加或删除的行数
HasRows	获取一个值，该值指示 SqlDataReader 是否包含一行或多行

下面介绍几个主要的属性。

（1）FieldCount 属性。获取当前行中的列数。

语法：

```
public override int FieldCount { get; }
```

属性值：如果未放在有效的记录集中，则为 0；否则为当前行中的列数。默认值为-1。

例如，获取 DataReader 对象中有几行数据，程序代码如下：

```
int Num=Datareader.FieldCount;
```

（2）RecordsAffected 属性。获取执行 Transact-SQL 语句所更改、插入或删除的行数。

语法：

```
public override int RecordsAffected { get; }
```

属性值：已更改、插入或删除的行数；如果没有任何行受到影响或语句失败，则为 0，直到读取了所有行并关闭 SqlDataReader 对象时，才会设置 RecordsAffected 属性。该属性的值是累积值。

例如，如果以批处理模式插入 3 个记录，则 RecordsAffected 属性的值将为 3。

DataReader 对象常用方法及说明如表 22.6 所示。

表 22.6　　DataReader 对象常用方法及说明

方　法	说　明
Read	使 DataReader 对象前进到下一条记录
Close	关闭 DataReader 对象
GetName	获取指定列的名称
GetValue	获取以本机格式表示的指定列的值

下面介绍几个主要的方法。

（1）Read 方法。使 SqlDataReader 前进到下一条记录。

语法：

```
public override bool Read()
```

返回值：布尔型，如果存在多个行，则为 true；否则为 false。

例如，判断 DataReader 对象中是否有记录。代码如下：

```
if (DataReader.Read())                                    //如果 SqlDataReader 对象中有记录
{
    MessageBox.Show("读取记录");
}
```

（2）GetName 方法。获取指定列的名称。

语法：

```
public override string GetName(int i)
```

参数说明如下：

i：Int32，从零开始的列序号。

返回值：String，指定列的名称。

例如，将 DataReader 对象中所有列的列名称存入到数组中。代码如下：

```
string[] MyArray = new string[DataReader.FieldCount];      //定义一个数组
for (int i = 0; i < DataReader.FieldCount; i++)            //遍历字段
{
    MyArray[i] = DataReader.GetName(i);                    //记录字段名
}
```

（3）GetValue 方法。获取以本机格式表示的指定列的值。

语法：

```
public override Object GetValue( int i)
```

参数说明如下：

i：Int32，从零开始的列序号。

返回值：Object，此方法对于空数据库列返回 DBNull。

例如，将 DataReader 对象中所有记录信息按照行和列存入到二维数组中。代码如下：

```
string[,] MyArray = new string[DataReader.VisibleFieldCount, DataReader.FieldCount];  //定义二维数组
int n = 0;                                                                            //用于存储当前记录的行数
while (DataReader.Read())                                                             //读取下一条记录
{
    for (int i = 0; i < DataReader.FieldCount; i++)                                   //遍历当前记录的各列
    {
        MyArray[n, i] = DataReader.GetValue(i).ToString();                            //获取当前记当的列信息
    }
    n++;
}
```

示例 21-03　SqlDataReader 对象的使用

将 SqlDataReader 对象中的列标题名称和数据信息添加到 ListView 控件中，也就是用

ListView 控件显示数据表的内容。程序代码如下：

代码 21-03　光盘位置：光盘\mingrisoft\sl\21\21_03

```
private void Form1_Load(object sender, EventArgs e)
{
    listView1.View = View.Details;                                          //显示列标题
    listView1.FullRowSelect = true;                                         //单击某项时选中行
    SqlConnection conn = new SqlConnection("server=.;database=db_22;uid=sa;pwd="); //连接数据源
    conn.Open();                                                            //打开连接的数据源
    SqlCommand com = new SqlCommand("select * from tb_EmployeeInfo", conn);//设置 SQL 语句与数据库的连接
    SqlDataReader dr = com.ExecuteReader();     //执行 SQL 语句并将结果返回到 SqlDataReader 对象中
    listView1.Items.Clear();                                                //清空当前控件
    for (int i = 0; i < dr.VisibleFieldCount; i++)                          //遍历数据表中的列字段
    {
        listView1.Columns.Add(dr.GetName(i));                               //添加列标题
    }
    while (dr.Read())                                                       //读取表中的行记录
    {
        ListViewItem lt = new ListViewItem(dr.GetValue(0).ToString());      //实例化 ListViewItem 对象
        for (int i = 1; i < dr.VisibleFieldCount; i++)                      //遍历当前记录行的各列信息
        {
            lt.SubItems.Add(dr.GetValue(i).ToString());                     //向集合中添加当前行指定列的信息
        }
        listView1.Items.Add(lt);                                            //添加行记录
    }
    dr.Close();                                                             //关闭
    conn.Close();                                                           //关闭
    listView1.ShowGroups = false;                                           //不以分组方式显示
    listView1.GridLines = true;                                             //在控件上显示网格线
}
```

按<Ctrl+F5>键运行程序，运行结果如图 22.3 所示。

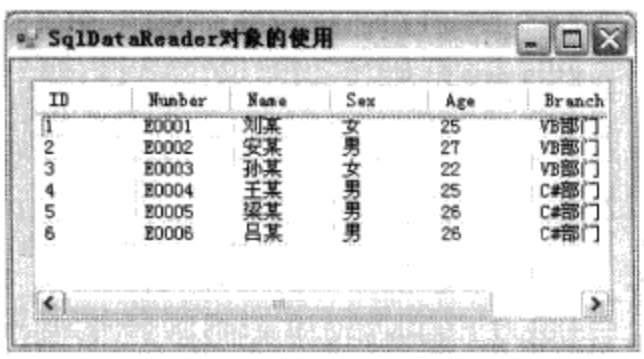


图 22.3　SqlDataReader 对象的使用

22.2.4　DataSet 对象

DataSet 对象是支持 ADO.NET 的断开式、分布式数据方案的核心对象。DataSet 对象是数据的内存驻留表示形式，无论数据源是什么，它都会提供一致的关系编程模型。它可以用于多种不同的数据源，用于 XML 数据，或用于管理应用程序本地的数据。DataSet 对象表示包括相关表、约束和表间关系在内的整个数据集，如图 22.4 所示。

1．DataTableCollection

一个 ADO.NET DataSet 对象包含 DataTable 对象所表示的零个或更多个表的集合。DataTable Collection 类包含 DataSet 对象中的所有 DataTable 对象。

DataTable 对象在 System.Data 命名空间中定义，表示内存驻留数据的单个表。其中包含由 DataColumnCollection 类表示的列集合以及由 ConstraintCollection 类表示的约束集合，这两个集合共同定义表的架构。DataTable 对象还包含 DataRowCollection 类所表示的行的集合，而 DataRowCollection 类则包含表中的数据，除了保存当前状态之外，DataRow 对象还会保留其当前版本和初始版本，以标识对行中存储值的更改。

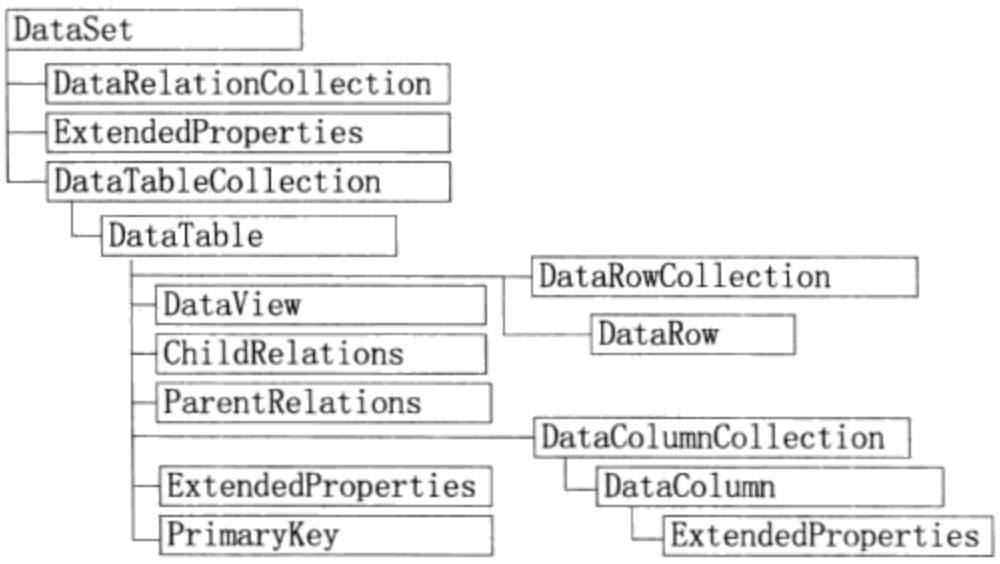


图 22.4　DataSet 对象模型

2．DataRelationCollection

DataSet 对象在其 DataRelationCollection 类中包含关系。关系由 DataRelation 对象来表示，它使一个 DataTable 对象中的行与另一个 DataTable 对象中的行相关联。关系类似于可能存在于关系数据库中的主键列和外键列之间的联接路径。DataRelation 对象标识 DataSet 对象中两个表的匹配列。

关系使用户能够在 DataSet 对象中从一个表导航至另一个表。DataRelation 对象的基本元素为关系的名称、相关表的名称以及每个表中的相关列。关系可以通过一个表的多个列来生成，方法是将一组 DataColumn 对象指定为键列。将关系添加到 DataRelationCollection 类中之后，可以选择添加 UniqueKeyConstraint 对象和 ForeignKeyConstraint 对象，在对相关列的值进行更改时，强制执行完整性约束。

3．ExtendedProperties

DataSet 对象、DataTable 对象和 DataColumn 对象全部具有 ExtendedProperties 属性。Extended Properties 属性是一个 PropertyCollection 对象，可以在其中加入自定义信息，例如，用于生成结果集的 SELECT 语句或生成数据的时间。ExtendedProperties 集合与 DataSet 对象的架构信息一起持久化。

4．从多个 DataAdapter 填充 DataSet

可以将任意数量的 DataAdapter 对象与 DataSet 对象一起使用。每个 DataAdapter 对象都可用于填充一个或多个 DataTable 对象并将更新解析返回给相关数据源。DataRelation 对象和 Constraint 对象可以在本地添加到 DataSet 对象，这样，用户就可以将来自多个不同数据源的数据（如 Sql Server 与 Access）相关联。

DataSet 对象常用属性及说明如表 22.7 所示。

表 22.7　　DataSet 对象常用属性及说明

属　性	说　明
CaseSensitive	获取或设置一个值，该值指示 DataTable 对象中的字符串比较是否区分大小写
Container	获取组件的容器
DataSetName	获取或设置当前 DataSet 对象的名称
DefaultViewManager	获取 DataSet 对象所包含数据的自定义视图，以允许使用自定义的 DataViewManager 属性进行筛选、搜索和导航
DesignMode	获取指示组件当前是否处于设计模式的值
EnforceConstraints	获取或设置一个值，该值指示在尝试执行任何更新操作时是否遵循约束规则

续表

属　性	说　明
ExtendedProperties	获取与 DataSet 对象相关的自定义用户信息的集合
HasErrors	获取一个值，指示在此 DataSet 对象中的任何 DataTable 对象中是否存在错误
IsInitialized	获取一个值，该值表明是否初始化 DataSet 对象
Locale	获取或设置用于比较表中字符串的区域设置信息
Namespace	获取或设置 DataSet 类的命名空间
Prefix	获取或设置一个 XML 前缀，该前缀是 DataSe 类的命名空间的别名
Relations	获取用于将表连接起来并允许从父表浏览到子表的关系的集合
RemotingFormat	为远程处理期间使用的 DataSet 对象获取或设置 SerializationFormat 属性
SchemaSerializationMode	获取或设置 DataSet 对象的 SchemaSerializationMode 属性
Site	获取或设置 DataSet 对象的 System.ComponentModel.Isite 接口
Tables	获取包含在 DataSet 对象中的表的集合

下面介绍几个主要的属性。

（1）DataSetName 属性。获取或设置当前 DataSet 对象的名称。

语法：

```
public stringDataSetName { get; set; }
```

属性值：DataSet 对象的名称。

示例

获取 DataSet 对象的名称。代码如下：

```
DataSet dataSet= newDataSet("HyWork");
MessageBox.Show (dataSet.DataSetName);
```

（2）Relations 属性。获取用于将表连接起来并允许从父表浏览到子表的关系的集合。

语法：

```
public DataRelationCollection Relations { get; }
```

属性值：包含一个 DataRelation 对象集合的 DataRelationCollection 对象。如果 DataRelation 对象不存在，将返回空集合。

例如，通过两个表中的相同字段 tb_PID，利用 DataSet 对象的 Relations 属性将这两个表关联起来，并实例化成新的 DataRelation 对象。代码如下：

```
DataSet DS = newDataSet();
DataRelation relation = DS.Relations.Add("t_People",DS.Tables["t_People"].Columns["tb_PID"],
DS.Tables["t_Class"].Columns["tb_PID"]);
```

（3）Tables 属性。获取包含在 DataSet 对象中的表的集合。

语法：

```
public DataTableCollection Tables { get; }
```

属性值：此 DataSet 对象包含的 DataTableCollection 对象。如果 DataTable 对象不存在，将返回空集合。

例如，显示 DataSet 对象中所有数据表的名称。代码如下：

```
private void ShowTableName(DataSet dataSet)
{
    foreach (DataTable table in dataSet.Tables)
    {
      string str = table.TableName;
      MessageBox.Show(str);
    }
}
```

示例 21-04 用 DataSet 对象读取多表

将多个表存储在 DataSet 对象中，并通过该对象中的表名，将指定的表显示在 DataGridView 控件上。程序代码如下：

代码 21-04 光盘位置：光盘\mingrisoft\sl\21\21_04

自定义方法 ConnDataSet 将两个数据表的内容添加到 DataSet 对象中。代码如下：

```
DataSet dataSet;
public void ConnDataSet()
{
    string strCon = " Server =(local);Database=db_22;User id=sa;PWD=";   //设置连接数据库的字符串
    SqlConnection sqlconn = new SqlConnection(strCon);                   //连接数据源
    sqlconn.Open();                                                      //打开数据源
    //创建一个 SqlDataAdapter 对象，并获取指定数据表的信息
    SqlDataAdapter SQLda = new SqlDataAdapter("select * from tb_People", sqlconn);
    dataSet = new DataSet();                                             //创建 DataSet 对象
    SQLda.Fill(dataSet, "tb_People");                                    //将数据表信息添加到 DataSet 对象中
    SQLda = new SqlDataAdapter("select * from tb_EmployeeInfo", sqlconn);
    SQLda.Fill(dataSet, "tb_EmployeeInfo");
}
```

在窗体的加载事件中将 DataSet 对象中的表名添加到 comboBox1 控件中。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    ConnDataSet();
    foreach (DataTable table in dataSet.Tables)                  //遍历 table 中的所有表名
    {
        comboBox1.Items.Add(table.TableName);                    //将表名添加到 comboBox1 控件中
    }
}
```

在“显示”按钮的单击事件中，通过 DataSet 对象中的表名，在 dataGridView1 控件中显示相应的表数据。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    dataGridView1.DataSource = dataSet.Tables[dataSet.Tables.IndexOf(comboBox1.Text)];
}
```

按<Ctrl+F5>键运行程序，运行结果如图 22.5 所示。

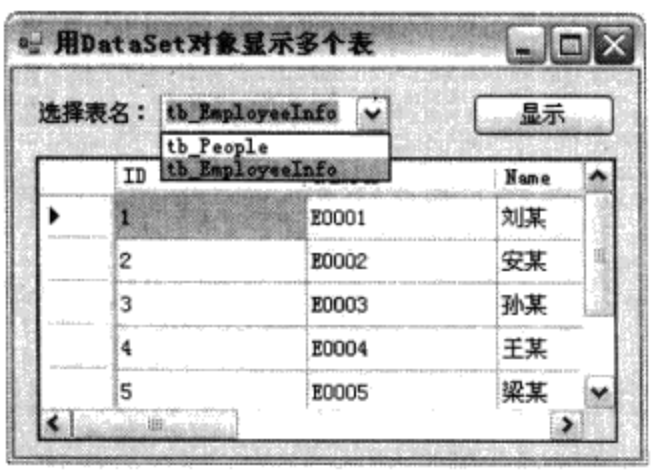


图 22.5 用 DataSet 对象读取多表

DataSet 对象常用方法及说明如表 22.8 所示。

表 22.8 DataSet 对象常用方法及说明

方　法	说　明
AcceptChanges	提交自加载此 DataSet 对象或上次调用 AcceptChanges 方法以来对其进行的所有更改
BeginInit	开始初始化在窗体上使用或由另一个组件使用的 DataSet 对象。初始化发生在运行时
Clear	通过移除所有表中的所有行来清除任何数据的 DataSet 对象
Clone	复制 DataSet 对象的结构，包括所有 DataTable 对象架构、关系和约束。不要复制任何数据
Copy	复制该 DataSet 对象的结构和数据
CreateDataReader	为每个 DataTable 对象返回带有一个结果集的 DataTableReader 对象，顺序与 Tables 集合中表的显示顺序相同
EndInit	结束在窗体上使用或由另一个组件使用的 DataSet 对象的初始化。初始化发生在运行时
GetChanges	获取 DataSet 对象的副本，该副本包含自上次加载以来或自调用 AcceptChanges 方法以来对该数据集进行的所有更改
GetHashCode	用作特定类型的哈希函数。GetHashCode 方法适合在哈希算法和数据结构（如哈希表）中使用
GetObjectData	用序列化 DataSet 对象所需的数据填充序列化信息对象
GetService	获取 IserviceProvider 接口的实施者
GetType	获取当前实例的 Type 属性
GetXml	返回存储在 DataSet 对象中的数据的 XML 表示形式
GetXmlSchema	返回存储在 DataSet 对象中的数据的 XML 表示形式的 XML 架构
HasChanges	获取一个值，该值指示 DataSet 对象是否有更改，包括新增行、已删除的行或已修改的行
InferXmlSchema	将 XML 架构应用于 DataSet 对象
Load	通过所提供的 IdataReader 接口，用某个数据源的值填充 DataSet 对象
Merge	将指定的 DataSet 对象、DataTable 对象或 DataRow 对象的数组合并到当前的 DataSet 对象或 DataTable 对象中
ReadXml	将 XML 架构和数据读入 DataSet 对象
ReadXmlSchema	将 XML 架构读入 DataSet 对象
ReferenceEquals	确定指定的 Object 对象实例是否是相同的实例
RejectChanges	回滚自创建 DataSet 对象以来或上次调用 DataSet.AcceptChanges 方法以来对其进行的所有更改
Reset	将 DataSet 对象重置为初始状态。子类应重写 Reset 方法，以便将 DataSet 对象还原到原始状态
WriteXml	从 DataSet 对象写 XML 数据，还可以选择写架构
WriteXmlSchema	写 XML 架构形式的 DataSet 对象结构

下面介绍几个主要的方法。

（1）ReadXml 方法。使用指定的文件将 XML 架构和数据读入 DataSet 对象。

语法：

```
public XmlReadMode ReadXml(string fileName)
```

参数说明如下。

FileName：读取源的文件名（包括路径）。

返回值：用于读取数据的 XmlReadMode 枚举值。

例如，通过 TryXml.xml 文件填充 DataSet 对象。代码如下：

```
DataSet newDataSet = new DataSet("HyWork");
newDataSet.ReadXml("TryXml.xml");
```

（2）GetXml 方法。返回存储在 DataSet 对象中的数据的 XML 表示形式。

语法：

```
public string GetXml ()
```

返回值：表示存储在 DataSet 对象中的数据的字符串。

例如，将数据集中的数据以 XML 文件形式显示给用户。代码如下：

```
DataSet dataSet = new DataSet("HyWork");                    //实例化 DataSet 对象
DataTable table = dataSet.Tables.Add("Items");              //添加 Items
table.Columns.Add("id", typeof(int));                       //添加 id 并设置类型
table.Columns.Add("Value", typeof(string));                 //添加 Value 关设置类型
DataRow row;
for (int i = 0; i < 3; i++)                                 //添加相应的值
{
    row = table.NewRow();
    row["id"] = i;
    row["Value"] = "this is a number of:" + i;
    table.Rows.Add(row);
}
MessageBox.Show(dataSet.GetXml());                          //显示设置后的 XML 文件
```

执行以上代码后，XML 文件的显示形式如下：

```
<HyWork>
  <Items>
    <id>0</id>
    <Value>this is a number of:0</Value>
  </Items>
  <Items>
    <id>1</id>
    <Value>this is a number of:1</Value>
  </Items>
  <Items>
    <id>2</id>
    <Value>this is a number of:2</Value>
  </Items>
</HyWork>
```

（3）Clear 方法。通过移除所有表中的所有行来清除任何数据的 DataSet 对象。

语法：

```
public void Clear ()
```

例如，清除所有表中所有行的 DataSet 对象。代码如下：

```
DataSet.Clear();
```

DataSet 对象使编程人员可以更灵活的操作数据，因为 DataSet 对象提供了关系 DataRelation 类。通过 DataRelation 类，设计人员可以以编程的方式动态地生成结果集（DataTable），下面示例从多个 DataAdapter 对象填充 DataSet 对象，然后通过 DataRelation 类生成新的 DataTable 对象。

示例 22-05　用 DataSet 对象实现多表关联

通过 DataSet 对象与 DataRelation 对象，将 DataTable 对象中的两个表进行关联。程序代码如下：

代码 22-05　光盘位置：光盘\mingrisoft\sl\22\22_05

```
public DataTable Associate()
{
    //实例 SqlDataAdapter 对象连接到数据源中
    SqlDataAdapter PelAdapter = new SqlDataAdapter("SELECT * FROM tb_People", Conn());
    SqlDataAdapter ClstAdapter = new SqlDataAdapter("SELECT * FROM tb_Class", Conn());
    DataSet DS = new DataSet();                             //实例化 DataSet 数据集
    PelAdapter.Fill(DS, "tb_People");                       //将指定表中的数据清加到 DataSet 对象中
    ClstAdapter.Fill(DS, "tb_Class");
    DataRelation relation = DS.Relations.Add("tb_People",DS.Tables["tb_People"].Columns["tb_PID"],
```

```
DS.Tables["tb_Class"].Columns["tb_PID"]);//实例化关系将 tb_People 表与 tb_Class 表通过 tb_PID 字段关连起来
    DataTable dt = new DataTable("NewTable");                    //实例化 DataTable 对象，用于存放查询结果
    DataColumn dcPName = new DataColumn();                       //实例化 DataColumn 对象
    dcPName.DataType = System.Type.GetType("System.String");     //执行区分大小写的搜索
    dcPName.AllowDBNull = true;                                  //该列允许为空
    dcPName.ColumnName = "姓名";                                 //设置列名称
    dt.Columns.Add(dcPName);                                     //添加指定表的列集合
    dt.Columns.Add("性别", typeof(String));
    DataColumn dcCClase = new DataColumn();                      //实例化 DataColumn 对象
    dcCClase.DataType = System.Type.GetType("System.String");    //执行区分大小写的搜索
    dcCClase.AllowDBNull = true;                                 //该列允许为空
    dcCClase.ColumnName = "班级";                                //设置列名称
    dt.Columns.Add(dcCClase);                                    //添加指定表的列集合
    DataColumn dcCNum = new DataColumn();                        //实例化 DataColumn 对象
    dcCNum.DataType = System.Type.GetType("System.Int32");       //执行区分大小写的搜索
    dcCNum.AllowDBNull = true;                                   //该列允许为空
    dcCNum.ColumnName = "人数";                                  //设置列名称
    dt.Columns.Add(dcCNum);                                      //添加指定表的列集合
    DataRow row;                                                 //实例化 DataRow 对象
    foreach (DataRow pRow in DS.Tables["tb_People"].Rows)        //遍历指定表的行记录
    {
        row = dt.NewRow();                  //通过 DataTabel 对象的 NewRow 方法生成 DataRow 对象
        row["姓名"] = pRow["tb_PName"].ToString().Trim();        //在行中添加指定列的信息
        row["性别"] = pRow["tb_PSex"].ToString();
        foreach (DataRow cRow in pRow.GetChildRows(relation))    //遍历关联表的行记录
        {
            row["班级"] = cRow["tb_CClase"].ToString();          //在行中添加指定列的信息
            row["人数"] = cRow["tb_CNum"].ToString();
        }
        dt.Rows.Add(row);                                        //把行信息添加到 DataTable 对象中
    }
    return dt;                                                   //返回新的结果集(DataTable)
}
```

按<Ctrl+F5>键运行程序，运行结果如图 22.6 所示。

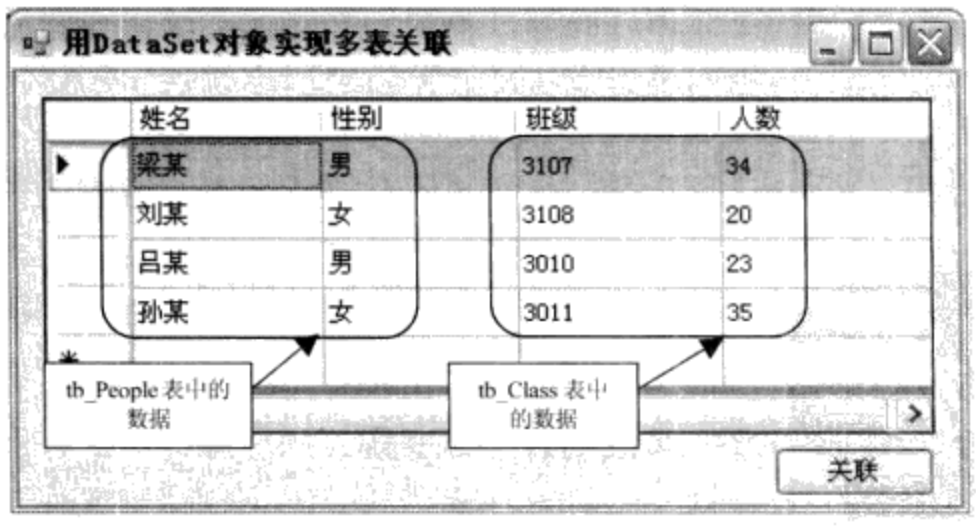


图 22.6　用 DataSet 对象实现多表关联

22.2.5　DataAdapter 对象

SqlDataAdapter 对象是 DataSet 对象和 SQL Server 之间的桥接器，用于检索和保存数据。SqlDataAdapter 对象通过对数据源使用适当的 Transact-SQL 语句映像 Fill 方法（它可更改 DataSet 对象中的数据以匹配数据源中的数据）和 Update 方法（它可更改数据源中的数据以匹配 DataSet 对象中的数据）来提供这一桥接。当 SqlDataAdapter 对象填充 DataSet 对象时，它为返回的数据创建必需的表和列（如果这些表和列尚不存在）。但是，除非将 MissingSchemaAction 属性设置为 AddWithKey，否则这个隐式创建的架构中不包括主键信息。用户也可以使用 FillSchema

方法，让 SqlDataAdapter 对象创建 DataSet 对象的架构，并在用数据填充它之前就将主键信息包含进去。

SqlDataAdapter 对象常用属性及描述如表 22.9 所示。

表 22.9　　SqlDataAdapter 的属性及描述

属　性	描　述
DeleteCommand	获取或设置一个 Transact-SQL 语句或存储过程，以从数据集删除记录
InsertCommand	获取或设置一个 Transact-SQL 语句或存储过程，以在数据源中插入新记录
SelectCommand	获取或设置一个 Transact-SQL 语句或存储过程，用于在数据源中选择记录
UpdateBatchSize	获取或设置每次到服务器的往返过程中处理的行数
UpdateCommand	获取或设置一个 Transact-SQL 语句或存储过程，用于更新数据源中的记录

下面介绍几个主要的属性。

（1）DeleteCommand 属性。获取或设置一个 Transact-SQL 语句或存储过程，用以从数据集删除记录。

语法：

```
public SqlCommand DeleteCommand { get; set; }
```

属性值：在 Update 过程中使用 SqlCommand 对象，用以在数据库中删除对应于 DataSet 对象中已删除行的记录。

例如，为 SqlDataAdapter 对象指定 DeleteCommand 属性。代码如下：

```
adapter.DeleteCommand = command;
```

（2）InsertCommand 属性。获取或设置一个 Transact-SQL 语句或存储过程，用以在数据源中插入新记录。

语法：

```
public SqlCommand InsertCommand { get; set; }
```

属性值：在 Update 过程中使用 SqlCommand 对象，用以在数据库中插入对应于 DataSet 对象中的新的一行记录。

例如，为 SqlDataAdapter 对象指定 InsertCommand 属性，程序代码如下：

```
adapter. InsertCommand= command;
```

（3）SelectCommand 属性。获取或设置一个 Transact-SQL 语句或存储过程，用于在数据源中选择记录。

语法：

```
public SqlCommand SelectCommand { get; set; }
```

属性值：在 Fill 过程中使用的 SqlCommand 对象，用来从数据库中为 DataSet 对象中的位置选择记录。

例如，为 SqlDataAdapter 对象指定 SelectCommand 属性，程序代码如下：

```
adapter.InsertCommand= command;
```

（4）UpdateCommand 属性。获取或设置一个 Transact-SQL 语句或存储过程，用于更新数据源中的记录。

语法：

```
public SqlCommand UpdateCommand { get; set; }
```

属性值：在 Update 过程中使用的 SqlCommand 对象，用于在数据库中更新对应于 DataSet 对象中已修改行的记录。

例如，为 SqlDataAdapter 对象指定 UpdateCommand 属性，程序代码如下：

```
adapter.UpdateCommand= command;
```

示例 22-06　DataAdapter 对象以及属性的应用

根据用户编写完的一条 SQL 语句，通过自定义方法 TransactSQL 获得 SQL 语句要实现的功能，然后用 TransactSQL 方法所返回 DataAdapter 的对象执行 SQL 语句。程序代码如下。

代码 22-06　光盘位置：光盘\mingrisoft\sl\22\22_06

自定义类 ClsDB 主要用于实现数据库的连接。代码如下：

```
class ClsDB
{
    private SqlConnection con;                                  //定义一个 SqlConnection 对象
    public SqlConnection Conn()                                 //自定义一个方法用于实现数据库的连接
    {
        con = new SqlConnection("server=.;pwd=;uid=sa;database=db_22");     //连接数据源
        con.Open();                                             //打开连连的数据源
        return con;                                             //返回连接数据源后的 SqlConnection 对象
    }
}
```

自定义方法 TransactSQL 主要判断 SQL 语句是添加、修改、删除或查询，然后将语句存入 SqlDataAdapter 对象的相应属性中，并返回设置后的 SqlDataAdapter 对象。代码如下：

```
public static SqlDataAdapter TransactSQL(string SqlSel, SqlConnection connection)
{
    SqlDataAdapter adapter = new SqlDataAdapter();              //实例化 SqlDataAdapter 对象
    //实例化 SqlCommand 对象，实现 SQL 语句与数据源的连接
    SqlCommand command = new SqlCommand(SqlSel, connection);
    if (SqlSel.IndexOf("insert") != -1)                         //判断 SQL 语句是否为添加语句
    {
        adapter.InsertCommand = command;                        //在添加属性中设置 SQL 语句
    }
    else if (SqlSel.IndexOf("update") != -1)                    //判断 SQL 语句是否为更新语句
    {
        adapter.UpdateCommand = command;                        //在更新属性中设置 SQL 语句
    }
    else if (SqlSel.IndexOf("select") != -1)                    //判断 SQL 语句是否为查询语句
    {
        adapter.SelectCommand = command;                        //在查询属性中设置 SQL 语句
    }
    else if (SqlSel.IndexOf("delete") != -1)                    //判断 SQL 语句是否为删除语句
    {
        adapter.DeleteCommand = command;                        //在删除属性中设置 SQL 语句
    }
    else
    {
        MessageBox.Show("SQL 语句不合法 ");
        return adapter;
    }
    adapter.SelectCommand = command;                            //设置数据源的选择记录
    return adapter;
}
```

注意：在用相应的 SQL 语句设置 SqlDataAdapter 对象的 InsertCommand、UpdateCommand 和 DeleteCommand 属性后，必须用 SQL 语句设置 SelectCommand 属性，否则将无法执行 SQL 语句。

“适配器”按钮的单击事件主要将设置的 SQL 语句传递到 TransactSQL 方法中，然后执行该方法所返回的 SqlDataAdapter 对象。代码如下：

```
SqlDataAdapter da;                                              //定义一个 SqlDataAdapter 对象
private void button1_Click(object sender, EventArgs e)
{
    ClsDB cb = new ClsDB();                                     //实例化自定义类 ClsDB
    DataTable dt = new DataTable("Resouce");                    //实例化 DataTable 对象
    dt.Clear();                                                 //清空 DataTable 对象
```

```
        if (textBox1.Text.Trim() != "")                          //如果设置了 SQL 语句
        {
            da = TransactSQL(textBox1.Text.Trim(), cb.Conn());   //调用自定义方法 TransactSQL
        }
        try
        {
            da.Fill(dt);                          //执行 SQL 语句，并将数据添加到 DataTable 对象中
            MessageBox.Show("OK");
        }
        catch { MessageBox.Show("Error"); }
        this.dataGridView1.DataSource = dt.DefaultView;          //显示信息
    }
```

按<Ctrl+F5>键运行程序，运行结果如图 22.7 所示。

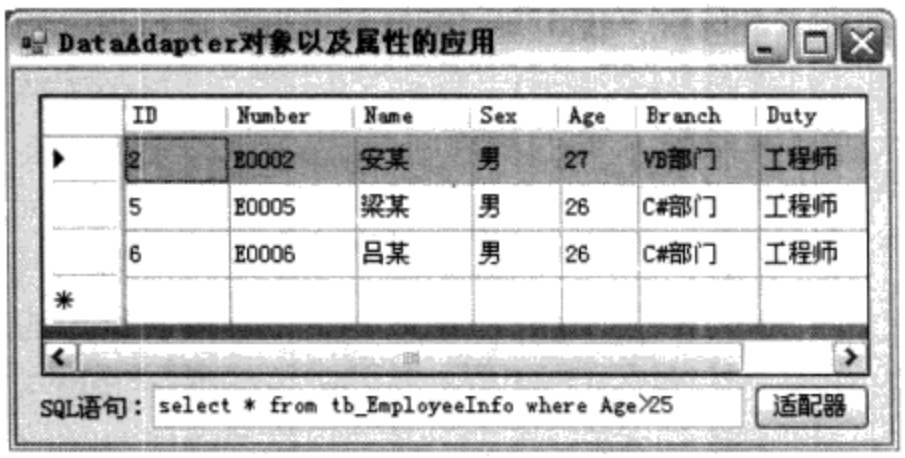


图 22.7　DataAdapter 对象以及属性的应用

注意：在使用 SqlDataAdapter 对象时，必须引用 using System.Data.SqlClient 命名空间。

22.3　数据库开发基本操作

下面以 SQL Server 为例讲解在编程时如何使用数据库进行开发。

22.3.1　打开数据库连接

实例位置：光盘\mingrisoft\dxyy\22\22_01

通过 SqlConnection 类完成数据库（SQL Server）的连接与断开，将与后台数据库有关的操作放在一个类中。程序运行结果如图 22.8 与图 22.9 所示。

图 22.8　连接数据库窗体

图 22.9　数据库状态提示窗体

程序开发步骤如下。

（1）打开 VS2008，然后新建一个应用程序（ConnData）。

（2）在解决方案管理器内添加一个文件夹 ClsDB 用于存放操作数据库的类，并在该文件夹中添加一个 ClsDBControl.cs 类，用于返回一个打开的数据库连接对象。

（3）主要程序程序代码如下。

在类文件 ClsDBControl 中，自定义一个 SqlConnection 对象的局部变量，用于连接指定的数据源。代码如下：

```
private SqlConnection con = new SqlConnection("server=.;uid=sa;pwd=;database=db_22");
```

在类文件 ClsDBControl 中，自定义一个 ConDB 方法，用于打开与数据库的连接并返回一个 SqlConnection 对象。代码如下：

```
public SqlConnection ConDB()
{
    if (con.State == ConnectionState.Closed)          //如要当前连接的数据库处于断开状态
    {
        con.Open();                                   //打开连接的数据库
    }
    return con;                                       //返回 SqlConnection 对象
}
```

在类文件 ClsDBControl 中，自定义一个 CloseDB 方法，用于断开与数据库的连接并返回一个 SqlConnection 对象。代码如下：

```
public SqlConnection CloseDB()
{
    if (con.State == ConnectionState.Open)            //如要当前连接的数据库处于打开状态
    {
        con.Close();                                  //断开连接的数据库
    }
    return con;
}
```

在“连接”按钮的单击事件中通过调用 ConDB 方法来实现数据库的连接。代码如下：

```
ClsDB.ClsDBControl Con = new OptDB.ClsDB.ClsDBControl();      //实例化 ClsDBControl 类
private void button1_Click(object sender, EventArgs e)
{
    SqlConnection conTry=Con.ConDB();                         //打开连接的数据库
    MessageBox.Show(conTry.State.ToString());                 //显示当前连接对象的状态
}
```

在“断开”按钮的单击事件中通过调用 CloseDB 方法来实现数据库的断开。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    SqlConnection conTry = Con.CloseDB();                     //断开连接的数据库
    MessageBox.Show(conTry.State.ToString());                 //显示当前连接对象的状态
}
```

注意：在使用 SqlConnection 对象时必须引用 using System.Data.SqlClient 命名空间。

22.3.2 向数据库中添加数据

在制作一些简单的应用程序时，可以直接用 SQL 语句对数据表进行更新操作。本节主要通过 SqlCommand 对象执行开发者手动编写的 SQL 语句，以实现数据表的更新操作。

实例位置：光盘\mingrisoft\dxyy\22\22_02

通过 SqlCommand 类的 CommandText 方法（为其构造 insert 语句）与 ExecuteNonQuery 方法实现向表中插入数据的功能，程序运行结果如图 22.10 与图 22.11 所示。

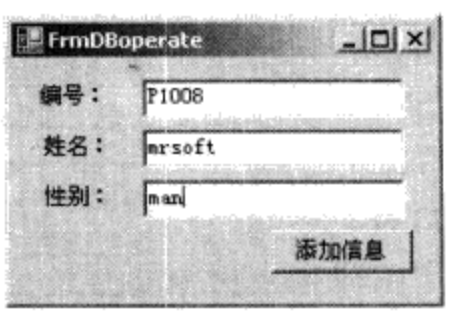


图 22.10 添加信息窗体

图 22.11 提示窗体

程序开发步骤如下。

（1）新建一个 Windows 应用程序（DataInsert），设置窗体名称为 FrmDBoperate，在窗体上添加相应的控件。

（2）在 ClsDBControl.cs 类中完成向数据库中添加数据的方法 AmendDB。

注意：ClsDBControl.cs 类的设置请参见 22.3.1 小节。

（3）主要程序代码如下。

自定义方法 AmendDB 主要通过参数 sql（SQL 语句）来构建 SqlCommand 对象，然后通过该对象的 ExecuteNonQuery 方法执行 SQL 语句。代码如下：

```
public bool AmendDB(String sql)
{
    using (SqlCommand cmd = new SqlCommand())            //实例化 SqlCommand 对象
    {
        try
        {
            cmd.CommandText = sql;                       //设置 SQL 语句
            cmd.Connection = ConDB();                    //调用打开数据库连接方法
            cmd.ExecuteNonQuery();                       //执行 SQL 语句
            return true;
        }
        catch
        {
            return false;
        }
    }
}
```

交叉链接：ConDB 方法请参见 22.3.1 小节。

单击“添加信息”按钮，将实现添加功能的 SQL 语句传给 AmendDB 方法，以实现数据的添加。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    ClsDB.ClsDBControl DBC = new DataInsert.ClsDB.ClsDBControl();
    //设置插入功能的 SQL 语句
    string strSql = "insert into tb_people values('" + this.textBox1.Text.Trim().ToString() + "','" +
this.textBox2.Text.Trim().ToString() + "','" + this.textBox3.Text.Trim().ToString() + "')";
    if (DBC. AmendDB(strSql))                            //执行插入操作
    {
        MessageBox.Show("OK");
    }
}
```

注意：在添加信息时，SQL 中的引用变量的格式为 '" +变量+ "'。

这里要说明的是自定义方法 AmendDB 可以执行任意格式的 SQL 语句，下面分别介绍修改和删除语句的设置。

修改语句的设置如下：

```
string strSql = "update t_people set tb_psex='" + this.textBox3.Text.Trim().ToString() +"',tb_pname= '" +
this.textBox2.Text.Trim().ToString() + "'   where tb_PID='" + this.textBox1.Text.Trim().ToString() + "'";
```

删除语句的设置如下：

```
string strSql = "delete from t_people where tb_PID='" + this.textBox1.Text.Trim().ToString() + "'";
```

22.3.3 修改数据库中数据

在前一节中已经讲解了如何直接用 SQL 语句对数据表进行添加、修改和删除的操作，该方法只适合于一些简单的数据操作。如果在一个工程中频繁地对数据表进行更新操作，要根据不同的数据表及操作方式编写不同的 SQL 语句，这样编写程序是十分笨拙的。

为了便于对不同的数据表进行更新操作，本节将填写更改信息的控件的名称设置为与表中字段名相关的名称（控件名中包含字段名），以便于通过控件名设置添加或修改的 SQL 语句。

实例位置：光盘\mingrisoft\dxyy\22\22_03

在对数据库中的数据进行修改时，为了便于对不同的数据表中的数据进行修改，本实例将通过对控件名的设置来获取指定表的字段信息，程序运行结果如图 22.12、图 22.13 和图 22.14 所示。

图 22.12　提示窗体

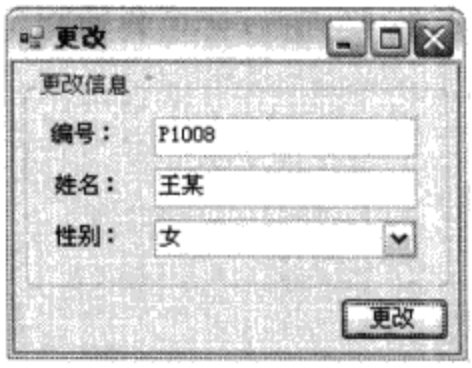


图 22.13　修改信息窗体

图 22.14　提示窗体

程序开发步骤如下。

（1）新建一个 Windows 应用程序（DataUpdate），设置窗体名称为 FrmUpdate，在窗体上添加相应的控件。

说明：在窗体上添加与表中字段相应的控件时，将控件的 Name 属性设置为“F_”+相应的表中的字段名，如 F_tb_PID（tb_PID 为 tb_People 表中的字段名）。

（2）在 ClsDBControl.cs 类中完成修改数据库中数据的方法 AmendDB。

（3）主要程序代码如下。

自定义方法 AmendDB 主要通过 Cont 参数获取指定容器中的控件集，然后对控件集中的控件进行遍历，获取指定数据表中的字段名和值，再通过参数 Sign 判断当前执行的是添加操作，还是更新操作，并生成相应的 SQL 语句，最后通过 SqlCommand 对象执行 SQL 语句。参数 TableName 表示修改数据表的名称，参数 Cont 表示控件集，参数 KeyWord 表示查询的条件字段（这里为条件字段的控件名），参数 Sign 是一个标识，如果为 0 表示添加数据，如果为 1 表示修改数据。代码如下。

```
public bool AmendDB(String TableName, Control.ControlCollection Cont, string KeyWord, int Sign)
{
    string FieldName = "";                                  //记录字段名
    string FieldValue = "";                                 //记录字段值
    string UpDataWhere = "";                                //记当更新数据的条件
    string Tem_Name = "";                                   //记录临时的单个字段名
    string Tem_Value = "";                                  //记录临时的单个字段值
    string sql = "";                                        //记录组合的 SQL 语句
    bool tem_b = false;                                     //判断控件名是否为字段名称
    foreach (Control C in Cont)                             //遍历指定容器下的控件
    {
        tem_b = false;                                      //设置初始值
        Tem_Name = "";                                      //设置初始值
        if (C.GetType().Name == "TextBox")                  //如果当前为 TextBox 控件
        {
            Tem_Name = ((TextBox)C).Name;                   //获取控件名称
            Tem_Value = ((TextBox)C).Text;                  //获取控件的值
            tem_b = true;
        }
        if (C.GetType().Name == "ComboBox")                 //如果当前为 ComboBox 控件
        {
            Tem_Name = ((ComboBox)C).Name;                  //获取控件名称
            Tem_Value = ((ComboBox)C).Text;                 //获取控件的值
            tem_b = true;
        }
        if (tem_b && Tem_Name.Length > 0)                   //如果当前控件符合要求
```

```
        {
            if (Tem_Name.Trim() == KeyWord)                                  //如果设置了主键控件
            {
                UpDataWhere = " WHERE " + Tem_Name.Substring(2, Tem_Name.Length - 2) + "='" +
Tem_Value + "'";                                                             //设置更新条件
                continue;
            }
            if (Tem_Name.Substring(0, 2) == "F_")                            //如果存在表中的字段名
            {
                FieldName += Tem_Name.Substring(2, Tem_Name.Length - 2) + ",";  //获取字段名
                FieldValue += "'" + Tem_Value + "',";                        //获取值
            }
        }
    }
    switch (Sign)
    {
        case 0:                                                              //添加操作
            {
                sql = "INSERT INTO " + TableName + " (" + FieldName.Substring(0, FieldName.Length - 1) + ")
values (" + FieldValue.Substring(0, FieldValue.Length - 1) + ")"; //设置 INSERT 语句
                break;
            }
        case 1:                                                              //修改操作
            {
                FieldName = FieldName.Substring(0, FieldName.Length - 1);    //取去字段组的最后一个逗号
                FieldValue = FieldValue.Substring(0, FieldValue.Length - 1);  //取去值组的最后一个逗号
                string[] ArrayName = FieldName.Split(',');                   //将字段名存入到数组中
                string[] ArrayValue = FieldValue.Split(',');                 //将值存入到数组中
                Tem_Name = "";                                               //清空
                for (int i = 0; i < ArrayName.Length; i++)                   //遍历数组
                {
                    Tem_Name += ArrayName[i] + "=" + ArrayValue[i] + ",";//将字段名与相应的值进行组合
                }
                sql = "UPDATE " + TableName + " SET " + Tem_Name.Substring(0, Tem_Name.Length - 1) +
UpDataWhere;                                                                 //设置 UPDATE 语句
                break;
            }
    }
    if (sql.Length == 0)                                                     //如果没有生成 SQL 语句
    {
        MessageBox.Show("没有设置更新方式");
        return false;
    }
    using (SqlCommand cmd = new SqlCommand())                                //实例化 SqlCommand 对象
    {
        try
        {
            cmd.CommandText = sql;                                           //设置 SQL 语句
            cmd.Connection = ConDB();                                        //调用打开数据库连接方法
            int n = cmd.ExecuteNonQuery();                                   //执行 SQL 语句
            if (n == 0)
            {
                MessageBox.Show("无更新记录");
                return false;
            }
            return true;
        }
        catch                                                                //发生异常
        {
            MessageBox.Show("SQL 语句不正确");
            return false;
        }
    }
}
```

注意：在使用该方法时，由于使用了 Control 对象和 SqlCommand 对象，必须引用 using System.Data.SqlClient 和 using System.Windows.Forms 命名空间。

单击“修改”按钮，将实现修改操作的表名、相关控件集、条件字段的控件名，以及标识传给 AmendDB 方法，通过该方法组合成 SQL 语句，并执行更改操作。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (this.F_tb_PID.Text.Trim().ToString() != "")                    //如果输入的了编号
    {
        ClsDB.ClsDBControl DBC = new DataUpdate.ClsDB.ClsDBControl();  //实例化 ClsDBControl 类
        if (DBC.AmendDB("tb_People", groupBox1.Controls, F_tb_PID.Name, 1))  //调用 AmendDB 方法
        {
            MessageBox.Show("OK");
        }
    }
    else                                                               //没有输入条件字段
    {
        MessageBox.Show("请写入编号信息");
    }
}
```

这里要说明的是自定义方法 AmendDB 不但可以执行修改操作，也可以执行添加操作，下面介绍添加操作的调用语句：

```
AmendDB("tb_People", groupBox1.Controls, "", 0);
```

说明：本实例只是为读者提供了一个便于更新数据的通用方法，读者可以根据不同的环境设置不同的方法。

22.3.4 删除数据库中数据

删除表中的数据可以使用 22.3.2 节中介绍的通过简单的 SQL 语句来实现删除操作，本节主要介绍如何根据不同的数据表对查询后的数据进行删除操作。

实例位置：光盘\mingrisoft\dxyy\22\22_04

在对数据库中的数据进行删除时，为了便于对数据表中的数据进行删除，可以自动获取数据表中的字段名，并根据删除条件显示要删除的数据，最后对查询后的数据进行删除，程序运行结果如图 22.15 所示。

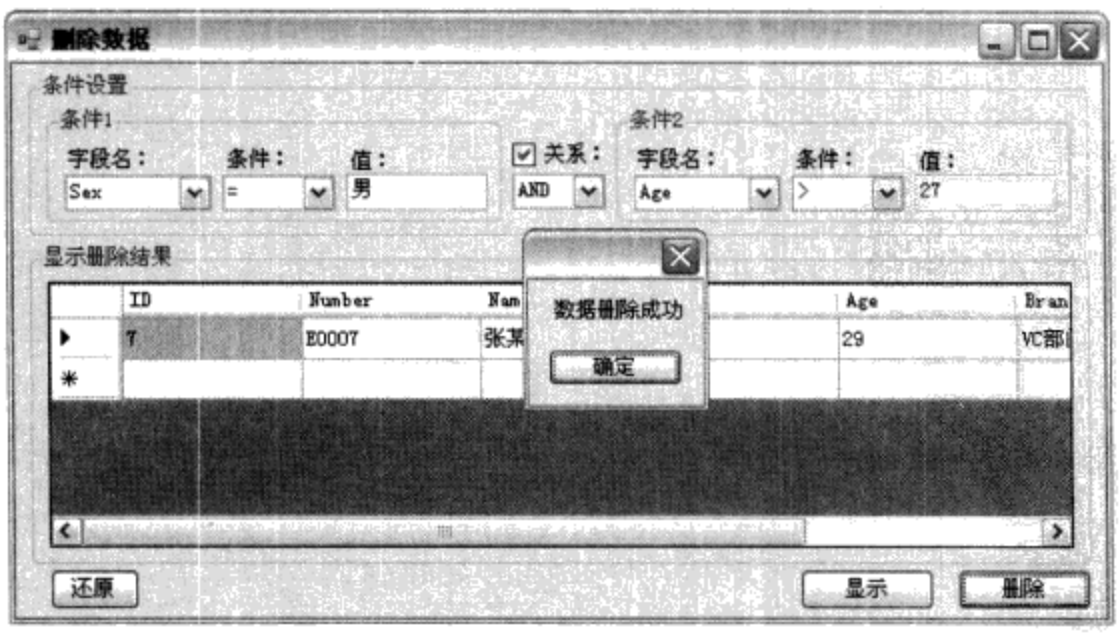


图 22.15 删除查找后的信息

程序开发步骤如下。

（1）新建一个 Windows 应用程序（DataDelete），设置窗体名称为 Frm_Main，在窗体上添加 1 个 DataGridView 控件、2 个 Button 控件。

（2）在工程中添加一个窗体，设置窗体的名称为 Frm_Delete，在窗体上添加控件，如图 22.15 所示。这里要注意的是，在“条件 1”和“条件 2”容器中，将设置“字段名”、“条件”和“值”的控件（ComboBox 或 TextBos 控件）的 Tag 属性分别设为 0、1、2，以便于组合查询条件。

（3）在 ClsDBControl.cs 类中设置相关的方法。

（4）主要程序代码如下。

● ClsDBControl.cs 类中主要方法的相关设置。

自定义方法 CoPassData 用于将指定数据表中的字段名添加到 ComboBox 控件中，参数 cobox 表示要显示字段名的 ComboBox 控件，参数 sql 表示具有查询功能的 SQL 语句，参数 AliasName 表示数据表的别名。代码如下：

```
public string CoPassData(ComboBox cobox, string sql, string AliasName)
{
    string SqlStr = "select * from (" + sql + ") as " + AliasName;//将查询语句以表别名的形式生成新的查询语句
    DataSet DSet = new DataSet();                                  //实例化 DataSet 对象
    DSet = SelectDB(SqlStr);                                       //执行查询功能的 SQL 语句
    cobox.Items.Clear();                                           //清空 ComboBox 控件
    for (int i = 0; i < DSet.Tables[0].Columns.Count; i++)         //将数据表的字段名称添加到 ComboBox 控件中
    {
        cobox.Items.Add(DSet.Tables[0].Columns[i].Caption);//添加表中的字段
    }
    return SqlStr;
}
```

自定义方法 SelectDB 用于实现数据表的查询功能，将查询的结果存入 DataSet 对象中并返回。参数 sql 表示具有查询功能的 SQL 语句。代码如下：

```
public DataSet SelectDB(String sql)
{
    DataSet dataSet = new DataSet();                                  //创建 DataSet 对象
    try
    {
        SqlDataAdapter SQLda = new SqlDataAdapter(sql, ConDB()); //建立 SQL 语句与数据库的连接
        SQLda.Fill(dataSet);                                          //将数据表信息添加到 DataSet 对象中
        return dataSet;
    }
    catch                                                             //发生异常
    {
        MessageBox.Show("SQL 语句不正确");
        return dataSet;
    }
}
```

自定义方法 AssortedTerm 主要遍历控件集中的相关控件，组合成具有子查询的查询条件。参数 Con 表示控件集，参数 TableName 表示表名，参数 AliasName 表示表别名。代码如下：

```
public string AssortedTerm(Control.ControlCollection Con, string TableName, string AliasName)
{
    string[] Array_Assorted = new string[3];                  /定义数组，用于存储查询条件的字段名、条件和值
    string tem_Value = "";                                    //记录查询条件
    foreach (Control C in Con)                                //遍历控件集
    {
        if (C.GetType().Name == "ComboBox")                   //如果是 ComboBox 控件
        {
            if ( ((ComboBox)C).Text.Trim().Length == 0))      //如果该控件为空，并且 Tag 属性值小于等于 1
            {
                return "";                                    //返回空值
            }
              //根据控件的 Tag 属性在数组中记录相应的字段名和条件
            Array_Assorted[Convert.ToInt32(((ComboBox)C).Tag.ToString())] = ((ComboBox)C).Text;
        }
        if (C.GetType().Name == "TextBox")                    //如果是 TextBox 控件
        {
```

```
                //根据控件的 Tag 属性在数组中记录相应的条件值
                Array_Assorted[Convert.ToInt32(((TextBox)C).Tag.ToString())] = ((TextBox)C).Text;
            }
        }
        //组合与 AliasName 表相关的查询条件
        tem_Value = " " + AliasName + "." + Array_Assorted[0] + Array_Assorted[1] + "'" + Array_Assorted[2] + "'";
        if (Array_Assorted[1].Trim() == "=" && Array_Assorted[2].Trim().Length == 0)      //如果值为空
        {
            tem_Value = "(" + tem_Value + " or " + AliasName + "." + Array_Assorted[0] + Array_Assorted[1] + "null" +
")";                                                    //设置查询空值的条件语句，空值分为""和 null
        }
        tem_Value = "(" + TableName + "." + Array_Assorted[0] + "=" + AliasName + "." + Array_Assorted[0] + " and " +
tem_Value + ")";                                        //将 TableName 表和 AliasName 的查询条件相关联
        return tem_Value;                               //返回查询条件的语句
    }
```

● 主窗体（Frm_Main）的设置。

在窗体的加载事件中调用 ClsDBControl 的 SelectDB 方法，显示要删除的数据表信息。代码如下：

```
ClsDB.ClsDBControl ClsDB = new DataDelete.ClsDB.ClsDBControl();//实例化 ClsDBControl 类
DataSet dataSet = new DataSet();                        //实例化 DataSet 类
private void Frm_Main_Load(object sender, EventArgs e)
{
    dataSet = ClsDB.SelectDB("select * from tb_EmployeeInfo");     //调用 SelectDB 方法获取删除表中的信息
    if (dataSet.Tables.Count > 0)                       //如果 SQL 语句正确
        dataGridView1.DataSource = dataSet.Tables[0];   //显示数据
}
```

在“删除”按钮的单击事件中调用“删除数据”窗体。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Frm_Delete F_Delete = new Frm_Delete();             //实例化 Frm_Delete 窗体
    F_Delete.DeleteWhere = "select * from tb_EmployeeInfo";     //设置要进行删除的查询语句
    F_Delete.DeleteAlias = "tb_EmployeeInfo_Delete";    //设置表别名
    F_Delete.DeleteTableName = "tb_EmployeeInfo";       //设置要进行删除的表名
    if (F_Delete.ShowDialog() == DialogResult.OK)       //调用窗体，如果窗体的返回值为 OK
    {
        dataSet = ClsDB.SelectDB("select * from tb_EmployeeInfo");//获取删除后的数据
        dataGridView1.DataSource = dataSet.Tables[0];   //显示数据
    }
}
```

● 删除窗体（Frm_Delete）的设置。

设置相关变量。代码如下：

```
public string DeleteWhere = "";                         //记录要进行删除的查询语句
public string DeleteAlias = "";                         //记录表别名
public string DeleteTableName = "";                     //记录要进行删除的表名
string Var_Rude = "";                                   //记录利用表别名生成的 SQL 查询语句
string Var_Sql = "";                                    //记录生成后的 SQL 删除语句
ClsDB.ClsDBControl ClsDB = new DataDelete.ClsDB.ClsDBControl();//实例化 ClsDBControl 类
DataSet dataSet = new DataSet();                        //实例化 DataSet 类
```

在窗体的加载事件中获取指定数据表的字段名，并将其添加到相应的 ComboBox 控件中。代码如下：

```
private void Frm_Delete_Load(object sender, EventArgs e)
{
    //判断调用当前窗体时，是否传递了相应的参数
    if (DeleteWhere.Length == 0 || DeleteAlias.Length == 0 || DeleteTableName.Length == 0)
    {
        MessageBox.Show("没有传递相应的参数，无法显示删除信息");
        Close();
    }
    else                                                //传递了相应的参数
```

```
    {
        dataSet = ClsDB.SelectDB(DeleteWhere);                          //获取要删除的数据
        dataGridView1.DataSource = dataSet.Tables[0];                   //显示数据
        //将表中的字段名添加到 comboBox1 控件中
        Var_Rude = ClsDB.CoPassData(comboBox1, DeleteWhere, DeleteAlias);
        comboBox1.SelectedIndex = 0;                                    //显示第一个选定项
    }
}
```

“显示”按钮的单击事件用于组合完整的 SQL 查询语句，并进行显示。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string tem_Sql = "";                                                //记录临时的 SQL 语句
    Var_Sql = "";                                                       //记录 SQL 语句
    //生成第一个条件的 SQL 语句
    tem_Sql = ClsDB.AssortedTerm(groupBox2.Controls,DeleteTableName, DeleteAlias);
    if (tem_Sql.Trim().Length != 0)                                     //如果 SQL 语句正确
    {
        Var_Sql = tem_Sql;                                              //记录条件语句
        tem_Sql = "";
        if (checkBox1.Checked == true)                                  //如果要设置第二个查询条件
        {
            //生成第二个条件的 SQL 语句
            tem_Sql = ClsDB.AssortedTerm(groupBox3.Controls,DeleteTableName, DeleteAlias);
            //如果设置了查询条件
            if (tem_Sql.Trim().Length != 0 && comboB_Logistic.Text.Trim().Length > 0)
            {
                Var_Sql = Var_Sql + " " + comboB_Logistic.Text + " " + tem_Sql; //组合查询条件
            }
            else
            {
                Var_Sql = "";
                MessageBox.Show("请设置查询条件");
            }
        }
    }
    else
    {
        Var_Sql = "";
        MessageBox.Show("请设置查询条件");
    }
    if (Var_Sql.Trim().Length != 0)                                     //如果不为空，表示查询条件正确
    {
        Var_Sql = Var_Rude + " WHERE " + Var_Sql;                       //设置子查询
        //设置子查询与指定数据表的关联
        Var_Sql = "from " + DeleteTableName + " WHERE EXISTS (" + Var_Sql + ")";
        dataSet= ClsDB.SelectDB("select * "+Var_Sql);                   //获取查找的数据
        dataGridView1.DataSource = dataSet.Tables[0];                   //显示数据
    }
}
```

“删除”按钮的单击事件用于组合完整的 SQL 删除语句。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (Var_Sql.Trim().Length == 0)                                     //如果生成了子查询语句
    {
        //将删删除语句和子查询语句建立关联
        ClsDB.AmendDB("Delete from " + DeleteTableName + " WHERE EXISTS (" + DeleteWhere + ")");
    }
    else
    {
        ClsDB.AmendDB("Delete " + Var_Sql);                             //删除整个数据表
    }
    MessageBox.Show("数据删除成功");                                    //弹出提示对话框
    this.DialogResult = DialogResult.OK;                                //设置对话框的返回值
```

```
        Var_Sql = "";
    }
```

注意：SQL 语句中通过子查询作为删除条件，主要是用关键字 EXISTS 连接的。例如，本实例所生成的 SQL 删除语句如下：

```
SELECT *
FROM tb_EmployeeInfo
WHERE EXISTS
    (SELECT *
     FROM
        (SELECT *
         FROM tb_EmployeeInfo)
         AS tb_EmployeeInfo_Delete
         WHERE (tb_EmployeeInfo.Sex = tb_EmployeeInfo_Delete.Sex
         AND tb_EmployeeInfo_Delete.Sex='男')
         AND (tb_EmployeeInfo.Age = tb_EmployeeInfo_Delete.Age
         AND tb_EmployeeInfo_Delete.Age>'26')
    )
```

以上代码中加粗部分也可以用关键字 EXISTS 连接一个子查询语句。

说明：本实例只是为读者提供了一个便于删除查询数据的通用方法，读者可以根据不同的环境设置不同的方法。

这里要说明的是“删除窗体”中的“显示”按钮，该按钮实现的是按指定条件对数据表进行查询功能，也就是说也可以通过该实例对数据表执行查询操作。

22.3.5 查询数据库中数据

在 22.3.2 小节和 22.3.4 小节中分别讲解了数据表的简单查询和复杂查询，本节主要对查询的结果进行单记录的显示。

实例位置：光盘\mingrisoft\dxyy\22\22_05

本实例主要用 SqlDataAdapter 类执行 SQL 查询语句，并将结果以行的形式填充到 DataSet 对象中，然后通过指定的按钮以上下滚动的形式进行显示。程序运行结果如图 22.16、图 22.17 和图 22.18 所示。

图 22.16 查询窗体

图 22.17 提示窗体

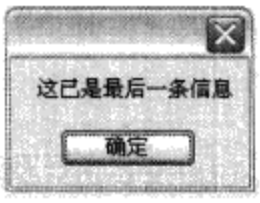


图 22.18 提示窗体

程序开发步骤如下。

（1）新建一个 windows 应用程序（DataSelect），设置窗体名称为 FrmSelect，在窗体上添加相应的控件（如图 22.16 所示）。

（2）在 ClsDBControl.cs 类中完成查询数据的方法 dsResult 并返回到 DataSet 对象中，以及获得表中行数的方法 Max。

（3）主要程序代码如下。

自定义方法 dsResult 主要通过 SqlDataAdapter 类的 Fill 方法，将查询结果存入到 dataSet 对象中并返回。代码如下：

```
Public DataSet dsResult(int currentIndex)
{
    int pageSize = 1;
    string orderSQL = "SELECT * FROM t_People ORDER BY tb_PID";//设置 SQL 查询语句
```

```
    SqlDataAdapter adapter = new SqlDataAdapter(orderSQL, ConDB());//建立 SQL 语句与数据库的连接
    DataSet dataSet = new DataSet("t_People");                      //实例化 dataSet 类
    adapter.Fill(dataSet, currentIndex, pageSize, "t_People");      //将查的数据表添加到 dataSet 类中
    returnDataSet;
}
```

自定义方法 Max 用于获得表中记录的总数。代码如下：

```
public int Max()
{
    SqlCommandcmd = new SqlCommand("select count(*) from t_people", ConDB());//获取查询数据的行数
    return int.Parse(cmd.ExecuteScalar().ToString());
}
```

在窗体加载事件中通过数据操作类 ClsDBControl 中的 Max 方法获得总行数，用于初始化变量 LastNum。代码如下：

```
public static int i = 0;                                          //存储当前记录的索引号
public static int LastNum;                                        //记录数据的总个数
ClsDB.ClsDBControl ClsDBC = new DataSelect.ClsDB.ClsDBControl();  //实例化 ClsDBControl 类
private void FrmSelect_Load(object sender, EventArgs e)
{
    LastNum = CDBLast.Max();                                      //获取查询数据的总个数
}
```

“第一条”按钮的单击事件用于显示首记录的信息。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    i = 0;                                                        //第一条记录的索引
    DataSet dsNew = ClsDBC.dsResult(i);                           //获取第一第记录的信息
    //显示信息
    this.textBox1.Text = dsNew.Tables[0].Rows[0][0].ToString();
    this.textBox2.Text = dsNew.Tables[0].Rows[0][1].ToString();
    this.textBox3.Text = dsNew.Tables[0].Rows[0][2].ToString();
}
```

“上一条”按钮的单击事件用于显示当前记录的上一条信息。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    i -= 1;                                                       //获取指定记录的索引号
    if (i >= 0)                                                   //如果没有到首记录
    {
        DataSet dsNew = ClsDBC.dsResult(i);                       //获取当前记录的信息
        //显示信息
        this.textBox1.Text = dsNew.Tables[0].Rows[0][0].ToString();
        this.textBox2.Text = dsNew.Tables[0].Rows[0][1].ToString();
        this.textBox3.Text = dsNew.Tables[0].Rows[0][2].ToString();
    }
    else
    {
        i += 1;                                                   //设置首记录的索引
        MessageBox.Show("这已是第一条信息");
    }
}
```

“下一条”按钮的单击事件用于显示当前记录的下一条信息。代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    i += 1;                                                       //获取指定记录的索引号
    if (i <= (LastNum - 1))                                       //如果不到记录尾
    {
        DataSet dsNew = ClsDBC.dsResult(i);                       //获取当前记录的信息
        //显示信息
        this.textBox1.Text = dsNew.Tables[0].Rows[0][0].ToString();
        this.textBox2.Text = dsNew.Tables[0].Rows[0][1].ToString();
        this.textBox3.Text = dsNew.Tables[0].Rows[0][2].ToString();
    }
    else
```

```
        {
            i -= 1;                                          //设置尾记录的索引号
            MessageBox.Show("这已是最后一条信息");
        }
    }
```

“最后一条”按钮的单击事件用于显示尾记当信息。代码如下：

```
private void button4_Click(object sender, EventArgs e)
{
    DataSet dsNew = ClsDBC.dsResult(LastNum - 1);            //获取当前记录的信息
    //显示信息
    this.textBox1.Text = dsNew.Tables[0].Rows[0][0].ToString();
    this.textBox2.Text = dsNew.Tables[0].Rows[0][1].ToString();
    this.textBox3.Text = dsNew.Tables[0].Rows[0][2].ToString();
    i = LastNum - 1;                                         //设置尾记录的索引号
}
```

22.3.6 使用事务

事务是作为单个逻辑工作单元执行的一系列操作。一个逻辑工作单元必须有 4 个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务。事务管理特性，强制保持事务的原子性和一致性。事务启动之后，就必须成功完成，否则 SQL Server 将撤销该事务启动之后对数据所做的所有修改。

实例位置：光盘\mingrisoft\dxyy\22\22_06

下面将应用事务的一致性来排除程序在运行过程中可能发生的一系列“危险”因素，确保数据的安全，例如，一个用户在取钱时，卡已放入取款机进行操作，等待取款，这时突然停电，通过事务可以帮助用户挽回损失，将信息恢复到操作之前的状态。程序运行结果如图 22.19、图 22.20 和图 22.21 所示。

图 22.19 查询窗体

图 22.20 提示窗体

图 22.21 提示窗体

程序开发步骤如下。

（1）新建一个 Windows 应用程序（DataAlternative），设置窗体名称为 FrmTran，在窗体上添加相应的控件，如图 22.19 所示。

（2）在 ClsDBControl.cs 类中设置实现事务的方法。

（3）主要程序代码如下。

自定义方法 Tran 主要通过事务来完成数据的添加与回滚，参数 strID、strName、strSex 表示要添加的记录内容，参数 strDelID 表示要删除的编号，参数 IfDO 表示是否进行事务回滚。代码如下。

```
public bool Tran(string strID,string strName,string strSex,string strDelID,bool IfDO)
{
    using (SqlConnection connection = ConDB());              //创建连接对象
    {
        SqlTransaction sqlTran = connection.BeginTransaction();  //开始事务
        SqlCommandcommand = connection.CreateCommand();      //创建 SqlCommand 对象
```

```
            command.Transaction = sqlTran;                          //将 SqlCommand 与 SqlTransaction 关联起来
        try
        {
            //设置添加的 SQL 语句
            command.CommandText = "insert into t_people values('" + strID + "','" + strName + "','" + strSex + "')";
            command.ExecuteNonQuery();                              //执行 SQL 语句
            //设置回滚的 SQL 语句
            command.CommandText = "delete from t_people where tb_PID='" + strDelID + "'";
            command.ExecuteNonQuery();                              //执行 SQL 语句
            if (IfDO)                                               //抛出异常，令其发生回滚
            {
                throw new Exception();
            }
            sqlTran.Commit();                                       //提交数据库事务
            connection.Close();                                     //关闭连接
            return True;
        }
        catch (Exception ex)                                        //如果添加错误
        {
            Console.WriteLine(ex.Message);
            sqlTran.Rollback();                                     //从挂起状态回滚事务
            return False;
        }
    }
}
```

“添加与删除信息”按钮主要设用 Tran 方法完成数据的添加，如果失败则对其进行回滚操作。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    ClsDB.ClsDBControl DBDT = new OptDB.ClsDB.ClsDBControl();       //实例化 ClsDBControl 类
    string strid, strname, strsex, strDelid;                        //定义变量
    //记录添加数据的信息
    strid = this.textBox1.Text.Trim().ToString();
    strname = this.textBox2.Text.Trim().ToString();
    strsex = this.textBox3.Text.Trim().ToString();
    strDelid = this.textBox4.Text.Trim().ToString();
    bool Bt;                                                        //定义返回变量
    if (this.radioButton1.Checked)                                  //如果添加数据
    {
        Bt = False;
    }
    else                                                            //事务回滚
    {
        Bt = True;
    }
    if (DBDT.Tran(strid, strname, strsex, strDelid, Bt))            //调用自定义方法 Tran
    {
        MessageBox.Show("事务供交成功");
    }
    else
    {
        MessageBox.Show("事务回滚成功");
    }
}
```

第 23 章

数据绑定控件

- DataGridView 控件
- GridView 控件
- DataList 控件
- Repeater 控件

数据绑定控件是指那些能将数据绑定到数据源上，并且可以在应用程序中轻松显示和修改的控件，比较常见的数据绑定控件有 DataGridView 控件、GridView 控件、DataList 控件和 Repeater 控件。本章将详细介绍这些数据绑定控件的属性、方法和使用。

23.1 DataGridView 控件

DataGridView 控件提供一种强大且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用以显示特大数据集的可编辑视图。

23.1.1 DataGridView 属性

DataGridView 控件常用属性及说明如表 23.1 所示。

表 23.1　　DataGridView 控件常用属性及说明

属　性	说　明
AllowUserToAddRows	是否向用户显示添加行的选项
AllowUserToDeleteRows	是否允许用户从 DataGridView 中删除行
AllowUserToResizeColumns	是否可以调整列的大小
AllowUserToResizeRows	是否可以调整行的大小
AutoGenerateColumns	在设置数据源 DataSource 或 DataMember 属性时，是否自动创建列
AutoSizeColumnsMode	设置一个值，该值指示如何确定列宽
BackgroundImage	设置在控件中显示的背景图像
BackgroundImageLayout	设置 ImageLayout 枚举中定义的背景图像布局
CanFocus	控件是否可以接收焦点
ColumnCount	设置 DataGridView 中显示的列数
ColumnHeadersVisible	是否显示列标题行
Columns	获取一个包含控件中所有列的集合
CurrentRow	获取包含当前单元格的行
DataMember	设置在 DataGridView 控件上显示的数据源 DataSource 中的列表或表的名称
DataSource	设置在 DataGridView 控件上显示数据的数据源
Dock	设置哪些控件的边框将停靠在父控件上，并确定控件如何随父控件一起调整大小
GridColor	设置网格线的颜色，网格线对 DataGridView 的单元格进行分隔
MultiSelect	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
ReadOnly	是否可以编辑 DataGridView 控件的单元格
RowCount	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
SelectedCells	获取用户选定的单元格的集合
SelectedColumns	获取用户选定的列的集合
SelectedRows	获取用户选定的行的集合
SortedColumn	获取 DataGridView 内容的当前排序所依据的列
SortOrder	设置 DataGridView 控件中的项排序方式（升序、降序或不排序）

下面对比较重要的属性进行详细介绍。

（1）AutoSizeColumnsMode 属性。设置一个值，该值指示如何确定列宽。语法格式如下：

```
public DataGridViewAutoSizeColumnsMode AutoSizeColumnsMode { get; set; }
```

属性值：DataGridViewAutoSizeColumnsMode 成员，默认值为 None。

DataGridViewAutoSizeColumnsMode 成员及说明如表 23.2 所示。

表 23.2　　DataGridViewAutoSizeColumnsMode 成员及说明

成　员	说　明
AllCells	列宽调整到适合列中所有单元格（包括标头单元格）的内容
AllCellsExceptHeader	列宽调整到适合列中除标头单元格以外所有单元格的内容
DisplayedCells	列宽调整到适合位于屏幕上当前显示的行中的列的所有单元格（包括标头单元格）的内容
DisplayedCellsExceptHeader	列宽调整到适合位于屏幕上当前显示的行中的列的所有单元格（不包括标头单元格）的内容
None	列宽不会自动调整
ColumnHeader	列宽调整到适合列标头单元格的内容
Fill	列宽调整到使所有列宽精确填充控件的显示区域，要求使用水平滚动的目的只是保持列宽大于 DataGridViewColumn…..MinimumWidth 属性值。相对列宽由相对 DataGrid ViewColumn…..FillWeight 属性值决定

示例 23-01　通过 AutoSizeColumnsMode 属性使数据完全填充到控件显示区域

本示例通过将 DataGridView 控件的 AutoSizeColumnsMode 属性设为 Fill，实现当控件加载数据时，将控件的列宽精确地填充到控件的显示区域，示例运行结果如图 23.1 所示。

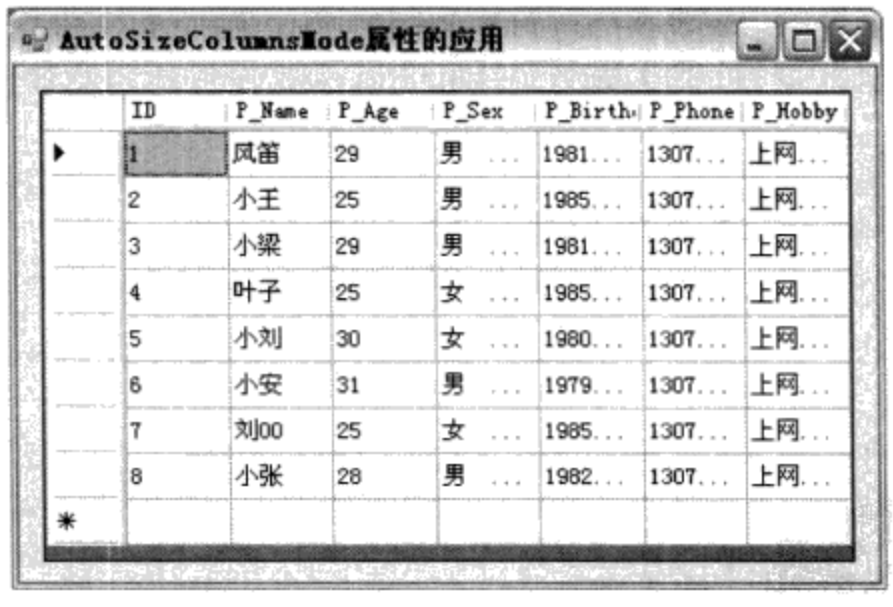


图 23.1　AutoSizeColumnsMode 属性的应用

程序代码如下。

代码 23-01　光盘位置：光盘\mingrisoft\sl\23\23_01

```
private void Form1_Load(object sender, EventArgs e)
{
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //创建 SqlDataAdapter 对象
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_Employee", conn);
    DataSet ds = new DataSet();                    //创建 DataSet 对象
    sda.Fill(ds,"tb_Employee");                    //填充该对象
    dataGridView1.DataSource = ds.Tables[0];       //设置控件数据源
    dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;//设置布局方式
}
```

(2) ColumnCount 属性。用于获取或设置 DataGridView 控件中显示的列数。语法格式如下：

```
public int ColumnCount { get; set; }
```

属性值：DataGridView 控件中显示的列数。

例如，指定 DataGridView 控件的 ColumnCount 属性值为 4，则在窗体中只显示 4 列数据，其代码如下：

```
dataGridView1.ColumnCount = 4;
```

(3) Dock 属性。使用 Dock 属性可以定义在调整控件的父控件大小时，如何自动调整控件的大小。语法格式如下：

```
[LocalizableAttribute(True)]
public virtual DockStyle Dock { get; set; }
```

属性值：指定控件停靠的位置和方式值之一。默认值为 None。

例如，将 Dock 属性值设置为 DockStyle.Left，将导致控件与其父控件的左边缘对齐，并在父控件调整大小时调整自身大小，其代码如下：

```
dataGridView1.Dock = DockStyle. Left
```

知识扩展：DockStyle 枚举指定控件停靠的位置和方式，其主要成员如表 23.3 所示。

表 23.3　　DockStyle 枚举成员

成员名称	说　明
Bottom	该控件的下边缘停靠在其包含控件的底部
Fill	控件的各个边缘分别停靠在其包含控件的各个边缘，并且适当调整大小
Left	该控件的左边缘停靠在其包含控件的左边缘
None	该控件未停靠
Right	该控件的右边缘停靠在其包含控件的右边缘
Top	该控件的上边缘停靠在其包含控件的顶端

(4) DataSource 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; }
```

属性值：包含 DataGridView 控件要显示的数据的对象。

交叉链接：关于 DataSource 属性的应用，可参见示例 23-01。

(5) Columns 属性。用于获取一个集合，该集合包含 DataGridfView 控件中所有列的集合。语法格式如下：

```
public DataGridViewColumnCollection Columns { get; }
```

属性值：DataGridView 控件中的 DataGridViewColumn 对象的集合，包含 DataGridView 控件中的所有列。

交叉链接：关于 Columns 属性的应用，可参见示例 23-02。

(6) ReadOnly 属性。使用 ReadOnly 属性可以定义用户是否可以编辑 DataGridView 控件的单元格。语法格式如下：

```
public bool ReadOnly { get; set; }
```

属性值：如果用户不能编辑 DataGridView 控件的单元格，则为 true，否则为 false，默认值为 false。

例如，在使用 DataGridView 控件显示数据信息时，可以通过设置 DataGridView 控件的 ReadOnly 属性值为 true，防止用户编辑 DataGridView 控件的单元格数据，其代码如下：

```
dataGridView1.ReadOnly = True;
```

(7) RowCount 属性。用于获取或设置 DataGridView 中显示的行数。语法格式如下：

```
public int RowCount { get; set; }
```

属性值：DataGridView 控件中显示的行数。

例如，指定 DataGridView 控件的 RowCount 属性值为 4，则在窗体中只显示 4 行数据，将多余的行数从 Rows 集合的末尾移除，其代码如下：

```
dataGridView1.RowCount = 4;
```

（8）Rows 属性。用于获取一个集合，该集合包含 DataGridView 控件中的所有行。语法格式如下：

```
public DataGridViewRowCollection Rows { get; }
```

属性值：DataGridView 控件中的 DataGridViewRow 对象的集合，包含 DataGridView 控件中的所有行。

示例 23-02　Columns 和 Rows 属性的使用

本示例主要通过设置 DataGridView 控件的 Columns 和 Rows 属性值，为数据控件 DataGridView 添加数据项，示例运行结果如图 23.2 所示。

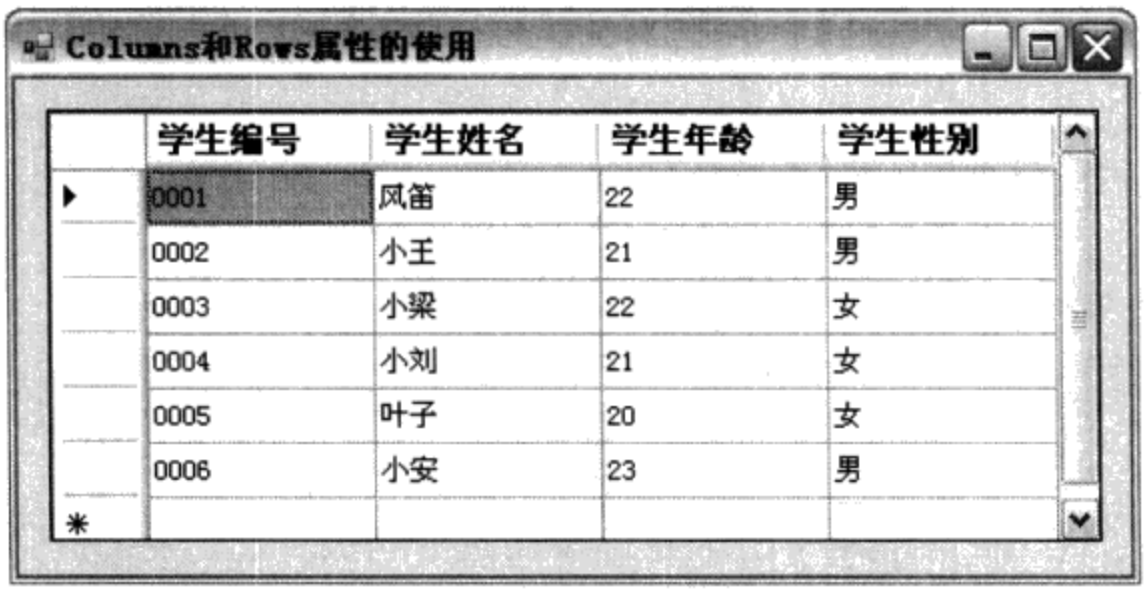


图 23.2　使用 DataGridView 控件的 Columns 和 Rows 属性值为控件添加数据项

程序代码如下。

代码 23-02　光盘位置：光盘\mingrisoft\sl\23\23_02

在窗体的 Load 事件下使用 DataGridView 控件的 Columns 和 Rows 属性值为控件添加数据项，并设置数据项的标题，其代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
        // 指定 DataGridView 控件显示的列数
        dataGridView1.ColumnCount = 4;
        dataGridView1.ColumnHeadersVisible = true;
        // 设置 DataGridView 控件标题列的样式
        DataGridViewCellStyle columnHeaderStyle = new DataGridViewCellStyle();
        columnHeaderStyle.BackColor = Color.Beige;
        columnHeaderStyle.Font = new Font("Verdana", 10, FontStyle.Bold);
        dataGridView1.ColumnHeadersDefaultCellStyle = columnHeaderStyle;
        //设置 DataGridView 控件的标题列名
        dataGridView1.Columns[0].Name = "学生编号";
        dataGridView1.Columns[1].Name = "学生姓名";
        dataGridView1.Columns[2].Name = "学生年龄";
        dataGridView1.Columns[3].Name = "学生性别";
        //设置控件数据源
        string[] row1 = new string[] { "0001", "风笛", "22","男" };
        string[] row2 = new string[] { "0002", "小王", "21", "男" };
        string[] row3 = new string[] { "0003", "小梁", "22", "女" };
        string[] row4 = new string[] { "0004", "小刘", "21", "女" };
        string[] row5 = new string[] { "0005", "叶子", "20", "女" };
        string[] row6 = new string[] { "0006", "小安", "23", "男" };
```

```
        object[] rows = new object[] { row1, row2, row3, row4, row5, row6 };
        //将数据添加到控件中
        foreach (string[] rowArray in rows)
        {
            dataGridView1.Rows.Add(rowArray);
        }
}
```

（9）SelectedCells 属性。用于获取用户选定的单元格的集合。语法格式如下：

```
public DataGridViewSelectedCellCollection SelectedCells { get; }
```

属性值：用户选定的单元格。

交叉链接：关于 SelectedCells 属性的应用，可参见示例 23-03。

23.1.2 DataGridView 方法

DataGridView 控件常用方法及说明如表 23.4 所示。

表 23.4　　DataGridView 控件常用方法及说明

方　法	说　明
AdjustColumnHeaderBorderStyle	为当前正在绘制的 DataGridView 控件的列标题单元格调整边框样式
AreAllCellsSelected	返回一个值，该值指示当前是否选择了所有的 DataGridView 控件的单元格
AutoResizeColumn	调整指定列的宽度以适应其单元格的内容
AutoResizeColumnHeadersHeight	调整列标题的高度以适应标题内容
AutoResizeColumns	调整所有列的宽度以适应其单元格的内容
AutoResizeRow	调整指定行的高度以适应其单元格的内容
AutoResizeRowHeadersWidth	调整行标题的宽度以适应标题内容
AutoResizeRows	调整某些或所有行的高度以适应其内容
BeginEdit	将当前的单元格置于编辑模式下
CancelEdit	取消当前选定单元格的编辑模式并丢弃所有更改
ClearSelection	取消对当前选定的单元格的选择
CommitEdit	将当前单元格中的更改提交到数据缓存，但不结束编辑模式
Contains	检索一个值，该值指示指定控件是否为一个控件的子控件
DisplayedColumnCount	返回向用户显示的列数
DisplayedRowCount	返回向用户显示的行数
Dispose	释放 DataGridView 控件使用的所有资源
DoDragDrop	开始拖放操作
Equals	确定两个 Object 实例是否相等
EndEdit	提交对当前单元格进行的编辑并结束编辑操作
FindForm	检索控件所在的窗体
GetCellCount	获取满足所提供筛选器的单元格的数目
GetChildAtPoint	检索指定位置的子控件
Invalidate	使控件的特定区域无效并向控件发送绘制消息
InvalidateCell	使 DataGridView 中的某个单元格无效，并强制对它进行重新绘制
InvalidateColumn	使 DataGridView 中的指定列无效，并强制对它进行重新绘制
InvalidateRow	使 DataGridView 中的指定行无效，并强制对它进行重新绘制
PointToClient	将指定屏幕点的位置计算成工作区坐标
PointToScreen	将指定工作区点的位置计算成屏幕坐标

续表

方　法	说　明
Refresh	强制控件使其工作区无效并立即重绘自己和任何子控件
Select	激活控件
SelectAll	选择 DataGridView 中的所有单元格
SelectNextControl	激活下一个控件
Show	向用户显示控件
Sort	对 DataGridView 控件的内容进行排序

下面对比较重要的方法进行详细介绍。

(1) AreAllCellsSelected 方法。返回一个值，该值指示当前是否选择了所有的 DataGridView 单元格。语法格式如下：

```
public bool AreAllCellsSelected (bool includeInvisibleCells )
```

参数说明如下。

includeInvisibleCells：如果包括 Visible 属性值为 False 的行和列，则为 true，否则为 false。

示例 23-03　AreAllCellsSelected 方法的使用

本示例主要是使用 AreAllCellsSelected 方法来判断用户是否选择了 DataGridView 控件上的所有单元格。当用户选择了 DataGridView 控件上的所有单元格，并单击“显示”按钮时将会弹出如图 23.3 所示的对话框。

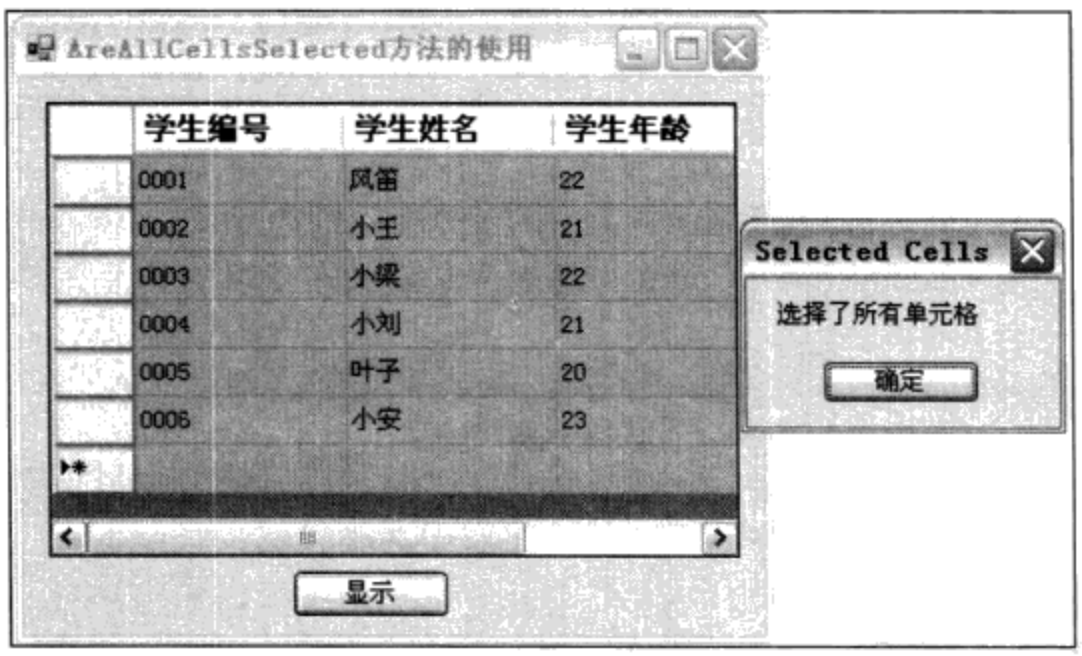


图 23.3　在数据控件 DataGridView 上显示信息

程序代码如下。

代码 23-03　光盘位置：光盘\mingrisoft\sl\23\23_03

当单击“显示”按钮时，将会触发 Button 按钮的 Click 事件，在该事件下，主要是使用 AreAllCellsSelected 方法来判断用户是否选择了 DataGridView 控件上的所有单元格，其代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //获取选择的单元格的数目
    Int32 selectedCellCount =dataGridView1.GetCellCount(DataGridViewElementStates.Selected);
    if (selectedCellCount > 0)                                    //判断其是否大于 0
    {
        if (dataGridView1.AreAllCellsSelected(true))              //如果选择了所有的单元格
```

```
        {
            MessageBox.Show("选择了所有单元格", "Selected Cells");          //弹出提示信息
        }
        else                                                                //否则
        {
            System.Text.StringBuilder sb =new System.Text.StringBuilder();  //声明 StringBuilder 对象
            for (int i = 0; i < selectedCellCount; i++)                     //遍历选择的单元格
            {
                //将选择的行和列的索引添加到 StringBuilder 对象中
                sb.Append("行: ");
                sb.Append(dataGridView1.SelectedCells[i].RowIndex.ToString());
                sb.Append(", 列: ");
                sb.Append(dataGridView1.SelectedCells[i].ColumnIndex.ToString());
                sb.Append(Environment.NewLine);
            }
            sb.Append("总单元格数: " + selectedCellCount.ToString());        //添加被选择的单元格数
            MessageBox.Show(sb.ToString(), "Selected Cells");               //显示获取的相关信息
        }
    }
}
```

（2）DisplayedColumnCount 方法。用于返回向用户显示的列数。语法格式如下：

```
public int DisplayedColumnCount (bool includePartialColumns )
```

参数说明如下。

includePartialColumns：如果要包括显示列数中的部分列，则为 true，否则为 false。

例如，在实现统计功能时，可以利用 DisplayedColumnCount 方法返回 DataGridView 控件中显示的列数，其代码如下：

```
MessageBox.Show(dataGridView1.DisplayedColumnCount(True).ToString());
```

（3）DisplayedRowCount 方法。用于返回向用户显示的行数。语法格式如下：

```
public int DisplayedRowCount (bool includePartialRow)
```

参数说明如下。

includePartialRow：如果要包括显示行数中的部分行，则为 true，否则为 false。

例如，可以利用 DisplayedRowCount 方法返回总成绩在 500 分以上的学生人数，其代码如下：

```
MessageBox.Show(dataGridView1.DisplayedRowCount(True).ToString());
```

（4）Select 方法。用于激活控件。语法格式如下：

```
public void Select ()
```

例如，如果一个控件能被选择，则激活该控件，代码如下：

```
public void ControlSelect(Control control)
{
    if (control.CanSelect)                          //判断控件是否能被选择
    {
        control.Select();                           //如果能选择则激活控件
    }
}
```

（5）SelectAll 方法。用于选择 DataGridView 控件中的所有单元格。语法格式如下：

```
public void SelectAll ()
```

例如，当单击 Button 按钮后，将 DataGridView 控件中的所有单元格选中，其代码如下：

```
private void selectAllButton_Click(object sender, EventArgs e)
{
    dataGridView1.SelectAll();                      //选中所有单元格
}
```


23.1.3 DataGridView 事件

DataGridView 控件常用事件及说明如表 23.5 所示。

表 23.5 DataGridView 控件常用事件及说明

事　件	说　明
AllowUserToAddRowsChanged	当 AllowUserToAddRows 属性的值更改时发生
AllowUserToDeleteRowsChanged	当 AllowUserToDeleteRowsChanged 属性的值更改时发生
AllowUserToOrderColumnsChanged	当 AllowUserToOrderColumns 属性的值更改时发生
AllowUserToResizeColumnsChanged	当 AllowUserToResizeColumns 属性的值更改时发生
AllowUserToResizeRowsChanged	当 AllowUserToResizeRows 属性的值更改时发生
AlternatingRowsDefaultCellStyleChanged	当 AlternatingRowsDefaultCellStyle 属性的值更改时发生
AutoGenerateColumnsChanged	当 AutoGenerateColumnsChanged 属性的值更改时发生
AutoSizeColumnModeChanged	当 DataGridViewAutoSizeRowsMode 属性的值更改时发生
AutoSizeRowsModeChanged	当 DataGridViewAutoSizeRowsMode 属性的值更改时发生
CancelRowEdit	在 DataGridView 控件的 VirtualMode 属性为 true 并且取消行中的编辑时发生
CausesValidationChanged	当 CausesValidation 属性的值更改时发生
CellBeginEdit	在为选定的单元格启动编辑模式时发生
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
CellEndEdit	在为当前选定的单元格停止编辑模式时发生
CellEnter	在 DataGridView 控件中的当前单元格更改或者该控件接收到输入焦点时发生
CellLeave	在单元格失去输入焦点，不再是当前单元格时发生
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
CellMouseDown	当鼠标指针位于单元格的边界内时，在用户按下鼠标按键时发生
CellMouseEnter	在鼠标指针进入单元格时发生
CellMouseLeave	在鼠标指针离开单元格时发生
CellMouseMove	在鼠标指针移到 DataGridView 控件上时发生
CellMouseUp	当鼠标指针停留在单元格上时，在用户松开鼠标按键时发生
CellPainting	在单元格需要绘制时发生
CellParsing	在单元格值已修改的情况下，当单元格退出编辑模式时发生
CellStateChanged	在单元格的状态更改时（例如，当单元格失去或获得焦点时）发生
CellValidated	在单元格完成验证后发生
CellValidating	在单元格失去输入焦点时发生，并启用内容验证功能
CellValueChanged	在单元格的值更改时发生
Click	在单击控件时发生
ColumnAdded	在向控件添加一列时发生
ColumnDividerDoubleClick	在用户双击两列之间的分隔符时发生

续表

事　　件	说　　明
ColumnHeaderCellChanged	在列标题单元格的内容更改时发生
ColumnHeaderMouseClick	在用户单击列标题时发生
ColumnHeaderMouseDoubleClick	在双击列标题时发生
ColumnNameChanged	当列的 Name 属性值更改时发生
ColumnRemoved	在从控件中移除列时发生
ColumnSortModeChanged	当列的 SortMode 属性值更改时发生
ColumnStateChanged	当列的状态更改（例如，获得或失去焦点）时发生
ControlAdded	在将新控件添加到 Control 对象的集合时发生
ControlRemoved	在从 Control 对象的集合中移除控件时发生
CurrentCellChanged	当 CurrentCell 属性更改时发生
CurrentCellDirtyStateChanged	在单元格的状态相对于其内容的更改而更改时发生
DataBindingComplete	在数据绑定操作完成之后发生
DataError	当外部数据分析或验证操作引发异常时，或者当尝试将数据提交到数据源失败时发生
DataMemberChanged	当 DataMember 属性的值更改时发生
DataSourceChanged	当 DataSource 属性的值更改时发生
DefaultValuesNeeded	当用户输入新记录的行时发生，以便可以在其中填入默认值
DockChanged	当 Dock 属性的值更改时发生
DoubleClick	在双击控件时发生
DragDrop	在完成拖放操作时发生
DragEnter	在将对象拖入控件的边界时发生
DragLeave	在将对象拖出控件的边界时发生
DragOver	在将对象拖到控件的边界上发生
EditingControlShowing	在显示用于编辑单元格的控件时发生
EditModeChanged	当 EditMode 属性的值更改时发生
EnabledChanged	在 Enabled 属性值更改后发生
Enter	进入控件时发生
GiveFeedback	在执行拖动操作期间发生
GotFocus	在控件接收焦点时发生
Invalidated	在控件的显示需要重绘时发生
KeyDown	在控件有焦点的情况下按下键时发生
KeyPress	在控件有焦点的情况下按下键时发生
KeyUp	在控件有焦点的情况下释放键时发生
Layout	在控件应重新定位其子控件时发生
Leave	在输入焦点离开控件时发生
LocationChanged	在 Location 属性值更改后发生

续表

事　件	说　明
LostFocus	当控件失去焦点时发生
MouseCaptureChanged	当控件失去鼠标捕获时发生
MouseClick	在单击该控件时发生
MouseDoubleClick	当双击控件时发生
MouseDown	当鼠标指针位于控件上并按下鼠标键时发生
MouseEnter	在鼠标指针进入控件时发生
MouseHover	在鼠标指针停放在控件上时发生
MouseLeave	在鼠标指针离开控件时发生
MouseMove	在鼠标指针移到控件上时发生
MouseUp	在鼠标指针在控件上并释放鼠标键时发生
MouseWheel	在移动鼠标并且控件有焦点时发生
Move	在移动控件时发生
RowHeaderCellChanged	在用户更改行标题单元格的内容时发生
RowHeaderMouseClick	当用户在行标题边界内单击时发生
RowHeaderMouseDoubleClick	在双击行标题时发生
RowsAdded	在向 DataGridView 中添加新行之后发生
RowsRemoved	在从 DataGridView 中删除一行或多行时发生
RowStateChanged	在行的状态更改（例如，失去或获得输入焦点）时发生
RowValidated	在行完成验证后发生
RowValidating	在验证行时发生
SelectionChanged	在当前选择出现更改时发生
SortCompare	在 DataGridView 对两个单元格的值进行比较以执行排序操作时发生
Sorted	在 DataGridView 控件完成排序操作时发生
TextChanged	当 Text 属性的值更改时发生
UserAddedRow	在用户完成向 DataGridView 控件中添加行时发生
UserDeletedRow	在用户完成从 DataGridView 控件中删除行时发生
UserDeletingRow	在用户从 DataGridView 控件中删除行时发生

下面对比较重要的事件进行详细介绍。

（1）CellClick 事件。用于在单元格的任何部分被单击时发生。语法格式如下：

```
public event DataGridViewCellEventHandler CellClick
```

示例 23-04　CellClick 事件的使用

本示例主要是在 DataGridView 控件的 CellClick 事件下，获取用户选择单元格的内容，运行结果如图 23.4 所示，当用户在 DataGridView 控件中选择任一单元格时，如选择“风笛”这个单元格，将会在 TextBox 文本框中显示“风笛”，运行结果如图 23.5 所示。

图 23.4　在数据控件 DataGridView 上显示信息

图 23.5　显示被选择的单元内容

程序代码如下。

代码 23-04　光盘位置：光盘\mingrisoft\sl\23\23_04

当用户单击任一单元格时，都会触发 DataGridView 控件的 CellClick 事件，在该事件下，将被选择的单元格内容显示出来。代码如下：

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        //获取用户选择的单元格内容
        textBox1.Text = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

（2）CellDoubleClick 事件。在用户双击单元格中的任何位置时发生。语法格式如下：

```
public event DataGridViewCellEventHandler CellDoubleClick
```

示例 23-05　CellDoubleClick 事件的使用

本示例主要是在 DataGridView 控件的 CellDoubleClick 事件下，显示被选择学生的详细信息，示例运行结果如图 23.6 所示，当用户在 DataGridView 控件中选择任一学生姓名时，都会将该学生的详细信息显示在界面上，运行结果如图 23.7 所示。

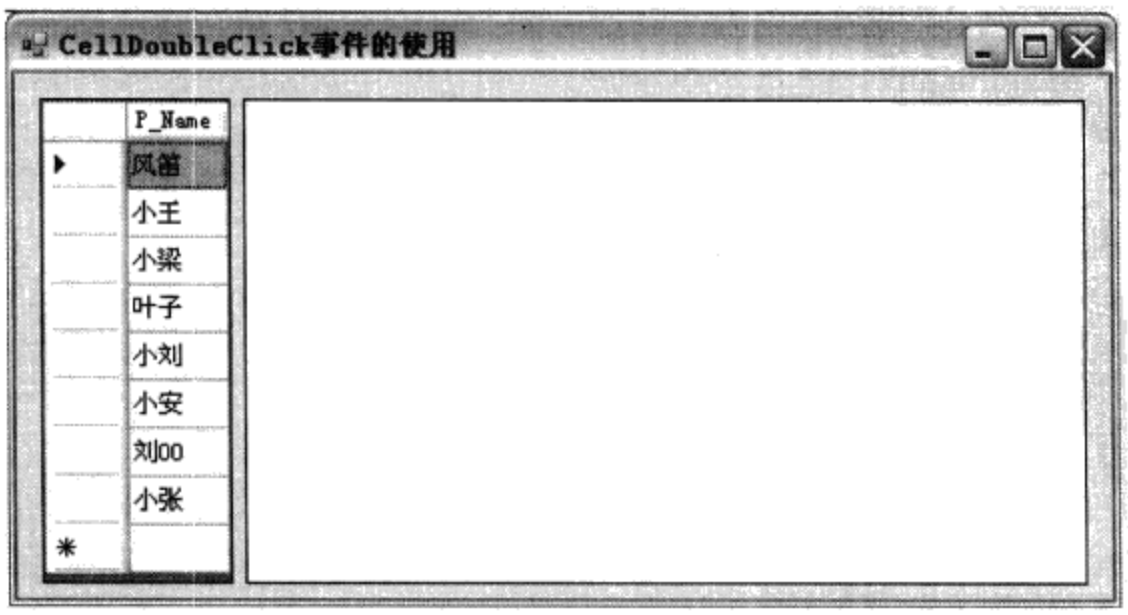


图 23.6　显示学生信息

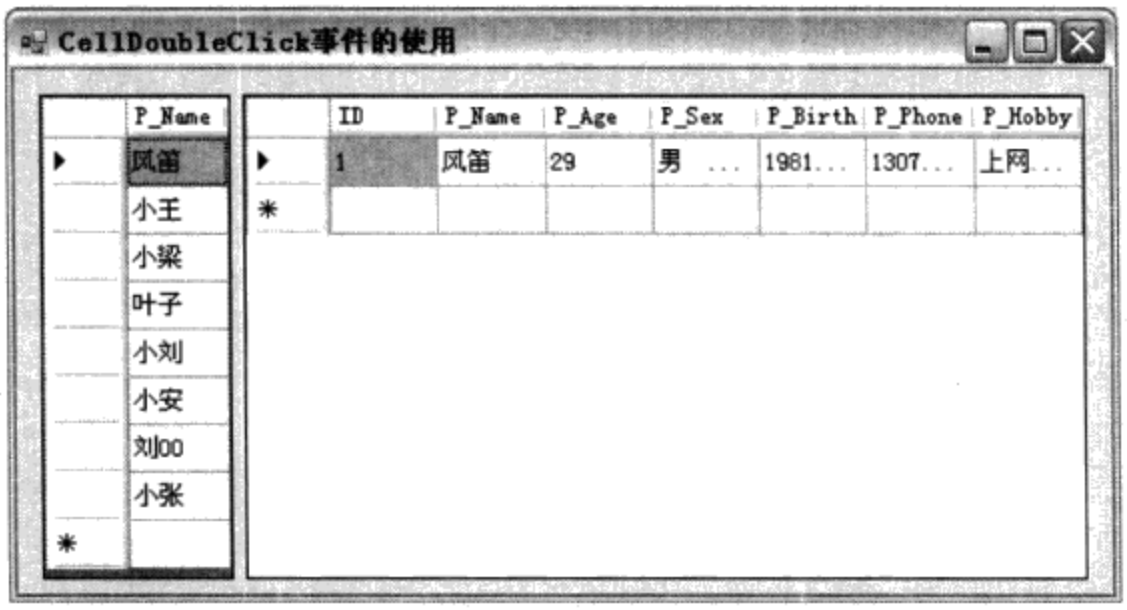


图 23.7　显示指定学生的详细信息

程序代码如下。

代码 23-05　光盘位置：光盘\mingrisoft\sl\23\23_05

当用户双击任一单元格中的学生姓名时，都会触发 DataGridView 控件的 CellDoubleClick 事件，在该事件下，显示被选择学生的详细信息。代码如下：

```
private void dataGridView1_CellDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        //获取选择的学生姓名
        string UserName = dataGridView1.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();
        //建立数据库连接
        SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        SqlDataAdapter sda = new SqlDataAdapter("select * from tb_Employee where P_Name='" + UserName + "'",
conn);
        //根据选择的学生姓名检索数据库
        DataSet ds = new DataSet();                                  //创建 DataSet 对象
        sda.Fill(ds, "tb_Employee");                                 //填充该对象
        dataGridView2.DataSource = ds.Tables[0];                     //设置控件数据源
        //设置控件布局方式
        dataGridView2.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    }
    catch (SqlException)                                             //如果发生异常
    {
        MessageBox.Show("获取数据失败！", "ERROR",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);          //显示提示信息
    }
}
```

23.1.4　DataGridView 控件应用实例

DataGridView 控件在 Windows 应用程序开发中被频繁的使用，该控件不仅操作简单，而且绑定数据也很方便，只需设置 Data Source 属性即可。DataGridView 控件支持标准 Windows 窗体数据绑定模型，因此可绑定到各种数据源，如 DataTable、DataSet 和 BindingList 等。下面通过一个实例来介绍如何操作 DataGridView 控件。

实例位置：光盘\mingrisoft\dxyy\23\23_01

本实例将数据表中的数据绑定到 DataGridView 控件中，当单击 DataGridView 控件中某一个行时，会将其详细信息显示到上方的员工数据中，并且可以对选择的信息进行修改，同时也可以添加新的数据，实例运行结果如图 23.8 所示。

图 23.8　DataGridView 控件的基本操作

程序开发步骤如下。

（1）新建一个 Windows 窗体应用程序，命名为“DataGridView 数据控件应用实例”。

（2）在项目中添加 3 个 TextBox 控件、4 个 ComboBox 控件、8 个 CheckBox 控件、1 个 DataGridView 控件和 2 个 Button 控件。3 个 TextBox 控件分别用来输入员工姓名、员工电话和显示员工的年龄，4 个 ComboBox 控件分别用来选择员工性别、选择员工出生的年、月和日。8 个 CheckBox 控件用来选择员工的爱好，DataGridView 控件用于显示所有的员工信息，2 个 Button 控件分别用来添加和修改员工信息。

（3）程序主要代码如下。

自定义的 BindData 方法，用于将数据表中的所有员工信息绑定到 DataGridView 控件中，并且在此方法中设置 DataGridView 控件的选择模式、是否允许选择多项以及是否只读等属性，其代码如下：

```
private void BindData()
{
        //建立数据库连接
        SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //检索数据库中所有数据
        SqlDataAdapter sda = new SqlDataAdapter("select ID as '编号',P_Name as '姓名',P_Age as '年龄',P_Sex as '
性别',P_Birthday as '生日',P_Phone as '电话',P_Hobby as '爱好' from tb_Employee", conn);
        //创建 DataSet 对象
        DataSet ds = new DataSet();
        //填充对象
        sda.Fill(ds, "tb_Employee");
        //设置控件数据源
        dataGridView1.DataSource = ds.Tables[0];
        //设置控件布局方式
        dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        //设置选择方式为正行选择
        dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
        //禁止多选
        dataGridView1.MultiSelect = false;
        //禁止显示添加行
        dataGridView1.AllowUserToAddRows = false;
```

```
        //设为只读
        dataGridView1.ReadOnly = true;
        cbbSex.SelectedIndex = 0;
    }
```

当选择 DataGridView 控件中的某一项时，会激发 DataGridView 控件的 CellClick 事件，在此事件中首先获取选择项的 ID 值，然后根据 ID 值检索数据库，将相关信息显示到相应的控件中，其代码如下：

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    //获取 ID 值
    int id = Convert.ToInt32(dataGridView1.SelectedRows[0].Cells[0].Value);
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //根据 ID 值检索数据库
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_Employee where ID="+id, conn);
    //创建一个 DataSet 对象
    DataSet ds = new DataSet();
    //填充该对象
    sda.Fill(ds, "tb_Employee");
    //显示姓名
    txtName.Text = ds.Tables[0].Rows[0][1].ToString();
    //显示性别
    cbbSex.SelectedItem = ds.Tables[0].Rows[0][3].ToString();
    //显示电话
    txtPhone.Text = ds.Tables[0].Rows[0][5].ToString();
    //显示年龄
    txtAge.Text = ds.Tables[0].Rows[0][2].ToString();
    //获取生日
    string Birthday = ds.Tables[0].Rows[0][4].ToString();
    //拆分成数组
    string[] birthday=Birthday.Split('-');
    //显示出生的年
    cbbYear.Text = birthday[0];
    //显示出生的月
    cbbMonth.Text = birthday[1];
    //显示出生的日
    cbbDay.Text = birthday[2];
    //获取爱好
    string Hobby = ds.Tables[0].Rows[0][6].ToString();
    //将所有的爱好拆分成数组
    string[] hobby = Hobby.Split('、');
    //遍历窗体中的所有控件
    foreach (Control cl in groupBox1.Controls)
    {
        //如果控件是 CheckBox 控件
        if (cl is System.Windows.Forms.CheckBox)
        {
            //首先将所有的 CheckBox 控件处于未选中状态
            ((CheckBox)cl).Checked = false;
            for (int i = 0; i < hobby.Length; i++)
            {
                //遍历存储爱好的数组，如果窗体中 CheckBox 控件显示的文本等于数组中的项
                if (hobby[i] == ((CheckBox)cl).Text)
                {
                    //使 CheckBox 控件处于选中状态
                    ((CheckBox)cl).Checked = true;
                }
            }
        }
    }
}
```

23.2 GridView 控件

以表格形式显示数据是网站开发中的一个常见功能，用户可以通过 GridView 控件来实现，并可编辑和删除多种不同数据源中的数据。

23.2.1 GridView 属性

GridView 控件常用属性及说明如表 23.6 所示。

表 23.6　　GridView 控件常用属性及说明

属　性	说　明
AllowPaging	是否启用分页功能
AllowSorting	是否启用排序功能
AlternatingRowStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的交替数据行的外观
AutoGenerateColumns	是否为数据源中的每个字段自动创建绑定字段
AutoGenerateDeleteButton	获取或设置一个值，该值指示每个带有“删除”按钮的 CommandField 字段列的数据行是否自动添加到 GridView 控件中
AutoGenerateEditButton	获取或设置一个值，该值指示每个带有“编辑”按钮的 CommandField 字段列的数据行是否自动添加到 GridView 控件中
AutoGenerateSelectButton	获取或设置一个值，该值指示每个带有“选择”按钮的 CommandField 字段列的数据行是否自动添加到 GridView 控件中
BackColor	设置 Web 服务器控件的背景色
BackImageUrl	设置要在 GridView 控件的背景中显示的图像的 URL
BorderColor	设置 Web 控件的边框颜色
BorderStyle	设置 Web 服务器控件的边框样式
BorderWidth	设置 Web 服务器控件的边框宽度
CellPadding	设置单元格的内容和单元格的边框之间的空间量
CellSpacing	设置单元格间的空间量
Columns	获取表示 GridView 控件中列字段的 DataControlField 对象的集合
Controls	获取复合数据绑定控件内的子控件的集合
CssClass	设置由 Web 服务器控件在客户端呈现的级联样式表（CSS）类
DataKeyNames	设置一个数组，该数组包含了显示在 GridView 控件中的项的主键字段的名称
DataKeys	获取一个 DataKey 对象集合，这些对象表示 GridView 控件中的每一行的数据键值
DataMember	当数据源包含多个不同的数据项列表时，获取或设置数据绑定控件绑定到的数据列表的名称
DataSource	设置对象，数据绑定控件从该对象中检索其数据项列表
DataSourceID	设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
EditIndex	设置要编辑的行的索引
EditRowStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中为进行编辑而选中的行的外观
Enabled	设置一个值，该值指示是否启用 Web 服务器控件

续表

属　性	说　明
EnableSortingAndPagingCallbacks	设置一个值，该值指示客户端回调是否用于排序和分页操作
EnableTheming	设置一个值，该值指示是否对此控件应用主题
EnableViewState	设置一个值，该值指示服务器控件是否向发出请求的客户端保持自己的视图状态，以及它所包含的任何子控件的视图状态
Font	获取与 Web 服务器控件关联的字体属性
FooterRow	获取表示 GridView 控件中的脚注行的 GridViewRow 对象
FooterStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的脚注行的外观
ForeColor	设置 Web 服务器控件的前景色
GridLines	设置 GridView 控件的网格线样式
HeaderRow	获取表示 GridView 控件中的标题行的 GridViewRow 对象
HeaderStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的标题行的外观
Height	设置 Web 服务器控件的高度
HorizontalAlign	设置 GridView 控件在页面上的水平对齐方式
ID	设置分配给服务器控件的编程标识符
Page	获取对包含服务器控件的 Page 实例的引用
PageCount	获取在 GridView 控件中显示数据源记录所需的页数
PageIndex	设置当前显示页的索引
PageSize	设置 GridView 控件在每页上所显示的记录的数目
Rows	获取表示 GridView 控件中数据行的 GridViewRow 对象的集合
RowStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的数据行的外观
SelectedDataKey	获取 DataKey 对象，该对象包含 GridView 控件中选中行的数据键值
SelectedIndex	设置 GridView 控件中的选中行的索引
SelectedRow	获取对 GridViewRow 对象的引用，该对象表示控件中的选中行
SelectedRowStyle	获取对 TableItemStyle 对象的引用，使用该对象可以设置 GridView 控件中的选中行的外观
SelectedValue	获取 GridView 控件中选中行的数据键值
SkinID	设置要应用于控件的外观
SortDirection	获取正在排序的列的排序方向
SortExpression	获取与正在排序的列关联的排序表达式
Visible	设置一个值，该值指示服务器控件是否作为 UI 呈现在页上
Width	设置 Web 服务器控件的宽度

下面对比较重要的属性进行详细介绍。

（1）AllowPaging 属性。是否启用分页功能。语法格式如下：

```
public virtual bool AllowPaging { get; set; }
```

属性值：如果启用分页功能，则为 true；否则为 false，默认值为 false。

示例 23-06　AllowPaging 属性的使用

本示例实现的是分页显示员工信息。示例运行结果如图 23.9 所示。

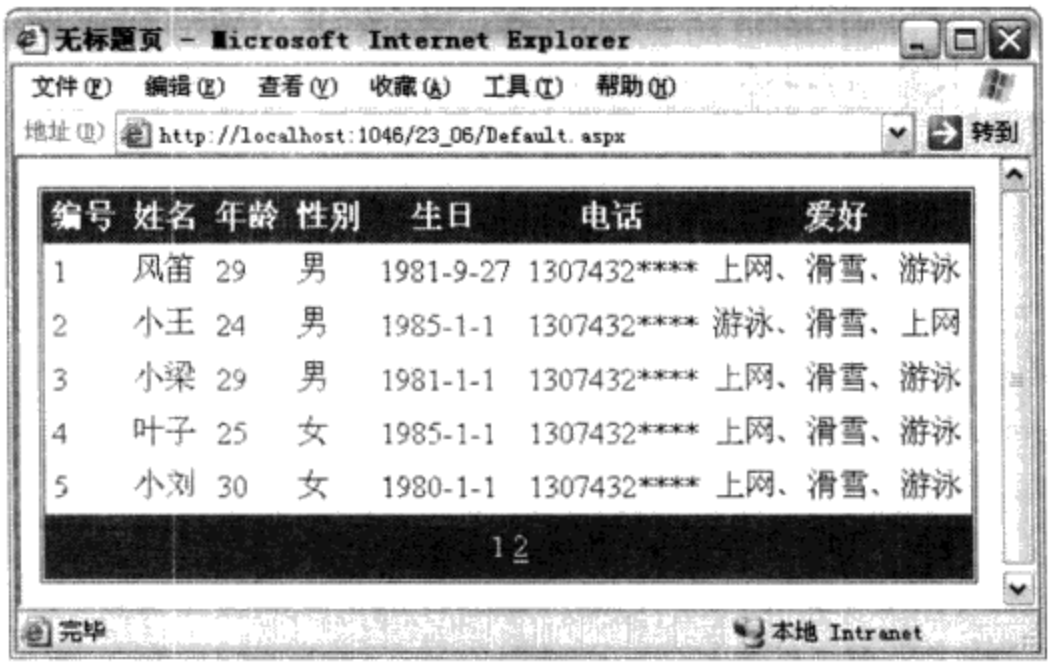


图 23.9 AllowPaging 属性的使用

程序代码如下。

代码 23-06 光盘位置：光盘\mingrisoft\sl\23\23_06

Default.aspx 页面在初始化时，将数据表中数据绑定到 GridView 控件中并显示出来。Default.aspx 页 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //启用分页
    GridView1.AllowPaging = true;
    //每页显示数据的数量
    GridView1.PageSize = 5;
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //检索 tb_Employee 表中所有信息
    SqlDataAdapter sda = new SqlDataAdapter("select ID as '编号',P_Name as '姓名',P_Age as '年龄',P_Sex as '性别
',P_Birthday as '生日',P_Phone as '电话',P_Hobby as '爱好' from tb_Employee", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充该对象
    sda.Fill(ds, "tb_Employee");
    //设置控件数据源
    GridView1.DataSource = ds.Tables[0].DefaultView;
    //绑定数据源
    GridView1.DataBind();
}
```

在实现 GridView 控件的分页功能时，使用了该控件自带的分页功能。首先将 GridView 控件的 AllowPaging 属性设置为 true，以允许该控件分页，然后触发其 PageIndexChanging 事件，并在该事件下编写 GridView 控件分页的代码。GridView 控件 PageIndexChanging 事件代码如下：

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    //获取当前被选择页的索引值
    GridView1.PageIndex = e.NewPageIndex;
    //重新对控件数据进行绑定
    GridView1.DataBind();
}
```

（2）DataKeyNames 属性。用于设置一个数组，该数组包含了显示在 GridView 控件中的项的主键字段的名称。语法格式如下：

```
public virtual string[] DataKeyNames { get; set; }
```

属性值：一个数组，其中包含了显示在 GridView 控件中项的主键字段的名称。

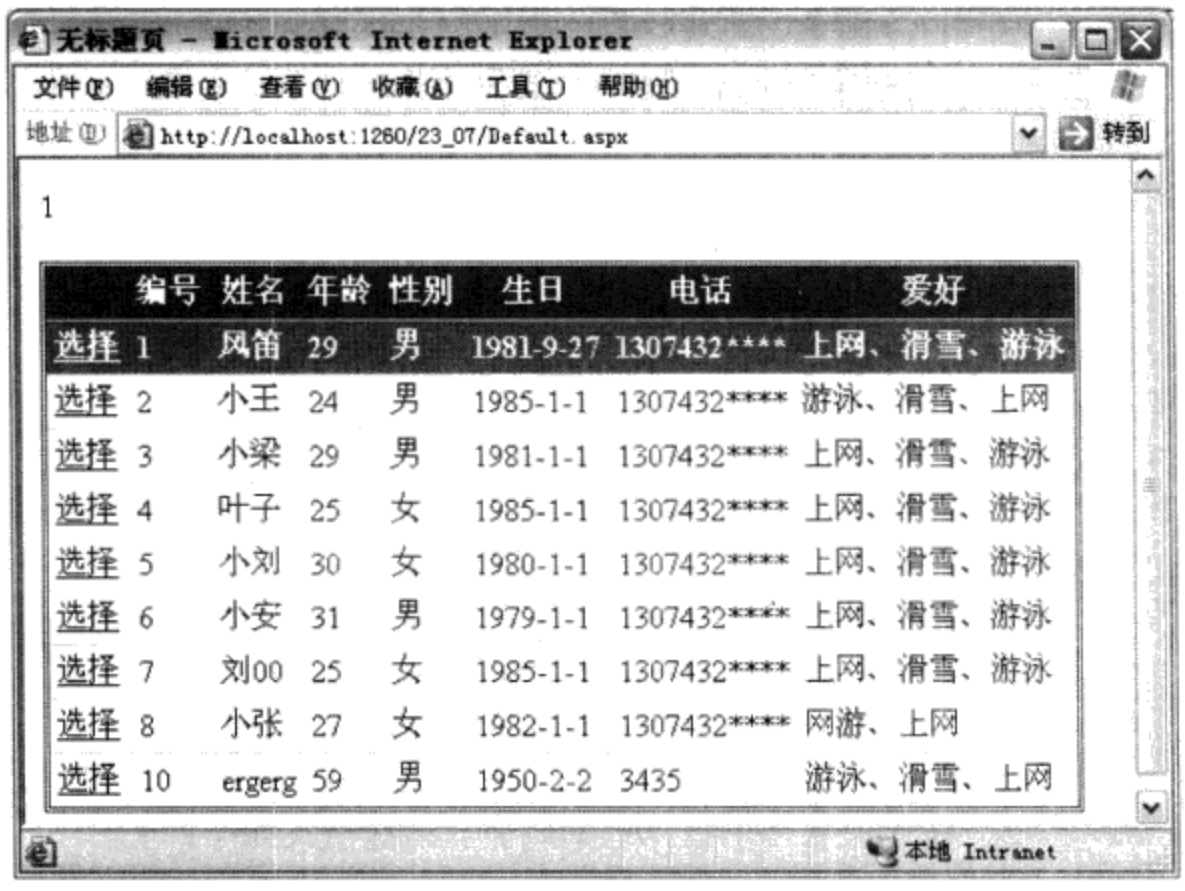
（3）DataKeys 属性。用于获取一个 DataKey 对象集合，这些对象表示 GridView 控件中的每一行的数据键值。语法格式如下：

```
public virtual DataKeyArray DataKeys { get; }
```

属性值：DataKey 对象的集合，包含 GridView 控件中每一行的数据键。

示例 23-07　DataKeyNames 和 DataKeys 属性的使用

本示例主要是介绍如何使用 DataKeys 属性来确定 GridView 控件中选中行的数据键值，当单击 GridView 控件中某一行的“选择”按钮后，将会在界面上显示该行的数据键值，结果如图 23.10 所示。



	编号	姓名	年龄	性别	生日	电话	爱好
选择	1	风笛	29	男	1981-9-27	1307432****	上网、滑雪、游泳
选择	2	小王	24	男	1985-1-1	1307432****	游泳、滑雪、上网
选择	3	小梁	29	男	1981-1-1	1307432****	上网、滑雪、游泳
选择	4	叶子	25	女	1985-1-1	1307432****	上网、滑雪、游泳
选择	5	小刘	30	女	1980-1-1	1307432****	上网、滑雪、游泳
选择	6	小安	31	男	1979-1-1	1307432****	上网、滑雪、游泳
选择	7	刘00	25	女	1985-1-1	1307432****	上网、滑雪、游泳
选择	8	小张	27	女	1982-1-1	1307432****	网游、上网
选择	10	ergerg	59	男	1950-2-2	3435	游泳、滑雪、上网

图 23.10　显示选中行的数据键值

程序代码如下。

代码 23-07　光盘位置：光盘\mingrisoft\sl\23\23_07

Default.aspx 页面在初始化时，将数据表中数据绑定到 GridView 控件中并显示出来。Default.aspx 页 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //显示选择按钮
    GridView1.AutoGenerateSelectButton = true;
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //检索 tb_Employee 表中的数据
    SqlDataAdapter sda = new SqlDataAdapter("select ID as '编号',P_Name as '姓名',P_Age as '年龄',P_Sex as '性别
',P_Birthday as '生日',P_Phone as '电话',P_Hobby as '爱好' from tb_Employee", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充对象
    sda.Fill(ds, "tb_Employee");
    //设置控件数据源
    GridView1.DataSource = ds;
    //设置每一行数据的键值
```

```
        GridView1.DataKeyNames = new string[] { "编号" };
        //绑定数据
        GridView1.DataBind();
    }
```

注意：当设置了 GridView 控件的 DataKeyNames 属性时，GridView 控件将自动为该控件中的每一行创建一个 DataKey 对象。

在使用 DataKeyNames 属性确定 GridView 控件中选中行的数据键值时，首先需要将 GridView 控件的 AutoGenerateSelectButton 属性值设置为 true，用以将“选择”按钮的 CommandField 字段列自动添加在 GridView 控件中，然后在 GridView 控件的 SelectedIndexChanged 事件下，获取被选中行的数据键值，并在界面上显示出来。代码如下：

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
    Response.Write(GridView1.DataKeys[GridView1.SelectedIndex].Value.ToString());
}
```

（4）DataSource 属性。用于获取或设置对象，数据绑定控件从该对象中检索其数据项列表。语法格式如下：

```
public virtual Object DataSource { get; set; }
```

属性值：一个表示数据源的对象，数据绑定控件从该对象中检索其数据。默认值为空引用。

交叉链接：关于 DataSource 属性的使用，可参见示例 23-07。

（5）EditIndex 属性。用于设置要编辑的行的索引。语法格式如下：

```
public virtual int EditIndex { get; set; }
```

属性值：要编辑的行从零开始索引。默认值为-1，指示没有正在编辑的行。

例如，在 GridView 控件的 RowCancelingEdit 事件下，使用 EditIndex 属性取消对指定信息进行编辑，其代码如下：

```
protected void GridView1_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    GridView1.EditIndex = -1;                    //取消编辑
    ClassBind();                                 //重新绑定数据
}
```

（6）PageIndex 属性。用于设置当前显示页的索引。语法格式如下：

```
public virtual int PageIndex { get; set; }
```

属性值：当前所显示页从零开始的索引。

例如，在实现 GridView 控件的分页功能时，使用 PageIndex 属性值，获取当前被选择页的索引值，其代码如下：

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    GridView1.PageIndex = e.NewPageIndex;        //获取当前被选择页的索引值
    GridView1.DataBind();                        //重新绑定数据
}
```

23.2.2 GridView 方法

GridView 控件常用方法及说明如表 23.7 所示。

表 23.7　　GridView 控件常用方法及说明

方　法	说　明
ApplyStyleSheetSkin	将页样式表中定义的样式属性应用到控件
DataBind	将数据源绑定到 GridView 控件
DeleteRow	从数据源中删除位于指定索引位置的记录
FindControl	在当前的命名容器中搜索指定的服务器控件
Focus	为控件设置输入焦点

续表

方　法	说　明
GetHashCode	用作特定类型的哈希函数
GetType	获取当前实例的 Type
HasControls	确定服务器控件是否包含任何子控件
IsBindableType	确定指定的数据类型是否能绑定到 GridView 控件中的列
ResolveClientUrl	获取浏览器可以使用的 URL
ResolveUrl	将 URL 转换为在请求客户端可用的 URL
Sort	根据指定的排序表达式和方向对 GridView 控件进行排序
UpdateRow	使用行的字段值更新位于指定行索引位置的记录

下面对比较重要的方法进行详细介绍。

（1）DataBind 方法。用于将数据源绑定到 GridView 控件。语法格式如下：

```
public override sealed void DataBind ()
```

交叉链接：关于 DataBind 方法的使用，可参见示例 23-07。

（2）DeleteRow 方法。用于从数据源中删除位于指定索引位置的记录。语法格式如下：

```
public virtual void DeleteRow (int rowIndex )
```

参数说明如下。

rowIndex：要删除行的索引。

例如，在 GridView 控件中，删除行的索引值为 1 的数据信息。其代码如下：

```
GridView1.DeleteRow(1);
```

23.2.3 GridView 事件

GridView 控件常用事件及说明如表 23.8 所示。

表 23.8　GridView 控件常用事件及说明

事　件	说　明
DataBinding	当服务器控件绑定到数据源时发生
DataBound	在服务器控件绑定到数据源后发生
Disposed	当从内存释放服务器控件时发生
Load	当服务器控件加载到 Page 对象中时发生
PageIndexChanged	在 GridView 控件处理分页操作之后发生
PageIndexChanging	在 GridView 控件处理分页操作之前发生
RowCancelingEdit	单击编辑模式中某一行的“取消”按钮以后，在该行退出编辑模式之前发生
RowCommand	当单击 GridView 控件中的按钮时发生
RowCreated	在 GridView 控件中创建行时发生
RowDataBound	在 GridView 控件中将数据行绑定到数据时发生
RowDeleted	单击某一行的“删除”按钮时，在 GridView 控件删除该行之后发生
RowDeleting	单击某一行的“删除”按钮时，在 GridView 控件删除该行之前发生
RowEditing	单击某一行的“编辑”按钮以后，GridView 控件进入编辑模式之前发生
RowUpdated	单击某一行的“更新”按钮，在 GridView 控件对该行进行更新之后发生
RowUpdating	单击某一行的“更新”按钮以后，GridView 控件对该行进行更新之前发生
SelectedIndexChanged	单击某一行的“选择”按钮，GridView 控件对相应的选择操作进行处理之后发生
SelectedIndexChanging	单击某一行的“选择”按钮以后，GridView 控件对相应的选择操作进行处理之前发生
Sorted	单击用于列排序的超级链接时，在 GridView 控件对相应的排序操作进行处理之后发生
Sorting	单击用于列排序的超级链接时，在 GridView 控件对相应的排序操作进行处理之前发生
Unload	当服务器控件从内存中卸载时发生

下面对比较重要的事件进行详细介绍。

（1）PageIndexChanging 事件。单击某一页“导航”按钮时，在 GridView 控件处理分页操作之后发生。语法格式如下：

```
public event EventHandler PageIndexChanging
```

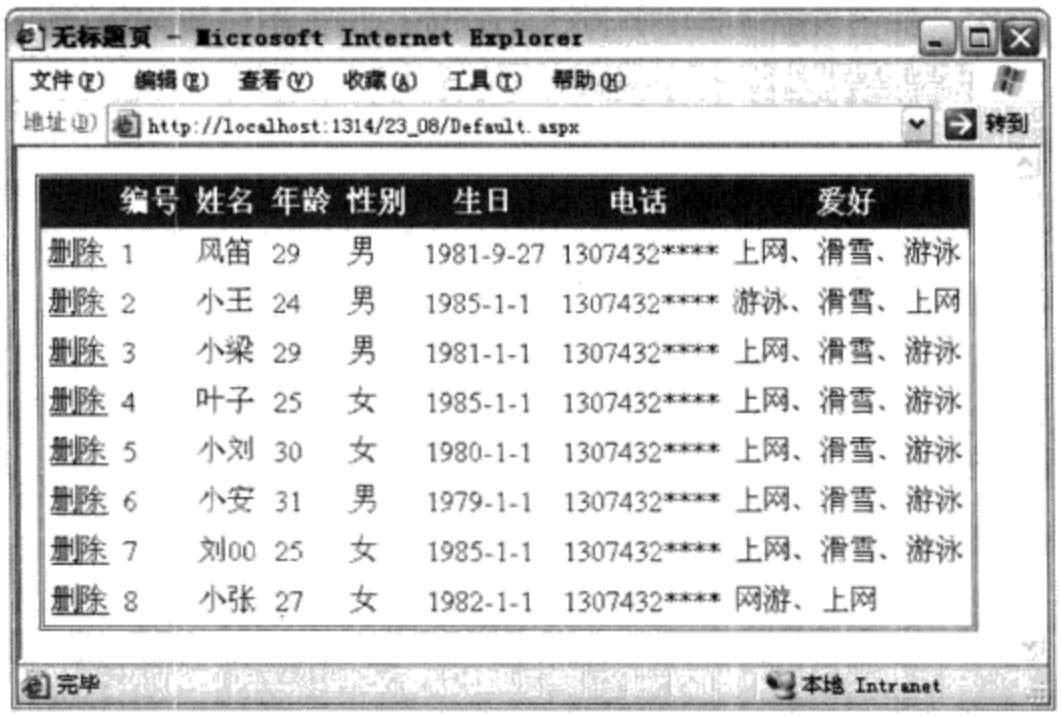
交叉链接：关于 PageIndexChanging 事件的使用，可参见示例 23-06。

（2）RowDeleting 事件。单击某一行的“删除”按钮时，在 GridView 控件删除该行之前发生。语法格式如下：

```
public event GridViewDeleteEventHandler RowDeleting
```

示例 23-08　RowDeleting 事件的使用

本示例实现的是当程序运行时，在页面 Default.aspx 中单击 GridView 控件上的“删除”按钮，将删除指定的信息，示例运行结果如图 23.11 所示。



	编号	姓名	年龄	性别	生日	电话	爱好
删除	1	风笛	29	男	1981-9-27	1307432****	上网、滑雪、游泳
删除	2	小王	24	男	1985-1-1	1307432****	游泳、滑雪、上网
删除	3	小梁	29	男	1981-1-1	1307432****	上网、滑雪、游泳
删除	4	叶子	25	女	1985-1-1	1307432****	上网、滑雪、游泳
删除	5	小刘	30	女	1980-1-1	1307432****	上网、滑雪、游泳
删除	6	小安	31	男	1979-1-1	1307432****	上网、滑雪、游泳
删除	7	刘00	25	女	1985-1-1	1307432****	上网、滑雪、游泳
删除	8	小张	27	女	1982-1-1	1307432****	网游、上网

图 23.11　RowDeleting 事件中删除指定的信息

程序代码如下。

代码 23-08　光盘位置：光盘\mingrisoft\sl\23\23_08

当单击 GridView 控件上的“删除”按钮时，将会触发 GridView 控件的 RowDeleting 事件删除指定信息，代码如下：

```
protected void GridView1_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    //声明 SQL 语句用于删除指定 ID 值的数据
    string sqlStr = "delete from tb_Employee where ID=" +
    Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value. ToString());
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //打开数据库
    conn.Open();
    //声明 SqlCommand 对象
    SqlCommand cmd = new SqlCommand(sqlStr,conn);
    //执行 SQL 语句
    int i = cmd.ExecuteNonQuery();
    //返回值大于 0 说明操作成功
    if (i > 0)
    {
        //弹出提示信息
        RegisterStartupScript("","<script>alert('删除成功')</script>");
```

```
            //重新绑定数据
            BindData();
            //关闭数据库连接
            conn.Close();
        }
    }
```

（3）RowCommand 事件。当单击 GridView 控件中的按钮时发生。语法格式如下：

```
public event GridViewCommandEventHandler RowCommand
```

说明：在使用 GridView 控件中的 RowCommand 事件时，需要设置 GridView 控件中的按钮（如 Button 按钮）的 CommandName 属性值，CommandName 属性值及其说明如表 23.9 所示。

表 23.9　CommandName 属性值及其说明

事　件	说　明
Cancel	取消编辑操作，并将 GridView 控件返回为只读模式
Delete	删除当前记录
Edit	将当前记录置于编辑模式
Page	执行分页操作，将按钮的 CommandArgument 属性设置为“First”、“Last”、“Next”、“Prev”或页码，以指定要执行的分页操作类型
Select	选择当前记录
Sort	对 GridView 控件进行排序
Update	更新数据源中的当前记录

示例 23-09　RowCommand 事件的使用

在 GridView 控件的模板列中添加一个 Button 按钮，并将 Button 按钮的 CommandName 属性值设置为“delete”，在 GridView 控件的 RowCommand 事件下便可以编写删除代码，删除指定行的数据，示例运行结果如图 23.12 所示。

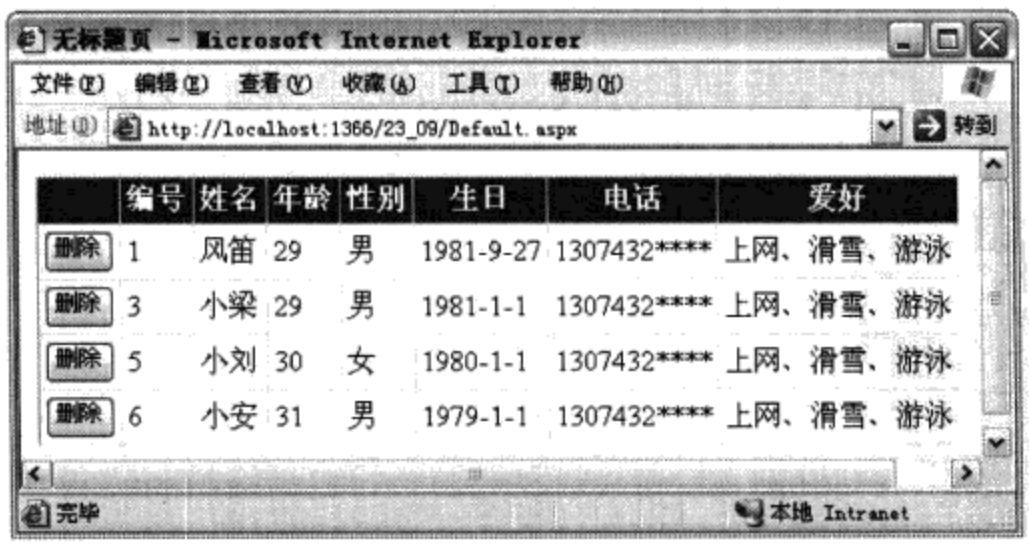


图 23.12　通过 RowCommand 事件删除指定行

程序代码如下。

代码 23-09　光盘位置：光盘\mingrisoft\sl\23\23_09

单击 GridView 控件属性面板中的 Columns 属性的…按钮，如图 23.13 所示。

单击此按钮之后打开“字段”窗口，在此窗口中可以向 GridView 控件中添加字段，如图 23.14 所示。

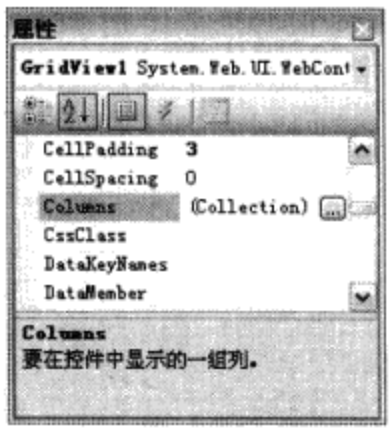


图 23.13　GridView 控件属性面板

图 23.14　GridView 控件的“字段”窗口

选择“ButtonField”，单击“添加”按钮，然后设置添加字段的 CommandName 属性为“Delete”，单击“确定”按钮完成字段的添加，如图 23.15 所示。

	Column0	Column1	Column2
删除	abc	abc	abc
删除	abc	abc	abc
删除	abc	abc	abc
删除	abc	abc	abc
删除	abc	abc	abc

图 23.15　新添加的字段

单击“删除”按钮，会激发 GridView 控件的 RowCommand 事件，在此事件中编写代码实现删除指定的数据，代码如下：

```
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
{
    //如果 CommandName 属性为“Delete”
    if (e.CommandName == "Delete")
    {
        //获取 ID 值
        int id =
        Convert.ToInt32(GridView1.DataKeys[Convert.ToInt32(e.CommandArgument.ToString())].Value. ToString());
        //声明 sql 语句根据 ID 值删除数据
        string sqlStr = "delete from tb_Employee where ID=" + id;
        //连接数据库
        SqlConnection conn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //打开数据库连接
        conn.Open();
        //创建 SqlCommand 对象
        SqlCommand cmd = new SqlCommand(sqlStr, conn);
        //执行 sql 语句
        int i = cmd.ExecuteNonQuery();
        //如果返回值大于 0 说明操作成功
        if (i > 0)
        {
            //弹出提示信息
            RegisterStartupScript("", "<script>alert('删除成功')</script>");
```

```
                //重新绑定数据
            BindData();
                //关闭数据库连接
            conn.Close();
        }
    }
}
```

（4）RowCancelingEdit 事件。单击编辑模式中某一行的“取消”按钮以后，在该行退出编辑模式之前发生。语法格式如下：

```
public event GridViewCancelEditEventHandler RowCancelingEdit
```

（5）RowEditing 事件。单击某一行的“编辑”按钮以后，在 GridView 控件进入编辑模式之前发生。语法格式如下：

```
public event GridViewEditEventHandler RowEditing
```

（6）RowUpdating 事件。单击某一行的“更新”按钮以后，在 GridView 控件对该行进行更新之前发生。语法格式如下：

```
public event GridViewUpdateEventHandler RowUpdating
```

示例 23-10　RowCancelingEdit、RowEditing 和 RowUpdating 事件的使用

本示例主要是介绍如何通过 GridView 控件的 RowCancelingEdit、RowEditing 和 RowUpdating 事件，当单击“编辑”按钮后，在文本框中修改数据，然后单击“更新”按钮，运行结果如图 23.16 所示。

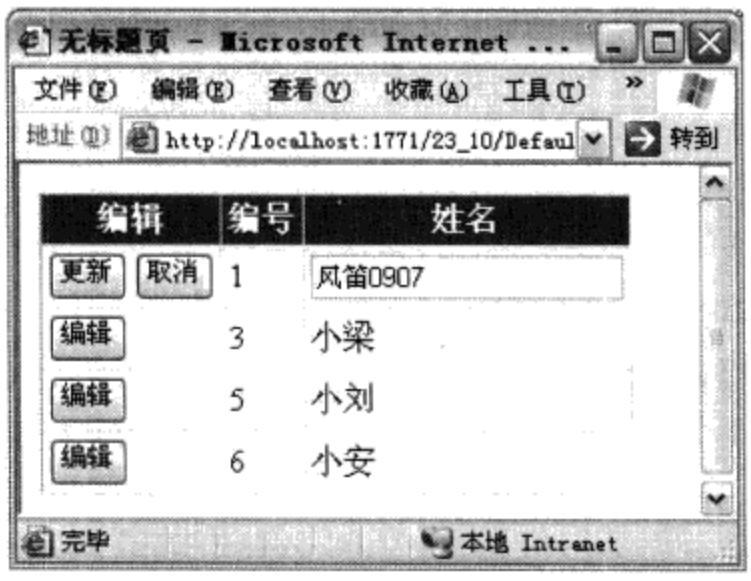


图 23.16　修改指定的信息

程序代码如下。

代码 23-10　光盘位置：光盘\mingrisoft\sl\23\23_10

当单击 GridView 控件上的“编辑”按钮时，将会触发 GridView 控件的 RowEditing 事件，在该事件下，指定需要编辑信息行的索引值。代码如下：

```
protected void GridView1_RowEditing(object sender, GridViewEditEventArgs e)
{
    //获取当前选择页的索引
    GridView1.EditIndex = e.NewEditIndex;
    //绑定数据
    BindData();
}
```

当单击 GridView 控件上的“更新”按钮时，将会触发 GridView 控件的 RowUpdating 事件，在该事件下，对指定信息进行更新。代码如下：

```
protected void GridView1_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    //获取 ID 值
    int ClassID = Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString());
```

```
        //获取编辑框中的数据
        string CName = ((TextBox)(GridView1.Rows[e.RowIndex].Cells[2].Controls[0])).Text.ToString();
        //声明一条 SQL 语句用于更新指定 ID 的 set P_Name 字段
        string sqlStr = "update tb_Employee set P_Name='" + CName + "' where ID=" + ClassID;
        //连接数据库
        SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //打开数据库
        myConn.Open();
        //创建 SqlCommand 对象
        SqlCommand myCmd = new SqlCommand(sqlStr, myConn);
        //执行 SQL 语句
        myCmd.ExecuteNonQuery();
        //释放对象占用的资源
        myCmd.Dispose();
        //关闭数据库连接
        myConn.Close();
        //取消编辑框
        GridView1.EditIndex = -1;
        //重新绑定数据
        BindData();
    }
```

当单击 GridView 控件上的“取消”按钮时，将会触发 GridView 控件的 RowCancelingEdit 事件，在该事件下，取消对指定信息进行编辑，其代码如下：

```
    protected void GridView1_RowCancelingEdit1(object sender, GridViewCancelEditEventArgs e)
    {
        //单击“取消”按钮，取消编辑
        GridView1.EditIndex = -1;
        //绑定数据
        BindData();
    }
```

23.2.4 GridView 控件应用实例

实例位置：光盘\mingrisoft\dxyy\23\23_02

在开发某些管理网站或系统时，常常需要通过选中某行前面的 CheckBox 复选框对该行信息进行操作。本实例通过在 GridView 控件中添加一个 CheckBox 模板列来删除指定行信息，运行结果如图 23.17 所示。

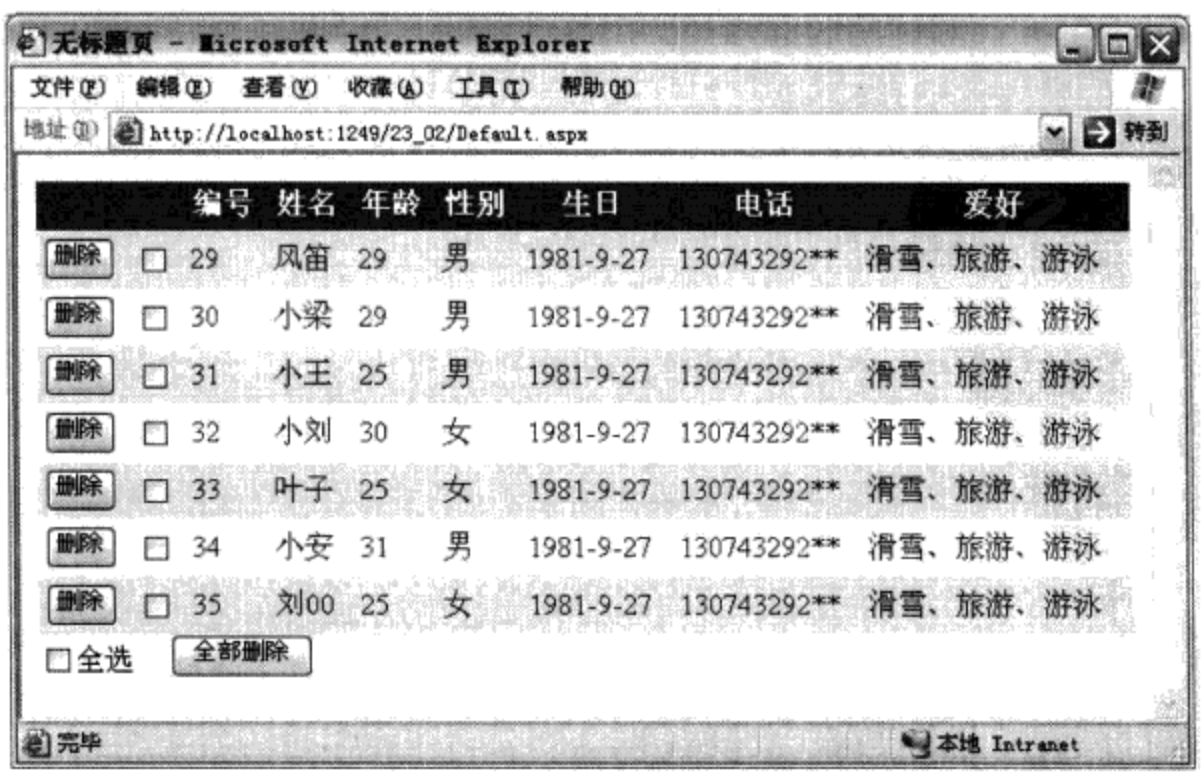


图 23.17 通过 CheckBox 删除选中记录

程序开发步骤如下。

（1）新建一个网站，命名为 23_02，其主页默认为 Default.aspx。

（2）在 Default.aspx 页面中添加 1 个 Table 表格，用来布局页面，然后在该 Table 表格中添加 1 个 GridView 控件、1 个 CheckBox 控件和 1 个 Button 控件，分别用来显示用户基本信息、将 GridView 控件中数据全部选中和删除选中信息的功能。

（3）主要程序代码如下。

当“全选”复选框选择状态发生改变时，循环访问 GridView 控件中的每一项，并使用 FindControl 方法取得 TemplateField 模板列中添加的 CheckBox 控件，然后建立该控件的引用对象，并使用该对象进行操作，实现全选功能。CheckBox 控件的 CheckedChanged 事件代码如下：

```
protected void CheckBox2_CheckedChanged(object sender, EventArgs e)
{
    //遍历控件数据的所有行
    for (int i = 0; i <= GridView1.Rows.Count-1;i++ )
    {
        //声明一个 CheckBox 对象
        CheckBox cbox = (CheckBox)GridView1.Rows[i].FindControl("CheckBox1");
        //如果选择了全选复选框
        if  (CheckBox2.Checked == true)
        {
            //选择所有的复选框
            cbox.Checked = true;
        }
        else//否则
        {
            //取消选择
            cbox.Checked = false;
        }
    }
}
```

单击“全部删除”按钮时，程序先判断 GridView 控件中哪些行被选中，并对选中行执行删除操作，然后对 GridView 控件进行重新绑定，以显示数据表中最新信息。“全部删除”按钮 Click 事件代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //遍历控件数据的所有行
    for (int i = 0; i <= GridView1.Rows.Count - 1; i++)
    {
        //声明一个 CheckBox 对象
        CheckBox cbox = (CheckBox)GridView1.Rows[i].FindControl("CheckBox1");
        //如果控件处于选择状态
        if (cbox.Checked == true)
        {
            //声明 SQL 语句删除指定 ID 的数据
            string sqlstr = "delete from tb_Employee where ID='" + GridView1.DataKeys[i].Value + "'";
            //连接数据库
            SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
            //创建 SqlCommand 对象
            SqlCommand myCmd = new SqlCommand(sqlstr, myConn);
            //打开数据库连接
            myConn.Open();
            //执行 SQL 语句
            myCmd.ExecuteNonQuery();
            //关闭数据库连接
            myConn.Close();
```

```
            }
        }
        //绑定数据
        BindData();
    }
```

23.3 DataList 控件

使用 DataList 控件可以显示模板定义的数据绑定列表，其内容可以通过模板进行控制，用户可以显示、选择和编辑多种不同数据源中的数据。

23.3.1 DataList 属性

DataList 控件常用属性及说明如表 23.10 所示。

表 23.10　　DataList 控件常用属性及说明

属　性	说　明
AlternatingItemStyle	获取 DataList 控件中交替项的样式属性
AlternatingItemTemplate	获取或设置 DataList 中交替项的模板
BackColor	获取或设置 Web 服务器控件的背景色
BorderColor	获取或设置 Web 控件的边框颜色
BorderStyle	获取或设置 Web 服务器控件的边框样式
BorderWidth	获取或设置 Web 服务器控件的边框宽度
CellPadding	获取或设置单元格的内容和单元格的边框之间的空间量
CellSpacing	获取或设置单元格间的空间量
CssClass	获取或设置由 Web 服务器控件在客户端呈现的级联样式表（CSS）类
DataKeyField	获取或设置由 DataSource 属性指定的数据源中的键字段
DataKeys	获取 DataKeyCollection 对象，该对象存储数据列表控件中每个记录的键值
DataMember	获取或设置多成员数据源中要绑定到数据列表控件的特定数据成员
DataSource	获取或设置源，该源包含用于填充控件中的项的值列表
DataSourceID	获取或设置数据源控件的 ID 属性，数据列表控件应使用它来检索其数据源
EditItemIndex	获取或设置 DataList 控件中要编辑的选定项的索引号
EditItemStyle	获取 DataList 控件中为进行编辑而选定的项的样式属性
EditItemTemplate	获取或设置一个值，该值指示是否对此控件应用主题
Enabled	获取或设置一个值，该值指示是否启用 Web 服务器控件
EnableTheming	获取或设置 DataGridView 的边框样式
EnableViewState	获取或设置一个值，该值指示服务器控件是否向发出请求的客户端保持自己的视图状态以及它所包含的任何子控件的视图状态
Font	获取与 Web 服务器控件关联的字体属性
FooterStyle	获取 DataList 控件的脚注部分的样式属性
FooterTemplate	获取或设置 DataList 控件的脚注部分的模板
ForeColor	获取或设置 Web 服务器控件的前景色
HeaderStyle	获取 DataList 控件的标题部分的样式属性

续表

属　性	说　明
HeaderTemplate	获取或设置 DataList 控件的标题部分的模板
Height	获取或设置 Web 服务器控件的高度
HorizontalAlign	获取或设置数据列表控件在其容器内的水平对齐方式
ID	获取或设置分配给服务器控件的编程标识符
Items	获取表示控件内单独项的 DataListItem 对象的集合
ItemStyle	获取 DataList 控件中项的样式属性
ItemTemplate	获取或设置 DataList 控件中项的模板
Page	获取对包含服务器控件的 Page 实例的引用
RepeatColumns	获取或设置要在 DataList 控件中显示的列数
RepeatDirection	获取或设置 DataList 控件是垂直显示还是水平显示
SelectedIndex	获取或设置 DataList 控件中的选定项的索引
SelectedItem	获取 DataList 控件中的选定项
SelectedItemStyle	获取 DataList 控件中选定项的样式属性
SelectedItemTemplate	获取或设置 DataList 控件中选定项的模板
SelectedValue	获取所选择的数据列表项的键字段的值
SeparatorStyle	获取 DataList 控件中各项间分隔符的样式属性
SeparatorTemplate	获取或设置 DataList 控件中各项间分隔符的模板
SkinID	获取或设置要应用于控件的外观
Visible	获取或设置一个值，该值指示服务器控件是否可见
Width	获取或设置 Web 服务器控件的宽度

下面对比较重要的属性进行详细介绍。

（1）DataKeyField 属性。用于设置由 DataSource 属性指定数据源中的键的字段。语法格式如下：

```
public virtual string DataKeyField { get; set; }
```

属性值：DataSource 属性指定的数据源中的键字段的名称。

（2）DataSource 属性。用于设置 DataList 控件所显示数据的数据源。语法格式如下：

```
public virtual Object DataSource { get; set; }
```

属性值：用于为 DataList 控件提供数据值的集合。

（3）DataKeys 属性。用于获取 DataKeyCollection 对象，在 DataKeyCollection 对象中存储数据列表控件中每个记录的键值。语法格式如下：

```
public DataKeyCollection DataKeys { get; }
```

属性值：返回 DataKeyCollection 对象。

（4）Items 属性。用于获取数据控件内单独项的 DataListItem 对象的集合。语法格式如下：

```
public virtual DataListItemCollection Items { get; }
```

属性值：返回 DataList 控件内单独项的 DataListItem 对象的集合。

示例 23-11　DataKeyField、DataSource、DataKeys 和 Items 属性的使用

运行示例显示所有的商品信息，单击“删除”按钮后，可以删除指定的商品信息，运行结果如图 23.18 所示。

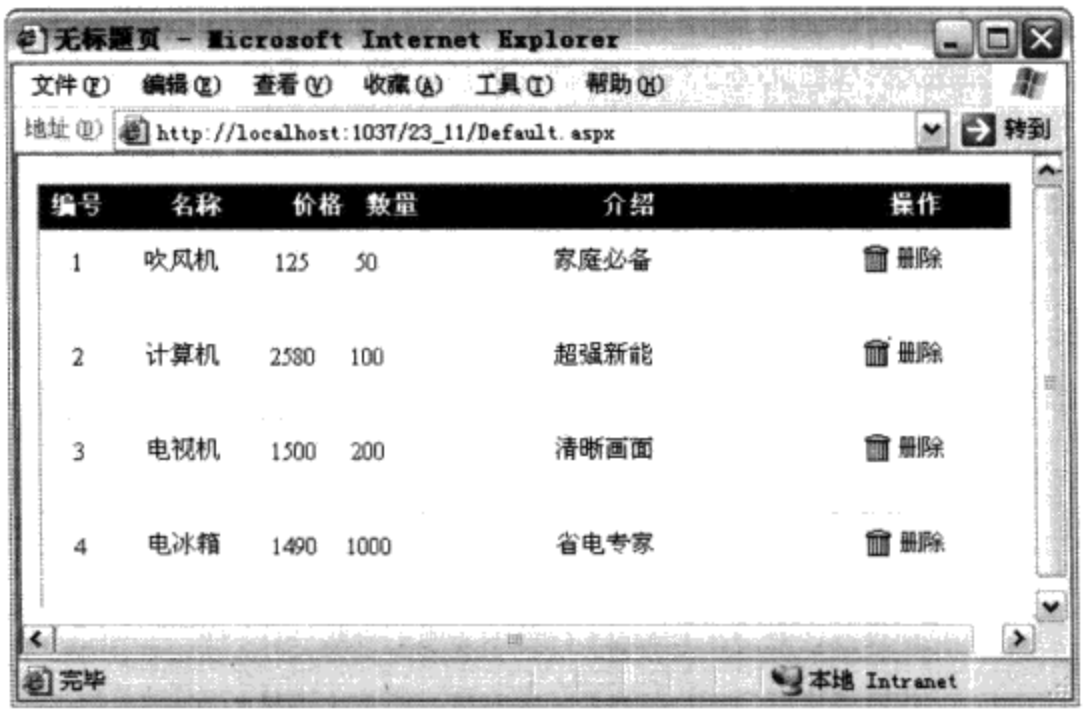


图 23.18　显示删除后的商品信息

程序代码如下。

代码 23-11　光盘位置：光盘\mingrisoft\sl\23\23_11

在绑定商品数据信息时，通过指定 DataList 控件的 DataSource 属性值，获取商品信息，并通过 DataKeyField 属性指定该商品信息中的键字段的名称。其代码如下：

```
private void BindData()
{
    //连接数据库
    SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //打开数据库连接
    myConn.Open();
    //声明 sql 语句检索所有数据
    string sqlStr = "select * from tb_Good";
    //创建 SqlDataAdapter 对象
    SqlDataAdapter da = new SqlDataAdapter(sqlStr, myConn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充 DataSet 对象
    da.Fill(ds, "Goods");
    //设置数据源
    DataList1.DataSource = ds.Tables["Goods"].DefaultView;
    //设置键值
    DataList1.DataKeyField = "ID";
    //绑定数据
    DataList1.DataBind();
    //关闭数据库连接
    myConn.Close();
}
```

在网页的 HTML 代码中，通过 Eval 绑定需要显示的字段，代码如下：

```
<asp:DataList ID="DataList1" runat="server" BackColor="White"
        BorderColor="#3366CC" BorderStyle="None" BorderWidth="1px" CellPadding="4"
        GridLines="Both" Width="377px" Font-Size="Small"
        onitemcommand="DataList1_ItemCommand">
        <FooterStyle BackColor="#99CCCC" ForeColor="#003399" />
        <ItemStyle BackColor="White" ForeColor="#003399" />
        <SelectedItemStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />
        <HeaderTemplate>
            <table cellpadding="0" cellspacing="0" class="style4">
                <tr style="color: #FFFFFF">
```

```
                <td class="style5" align="center">
                    编号</td>
                <td align="center">
                    名称</td>
                <td class="style6" align="center">
                    价格</td>
                <td class="style7" align="center">
                    数量</td>
                <td class="style8" align="center">
                    介绍</td>
                <td align="center">
                    操作</td>
            </tr>
        </table>
    </HeaderTemplate>
    <HeaderStyle BackColor="#003399" Font-Bold="True" Font-Italic="False"
        Font-Names="宋体" Font-Overline="False" Font-Size="Small" Font-Strikeout="False"
        Font-Underline="False" ForeColor="#CCCCFF" HorizontalAlign="Center"
        VerticalAlign="Middle" />
    <ItemTemplate>
        <table cellpadding="0" cellspacing="0" class="style4">
            <tr>
                <td class="style5" align="center">
                    <%#Eval("ID") %></td>                    //绑定 ID 字段
                <td align="center">
                    <%#Eval("goodName")%></td>               //绑定 goodName 字段
                <td class="style6" align="center">
                    <%#Eval("goodPrice")%></td>              //绑定 goodPrice 字段
                <td class="style7" align="center">
                    <%#Eval("goodAmount")%></td>             //绑定 goodAmount 字段
                <td class="style8" align="center">
                    <%#Eval("goodIntroduction")%></td>       //绑定 goodIntroduction 字段
                <td align="center">
                    <asp:ImageButton ID="ImageButton1" runat="server" CommandName="delete"
                        ImageUrl="~/del_2.gif" />
                </td>
            </tr>
        </table>
        <br />
    </ItemTemplate>
</asp:DataList>
```

当单击“删除”按钮后，在 DataList 控件的 ItemCommand 事件下，使用 DataList 控件的 DataKeys 属性值获取数据列表控件中被删除记录的键值，实现删除功能，其代码如下：

```
protected void DataList1_ItemCommand(object source, DataListCommandEventArgs e)
{
    //如果 CommandName 属性为 delete
    if (e.CommandName == "delete")
    {
        //获取 ID 值
        int GoodsID = Convert.ToInt32(DataList1.DataKeys[e.Item.ItemIndex]);
        //声明 sql 语句根据指定的 ID 值删除数据
        string sqlStr = "delete   from tb_Good where ID=" + GoodsID;
        //连接数据库
        SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //打开数据库连接
        myConn.Open();
        //创建 SqlCommand 对象
        SqlCommand myCmd = new SqlCommand(sqlStr, myConn);
        //执行 sql 语句
        myCmd.ExecuteNonQuery();
        //释放 SqlCommand 对象
        myCmd.Dispose();
```

```
            //释放数据库连接
            myConn.Dispose();
            //重新绑定数据
            BindData();
        }
    }
```

（5）EditItemIndex 属性。用于获取或设置 DataList 控件中要编辑的选定项的索引号。语法格式如下：

```
public virtual int EditItemIndex { get; set; }
```

属性值：DataList 控件中要编辑的选定项的索引号。

例如，使用 EditItemIndex 属性指定要编辑的项，其代码如下：

```
DataList1.EditItemIndex = e.Item.ItemIndex;
```

23.3.2 DataList 方法

DataList 控件常用方法及说明如表 23.11 所示。

表 23.11　　DataList 控件常用方法及说明

方　法	说　明
DataBind	将数据源绑定到 DataList 控件
FindControl	在当前的命名容器中搜索指定的服务器控件
Focus	为控件设置输入焦点
GetHashCode	用作特定类型的哈希函数
GetType	获取当前实例的 Type
HasControls	确定服务器控件是否包含任何子控件
IsBindableType	确定指定的数据类型是否能绑定到 DataList 控件中的列
ToString	返回表示当前 Object 的 String

DataList 控件的方法中比较重要的是 DataBind 方法，下面对其进行详细介绍，此方法用于将数据源绑定到 DataList 控件。

语法：

```
public override void DataBind ()
```

交叉链接：关于 DataBind 方法的使用，可参见示例 23-11。

23.3.3 DataList 事件

DataList 控件常用事件及说明如表 23.12 所示。

表 23.12　　DataList 控件常用事件及说明

事　件	说　明
CancelCommand	对 DataList 控件中的某个项，单击“Cancel”按钮时发生
DeleteCommand	对 DataList 控件中的某个项，单击“Delete”按钮时发生
EditCommand	对 DataList 控件中的某个项，单击“Edit”按钮时发生
ItemCommand	当单击 DataList 控件中的任一按钮时发生
ItemCreated	当在 DataList 控件中创建项时在服务器上发生
ItemDataBound	当项被数据绑定到 DataList 控件时发生
Load	在 GridView 控件中创建行时发生
SelectedIndexChanged	当数据列表控件中选择了不同项时发生
Unload	当服务器控件从内存中卸载时发生
UpdateCommand	对 DataList 控件中的某个项，单击 Update 按钮时发生

下面对比较重要的事件进行详细介绍。

（1）CancelCommand 事件。对 DataList 控件中的某个项，单击“Cancel”按钮时发生。语法格式如下：

```
public event DataListCommandEventHandler CancelCommand
```

（2）DeleteCommand 事件。对 DataList 控件中的某个项，单击“Delete”按钮时发生。语法格式如下：

```
public event DataListCommandEventHandler DeleteCommand
```

（3）EditCommand 事件。对 DataList 控件中的某个项，单击“Edit”按钮时发生。语法格式如下：

```
public event DataListCommandEventHandler EditCommand
```

（4）UpdateCommand 事件。对 DataList 控件中的某个项，单击“Update”按钮时发生。语法格式如下：

```
public event DataListCommandEventHandler UpdateCommand
```

示例 23-12 CancelCommand、DeleteCommand、EditCommand 和 UpdateCommand 事件的使用

本示例主要对商品信息进行编辑、修改和删除操作。示例运行结果如图 23.19 所示，当单击“编辑”按钮后，将会弹出如图 23.20 所示的界面；当单击“更新”按钮后，更新商品信息，并弹出如图 23.21 所示的界面；当在图 23.20 所示的界面中，单击“删除”按钮后，将会删除指定商品的信息。

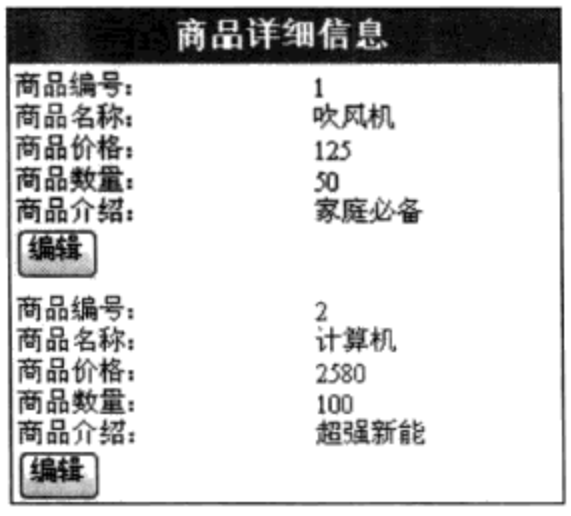


图 23.19 显示商品信息

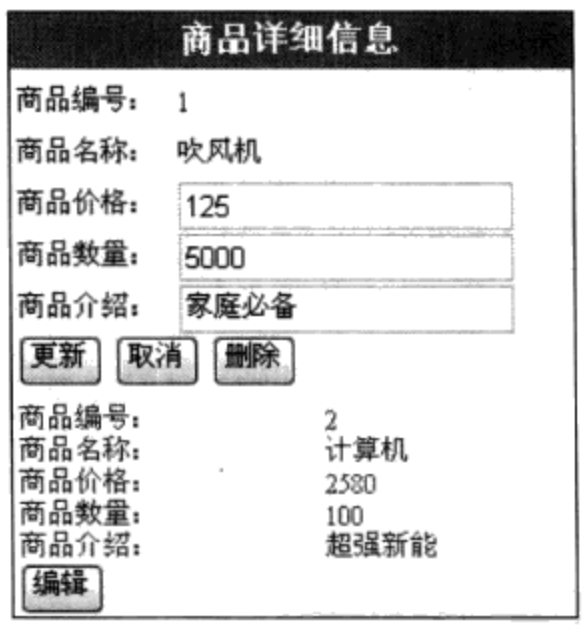


图 23.20 编辑商品信息

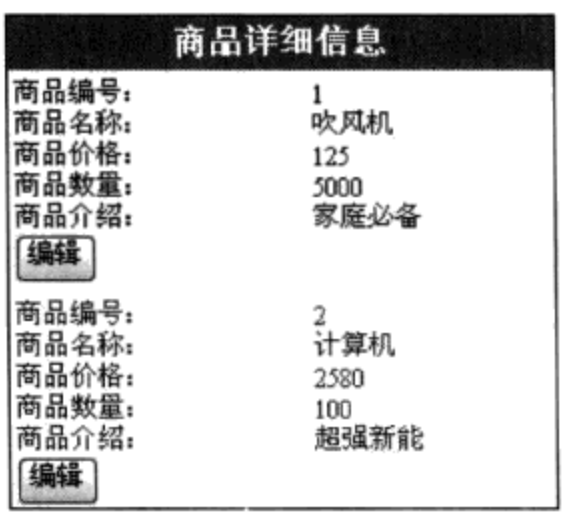


图 23.21 修改后的商品信息

程序代码如下。

代码 23-12　光盘位置：光盘\mingrisoft\sl\23\23_12

当单击商品显示信息列表中的“编辑”按钮时，将会触发 DataList 控件的 EditCommand 事件，在该事件下，设置该控件的编辑项的索引为当前选择的行的索引。其代码如下：

```
protected void DataList1_EditCommand(object source, DataListCommandEventArgs e)
{
    //编辑项的索引为当前选择行的索引
    DataList1.EditItemIndex = e.Item.ItemIndex;
    //重新绑定数据
    BindData();
}
```

在编辑商品信息对话框中，当单击“取消”按钮时，将会触发 DataList 控件的 CancelCommand 事件，在该事件下，设置该控件的编辑项的索引为-1。其代码如下：

```
protected void DataList1_CancelCommand(object source, DataListCommandEventArgs e)
{
    //取消编辑
    DataList1.EditItemIndex = -1;
    //重新绑定数据
    BindData();
}
```

在编辑商品信息对话框中，当单击“更新”按钮时，首先取得当前编辑行的记录的键值，其次取得在文本框中输入的内容，然后更新数据库内容。其代码如下：

```
protected void DataList1_UpdateCommand(object source, DataListCommandEventArgs e)
{
    //获取 ID 值
    int GoodsID = Convert.ToInt32(DataList1.DataKeys[e.Item.ItemIndex]);
    //获取编辑后的价格
    string Price = ((TextBox)e.Item.FindControl("txtPrice")).Text;
    //设置编辑后的数量
    string Amount = ((TextBox)e.Item.FindControl("txtAmount")).Text;
    //设置编辑后的产品介绍
    string Introduction = ((TextBox)e.Item.FindControl("txtIntroduction")).Text;
    //声明 SQL 语句用于修改数据表中指定 ID 的数据
    string sqlStr = "update tb_Good set goodPrice='" + Price + "', goodAmount='" + Amount + "',
goodIntroduction='"+ Introduction+"'";
    sqlStr += " where ID=" + GoodsID;
    //连接数据库
    SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //打开数据库连接
    myConn.Open();
    //创建 SqlCommand 对象
    SqlCommand myCmd = new SqlCommand(sqlStr, myConn);
    //执行 sql 语句
    myCmd.ExecuteNonQuery();
    //释放 SqlCommand 对象
    myCmd.Dispose();
    //释放数据库连接
    myConn.Dispose();
    //取消编辑
    DataList1.EditItemIndex = -1;
    //重新绑定数据
    BindData();
}
```

在编辑商品信息对话框中，当单击“删除”按钮时，首先取得当前删除行的记录的键值，然后删除指定行的内容。其代码如下：

```
protected void DataList1_DeleteCommand(object source, DataListCommandEventArgs e)
{
    //获取 ID 值
```

```
        int GoodsID = Convert.ToInt32(DataList1.DataKeys[e.Item.ItemIndex]);
        //声明 sql 语句用于根据指定的 ID 删除数据
        string sqlStr = "delete   from tb_Good where ID=" + GoodsID;
        //连接数据库
        SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //打开数据库连接
        myConn.Open();
        //创建 SqlCommand 对象
        SqlCommand myCmd = new SqlCommand(sqlStr, myConn);
        //执行 SQL 语句
        myCmd.ExecuteNonQuery();
        //释放 SqlCommand 对象
        myCmd.Dispose();
        //释放数据库连接
        myConn.Dispose();
        //取消编辑
        DataList1.EditItemIndex = -1;
        //重新绑定数据
        BindData();
    }
```

（5）ItemCommand 事件。当单击 DataList 控件中的任一按钮时发生。

交叉链接：关于 ItemCommand 事件的使用，可参见示例 23-11。

23.3.4 DataList 控件应用实例

实例位置：光盘\mingrisoft\dxyy\23\23_03

在网站开发中，经常需要使用 DataList 控件显示具有主次结构的数据。本实例中，通过在 DataList 控件中嵌套一个 DataList 控件来根据员工姓名显示员工的详细信息。实例运行结果如图 23.22 所示。

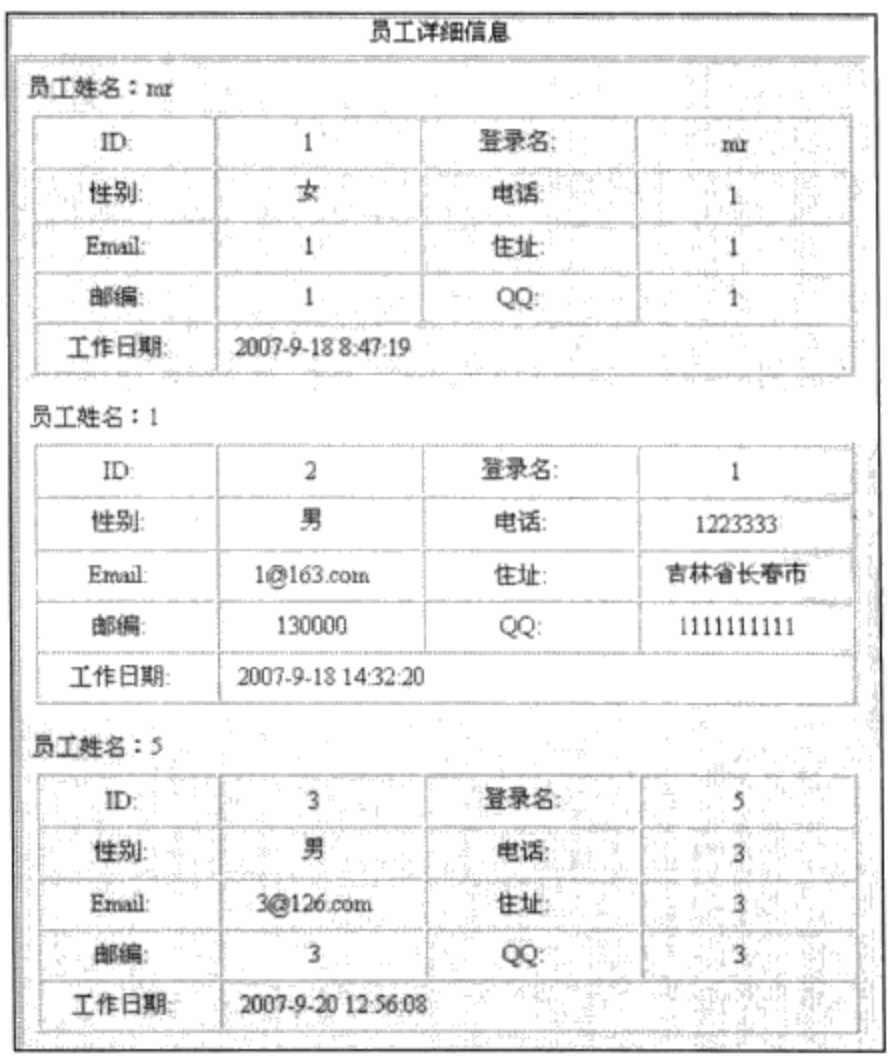


图 23.22 使用 DataList 控件显示具有主次结构的数据

程序开发步骤如下。

（1）新建一个网站，将其命名为 23_03，默认主页名为 Default.aspx。

（2）Default.aspx 页面涉及的控件如表 23.13 所示。

表 23.13　　Default.aspx 页面用到的控件

控件类型	控件名称	用　途
HTML/Table		布局页面/DataList 项模板
标准/Label	labUserID	员工 ID
	labName	登录名
	labSex	性别
	labTel	电话
	labEmail	E-mail 地址
	labAddress	住址
	labPostCode	邮编
	labQQ	QQ
	labDate	工作日期
数据/DataList	DataList1	绑定主表
	DataList2	绑定符合主表中字段的详细信息

（3）主要程序代码如下。

在 Default.aspx 页的 Page_Load 事件中，首先从数据表中检索数据，同时取得要显示的主数据和从数据，填充到 DataSet 中，然后根据用户姓名创建 DataRelation 对象，并使用数据绑定方法将相关信息绑定到 DataList 控件中。Default.aspx 页 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //连接数据库
    SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
    //声明 sql 语句检索数据库
    string sqlStr = "select top 3* from tb_EmpID select top 3a.*,b.UserID from tb_User as a inner join tb_EmpID as b
on a.UserID=b.UserID";
    //创建 SqlDataAdapter 对象
    SqlDataAdapter myda = new SqlDataAdapter(sqlStr, myConn);
    //创建 DataSet 对象
    DataSet myds = new DataSet();
    //打开数据库连接
    myConn.Open();
    //填充 DataSet 对象
    myda.Fill(myds);
    //设置 DataSet 中第一个表的名称
    myds.Tables[0].TableName = "tb_EmpID";
    //设置 DataSet 中第二个表的名称
    myds.Tables[1].TableName = "tb_Employee";
    //创建两个 DataColumn 对象
    DataColumn Parent = myds.Tables["tb_EmpID"].Columns["UserLoginName"];
    DataColumn Child = myds.Tables["tb_Employee"].Columns["UserLoginName"];
    DataRelation tableRelation = new DataRelation("tableRelation", Parent, Child, false);
    myds.Relations.Add(tableRelation);
    //设置数据源
    DataList1.DataSource = myds.Tables["tb_EmpID"].DefaultView;
    //绑定数据
    DataList1.DataBind();
    //关闭数据库连接
    myConn.Close();
}
```

23.4 Repeater 控件

Repeater 控件是一个基本容器控件，它可以从页的任何可用数据中创建出自定义列表。Repeater 控件支持几个模板，其中最重要的是 ItemTemplate 模板，用户可以在此模板中给 Repeater 控件添加绑定代码，以显示数据源中字段。

23.4.1 Repeater 属性

Repeater 控件常用属性及说明如表 23.14 所示。

表 23.14　　Repeater 控件常用属性及说明

属　性	说　明
AlternatingItemTemplate	定义如何显示控件中的交替项
DataMember	获取或设置 DataSource 属性要绑定到控件的特定表
DataSource	获取或设置为填充列表提供数据的数据源
DataSourceID	设置数据源控件的 ID 属性，Repeater 控件使用数据源控件的 ID 属性检索其数据源
EnableTheming	获取或设置一个值，该值指示主题是否应用于此控件
FooterTemplate	定义如何显示 Repeater 控件的注脚部分
HeaderTemplate	定义如何显示 Repeater 控件的标头部分
ID	获取或设置分配给服务器控件的编程标识符
Items	获取 Repeater 中 RepeaterItem 对象的集合
ItemTemplate	定义如何显示 Repeater 控件中的项
SkinID	获取或设置要应用于控件的外观
Visible	获取或设置一个值，该值指示服务器控件是否可见

下面对比较重要的属性进行详细介绍。

（1）DataSource 属性。用于获取或设置为填充列表提供数据的数据源。语法格式如下：

```
public virtual Object DataSource { get; set; }
```

属性值：用于为 Repeater 控件提供数据的值的集合。

（2）AlternatingItemTemplate 属性。根据 ItemTemplate 中指定的内容，为 Repeater 控件中的交替项提供不同的外观。语法格式如下：

```
public virtual ITemplate AlternatingItemTemplate { get; set; }
```

属性值：用于定义如何显示交替项。

（3）FooterTemplate 属性。该属性创建一个模板，控制如何显示 Repeater 控件的注脚部分。语法格式如下：

```
public virtual ITemplate FooterTemplate { get; set; }
```

属性值：定义如何显示 Repeater 控件的注脚部分。

（4）HeaderTemplate 属性。该属性创建一个模板，控制如何显示 Repeater 控件的标头部分。语法格式如下：

```
public virtual ITemplate HeaderTemplate { get; set; }
```

属性值：定义如何显示 Repeater 控件的标头部分。

（5）ItemTemplate 属性。该属性创建一个模板，控制如何显示 Repeater 控件中的项。语法格式如下：

```
public virtual ITemplate ItemTemplate { get; set; }
```

属性值：定义如何显示 Repeater 控件中的项。

示例 23-13　DataSource、AlternatingItemTemplate、FooterTemplate、HeaderTemplate 和 ItemTemplate 属性的使用

本示例主要是通过属性 DataSource 将数据信息绑定到 Repeater 控件中，并通过属性 AlternatingItemTemplate、FooterTemplate、HeaderTemplate 和 ItemTemplate，设置 Repeater 控件的外观，同时将数据显示出来，示例运行结果如图 23.23 所示。

图 23.23　利用 Repeater 控件显示数据信息

程序代码如下。

代码 23-13　光盘位置：光盘\mingrisoft\sl\23\23_13

Default.aspx 页面在初始化时，通过设置 Repeater 控件的 DataSource 属性值，将一维数组绑定到 Repeater 控件中。Default.aspx 页 Page_Load 事件代码如下：

```
//声明 ArrayList 动态数组
protected ArrayList ItemList = new ArrayList();
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        //向动态数组中添加数据
        ItemList.Add("Monday");
        ItemList.Add("Tuesday");
        ItemList.Add("Wednesday");
        temList.Add("Thursday");
        temList.Add("Friday");
        ItemList.Add("Saturday");
        ItemList.Add("Sunday");
        //设置控件数据源
        Repeater1.DataSource = ItemList;
        //绑定数据
        Repeater1.DataBind();
    }
}
```

在 Default.aspx 页的 HTML 代码中，通过属性 AlternatingItemTemplate、FooterTemplate、HeaderTemplate 和 ItemTemplate，设置 Repeater 控件的外观，并将数据显示出来，其代码如下：

```
<asp:Repeater id=Repeater1 runat="server">
    <HeaderTemplate>
        <table border=1>
        <tr>
        <td style="background-color:AliceBlue; font-size :9pt; font-family :宋体;">
            <b>星期列表</b>
        </td>
```

```
    </tr>
</HeaderTemplate>
<AlternatingItemTemplate>
    <tr>
    <td style="background-color: lightblue ;font-size :9pt; font-family :宋体;">
        <b><%# Container.DataItem %></b>
      </td>
    </tr>
</AlternatingItemTemplate>
<ItemTemplate>
    <tr>
     <td style="background-color:LightGrey;font-size :9pt; font-family :宋体;">
        <%# Container.DataItem %>
       </td>
     </tr>
</ItemTemplate>
<FooterTemplate>
     </table>
  </FooterTemplate>
</asp:Repeater>
```

23.4.2 Repeater 方法

Repeater 控件常用方法及说明如表 23.15 所示。

表 23.15　　Repeater 控件常用方法及说明

方　法	说　明
DataBind	将数据源绑定到 Repeater 控件
FindControl	在当前的命名容器中搜索指定的服务器控件
Focus	为控件设置输入焦点
GetHashCode	用作特定类型的哈希函数
GetType	获取当前实例的 Type
HasControls	确定服务器控件是否包含任何子控件
ResolveClientUrl	获取浏览器可以使用的 URL
ResolveUrl	将 URL 转换为在请求客户端可用的 URL
ToString	返回表示当前 Object 的 String

下面对比较重要的方法进行详细介绍。

（1）DataBind 方法。用于将数据源绑定到 Repeater 控件。语法格式如下：

```
public override void DataBind ()
```

交叉链接：关于 DataBind 方法的使用，可参见示例 23-13。

（2）FindControl 方法。用于在当前的命名容器中搜索带指定 id 参数的服务器控件。语法格式如下：

```
public virtual Control FindControl (string id )
```

参数说明如下。

Id：要查找的控件的标识符。

例如，在 Button 按钮的 Click 事件下，使用 FindControl 方法查找 ID 属性为 TextBox2 的控件，其代码如下：

```
private void Button1_Click(object sender, EventArgs MyEventArgs)
{
    //创建 Control 对象
    Control myControl1 = FindControl("TextBox2");
    if (myControl1 != null)//如果存在控件
    {
```

```
            //获取 TextBox2 控件的父控件
            Control myControl2 = myControl1.Parent;
            //显示父控件 ID
            Response.Write("TextBox2 控件的父控件的 ID 是 :" + myControl2.ID);
        }
        else                                                    //否则
        {
            //提示控件不存在
            Response.Write("TextBox2 控件不存在");
        }
    }
```

23.4.3 Repeater 事件

Repeater 控件常用事件及说明如表 23.16 所示。

表 23.16　　Repeater 控件常用事件及说明

事　件	说　明
DataBinding	当服务器控件绑定到数据源时发生
ItemCommand	当单击 Repeater 控件中的按钮时发生
ItemCreated	当在 Repeater 控件中创建项时在服务器上发生
ItemDataBound	当项被数据绑定到 Repeater 控件时发生
Load	当服务器控件加载到 Page 对象中时发生
Unload	当服务器控件从内存中卸载时发生

ItemCommand 事件：当单击 Repeater 中的按钮时引发该事件。语法格式如下：

```
public event RepeaterCommandEventHandler ItemCommand
```

示例 23-14　ItemCommand 事件的使用

本示例主要是介绍 Repeater 控件的 ItemCommand 事件的使用，示例运行结果如图 23.24 所示，当单击“添加”按钮后，将在界面上显示被单击的控件名，运行结果如图 23.25 所示。

图 23.24　控件被单击前

图 23.25　控件被单击后

程序代码如下。

代码 23-14　光盘位置：光盘\mingrisoft\sl\23\23_14

Default.aspx 页面在初始化时，通过设置 Repeater 控件的 DataSource 属性值，将一维数组绑定到 Repeater 控件中。Default.aspx 页 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        //声明一个 ArrayList 动态数组
        ArrayList ItemList = new ArrayList();
        //向数组中添加数据
        ItemList.Add(new PositionData("添加", "添加"));
```

```
            ItemList.Add(new PositionData("删除", "删除"));
            ItemList.Add(new PositionData("修改", "修改"));
            ItemList.Add(new PositionData("取消", "取消"));
            //设置控件的数据源
            Repeater1.DataSource = ItemList;
             //绑定数据
            Repeater1.DataBind();
        }
    }
```

在 Repeater 控件的 ItemCommand 事件下，添加程序代码，将被选择 Button 按钮显示出来，其代码如下：

```
protected void Repeater1_ItemCommand(object source, RepeaterCommandEventArgs e)
{
    //显示单击按钮的类型
    Label2.Text = "单击了" + ((Button)e.CommandSource).Text + "按钮。 <br>";
}
```

23.4.4 Repeater 控件应用实例

实例位置：光盘\mingrisoft\dxyy\23\23_04

在开发网站时，常常需要在页面中比较全面地显示一些信息，但如果要显示的信息记录较多，用一个页面显示所有记录可能给用户的浏览带来不便，为了解决这个问题，开发人员可以使用分页技术来限定一个页面中显示的记录数。本实例主要是通过使用 Repeater 控件的分页技术显示员工信息，实例运行结果如图 23.26 所示。

使用Repeater控件分页显示员工信息			
ID:	111	登录名:	222
性别:	男	电话:	111
Email:	lxkou6666@sohu.com	住址:	长春市人民大街29号
邮编:	130021	QQ:	124067691
工作日期:	2007-9-8 16:38:37		
ID:	222	登录名:	风笛
性别:	男	电话:	00000000
Email:	lovelvshuang@163.com	住址:	吉林省长春市
邮编:	130031	QQ:	49490000
工作日期:	2009-5-30 13:08:01		

当前页码为：[6] 总页码为：[6] 第一页 上一页 下一页 最后一页

图 23.26 分页显示员工信息

程序开发步骤如下。

(1) 新建一个网站，将其命名为 23_04，默认主页名为 Default.aspx。

(2) Default.aspx 页面涉及的控件如表 23.17 所示。

表 23.17 Defaultaspx 页面用到的控件

控件类型	控件名称	用途
HTML/Table	table1	布局页面/Repeater 项模板
标准/Label	labUserID	员工 ID
	labName	登录名
	labSex	性别
	labTel	电话
	labEmail	E-mail 地址

续表

控件类型	控件名称	用　途
标准/Label	labAddress	住址
	labPostCode	邮编
	labQQ	QQ
	labDate	工作日期
	labPage	显示当前页
	LabBackPage	显示总页码
标准/LinkButton	lnkbtnOne	第一页
	lnkbtnUp	上一页
	lnkbtnNext	下一页
	lnkbtnBack	最后一页
数据/Repeater	Repeater1	绑定员工信息

（3）主要程序代码如下。

Default.aspx 页面在初始化时，调用自定义方法 DBBind 对 Repeater 控件进行数据绑定并实现分页功能，Default.aspx 页 Page_Load 事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)          //页面第一次加载时
        DBBind();             //绑定数据
}
```

自定义方法 DBBind，主要是用来从数据库中查询出符合指定条件的记录，并绑定到 Repeater 控件中，然后通过设置 PagedDataSource 类对象的 AllowPaging 属性值为 true，实现 Repeater 控件的分页功能。DBBind 方法关键代码如下：

```
public void DBBind()
{
        //获取当前页码
        int curpage = Convert.ToInt32(this.labPage.Text);
        //实例化 PagedDataSource
        PagedDataSource ps = new PagedDataSource();
        //连接数据库
        SqlConnection myConn = new SqlConnection("server=.;database=db_23;uid=sa;pwd=");
        //打开数据库连接
        myConn.Open();
        //声明 sql 语句检索 tb_Employee 表中所有数据
        string sqlStr = "select * from tb_Employee";
        //实例化 SqlDataAdapter 对象
        SqlDataAdapter myAdapter = new SqlDataAdapter(sqlStr, myConn);
        //创建 DataSet 对象
        DataSet ds = new DataSet();
        //填充 DataSet 对象
        myAdapter.Fill(ds);
        //设置控件数据源
        ps.DataSource = ds.Tables[0].DefaultView;
        //设置是否可以分页
        ps.AllowPaging = True;
        //每页显示数据的数目
        ps.PageSize = 2;
        //取得当前页的页码
        ps.CurrentPageIndex = curpage - 1;
        //设置按钮状态
        this.lnkbtnUp.Enabled = True;
        this.lnkbtnNext.Enabled = True;
        this.lnkbtnBack.Enabled = True;
        this.lnkbtnOne.Enabled = True;
```

```
        //如果 curpage 等于 1 则说明当前页为第一页
        if (curpage == 1)
        {
            this.lnkbtnOne.Enabled = False;//不显示第一页按钮
            this.lnkbtnUp.Enabled = False;//不显示上一页按钮
        }
        //当 curpage 等于 PageCount，说明当前页是最后一页
        if (curpage == ps.PageCount)
        {
            this.lnkbtnNext.Enabled = False;//不显示下一页
            this.lnkbtnBack.Enabled = False;//不显示最后一页
        }
        //显示总页码
        this.labBackPage.Text = Convert.ToString(ps.PageCount);
        //设置控件数据源
        this.Repeater1.DataSource = ps;
        //绑定数据
        this.Repeater1.DataBind();
        //关闭数据库连接
        myConn.Close();
    }
```

当用户单击用于操作分页的 LinkButton 控件时，程序根据当前页码执行指定操作。用于控制分页的 LinkButton 控件的 Click 事件代码如下：

```
    //第一页
    protected void lnkbtnOne_Click(object sender, EventArgs e)
    {
        this.labPage.Text = "1";
        this.DBBind();
    }
    //上一页
    protected void lnkbtnUp_Click(object sender, EventArgs e)
    {
        this.labPage.Text = Convert.ToString(Convert.ToInt32(this.labPage.Text) - 1);
        this.DBBind();
    }
    //下一页
    protected void lnkbtnNext_Click(object sender, EventArgs e)
    {
        this.labPage.Text = Convert.ToString(Convert.ToInt32(this.labPage.Text) + 1);
        this.DBBind();
    }
    //最后一页
    protected void lnkbtnBack_Click(object sender, EventArgs e)
    {
        this.labPage.Text = this.labBackPage.Text;
        this.DBBind();
    }
```

第 24 章

LINQ 技术

- LINQ 概述
- LINQ 语法基础

24.1 LINQ 概述

LINQ 是微软公司提供的一项新技术，它能够将查询功能直接引入.Net Framework 3.5 所支持的编程语言（如 C#、Visual Basic.NET 等）中。查询操作可以通过编程语言自身来传达，而不是以字符串嵌入到应用程序代码中。LINQ 主要包括 4 个组件：LINQ to Objects、LINQ to DataSet、LINQ to SQL 和 LINQ to XML。使用 LINQ 查询和处理数据，具有以下 5 个优点。

（1）LINQ 查询语法简单、容易书写。在创建查询表达式时，Visual Studio 2008 集成开发环境还提供了智能提示功能。

（2）由于查询表达式被嵌入在编程语言中，因此，编译器将检查查询表达式的语法错误和查询数据的类型安全。

（3）LINQ 提供了强大的过滤、排序、数据分区、分组等处理数据的功能。

（4）使用 LINQ 可以直接处理 XML 元素，并为内存中的 XML 文档提供强大的处理功能。

（5）容易处理多数据源和多数据格式的数据。

24.2 LINQ 语法基础

24.2.1 LINQ to Objects

LINQ to Objects 是一种新的处理集合的方法，采用 LINQ to Objects 技术，只需编写描述要检索内容的声明性代码。LINQ to Objects 能够直接使用 LINQ 查询 IEnumerable 或 IEnumerable<T> 集合，而不需要使用 LINQ 提供程序或 API，可以说，使用 LINQ 能够查询任何可枚举的集合，例如数组、泛型列表等。

1．LINQ 与静态数组

长度不能改变的数组被称为静态数组，通过 LINQ 技术可以很方便地查询静态数组。

例如，声明一个静态数组，然后通过 LINQ 查询长度大于 3 的字符串，代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    //声明一个字符串数组
    string[] userName= { "风笛","芸烨湘枫","小梁","小王"};
    //声明 LINQ 查询语句
    var values = from s in userName where s.Length > 3 select s;
    foreach (var v in values)
    {
        //输出查询结果
        MessageBox.Show(v.ToString());
    }
}
```

当查询整型数组时，还可以将查询结果进行排序，主要是使用 Descending（降序）和 Ascending（升序）实现的。

例如，声明一个整型数组，然后通过 LINQ 查询大于 3 的元素，并对查询结果进行降序排列，代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    //声明一个 int 类型的数组
    int[] P_int = { 0,1,3,5,7,8,9,10,12};
    //通过 LINQ 查询大于 3 的元素
    var values = from s in P_int where s> 3 orderby s descending select s;
    foreach (var v in values)
```

```
    {
        //输出查询结果
        listBox1.Items.Add(v.ToString());
    }
}
```

2．LINQ 与动态数组

长度可以修改的数组被称为动态数组，例如 ArrayList 等。使用 LINQ 能够查询动态数组。

例如，首先声明一个 ArrayList 动态数组，用于存储考生考试成绩，然后通过 LINQ 查询动态数组中分数大于 95 的考生，代码如下：

```
//创建一个公共类
public class Student
 {
     //定义两个公共变量
     public string strName { get; set; }
     public int[] Scores { get; set; }
 }
 private void Form1_Load(object sender, EventArgs e)
 {
     //实例化一个 ArrayList 动态数组
     ArrayList arrList = new ArrayList();
     //向动态数组中添加数据
     arrList.Add(new Student{strName = "风笛",Scores = new int[] { 98, 92, 81, 60 }});
     arrList.Add(new Student{strName = "芸烨湘枫",Scores = new int[] { 75, 84, 91, 39 }});
     arrList.Add(new Student{strName = "小梁",Scores = new int[] { 88, 94, 65, 91 }});
     arrList.Add(new Student{strName = "小王",Scores = new int[] { 97, 89, 85, 82 }});
     //查询分数大于 95 的考生信息
     var query = from Student student in arrListwhere student.Scores[0] > 95 select student;
     foreach (Student s in query)
         listBox1.Items.Add(s.strName + ": " + s.Scores[0]);//将查询结果添加到 ListBox 控件中
 }
```

3．LINQ 与泛型列表 LIST<T>

泛型列表 List<T>能够通过索引访问对象的强类型列表，T 表示数据类型，功能上和动态数组 ArrayList 效果相同。

例如，首先声明一个泛型列表，用于存储考生考试成绩，然后通过 LINQ 查询动态数组中分数大于 95 的考生，读者可以将其与 LINQ 查询 ArrayList 动态数组的例子做比较，可以看出其实现的效果是相同的，代码如下：

```
//创建一个公共类
public class Student
 {
     //定义两个公共变量
     public string strName { get; set; }
     public int[] Scores { get; set; }
 }
private void Form1_Load(object sender, EventArgs e)
{
     //实例化一个泛型列表
    List<Student> lt = new List<Student>();
     //向泛型列表中添加数据
    lt.Add(new Student{strName = "风笛",Scores = new int[] { 98, 92, 81, 60 }});
    lt.Add(new Student{strName = "芸烨湘枫",Scores = new int[] { 75, 84, 91, 39 }});
    lt.Add(new Student{strName = "小梁",Scores = new int[] { 88, 94, 65, 91 }});
    lt.Add(new Student{strName = "小王",Scores = new int[] { 97, 89, 85, 82 }});
     //查询分数大于 95 的考生信息
    var query = from Student student in lt where student.Scores[0] > 95 select student;
    foreach (Student s in query)
        listBox1.Items.Add(s.strName + ": " + s.Scores[0]);//将查询结果天骄到 ListBox 控件中
}
```

下面通过示例演示如何通过 LINQ to Objects 技术获取指定文件的详细信息。

示例 24-01　使用 LINQ 技术获取文件详细信息

本示例通过 LINQ to Objects 技术对数组及集合进行操作，实现获取文件详细信息的功能，运行本示例，单击【…】按钮选择文件夹，将选中文件夹中所包含的所有文件显示在窗体列表中，这时选择窗体列表中的文件，即可将选中文件的详细信息显示在窗体下方“详细信息”区域的相应文本框中。运行结果如图 24.1 所示。

图 24.1　使用 LINQ 技术获取文件详细信息

程序代码如下。

代码 24-01　光盘位置：光盘\mingrisoft\sl\24\24_01

单击【…】按钮选择文件夹，通过 LINQ to Objects 技术遍历指定文件夹中的文件，并添加到 ListView 控件中，代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    //如果选择了文件夹
    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        //清空 ListView 控件
        listView1.Items.Clear();
        //显示选择的文件夹路径
        textBox1.Text = folderBrowserDialog1.SelectedPath;
        //实例化一个泛型列表
        List<FileInfo> myFiles = new List<FileInfo>();
        //遍历选择文件夹下的所有文件
        foreach (string strFile in Directory.GetFiles(textBox1.Text))
        {
            //将文件路径添加到泛型列表中
            myFiles.Add(new FileInfo(strFile));
        }
        //通过 LINQ 获取泛型列表中文件的相关信息
        var values = from strFile in myFiles
                     group strFile by strFile.Extension into FExten
                     orderby FExten.Key
                     select FExten;
        //将文件的相关信息添加到 ListView 控件中
```

```
            foreach (var vFiles in values)
            {
                foreach (var f in vFiles)
                    listView1.Items.Add(f.FullName);
            }
        }
    }
```

当选择某项后，根据 LINQ to Objects 技术获取指定文件的详细信息，并将信息显示到相应的控件中，代码如下：

```
    private void listView1_SelectedIndexChanged(object sender, EventArgs e)
    {
        //如果选择了某个文件
        if (listView1.SelectedItems.Count != 0)
        {
            //实例化 FileInfo 对象
            FileInfo myFile = new FileInfo(listView1.SelectedItems[0].Text);
            //声明一个字符串数组，存储文件的相关信息
            string[] strAttribute = new string[] { myFile.Name, Convert.ToDouble(myFile.Length / 1024).ToString(),
myFile.Extension, myFile.CreationTime.ToString(), myFile.IsReadOnly.ToString(), myFile.LastWriteTime.ToString() };
            //通过 LINQ 获取文件的相关信息
            var values = from str in strAttribute
                         select new
                         {
                             Name = strAttribute[0].ToString(),
                             Size = strAttribute[1].ToString(),
                             Exten = strAttribute[2].ToString(),
                             CTime = strAttribute[3].ToString(),
                             ReadOnly = strAttribute[4].ToString(),
                             WTime = strAttribute[5].ToString()
                         };
            //将信息显示到相应的控件中
            foreach (var v in values)
            {
                textBox2.Text = v.Name.ToString();//文件名
                textBox4.Text = v.Size.ToString();//文件大小
                textBox3.Text = v.Exten.ToString();//文件扩展名
                textBox5.Text = v.CTime.ToString();//创建时间
                textBox6.Text = v.WTime.ToString();//最后修改时间
                textBox7.Text = v.ReadOnly.ToString();//是否只读
            }
        }
    }
```

24.2.2 LINQ to DataSet

LINQ to DataSet 是基于 ADO.NET 的一种 LINQ 功能模块。使用 LINQ to DataSet 能够更快更容易地查询 DataSet 对象中的数据。LINQ to DataSet 功能通过 DataRowExtensions 和 DataTable Extensions 静态类中的扩展方法来实现。如果要使用 LINQ to DataSet 查询 DataSet 对象中的数据，那么首先要通过 ADO.NET 中的组件填充 DataSet 对象，之后才能够进行查询。LINQ to DataSet 的查询操作绝大多数是基于 DataRow 或者 DataTable 对象的，DataRowExtensions 和 DataTable Extensions 静态类分别为 DataRow 和 DataTable 对象提供了相应的扩展方法。

1．DataRowExtensions 类

DataRowExtensions 类定义了 DataRow 类的扩展方法，主要用来操作 DataRow 类的实例中的数据。例如 Field<T>方法和 SetField<T>方法，下面主要介绍这两个方法。

（1）Field<T>方法。此方法可以获取 DataRow 对象中指定列的值，其语法格式如下：

```
public static T Field<T>( string columnName)
```

参数说明如下。

T：指定列的返回类型的泛型参数。

columnName：要返回其值的列的名称。

示例 24-02　Field<T>方法获取指定列的值

本示例实现将数据表中性别为“男”的所有信息检索出来，主要通过 Field<T>方法设置检索数据表时的条件，通过此方法可以指定检索的条件是性别为“男”，运行结果如图 24.2 所示。

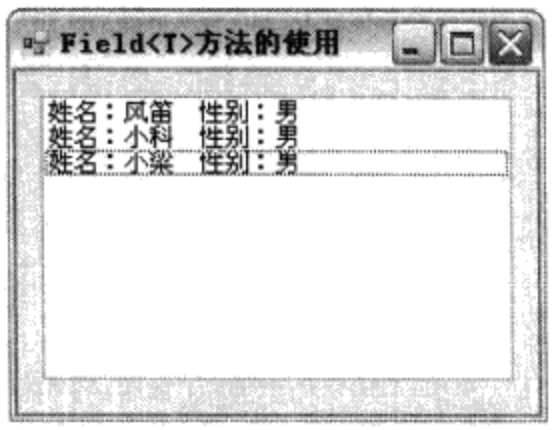


图 24.2　检索性别为“男”的所有数据

程序代码如下。

代码 24-02　光盘位置：光盘\mingrisoft\sl\24\24_02

```
private void Form1_Load(object sender, EventArgs e)
{
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_24;uid=sa;pwd=");
    //创建 SqlDataAdapter 对象
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_User", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充 DataSet 对象
    sda.Fill(ds,"user");
    //通过 LINQ 查询性别为“男”的信息
    var users = from u in ds.Tables["user"].AsEnumerable()
                where u.Field<string>("User_Sex").Trim() == "男"
                select new { username = (string)u["User_Name"], usersex = (string)u["User_Sex"] };
     //将查询结果添加到 ListBox 控件中
    foreach (var item in users)
    {
        listBox1.Items.Add("姓名："+item.username+"　性别："+item.usersex);
    }
}
```

（2）SetField<T>方法。该方法用于设置 DataRow 对象中指定列的值，其语法格式如下：

```
public static void SetField<T>(string columnName,T value)
```

参数说明如下。

T：指定列的值类型的泛型参数。

columnName：要设置其值的列的名称。

value：指定列的新行值，类型为 T。

示例 24-03　SetField<T>方法设置指定列的值

本示例通过 SetField<T>方法设置“User_Marriage”列的值为“已婚”，运行结果如图 24.3 所示。

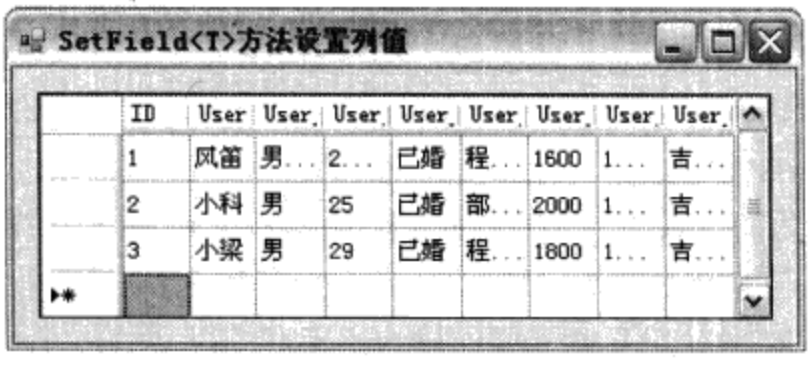


图 24.3　修改指定列的值

程序代码如下。

代码 24-03　光盘位置：光盘\mingrisoft\sl\24\24_03

```
private void Form1_Load(object sender, EventArgs e)
{
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_24;uid=sa;pwd=");
    //创建 SqlDataAdapter 对象
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_User", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充 DataSet 对象
    sda.Fill(ds, "user");
    //通过 LINQ 检索性别为“男”的信息
    IEnumerable<DataRow> q = from u in ds.Tables["user"].AsEnumerable()
                             where u.Field<string>("User_Sex").Trim() == "男"
                             select u;
    foreach (DataRow r in q)
    {
        //通过 SetField 方法修改列值
        r.SetField<string>("User_Marriage", "已婚");
    }
    DataTable dt = q.CopyToDataTable<DataRow>();
    dataGridView1.DataSource = dt;//设置控件数据源
}
```

2．DataTableExtensions 类

DataTableExtensions 类定义了 DataTable 类的扩展方法，主要用来操作 DataTable 类的实例中的数据，下面主要介绍 DataTableExtensions 类中的 AsEnumerable 方法和 CopyToDataTable 方法。

（1）AsEnumerable 方法。该方法用于将 DataTable 对象转换为 EnumerableRowCollection < DataRow>对象，其语法格式如下：

```
public static EnumerableRowCollection<DataRow> AsEnumerable()
```

返回值：EnumerableRowCollection<DataRow>对象。

示例 24-04　AsEnumerable 方法转换 DataTable 对象

本示例通过 AsEnumerable 方法将 DataTable 对象转换为 EnumerableRowCollection<DataRow>对象，然后输出相应的字段，运行结果如图 24.4 所示。

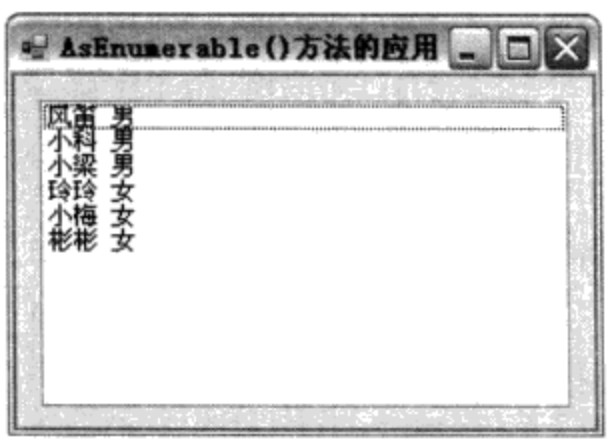


图 24.4　输出数据

程序代码如下。

代码 24-04　光盘位置：光盘\mingrisoft\sl\24\24_04

```
private void Form1_Load(object sender, EventArgs e)
{
    //连接数据库
```

```
    SqlConnection conn = new SqlConnection("server=.;database=db_24;uid=sa;pwd=");
    //创建 SqlDataAdapter 对象
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_User", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充 DataSet 对象
    sda.Fill(ds, "user");
    //创建 DataTable 对象
    DataTable dt = ds.Tables["user"];
    // AsEnumerable 方法转换 DataTable 对象
    EnumerableRowCollection<DataRow> r = dt.AsEnumerable();
    foreach (var item in r)
    {
        //将信息添加到 ListBox 控件中
        listBox1.Items.Add(item.Field<string>("User_Name") + " " + item.Field<string>("User_Sex"));
    }
}
```

（2）CopyToDataTable 方法。此方法用于将 IEnumerable<T>对象中的数据复制到 DataTable 对象中，其语法格式如下：

```
public static DataTable CopyToDataTable<T>()
```

参数说明如下。

T：源数据类型。

返回值：DataTable 对象。

示例 24-05　CopyToDataTable 方法复制数据

本示例通过 CopyToDataTable 方法将 IEnumerable<T>对象中的数据复制到 DataTable 对象中，然后将 DataTable 对象设为 DataGridView 控件的数据源，以便显示筛选的数据，运行结果如图 24.5 所示。

图 24.5　复制数据

程序代码如下。

代码 24-05　光盘位置：光盘\mingrisoft\sl\24\24_05

```
private void Form1_Load(object sender, EventArgs e)
{
    //连接数据库
    SqlConnection conn = new SqlConnection("server=.;database=db_24;uid=sa;pwd=");
    //创建 SqlDataAdapter 对象
    SqlDataAdapter sda = new SqlDataAdapter("select * from tb_User", conn);
    //创建 DataSet 对象
    DataSet ds = new DataSet();
    //填充 DataSet 对象
    sda.Fill(ds, "user");
    //检索年龄小于 30 的信息
    IEnumerable<DataRow> r = from u in ds.Tables["user"].AsEnumerable()
                             where Convert.ToInt32(u.Field<string>("User_Age")) < 30
                             select u;
    // CopyToDataTable 方法复制数据
    DataTable dt = r.CopyToDataTable<DataRow>();
    //设置控件数据源
    dataGridView1.DataSource = dt;
}
```


24.2.3　LINQ to SQL

LINQ 是 Visual Studio 2008 和.Net Framework 3.5 版中一项突破性的创新，它在对象领域和数据领域之间架起了一座桥梁，LINQ to SQL 是 LINQ 中最重要的一个组件，它可以为关

系数据库提供一个对象模型，并在该对象模型基础上实现对数据的查询、添加、修改和删除等功能。Visual Studio 2008 为 SQL Server 数据库 LinqDB 提供创建 DBML 文件的功能，通过该文件可以更好地操作数据库，下面将详细介绍如何创建 DBML 文件以及对数据库的常用操作。

1．创建 DBML 文件

（1）在 Visual Studio 2008 开发环境中，右键单击“解决方案资源管理器”面板中的项目名称，在右键菜单中选择“添加”/“新建项”命令，如图 24.6 所示。

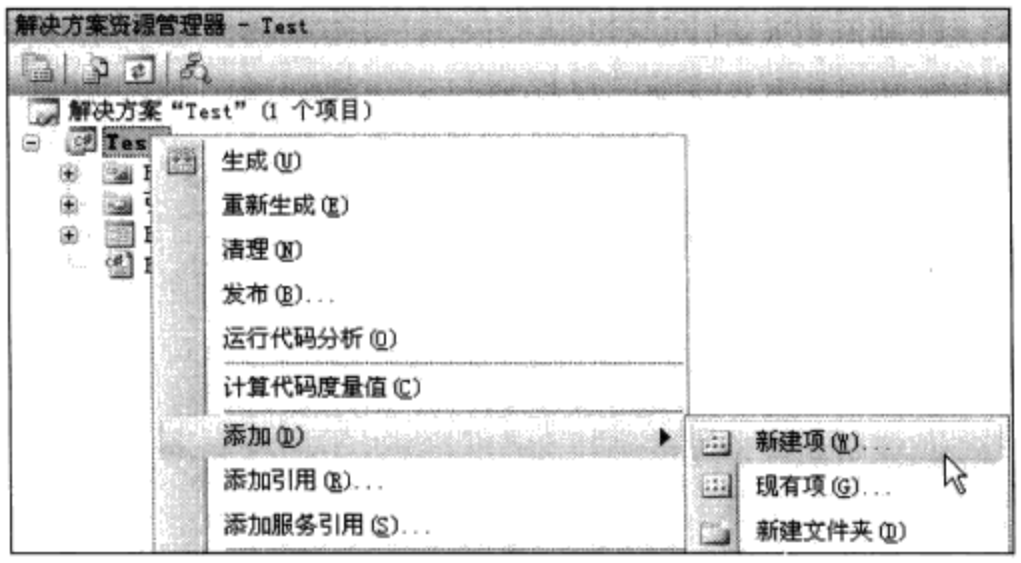


图 24.6　添加“新建项”界面

（2）选择“新建项”命令，弹出“添加新项”对话框，选择“LINQ to SQL 类”选项，并在“名称”输入框中输入 LinqToSql.dbml，如图 24.7 所示。

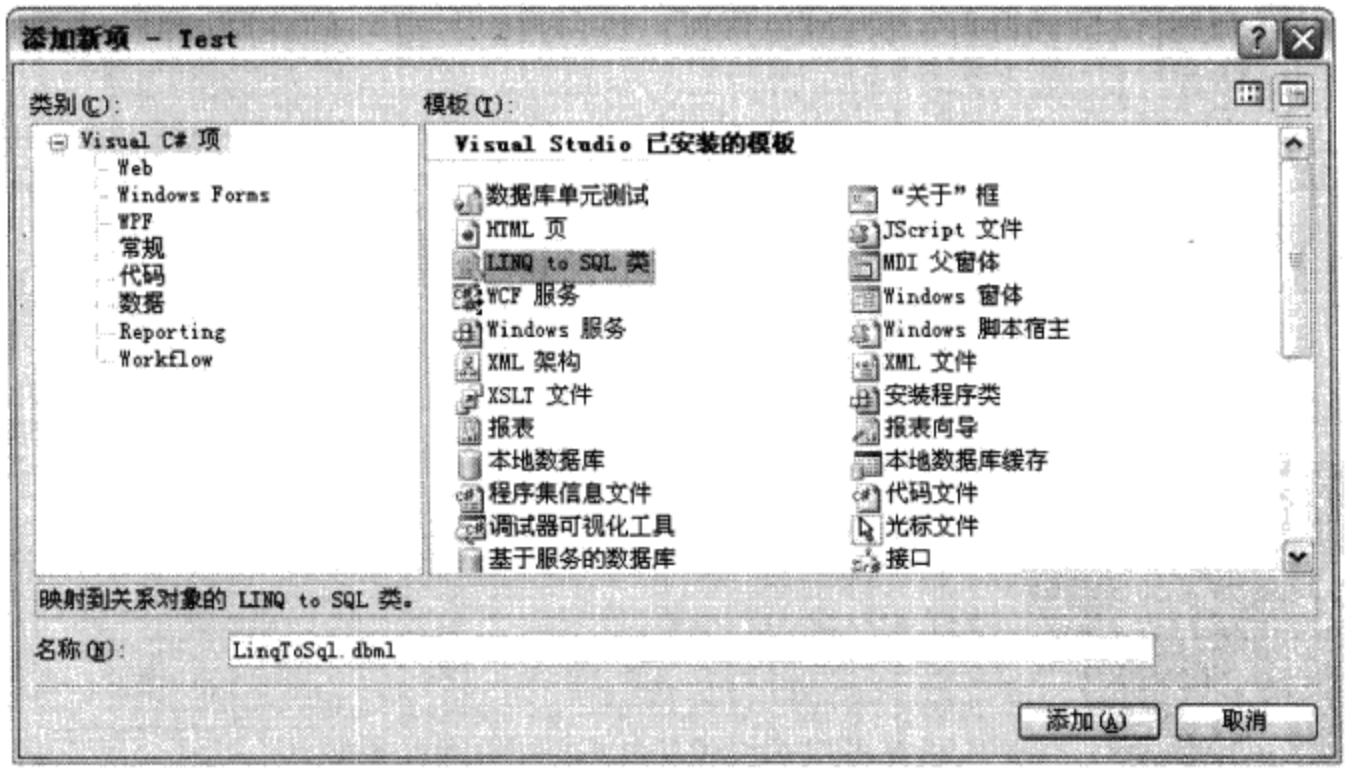


图 24.7　“添加新项”对话框

（3）单击“添加”按钮，将 DBML 文件添加到项目中。然后在“服务器资源管理器”面板中选择数据库中的各个表，并将其拖放到 LinqToSql.dbml 文件的视图面板中，如图 24.8 所示。

（4）单击按钮保存，即可创建 DBML 文件。

2．查询数据库

使用 LINQ to SQL 技术查询数据库中的数据，首先要创建一个 DBML 文件，然后为该数据

库创建数据上下文类 LinqToSqlDataContext，该类继承于 DataContext 类，定义该类的代码如下：

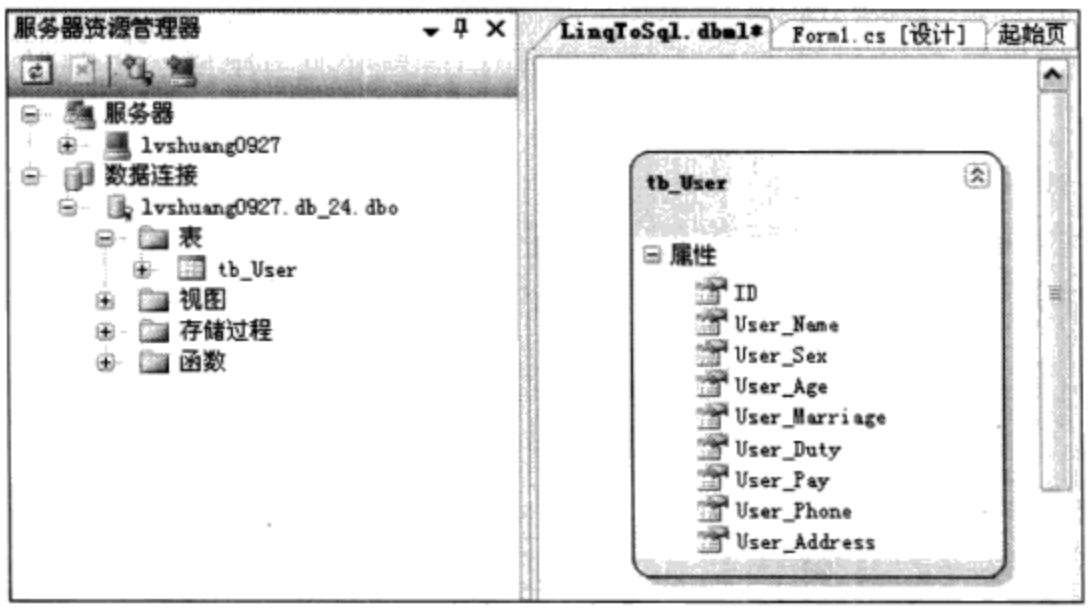


图 24.8　将数据表添加到 LinqToSql.dbml 文件的视图面板中

```
string strCon = "Data Source=(local);Database=db_24;Uid=sa;Pwd=;";
LinqToSqlDataContext linq;                                  //声明 Linq 连接对象
linq = new LinqToSqlDataContext(strCon);                    //实例化 Linq 连接对象
```

下面通过示例演示如何通过 LINQ to SQL 技术查询数据库。

示例 24-06　使用 LINQ 技术查询 SQL 数据库中的数据

本示例通过 LINQ to SQL 技术查询数据，可以根据输入的关键字，在数据表中的姓名、性别、年龄和职位字段中检索数据，并将检索出来的数据绑定到控件中显示出来。运行结果如图 24.9 所示。

图 24.9　使用 LINQ 技术查询 SQL 数据库中的数据

程序代码如下。

代码 24-06　光盘位置：光盘\mingrisoft\sl\24\24_06

```
string strCon = "Data Source=(local);Database=db_11;Uid=sa;Pwd=;";      //定义数据库连接字符串
LinqtosqlDataContext linq;                                              //声明 Linq 连接对象
private void SearchInfo()
{
    linq = new LlinqtosqlDataContext(strCon);                           //实例化 Linq 连接对象
    if (txtKey.Text == "")                                              //如果没有输入查询的关键字
```

```
{
    var result = from info in linq.tb_User                          //查找数据库中所有的员工信息
                 select new
                 {
                     编号 = info.ID,                                //显示编号
                     姓名 = info.User_Name.Trim(),                  //姓名
                     性别 = info.User_Sex.Trim(),                   //性别
                     年龄 = info.User_Age.Trim(),                   //年龄
                     婚姻状况 = info.User_Marriage.Trim(),          //婚姻状况
                     职位 = info.User_Duty.Trim(),                  //职位
                     联系电话 = info.User_Phone.Trim(),             //联系电话
                     联系地址 = info.User_Address.Trim()            //联系地址
                 };
    dataGridView1.DataSource = result;                              //将检索的数据绑定到dataGridView1控件
}
else                                                                //如果输入了关键字
{
    int i = comboBox1.SelectedIndex;                                //获取查询的范围
    switch (i)
    {
        case 0:                                                     //如果根据姓名查找
            var resultName = from info in linq.tb_User
                             where info.User_Name.IndexOf(txtKey.Text)>=0//模糊查询
                             select new
                             {
                                 编号 = info.ID,
                                 姓名 = info.User_Name,
                                 性别 = info.User_Sex,
                                 年龄 = info.User_Age,
                                 婚姻状况 = info.User_Marriage,
                                 职位 = info.User_Duty,
                                 联系电话 = info.User_Phone,
                                 联系地址 = info.User_Address
                             };
            dataGridView1.DataSource = resultName;          //将检索的数据绑定到 dataGridView1 控件
            break;
        case 1:                                                     //如果根据性别查找
            var resultSex = from info in linq.tb_User
                            where info.User_Sex==txtKey.Text.Trim()//判断员工性别是否等于输入的性别
                            select new
                            {
                                编号 = info.ID,
                                姓名 = info.User_Name,
                                性别 = info.User_Sex,
                                年龄 = info.User_Age,
                                婚姻状况 = info.User_Marriage,
                                职位 = info.User_Duty,
                                联系电话 = info.User_Phone,
                                联系地址 = info.User_Address
                            };
            dataGridView1.DataSource = resultSex;           //将检索的数据绑定到 dataGridView1 控件
            break;
        case 2:                                             //如果根据年龄查找
            //判断数据库中的员工年龄是否以输入的关键字开头
            var resultAge = from info in linq.tb_User
                            where info.User_Age.StartsWith(txtKey.Text)
                            select new
                            {
                                编号 = info.ID,
                                姓名 = info.User_Name,
                                性别 = info.User_Sex,
                                年龄 = info.User_Age,
```

```
                                婚姻状况 = info.User_Marriage,
                                职位 = info.User_Duty,
                                联系电话 = info.User_Phone,
                                联系地址 = info.User_Address
                            };
                dataGridView1.DataSource = resultAge;          //将检索的数据绑定到 dataGridView1 控件
                break;
            case 3:                                            //如果根据职位查找
                var resultDuty = from info in linq.tb_User
                                 where info.User_Duty == txtKey.Text.Trim()//判断员工职位是否等于输入的关键字
                                 select new
                                 {
                                     编号 = info.ID,
                                     姓名 = info.User_Name,
                                     性别 = info.User_Sex,
                                     年龄 = info.User_Age,
                                     婚姻状况 = info.User_Marriage,
                                     职位 = info.User_Duty,
                                     联系电话 = info.User_Phone,
                                     联系地址 = info.User_Address
                                 };
                dataGridView1.DataSource = resultDuty;         //将检索的数据绑定到 dataGridView1 控件
                break;
        }
    }
}
```

如果想查询数据库中前几名的数据，可以通过使用 Take 方法。该方法从序列的开头返回指定数量的连续元素，其语法格式如下：

```
public static IEnumerable<TSource> Take<TSource>(this IEnumerable<TSource> source,int count)
```

参数说明如下。

source：表示要从其返回元素的序列。

count：表示要返回的元素数量。

返回值：一个 IEnumerable<T>，包含输入序列开头的指定数量的元素。

示例 24-07　使用 LINQ 技术查询前 5 名数据

本示例主要通过 LINQ to SQL 技术获取编号前五名的数据，实例首先将数据库中的数据检索出来显示到控件中，然后单击“获取编号前 5 名的数据”按钮。运行结果如图 24.10 所示。

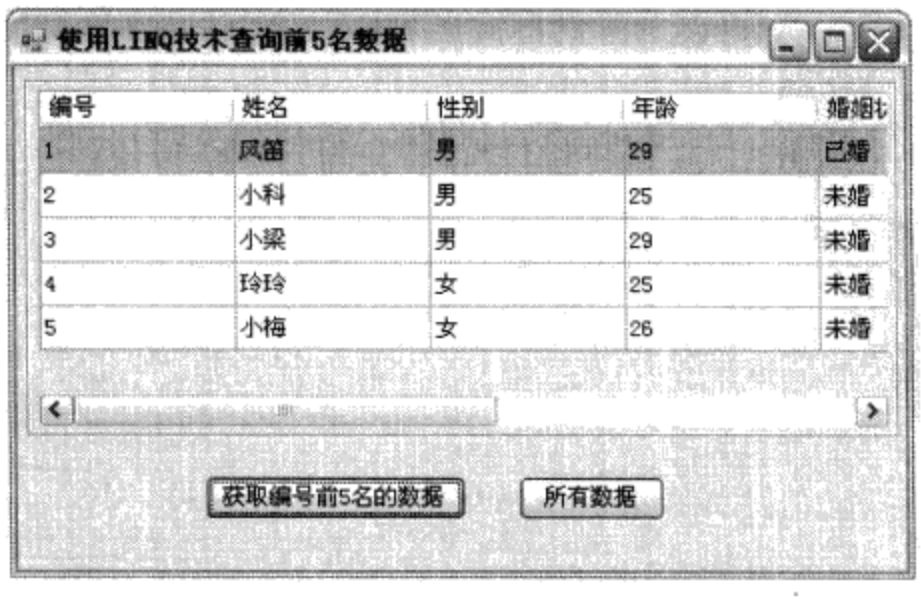


图 24.10　使用 LINQ 技术查询前 5 名数据

程序代码如下。

代码 24-07　光盘位置：光盘\mingrisoft\sl\24\24_07

```
private void button1_Click(object sender, EventArgs e)
{
    linq = new LinqToSqlDataContext (strCon);                          //实例化 linq
    var result = (from info in linq.tb_User orderby info.ID ascending select new //按照升序排列数据表中的数据
    {
        编号 = info.ID,                                                //显示编号
        姓名 = info.User_Name.Trim(),                                  //显示姓名
        性别 = info.User_Sex.Trim(),                                   //显示性别
        年龄 = info.User_Age.Trim(),                                   //显示年龄
        婚姻状况 = info.User_Marriage.Trim(),                          //显示婚姻状况
        职位 = info.User_Duty.Trim(),                                  //显示职位
        联系电话 = info.User_Phone.Trim(),                             //显示联系电话
        联系地址 = info.User_Address.Trim()                            //显示联系地址
    }).Take(5);                                                        //然后取前 5 名数据
    dataGridView1.DataSource = result;                                 //绑定数据源
}
```

3．添加数据

LINQ to SQL 向数据库中插入数据的功能由 Table<T>类的 InsertOnSubmit()方法实现，InsertOnSubmit()方法将单个实体添加到 Table<T>类的实例中，然后通过 SubmitChanges 方法提交对数据库的更改，下面详细介绍这两种方法。

（1）InsertOnSubmit 方法。InsertOnSubmit 方法用来将处于 pending insert 状态的实体添加到表中。其语法格式如下：

```
void InsertOnSubmit(Object entity)
```

参数说明如下。

entity：表示要添加的实体。

（2）SubmitChanges 方法。SubmitChanges 方法用来计算要插入、更新或删除的已修改对象的集，并执行相应命令以实现对数据库的更改。其语法格式如下：

```
public void SubmitChanges()
```

示例 24-08　使用 LINQ 技术向 SQL 数据库中添加数据

本示例使用 LINQ to SQL 技术将输入的信息添加到数据库中，首先断信息是否输入完整，然后还要判断输入的电话号码和年龄是否为数字，如果输入的信息无误，单击“添加数据”按钮，将输入的数据添加到数据库中。运行结果如图 24.11 所示。

图 24.11　使用 LINQ 技术向 SQL 数据库中添加数据

程序代码如下。

代码 24-08　光盘位置：光盘\mingrisoft\sl\24\24_08

```
private void button1_Click(object sender, EventArgs e)                    //添加数据按钮
{
    //判断是否输入姓名、年龄、电话和家庭住址
    if (txtaddress.Text != "" && txtage.Text != "" && txtName.Text != "" && txtphone.Text != "")
    {
        if (txtphone.Text.Length != 11)                                   //判断输入的电话号码是否合法
        {
            MessageBox.Show("电话号码位数不正确");                         //不合法则弹出提示
        }
        else                                                              //如果输入的电话号码合法
        {
            linq = new LinqToSqlDataContext (strCon);                     //实例化 linq 连接对象
            tb_User users = new tb_User();                                //实例化 tb_User 类
            users.User_Name = txtName.Text.Trim();                        //设置姓名
            users.User_Sex = cbbSex.Text;                                 //性别
            users.User_Age = txtage.Text;                                 //年龄
            users.User_Marriage = cbbmary.Text;                           //婚姻状况
            users.User_Duty = cbbduty.Text;                               //职位
            users.User_Phone = txtphone.Text;                             //电话
            users.User_Address = txtaddress.Text;                         //家庭住址
            linq.tb_User.InsertOnSubmit(users);                           //提交数据
            linq.SubmitChanges();                                         //执行对数据库的修改
            binginfo();                                                   //重新绑定数据
            MessageBox.Show("添加成功");                                  //弹出提示
        }
    }
}
```

4．修改数据

LINQ to SQL 修改数据库中的数据的功能主要是通过 SubmitChanges 方法实现的，该方法在“添加数据”一节中已经做过介绍，所以此处不再赘述，下面通过示例演示如何通过 LINQ to SQL 修改数据库中的数据。

示例 24-09　使用 LINQ 技术在 SQL 数据库中修改数据

本示例主要通过 LINQ to SQL 技术修改指定的数据，示例首先查询数据库中所有的数据并显示出来，然后选择某个数据后会显示其详细信息，修改某个信息后，单击“修改数据”按钮修改数据。运行结果如图 24.12 所示。

图 24.12　使用 LINQ 技术在 SQL 数据库中修改数据

程序代码如下。

代码 24-09　光盘位置：光盘\mingrisoft\sl\24\24_09

```
private void button1_Click(object sender, EventArgs e)
{
    //判断是否输入姓名、年龄、电话和家庭住址
    if (txtaddress.Text != "" && txtage.Text != "" && txtName.Text != "" && txtphone.Text != "")
    {
        if (txtphone.Text.Length != 11)                                  //如果电话号码错误
        {
            MessageBox.Show("电话号码位数不正确");                        //弹出提示
        }
        else
        {
            linq = new LinqtosqlDataContext(strCon);
            var resultChange = from info in linq.tb_User
                               where info.ID == Pid                      //实例化 linq，设置其根据 ID 值进行修改
                               select info;
            foreach (tb_User users in resultChange)
            {
                users.User_Name = txtName.Text;                          //姓名
                users.User_Sex = cbbSex.Text;                            //性别
                users.User_Age = txtage.Text;                            //年龄
                users.User_Marriage = cbbmary.Text;                      //婚姻状况
                users.User_Duty = cbbduty.Text;                          //职位
                users.User_Phone = txtphone.Text;                        //电话
                users.User_Address = txtaddress.Text;                    //家庭地址
                linq.SubmitChanges();                                    // SubmitChanges 方法提交修改
            }
            MessageBox.Show("修改成功");                                 //弹出成功的提示
            binginfo();                                                  //重新绑定数据
        }
    }
}
```

5．删除数据

LINQ to SQL 从数据库中删除数据的功能主要是通过 SubmitChanges 方法和 DeleteAllOnSubmit 方法实现的，SubmitChanges 方法在“添加数据”一节中已经做过介绍，所以此处不再赘述，下面主要介绍 DeleteAllOnSubmit 方法。

DeleteAllOnSubmit 方法从 Table<T>类的实例中删除实体的集合。其语法格式如下：

```
void DeleteAllOnSubmit(IEnumerable entities)
```

参数说明如下。

entities：表示要移除所有项的集合。

示例 24-10　使用 LINQ 技术在 SQL 数据库中删除数据

本示例主要通过 LINQ to SQL 技术删除指定的数据，示例首先将数据库中的数据检索出来并显示在控件中，然后选择某项后，单击“删除”按钮删除数据。运行结果如图 24.13 所示。

图 24.13　使用 LINQ 技术在 SQL 数据库中删除数据

程序代码如下。

代码 24-10　光盘位置：光盘\mingrisoft\sl\24\24_10

```
private void button1_Click(object sender, EventArgs e)
 {
      if (dataGridView1.SelectedRows.Count != 0)              //判断是否选择了项目
      {
           linq = new linqtosqlDataContext(strCon);
           var result = from info in linq.tb_User
                        where info.ID == id
                        select info;                          //实例化 linq，并设置删除的条件
           linq.tb_User.DeleteAllOnSubmit(result);            //通过 DeleteAllOnSubmit 方法删除指定的数据
           linq.SubmitChanges();                              //提交对数据表的修改
           MessageBox.Show("删除成功");                        //弹出提示
           bindinfo();                                        //重新绑定数据
      }
      else                                                    //如果没有选择项
      {
           MessageBox.Show("请选择删除项");                    //显示提示信息
      }
 }
```

24.2.4　LINQ to XML

LINQ to XML 是 LINQ 技术中的一种，它提供了修改文档对象模型的内存文档，并支持 LINQ 查询表达式等功能。对 XML 文件进行操作时，主要用到 LINQ to XML 技术中的 XElement 类的 Load 方法、SetAttributeValue 方法、Add 方法、ReplaceNodes 方法及 Save 方法，XDocument 类的 Save 方法，XDeclaration 类、IEnumerable<T>泛型接口的 First 方法及 Remove 方法，下面详细介绍这几个方法。

（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。其语法格式如下：

```
public static XElement Load(string uri)
```

参数说明如下。

uri：一个 URI 字符串，用来引用要加载到新 XElement 中的文件。

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。其语法格式如下：

```
public void SetAttributeValue(XName name,Object value)
```

参数说明如下。

name：一个 XName，其中包含要更改的属性的名称。

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。其语法格式如下：

```
public void Add(Object content)
```

参数说明如下。

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。其语法格式如下：

```
public void ReplaceNodes(Object content)
```

参数说明如下。

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。其语法格式如下：

```
public void Save(string fileName)
```

参数说明如下。

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。其语法格式如下：

```
public void Save(string fileName)
```

参数说明如下。

fileName：一个包含文件名称的字符串。

（7）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。其语法格式如下：

```
public static TSource First<TSource>(this IEnumerable<TSource> source)
```

参数说明如下。

source：要返回其第一个元素的 IEnumerable<T>。

返回值：返回指定序列中的第一个元素。

（8）IEnumerable 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。其语法格式如下：

```
public static void Remove<T>(this IEnumerable<T> source) where T : XNode
```

参数说明如下。

source：一个包含源集合的 XNode 的 IEnumerable<T>。

示例 24-11　使用 LINQ 技术对 XML 文件进行操作

本示例使用 LINQ to XML 技术来实现创建、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，单击“修改”和“删除”按钮，即可修改和删除 XML 文件中的元素。运行结果如图 24.14 所示。

图 24.14　使用 LINQ 技术对 XML 文件进行操作

程序代码如下。

代码 24-11　光盘位置：光盘\mingrisoft\sl\24\24_11

Form1 窗体的后台代码中，首先定义两个字符串类型的全局变量，分别用来记录 XML 文件路径及选中的 ID 编号。代码如下：

```
static string strPath = "Employee.xml";                    //记录 XML 文件路径
static string strID = "";                                  //记录选中的 ID 编号
```

Form1 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。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。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();                          //实例化 DataSet 数据集对象
    myds.ReadXml(strPath);                                 //读取 XML 结构
    dataGridView1.DataSource = myds.Tables[0];             //在 DataGridView 中显示 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。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    XDocument doc = new XDocument(                         //实例化 XML 文档对象
        new XDeclaration("1.0", "utf-8", "yes"),           //添加 XML 文件声明
        new XElement(textBox1.Text,                        //实例化 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 LinqToXML 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。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")  //实例化 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(                                    //实例化 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 LinqToXML 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 LinqToXML 技术修改 XML 文件中的指定记录，并重新保存 XML 文件。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 LinqToXML 技术删除 XML 文件中的指定记录，并重新保存 XML 文件。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();
    }
```

第 25 章

高性能数据处理技术

- 存储过程概述
- 存储过程的建立、修改和删除
- 在 C#中调用存储过程
- 视图的概述
- 视图的建立、修改和删除
- 在 C#中调用视图
- 触发器的应用
- 游标的应用

25.1 存储过程概述

存储过程是 SQL Server 服务器上一组预先编译好的 Transact-SQL 语句，并作为一个独立的数据库对象或单元，供用户在应用程序中调用。存储过程存储在数据库内，并且支持用户声明的变量以及其他强大的编程功能。存储过程可以接收和输出参数，并且返回执行存储过程的状态值，而且还可以嵌套调用。

1．临时存储过程

SQL Server 支持两种临时存储过程：局部临时存储过程和全局临时存储过程。局部临时存储过程只能由创建该存储过程的连接使用，而全局临时存储过程则可由所有连接使用。局部临时存储过程在当前会话结束时自动除去，全局临时存储过程在使用该存储过程的最后一个会话结束时除去，通常是在创建该存储过程的会话结束时。

临时存储过程用#和##命名，可以由任何用户创建。创建存储过程后，局部存储过程的所有者是惟一可以使用该过程的用户。执行局部临时存储过程的权限不能授予其他用户。如果创建了全局临时存储过程，则所有用户均可以访问该过程，权限不能显式废除。只有在 tempdb 数据库中具有显式 CREATE PROCEDURE 权限的用户，才可以在该数据库中显式创建临时存储过程（不使用编号符命名）。用户可以授予或废除这些存储过程中的权限。

2．自动执行存储过程

SQL Server 启动时可以自动执行一个或多个存储过程，这些存储过程必须由系统管理员创建，并在 sysadmin 固定服务器角色下作为后台过程执行，另外，这些存储过程不能有任何输入参数。

对启动过程的数目没有限制，但是每个启动过程在执行时都会占用一个连接。如果必须在启动时执行多个存储过程，但不需要并行执行，则可以指定一个存储过程作为启动存储过程，让该存储过程调用其他存储过程。这样就只占用一个连接。

25.2 存储过程的建立、修改和删除

创建存储过程的方法很多，本节将详细介绍如何在 SQL 查询分析器中通过模板完成存储过程的建立、修改和删除操作。

25.2.1 存储过程的建立

首先在“SQL 查询分析器”的“对象浏览器”中选取“模板”选项卡，然后在模板列表中展开“Create Procedure”节点，双击“存储过程”子节点，在右侧窗口将显示出相应的模板代码信息。具体操作如图 25.1 所示。

建立存储过程的语法：

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
[ { @parameter data_type }
[ VARYING ] [ = default ] [ OUTPUT ]
] [ ,...n ]
[ WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

建立存储过程语法中各参数及说明如表 25.1 所示。

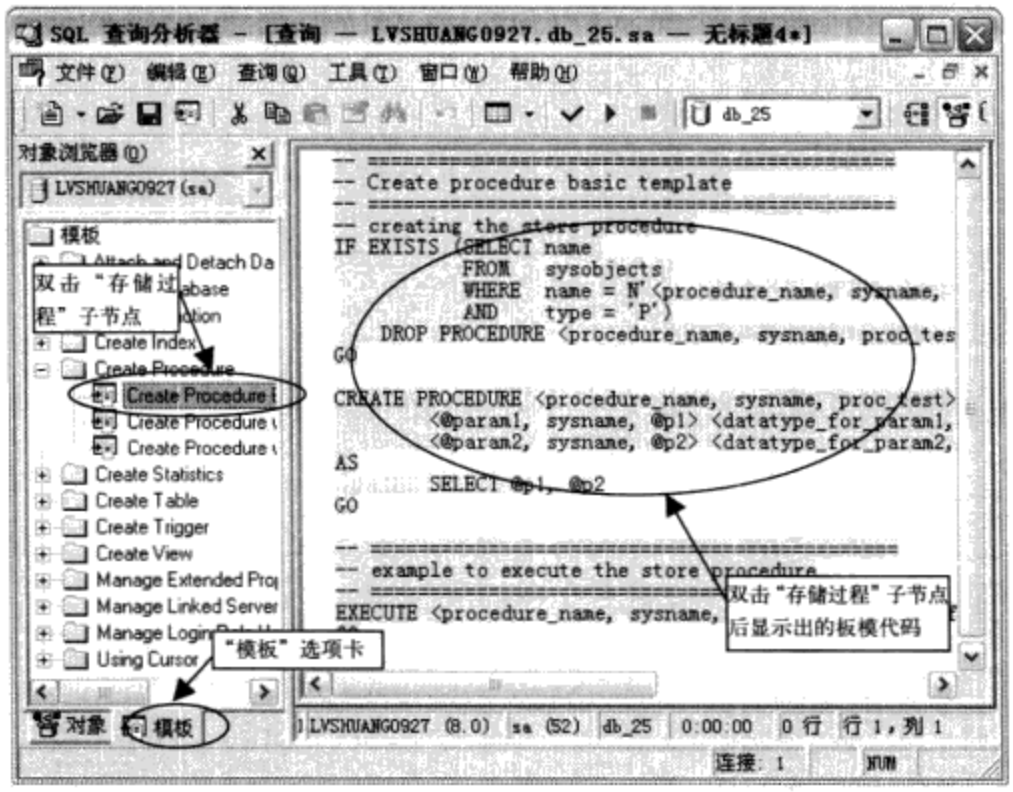


图 25.1 建立存储过程

表 25.1 建立存储过程语法中的各参数及说明

参 数	说 明
procedure_name	新存储过程的名称。过程名必须符合标识符规则，且对于数据库及其所有者必须惟一。要创建局部临时存储过程，可以在 procedure_name 前面加一个编号(#procedure_name)，要创建全局临时存储过程，可以在 procedure_name 前面加两个编号符(##procedure_name)。完整的名称（包括#或##）不能超过 128 个字符。指定过程所有者的名称是可选的
number	是可选的整数，用来对同名的过程分组，以便用一条 DROP PROCEDURE 语句即可将同组的过程一起除去。例如，名为 orders 的应用程序使用的过程可以命名为 orderproc;1、orderproc;2 等。DROP PROCEDURE orderproc 语句将除去整个组。如果名称中包含定界标识符，则数字不应包含在标识符中，只应在 procedure_name 前后使用适当的定界符
@parameter	过程中的参数。在 CREATE PROCEDURE 语句中可以声明一个或多个参数。用户必须在执行过程时提供每个所声明参数的值（除非定义了该参数的默认值）。存储过程最多可以有 2100 个参数。使用@符号作为第一个字符来指定参数名称。参数名称必须符合标识符的规则。每个过程的参数仅用于该过程本身，相同的参数名称可以用在其他过程中
data_type	参数的数据类型。所有数据类型（包括 text、ntext 和 image）均可以用作存储过程的参数。不过，cursor 数据类型只能用于 OUTPUT 参数。如果指定的数据类型为 cursor，也必须同时指定 VARYING 和 OUTPUT 关键字
VARYING	指定作为输出参数支持的结果集（由存储过程动态构造，内容可以变化）。仅适用于游标参数
default	参数的默认值。如果定义了默认值，不必指定该参数的值即可执行过程。默认值必须是常量或 NULL。如果过程将对该参数使用 LIKE 关键字，那么默认值中可以包含通配符（%、_、[]和[^]）
OUTPUT	表明参数是返回参数。该选项的值可以返回给 EXEC[UTE]。使用 OUTPUT 参数可将信息返回给调用过程。Text、ntext 和 image 参数可用作 OUTPUT 参数。使用 OUTPUT 关键字的输出参数可以是游标占位符
n	表示最多可以指定 2100 个参数的占位符
{RECOMPILE |ENCRYPTION |RECOMPILE, ENCRYPTION}	RECOMPILE 表明 SQL Server 不会缓存该过程的计划，该过程将在运行时重新编译。在使用非典型值或临时值而不希望覆盖缓存在内存中的执行计划时，请使用 RECOMPILE 选项。ENCRYPTION 表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本的条目。使用 ENCRYPTION 可防止将过程作为 SQL Server 复制的一部分发布
FOR REPLICATION	指定不能在订阅服务器上执行为复制创建的存储过程。使用 FOR REPLICATION 选项创建的存储过程可用作存储过程筛选，且只能在复制过程中执行。本选项不能和 WITH RECOMPILE 选项一起使用

续表

参　数	说　明
AS	指定过程要执行的操作
sql_statement	过程中要包含的任意数目和类型的 Transact-SQL 语句
n	是表示此过程可以包含多条 Transact-SQL 语句的占位符

示例 25-01　建立存储过程

本示例完成了存储过程的建立，用于实现向 tb_People 表中添加信息，同时生成自动编号。程序代码如下：

代码 25-01　光盘位置：光盘\mingrisoft\sl\25\25_01

```
IF EXISTS (SELECT name
        FROM sysobjects
        WHERE name = 'Proc_InsertOfPelple'
        AND type = 'P')
   DROP PROCEDURE Proc_InsertOfPelple
GO
CREATE PROCEDURE Proc_InsertOfPelple
@tb_PName varchar(20),
@tb_PSex varchar(10)
AS
begin
     declare @tb_PID varchar(10)
     select @tb_PID=Max(tb_PID) from t_People
     if(@tb_PID is null)
          set @tb_PID='P1001'          --做到见编号就可以知道是对哪个表操作(因为编号的首写字母取自表名)
     else
          set @tb_PID='P'+cast(cast(substring(@tb_PID,2,4) as int)+1 as varchar(210))
     begin
          insert into t_People values(@tb_PID,@tb_PName,@tb_PSex)
     end
end
```

25.2.2　存储过程的修改

在“SQL 查询分析器”的“对象浏览器”中选取“存储过程”，右键单击要修改的存储过程名，在弹出的快捷菜单中选择“编辑”命令，这时在右侧窗口将显示修改存储过程的 SQL 语句，可以直接在上面更改。具体操作如图 25.2 所示。

示例 25-02　修改存储过程

本示例主要完成了存储过程的修改操作。

程序代码如下：

代码 25-02　光盘位置：光盘\mingrisoft\sl\25\25_02

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
ALTER PROCEDURE Proc_SelectOfPelple
@tb_PName varchar(20)='',
@tb_PSex varchar(10)=''
AS
begin
     declare @SQL varchar(200)
     set @SQL='select tb_PID 人员编号,tb_PName 姓名,tb_PSex 性别 from t_People
     where tb_PName like "%'+@tb_PName+'%" and tb_PName like "%'+@tb_PSex+'%"'
```

```
    print @SQL
    exec (@SQL)
end
GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

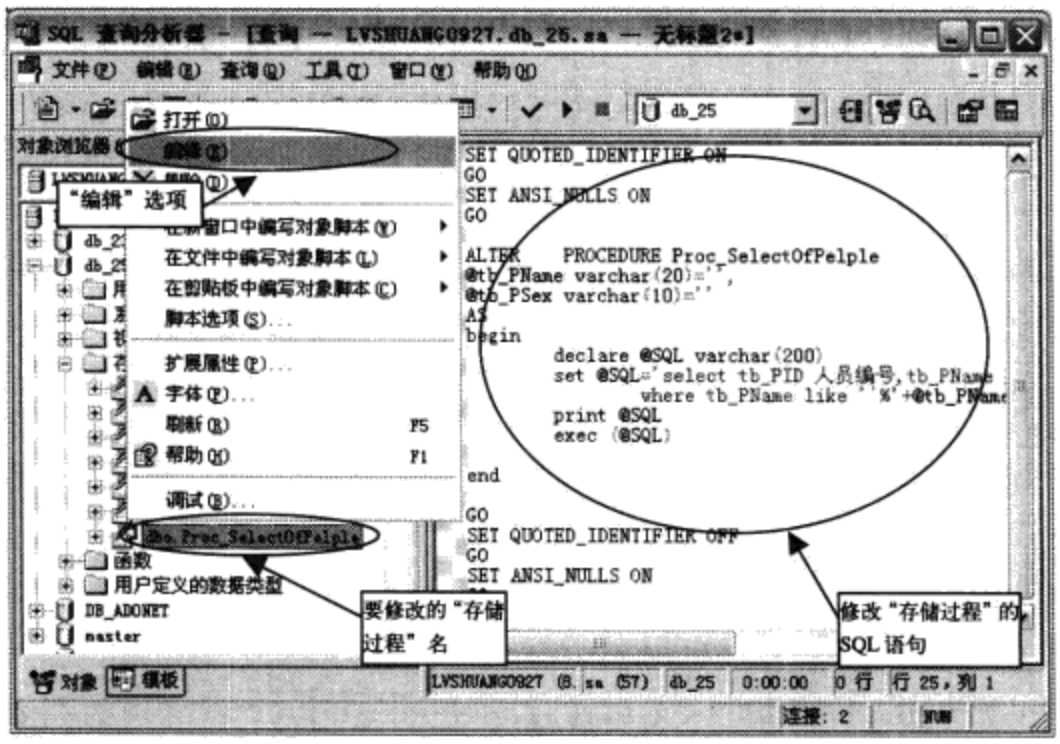


图 25.2　修改存储过程

注意：修改存储过程的关键字是 ALTER，语法为 ALTER PROCEDURE 存储过程名。

25.2.3　存储过程的删除

可以通过 SQL 查询分析器和 SQL 代码两种方法删除存储过程。

1．通过 SQL 查询分析器删除存储过程

在"SQL 查询分析器"的"对象浏览器"中选取"存储过程"，右键单击要删除的存储过程名，在弹出的快捷菜单中选择"删除"命令即可将选择的存储过程删除，如图 25.3 所示。

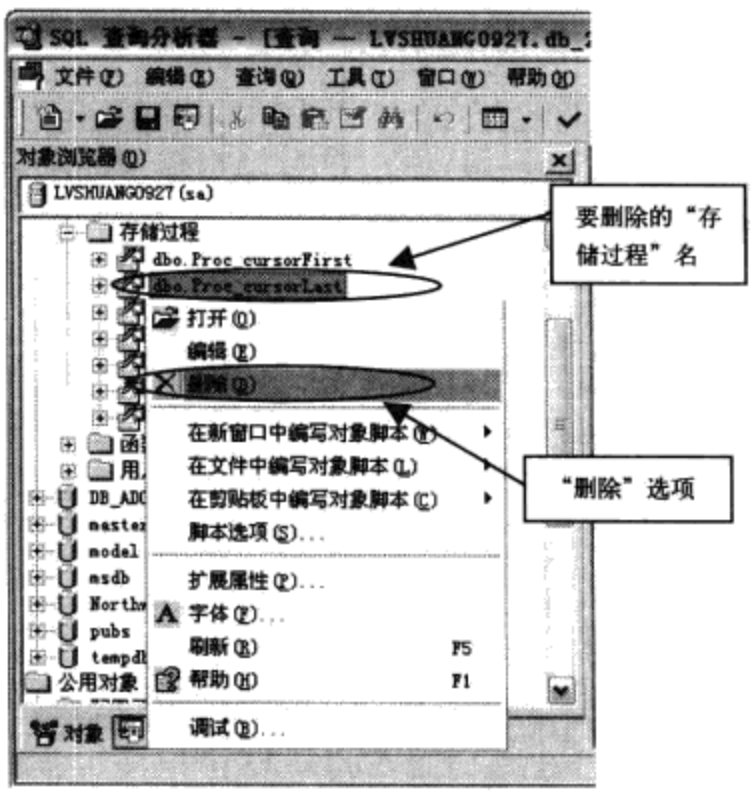


图 25.3　删除存储过程

2．通过 SQL 代码删除存储过程

通过 SQL 代码实现删除“存储过程”的代码如下：

```
drop proc 存储过程名
```

25.3 在 C#中调用存储过程

调用存储过程时主要会涉及两种状况：一种是执行而不需要返回值，如删除和修改等；另一种是执行并且要求有返回值，如查询。在 C#中调用存储过程时主要会用到两个类 SqlCommand 和 SqlDataAdapter，SqlCommand 类的 CommandType 属性可以获取或设置要对数据源执行的 Transact-SQL 语句或存储过程。当 SqlCommand 与存储过程关联起来之后就可以通过 SqlCommand 类来执行。另一种执行方法则是通过 SqlDataAdapter 类的 Fill 方法，该方法不仅会执行存储过程同时会将结果放入结果集中。

示例 25-03 C# 中调用存储过程

首先创建一个用于添加信息的存储过程（Proc_InsertOfPelple）与一个用于查找信息的存储过程（Proc_SelectOfPelple），然后定义一个数据操作类（ClsDBProcControl）用于实现对表（t_People）的各种操作，ClsDBProcControl 类中包括 ConDB 方法、Insert_Proc 方法、select_Proc 方法以及 t_People 表对应的实体类。在程序运行时，当用户单击“添加信息”按钮时，将姓名和性别信息赋值给 t_People 表的属性，然后通过调用 Insert_Proc 方法向数据表中插入新数据，当单击“查询信息”按钮时，将姓名和性别信息赋值给 t_People 表的属性，然后通过调用 select_Proc 方法查询指定信息。示例运行结果如图 25.4 和图 25.5 所示。

图 25.4 插入信息窗体

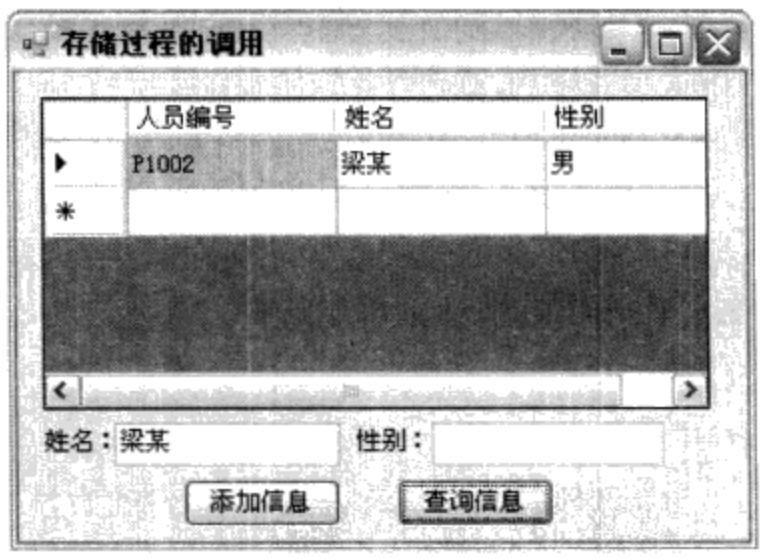


图 25.5 查询信息窗体

代码 25-03 光盘位置：光盘\mingrisoft\sl\25\25_03

在“添加信息”按钮的单击事件中，首先将姓名、性别信息赋值给属性 str_Name 和 str_Sex，然后再调用 Insert_Proc 方法实现插入信息功能。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    ClsDB.ClsDBProcControl CBDP = new OptDB.ClsDB.ClsDBProcControl();      //实例化 ClsDBProcControl 类
    CBDP.str_Name = this.textBox2.Text.Trim().ToString();                  //记录添加的姓名
    CBDP.str_Sex = this.textBox3.Text.Trim().ToString();                   //记录添加的性别
    if (CBDP.Insert_Proc(CBDP))                                            //调用存储过程
    {
        MessageBox.Show("插入成功");
        CBDP.str_Name =string.Empty;                                       //记录姓名
        CBDP.str_Sex = string.Empty;                                       //记录性别
```

```
        //调用存储过程查询添加后的记录
        this.dataGridView1.DataSource = CBDP.select_Proc(CBDP).Tables[0].DefaultView;
    }
}
```

在“查询信息”按钮的单击事件中，首先将姓名、性别信息赋予属性 str_Name 和 str_Sex，然后再调用 select_Proc 方法，select_Proc 方法将查询指定条件的数据，并返回一个 DataSet 对象用于将结果显示给用户。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    ClsDB.ClsDBProcControl CBDP = new OptDB.ClsDB.ClsDBProcControl();      //实例化 ClsDBProcControl 类
    CBDP.str_Name = this.textBox2.Text.Trim().ToString();                  //记录查询的姓名
    CBDP.str_Sex = this.textBox3.Text.Trim().ToString();                   //记录查询的性别
    //设用存储过程查询指定的数据
    this.dataGridView1.DataSource = CBDP.select_Proc(CBDP).Tables[0].DefaultView;
}
```

t_People 表对应实体，主要通过 GET、SET 访问器定义属性 str_Name 和 str_Sex。代码如下：

```
#region//表对应的实体
private string strName;                                                    //定义变量
public string str_Name
    {
        get                                                                //访问器
        {
            return this.strName;
        }
        set                                                                //访问器
        {
            strName = value;
        }
    }
private string strSex;
public string str_Sex
{
    get
    {
        return this.strSex;
    }
    set
    {
        strSex = value;
    }
}
#endregion
```

自定义方法 ConDB 主要用来打开连接的数据库。代码如下：

```
public SqlConnection ConDB()
{
    con = new SqlConnection("server=.;uid=sa;pwd=;database=DB_ADONET");    //连接数据库
    if (con.State == ConnectionState.Closed)                               //如果没有打开连接
    {
        con.Open();                                                        //打开连接的数据库
    }
    return con;
}
```

自定义方法 Insert_Proc 首先获取属性值 str_Name 和 str_Sex，然后将值传给存储过程(Proc_InsertOfPelple)，最后利用 SqlCommand 对象的 ExecuteNonQuery 方法执行存储过程（Proc_Insert OfPelple），完成信息添加功能。代码如下：

```
public bool Insert_Proc(ClsDBProcControl CDB)
{
    using(cmd = new SqlCommand())
    {
```

```
        try
        {
            cmd.CommandText = "Proc_InsertOfPelple";                                    //记录存储过程名
            cmd.Connection = ConDB();                                                   //打开连接的数据源
            cmd.CommandType = CommandType.StoredProcedure;                              //类型为存储过程
            cmd.Parameters.Add("@tb_PName", SqlDbType.VarChar);                         //添加参数
            cmd.Parameters["@tb_PName"].Value = CDB.str_Name;                           //添加参数值
            cmd.Parameters.Add("@tb_PSex", SqlDbType.VarChar).Value = CDB.str_Sex; //添加参数值
            cmd.ExecuteNonQuery();                                                      //执行存储过程
            return True;
        }
        catch
        {
            return False;
        }
        finally
        {
            if (con.State == ConnectionState.Open)
            {
                con.Close();                                                            //关闭数据库的连接
                con.Dispose();                                                          //释放对象
            }
        }
    }
}
```

自定义方法 select_Proc 首先获取属性值 str_Name 和 str_Sex，然后将值传给存储过程（Proc_SelectOfPelple），最后使用 SqlDataAdapter 类的 Fill 方法执行存储过程（Proc_SelectOfPelple）并将查询的结果填充给 DataSet 对象。代码如下：

```
public DataSet select_Proc(ClsDBProcControl CDB)
{
    try
    {
        //建立存储过程与数据源的连接
        SqlDataAdapter da = new SqlDataAdapter("Proc_SelectOfPelple", ConDB());
        da.SelectCommand.CommandType = CommandType.StoredProcedure;                 //设置类型为存储过程
        da.SelectCommand.Parameters.Add("@tb_PName", SqlDbType.VarChar).Value = CDB.str_Name;//添加参数
        da.SelectCommand.Parameters.Add("@tb_PSex", SqlDbType.VarChar).Value = CDB.str_Sex;    //添加参数
        DataSet ds = new DataSet();                                                 //实例化 DataSet 类
        da.Fill(ds);                                                        //执行存储过程，并将结果存入到 DataSet 类中
        return ds;                                                                  //返回查询的结果
    }
    catch
    {
        return null;
    }
}
```

25.4 视图的概述

视图是一种常用的数据库对象，可以把它看成虚拟表或存储在数据库中的查询，它为查看和存取数据提供了另外一种途径。大多数查询操作可以使用视图完成，而且通过视图还可以简化数据操作，提高数据库的安全性。

可通过视图访问的数据不作为独特的对象存储在数据库内，数据库内存储的是 SELECT 语句。SELECT 语句的结果集构成视图所返回的虚拟表。用户可以用引用表的方法，在 Transact-SQL 语句中通过引用视图名称来使用虚拟表。使用视图可以实现下列功能。

- 将用户限定在表中的特定行上。例如，只允许雇员看见工作跟踪表内记录其工作的行。
- 将用户限定在特定列上。例如，对于那些不负责处理工资单的雇员，只允许他们看见雇

员表中的姓名、办公室、工作电话和部门等字段，而不能看见任何包含工资信息或个人信息的字段。

- 将多个表中的列连接起来，使它们看起来像一个表。
- 聚合信息而非提供详细信息。例如，显示一个列的和，或列中数据的最大值或最小值。

25.5 视图的建立、修改和删除

创建视图的方法很多，本节将详细介绍如何在 SQL 查询分析器中通过模板完成对视图的操作。

25.5.1 视图的建立

首先在“SQL 查询分析器”的“对象浏览器”中选取“模板”选项卡，然后在模板列表中展开“Create View”节点，双击子节点，在右侧窗口显示出相应的模板信息，具体操作如图 25.6 所示。

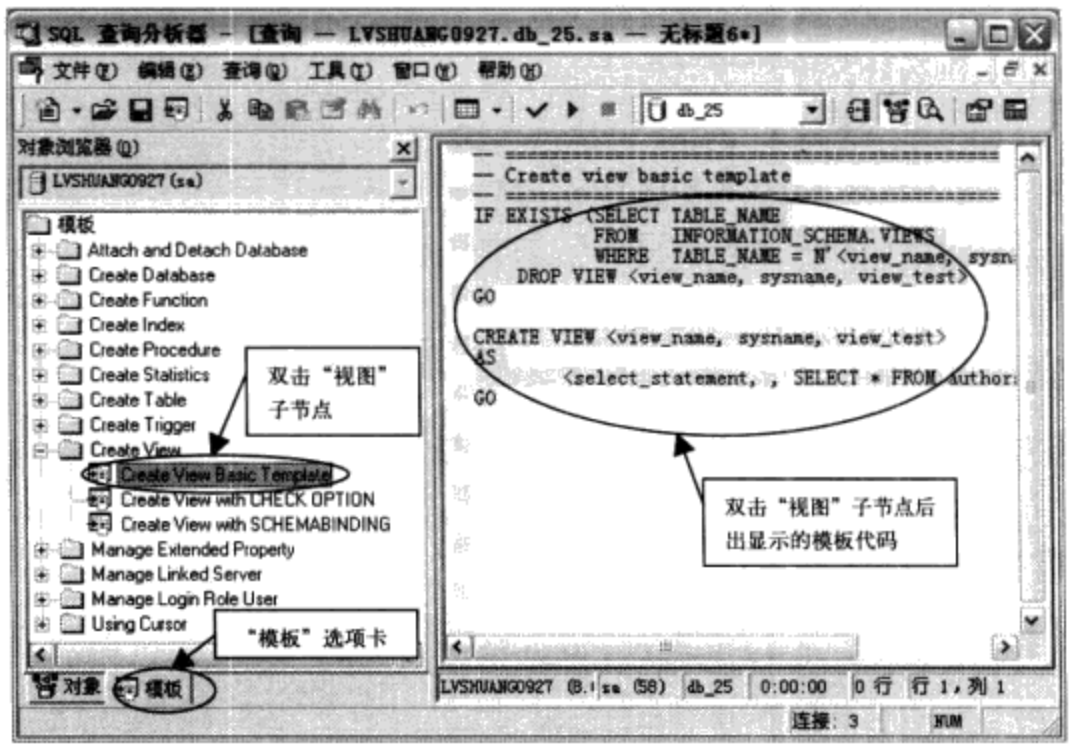


图 25.6 建立视图

建立视图的语法：

```
CREATE VIEW [ < database_name > .] [ < owner > .] view_name [ ( column [ ,...n ] ) ]
[ WITH < view_attribute > [ ,...n ] ]
AS
select_statement
[ WITH CHECK OPTION ]
```

建立视图语法中各参数及说明如表 25.2 所示。

表 25.2 建立视图语法中各参数及说明

参　数	说　明
view_name	视图的名称。视图名称必须符合标识符规则。可以选择是否指定视图所有者名称
column	视图中的列名。只有在下列情况下，才必须命名 CREATE VIEW 中的列：当列是从算术表达式、函数或常量派生的，两个或更多的列可能会具有相同的名称（通常是因为连接），视图中的某列被赋予了不同于派生来源列的名称。还可以在 SELECT 语句中指派列名
n	表示可以指定多列的占位符

续表

参　　数	说　　明
AS	视图要执行的操作
select_statement	定义视图的 SELECT 语句。该语句可以使用多个表或其他视图。若要从创建视图的 SELECT 子句所引用的对象中选择，必须具有适当的权限
WITH CHECK OPTION	强制视图上执行的所有数据修改语句都必须符合由 select_statement 设置的准则。通过视图修改行时，WITH CHECK OPTION 可确保提交修改后，仍可通过视图看到修改的数据

示例 25-04　建立视图

本示例主要完成视图的建立过程，它用于实现将两表信息组合成新的虚拟表。

程序代码如下：

代码 25-04　光盘位置：光盘\mingrisoft\sl\25\25_04

```
IF EXISTS (SELECT TABLE_NAME
        FROM INFORMATION_SCHEMA.VIEWS
        WHERE   TABLE_NAME ='View_Try')
    DROP VIEW View_Try
GO
CREATE VIEW View_Try
AS
select tc.tb_PID,tc.tb_CClase,tc.tb_CNum,tp.tb_PName,tp.tb_PSex
from t_Class tc,t_People tp
where tp.tb_PID=tc.tb_PID
GO
```

25.5.2　视图的修改

在“SQL 查询分析器”的“对象浏览器”中选择“视图”，右键单击要修改的视图名，在弹出的快捷菜单中选择“编辑”命令，这时在右侧窗口将显示修改视图的 SQL 语句，可以直接在上面更改 SQL 语句。具体操作如图 25.7 所示。

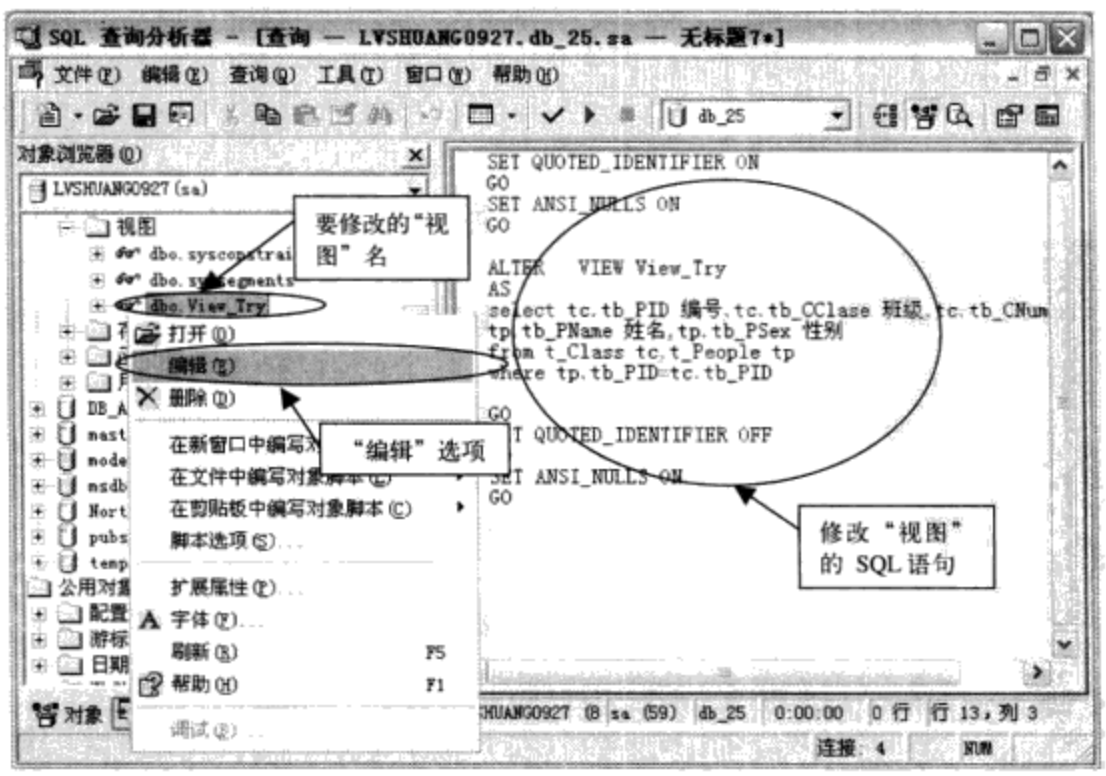


图 25.7　修改视图

示例 25-05　修改视图

程序代码如下：

代码 25-05　光盘位置：光盘\mingrisoft\sl\25\25_05

```
ALTER VIEW View_Try
AS
select tc.tb_PID 编号,tc.tb_CClase 班级,tc.tb_CNum 人数,
tp.tb_PName 姓名,tp.tb_PSex 性别
from t_Class tc,t_People tp
where tp.tb_PID=tc.tb_PID
```

注意：修改视图的关键字是 ALTER，语法为 ALTER VIEW 视图名。

25.5.3 视图的删除

可以通过 SQL 查询分析器和 SQL 代码两种方法删除视图。

1．通过 SQL 查询分析器删除视图

在“SQL 查询分析器”的“对象浏览器”中选取“视图”，右键单击要删除的视图，在弹出的快捷菜单中选择“删除”命令即可将选择的视图删除，如图 25.8 所示。

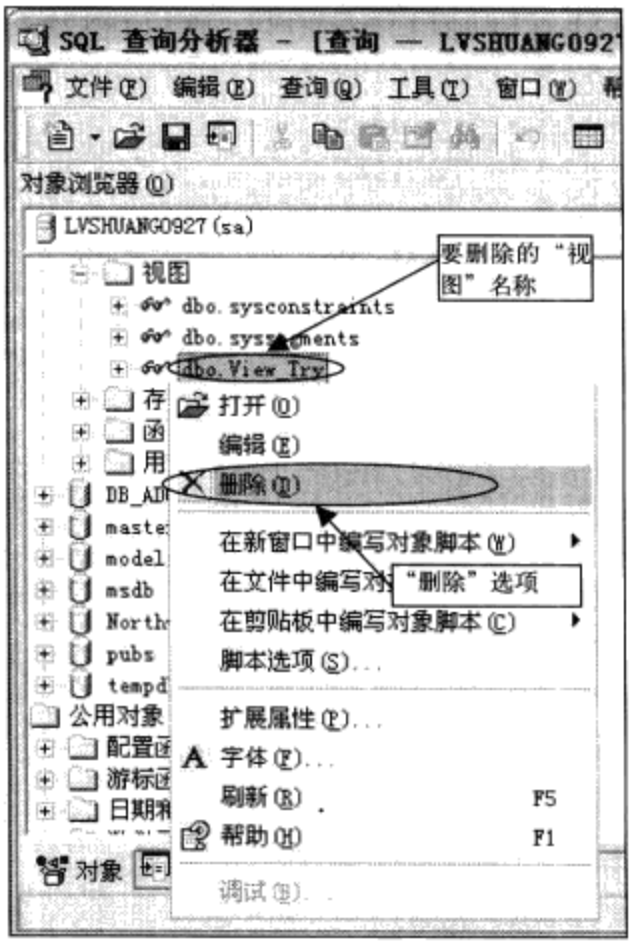


图 25.8　删除视图

2．通过 SQL 代码删除视图

通过 SQL 代码实现删除“视图”的代码如下：

```
drop View 视图名
```

25.6　在 C#中调用视图

视图的调用方法与表的调用是一样的，由于视图是虚拟表，所以一般在查询中使用。例如，在 C#中调用视图中的结果时通常会用到 SqlDataAdapter 类，然后将视图中的结果通过 Fill 方法填充结果集（DataSet）。

示例 25-06　C#中调用视图

首先创建一个用于显示查询结果的视图 View_Try，然后建立一个数据操作类 ClsView.cs，

其中包括 ConDB 和 dsView 方法（用于查询视图）。当程序运行时，通过调用数据操作类（ClsView）实现对虚拟表（由 t_People 表和 t_Class 表组成）的查询操作，同时也实现了对查询结果的某一字段过滤与排序的操作，示例运行结果如图 25.9 所示。

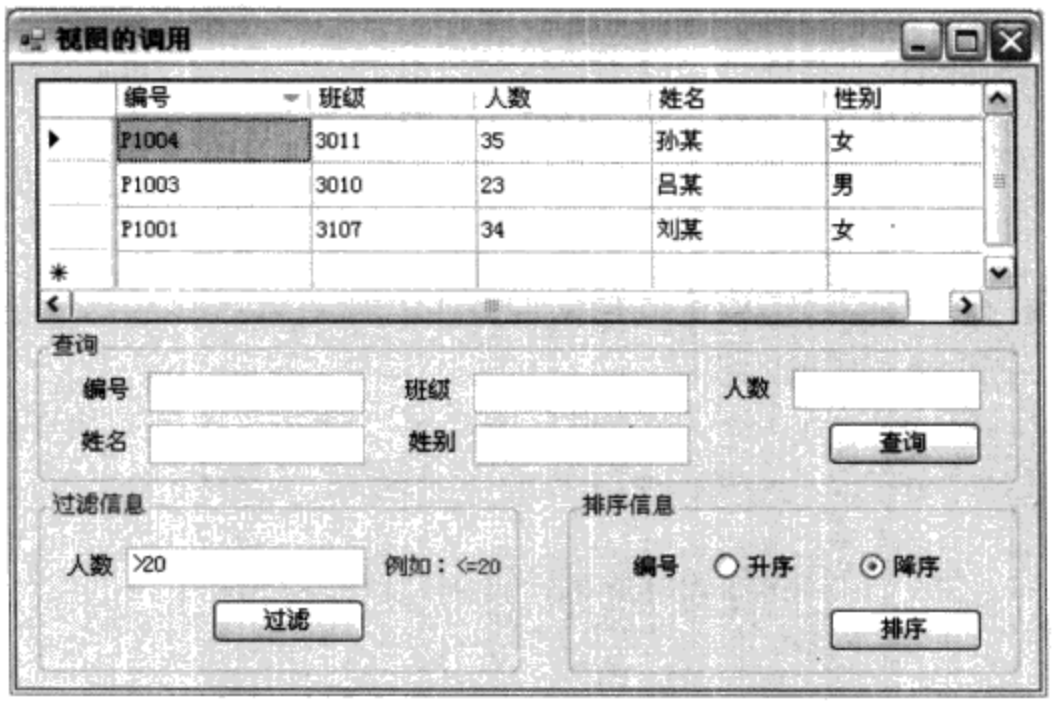


图 25.9　视图演示窗体

代码 25-06　光盘位置：光盘\mingrisoft\sl\25\25_06

（1）ConDB 方法用于建立数据库的连接。

（2）dsView 方法用于执行视图（View_Try）。

自定义方法 ConDB 用于连接打开的数据源。代码如下：

```
public SqlConnection ConDB()
{
    con = new SqlConnection("server=.;uid=sa;pwd=;database=DB_ADONET");    //连接数据源
    if (con.State == ConnectionState.Closed)                               //如果没有打开数据源
    {
        con.Open();                                                        //打开数据源
    }
    return con;
}
```

自定义方法 dsView 用于执行 View_Try 视图。代码如下：

```
public DataSet dsView(string SQLWhere)
{
    string SQL = "select * from View_Try";                                 //记录视图的 SQL 语句
    SQL += SQLWhere;                                                       //添加查询条件
    try
    {
        DataSet ds = new DataSet();                                        //实例化 DataSet 对角
        da = new SqlDataAdapter(SQL, ConDB());                             //连接数据源与 SQL 语句
        da.Fill(ds);                                          //执行 SQL 语句，并将数据添加到 DataSet 对象中
        return ds;
    }
    catch
    {
        return null;
    }
}
```

在“查询”按钮的单击事件中，首先生成有查询条件的 SQL 语句，然后将 SQL 语句传递给 dsView 方法，实现信息查询。代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
```

```
    StringBuilder strSQLWhere = new StringBuilder();                                    //定义一个可变字符串变量
    //添加查询条件
    strSQLWhere.Append(" where ");
    strSQLWhere.Append(" 编号 like '%" + this.textBox1.Text.Trim().ToString() + "%' and");
    strSQLWhere.Append(" 班级 like '%" + this.textBox2.Text.Trim().ToString() + "%' and");
    strSQLWhere.Append(" 人数 like '%" + this.textBox3.Text.Trim().ToString() + "%' and");
    strSQLWhere.Append(" 姓名 like '%" + this.textBox4.Text.Trim().ToString() + "%' and");
    strSQLWhere.Append(" 性别 like '%" + this.textBox5.Text.Trim().ToString() + "%'");
    ClsDB.ClsView CV = new OptDB.ClsDB.ClsView();                                       //执行视图
    this.dataGridView1.DataSource=CV.dsView( strSQLWhere.ToString()).Tables[0];         //显示查询的结果
}
```

自定义方法 ChangeRowFilter 按照指定的条件对数据源进行筛选，代码如下：

```
private void ChangeRowFilter(string str)
{
    if (this.textBox6.Text.Trim().ToString() != "")                                     //如果设置了筛选条件
    {
      //将 dataGridView1 控件中的数据存入到 DataTable 中
      DataTable custDV = (DataTable)this.dataGridView1.DataSource;
      if (custDV != null)                                                               //如果 DataTable 不为空
      {
            custDV.DefaultView.RowFilter = "人数" + str;                                //设置筛选条件
            this.dataGridView1.DataSource = custDV;                                     //显示筛选结果
      }
      else
      {
            MessageBox.Show("无操作数据");
      }
    }
}
```

自定义方法 SortRow 用于实现对数据进行升序和降序的排序，代码如下：

```
private void SortRow(string str)
{
    //将 dataGridView1 控件中的数据存入到 DataTable 中
    DataTable custDV = (DataTable)this.dataGridView1.DataSource;
    if (custDV != null)                                                                 //如果 DataTable 不为空
    {
      custDV.DefaultView.Sort = " 编号"+str;                                            //设置排序条件
      this.dataGridView1.DataSource = custDV;                                           //显示排序后的结果
    }
    else
    {
       MessageBox.Show("无操作数据");
    }
}
```

在“过滤”按钮的单击事件中，调用 ChangeRowFilter 方法实现对数据的过滤功能。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    this.ChangeRowFilter(this.textBox6.Text.Trim().ToString());
}
```

在“排序”按钮的单击事件中，调用 SortRow 方法实现对数据的排序功能。代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    string strState;
    if (this.radioButton1.Checked)                                                      //如果选中升序单选项
      strState = "asc";                                                                 //设置升序条件
    else
      strState = "desc";                                                                //设置降序条件
    SortRow(strState);                                                                  //调用自定义方法
}
```

25.7 触发器的应用

触发器是一种特殊的存储过程，与数据表紧密联系。当用户修改数据表中的数据时，触发器将自动执行，触发器可以使用 Transact-SQL 语句进行复杂的逻辑处理，它基于一个数据表创建，但是可以对多个数据表进行操作。创建触发器的语法代码如下：

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
 [ WITH APPEND ]
 [ NOT FOR REPLICATION ]
AS
[ { IF UPDATE ( column )
[ { AND | OR } UPDATE ( column ) ]
[ ...n ]
| IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
{ comparison_operator } column_bitmask [ ...n ]
} ]
sql_statement [ ...n ]
}
}
```

创建触发器语法中各参数及说明如表 25.3 所示。

表 25.3 创建触发器语法中各参数及说明

参　数	说　明
trigger_name	触发器的名称。触发器名称必须符合标识符规则，并且在数据库中必须惟一。可以选择是否指定触发器所有者名称
Table | view	在其上执行触发器的表或视图，有时称为触发器表或触发器视图。可以选择是否指定表或视图的所有者名称
WITH ENCRYPTION	加密 syscomments 表中包含 CREATE TRIGGER 语句文本的条目。使用 WITH ENCRYPTION 可防止将触发器作为 SQL Server 复制的一部分发布
AFTER	指定触发器只有在触发 SQL 语句中指定的所有操作都已成功执行后才激发。所有的引用级联操作和约束检查也必须成功完成后，才能执行此触发器
INSTEAD OF	指定执行触发器而不是执行触发 SQL 语句，从而替代触发语句的操作。在表或视图上，每个 INSERT、UPDATE 或 DELETE 语句最多可以定义一个 INSTEAD OF 触发器。然而，可以在每个具有 INSTEAD OF 触发器的视图上定义视图。 INSTEAD OF 触发器不能在 WITH CHECK OPTION 的可更新视图上定义。如果向指定了 WITH CHECK OPTION 选项的可更新视图添 INSTEAD OF 触发器，SQL Server 将产生一个错误。用户必须用 ALTER VIEW 删除该选项后才能定义 INSTEAD OF 触发器
{ [DELETE] [,] [INSERT] [,] [UPDATE] }	指定在表或视图上执行哪些数据修改语句时将激活触发器的关键字。必须至少指定一个选项。在触发器定义中允许使用以任意顺序组合的这些关键字。如果指定的选项多于一个，需用逗号分隔这些选项。 对于 INSTEAD OF 触发器，不允许在具有 ON DELETE 级联操作引用关系的表上使用 DELETE 选项。同样，也不允许在具有 ON UPDATE 级联操作引用关系的表上使用 UPDATE 选项
WITH APPEND	指定应该添加现有类型的其他触发器。只有当兼容级别是 65 或更低时，才需要使用该可选子句。如果兼容级别是 70 或更高，则不必使用 WITH APPEND 子句添加现有类型的其他触发器

续表

参　数	说　明
NOT FOR REPLICATION	表示当复制进程更改触发器所涉及的表时，不应执行该触发器
AS	触发器要执行的操作
sql_statement	触发器的条件和操作。触发器条件指定其他准则，以确定 DELETE、INSERT 或 UPDATE 语句是否导致执行触发器操作。 当尝试 DELETE、INSERT 或 UPDATE 操作时，Transact-SQL 语句中指定的触发器操作将生效

指定触发器何时触发分为以下两种情况。

- AFTER 触发器在触发操作（INSERT、UPDATE 或 DELETE）后和处理完任何约束后触发。可通过指定 AFTER 或 FOR 关键字来请求 AFTER 触发器。因为 FOR 关键字与 AFTER 的效果相同，所以具有 FOR 关键字的触发器也归为 AFTER 触发器。
- INSTEAD OF 触发器代替触发动作进行触发，并在处理约束之前触发。对于每个触发操作（UPDATE、DELETE 和 INSERT），每个表或视图只能有一个 INSTEAD OF 触发器。而一个表对于每个触发操作可以有多个 AFTER 触发器。

示例 25-07　触发器的建立与应用

本示例实现的是，当对班级表进行删除操作时，其学生表中属于被删除班的学生信息自动删除。

程序代码如下：

代码 25-07　光盘位置：光盘\mingrisoft\sl\25\25_07

```
IF EXISTS (SELECT name
        FROM sysobjects
        WHERE name = 'Trig_Del'
        AND type = 'TR')
    DROP TRIGGER Trig_Del
GO
CREATE TRIGGER Trig_Del
ON t_TClass
FOR DELETE
AS
BEGIN
    declare @ID varchar(10)
    select @ID=t_trigClss from deleted
    delete from t_TStu where t_trigClass=@ID
END
```

注意：触发器的特性是自动执行而不用对其进行调用，用户如果想要调用触发器可以借助存储过程实现。

25.8　游标的应用

对关系数据库中的操作会对整个行集产生影响。由 SELECT 语句返回的行集包括所有满足该语句 WHERE 子句中条件的行，由语句所返回的完整的行集被称为结果集。应用程序，特别是交互式联机应用程序，并不总能将整个结果集作为一个单元来有效地处理，这些应用程序需要一种机制以便每次处理一行或一部分行，游标就是提供这种机制的结果集扩展。

游标通过以下方式扩展结果处理。

- 允许定位在结果集的特定行。
- 从结果集的当前位置检索一行或多行。
- 支持对结果集中当前位置的行进行数据修改。
- 为由其他用户对显示在结果集中的数据所做的更改提供不同级别的可见性支持。
- 提供脚本、存储过程和触发器中使用的访问结果集中的数据的 Transact-SQL 语句。

建立游标的语法如下：

```
DECLARE cursor_name CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_statement
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

建立游标语法中各参数及说明如表 25.4 所示。

表 25.4　　建立游标语法中各参数及说明

参　数	说　明
Cursor_name	所定义的 Transact-SQL 服务器游标名称
LOCAL	指定该游标的作用域对在其中创建它的批处理、存储过程或触发器是局部的。该游标名称仅在这个作用域内有效。在批处理、存储过程、触发器或存储过程 OUTPUT 参数中，该游标可由局部游标变量引用。OUTPUT 参数用于将局部游标传递回调用批处理、存储过程或触发器，它们可在存储过程终止后给游标变量指派参数使其引用游标。除非 OUTPUT 参数将游标传递回来，否则游标将在批处理、存储过程或触发器终止时隐性释放。如果 OUTPUT 参数将游标传递回来，游标在最后引用它的变量释放或离开作用域时释放
GLOBAL	指定该游标的作用域对链接是全局的。在由链接执行的任何存储过程或批处理中，都可以引用该游标名称。该游标仅在脱接时隐性释放
FORWARD_ONLY	指定游标只能从第一行滚动到最后一行。FETCH NEXT 是惟一受支持的提取选项。如果在指定 FORWARD_ONLY 时不指定 STATIC、KEYSET 和 DYNAMIC 关键字，则游标作为 DYNAMIC 游标进行操作。如果 FORWARD_ONLY 和 SCROLL 均未指定，除非指定 STATIC、KEYSET 或 DYNAMIC 关键字，否则默认为 FORWARD_ONLY。STATIC、KEYSET 和 DYNAMIC 游标默认为 SCROLL。与 ODBC 和 ADO 这类数据库 API 不同，STATIC、KEYSET 和 DYNAMIC Transact-SQL 游标支持 FORWARD_ONLY
STATIC	定义一个游标，以创建将由该游标使用的数据的临时副本。对游标的所有请求都从 tempdb 中的该临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中不反映对基表所做的修改，并且该游标不允许修改
KEYSET	指定当游标打开时，游标中行的成员资格和顺序已经固定。对行进行惟一标识的键集内置在 tempdb 内一个称为 keyset 的表中。对基表中的非键值所做的更改（由游标所有者更改或由其他用户提交）在用户滚动游标时是可视的。其他用户进行的插入是不可视的（不能通过 Transact-SQL 服务器游标进行插入）。如果某行已删除，则对该行的提取操作将返回 @@FETCH_STATUS 值-2。从游标外更新键值类似于删除旧行后接着插入新行的操作。含有新值的行不可视，对含有旧值的行的提取操作将返回@@FETCH_STATUS 值-2。如果通过指定 WHERE CURRENT OF 子句用游标完成更新，则新值可视
DYNAMIC	定义一个游标，以反映在滚动游标时对结果集内的行所做的所有数据更改。行的数据值、顺序和成员在每次提取时都会更改。动态游标不支持 ABSOLUTE 提取选项
FAST_FORWARD	指定启用了性能优化的 FORWARD_ONLY、READ_ONLY 游标
READ_ONLY	禁止通过该游标进行更新。在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不能引用游标。该选项替代要更新的游标的默认功能
SCROLL_LOCKS	指定确保通过游标完成的定位更新或定位删除可以成功。当将行读入游标以确保它们可用于以后的修改时，Microsoft® SQL Server™会锁定这些行。如果还指定了 FAST_FORWARD，则不能指定 SCROLL_LOCKS

续表

参　数	说　明
OPTIMISTIC	指定如果行自从被读入游标以来已得到更新，则通过游标进行的定位更新或定位删除不成功。当将行读入游标时 SQL Server 不锁定行。相反，SQL Server 使用 timestamp 列值的比较，或者如果表没有 timestamp 列则使用校验值，以确定将行读入游标后是否已修改该行
TYPE_WARNING	指定如果游标从所请求的类型隐性转换为另一种类型，则给客户端发送警告消息

示例 25-08　建立游标

建立游标，通过游标来完成特定信息的提取功能，如果 People_cursor 数据表中存在 5 条记录，通过游标可以提取一条指定员工信息。

程序代码如下：

代码 25-08　光盘位置：光盘\mingrisoft\sl\25\25_08

```
create proc Proc_cursorNum
@Num int
as
begin
    DECLARE People_cursor SCROLL CURSOR FOR
    SELECT * FROM t_People
    ORDER BY tb_PID
    open People_cursor
    FETCH ABSOLUTE @Num FROM People_cursor
    deallocate People_cursor
end
```

示例 25-09　使用游标

当程序运行时，调用存储过程，并通过存储过程操作游标来完成特定信息的提取（如获取工编号为 P1004 的员工信息）。示例运行结果如图 25.10 所示。

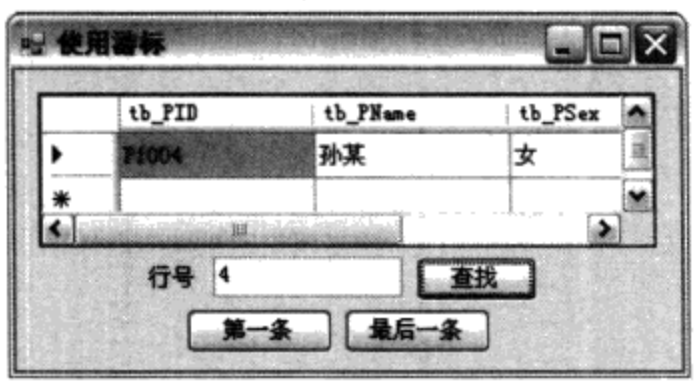


图 25.10　游标演示窗体

代码 25-09　光盘位置：光盘\mingrisoft\sl\25\25_09

自定义方法 Num_ProcOfCussor 用于调用存储过程 Proc_cursorNum，同时传入所需的信息编号，将返回结果放入结果集中。代码如下：

```
public DataSet Num_ProcOfCussor(int Num)
{
    try
    {
        SqlDataAdapter da = new SqlDataAdapter("Proc_cursorNum", ConDB());      //建立存储过程与数据源的连接
        da.SelectCommand.CommandType = CommandType.StoredProcedure;             //设置类型为存储过程
        da.SelectCommand.Parameters.Add("@Num", SqlDbType.Int).Value = Num;     //添加参数
        DataSet ds = new DataSet();                                             //实例化 DataSet 对象
        da.Fill(ds);                                                            //将查询的数据添加到 DataSet 对象中
        return ds;
    }
```

```
        catch
        {
            return null;                                          //如果发生异常返回空
        }
    }
```

在“查找”按钮的单击事件中，将行号传给 Num_ProcOfCussor 方法，并获得指定行的数据信息。代码如下：

```
    private void button3_Click(object sender, EventArgs e)
    {
        ClsDB.ClsCussor CU = new OptDB.ClsDB.ClsCussor();         //实例化 ClsCussor 类
        int Num;                                                  //记录行号
        try
        {
            Num = int.Parse(this.textBox1.Text.Trim().ToString());  //获取设置的行号
        }
        catch (Exception ey)
        {
            MessageBox.Show(ey.Message);
            return;
        }
        this.dataGridView1.DataSource = CU.Num_ProcOfCussor(Num).Tables[0].DefaultView;   //显示指定行的信息
    }
```

第 5 篇

文件技术篇

C#

第 26 章

文件处理技术

- System.IO 命名空间
- 文件的基本操作
- 文件夹的基本操作
- 文件写入和读取

26.1 System.IO 命名空间

输入/输出是操作系统的一个重要组成部分。.NET 框架提供了基于流的操作方式，在.NET 框架中所有的输入和输出工作都要使用流。流是串行化设备的抽象表示，串行化设备可以以线性方式存储数据，并可以以同样的方式访问数据。此设备可以是磁盘文件、打印机、内存位置和任何其他支持以线性方式读写的对象。这样就大大减轻了开发者的工作，因为可以对一系列的通用对象进行操作，而不必关心该 I/O 操作是本机的文件还是和网络中的数据有关。

在 System.IO 命名空间中有众多的类，用于进行数据文件和流的读写操作，这些操作可以同步进行也可以异步进行。

System.IO 命名空间中的类及说明如表 26.1 所示。

表 26.1　　System.IO 命名空间中的类及说明

类	说　明
BinaryReader	用特定的编码将基元数据类型读作二进制值
BinaryWriter	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
BufferedStream	给另一流上的读写操作添加一个缓冲层。无法继承此类
Directory	公开用于创建、移动和枚举通过目录和子目录的静态方法。无法继承此类
DirectoryInfo	公开用于创建、移动和枚举目录和子目录的实例方法。无法继承此类
DirectoryNotFoundException	当找不到文件或目录的一部分时所引发的异常
DriveInfo	提供对有关驱动器的信息的访问
DriveNotFoundException	当尝试访问的驱动器或共享不可用时引发的异常
EndOfStreamException	读操作试图超出流的末尾时引发的异常
ErrorEventArgs	为 Error 事件提供数据
File	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
FileInfo	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。无法继承此类
FileLoadException	当找到托管程序集却不能加载它时引发的异常
FileNotFoundException	试图访问磁盘上不存在的文件失败时引发的异常
FileStream	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
FileSystemEventArgs	提供目录事件的数据：Changed、Created、Deleted
FileSystemInfo	为 Fileinfo 和 Directoryinfo 对象提供基类
FileSystemWatcher	侦听文件系统更改通知，并在目录或目录中的文件发生更改时引发事件
InternalBufferOverflowException	内部缓冲区溢出时引发的异常
InvalidDataException	在数据流的格式无效时引发的异常
IODescriptionAttribute	设置可视化设计器在引用事件、扩展程序或属性时可显示的说明
IOException	发生 I/O 错误时引发的异常
MemoryStream	创建其支持存储区为内存的流
Path	对包含文件或目录路径信息的 String 实例执行操作。这些操作是以跨平台的方式执行的
PathTooLongException	当路径名或文件名超过系统定义的最大长度时引发的异常
RenamedEventArgs	为 Renamed 事件提供数据
STREAM	提供字节序列的一般视图
StreamReader	实现一个 Textreader，使其以一种特定的编码从字节流中读取字符
StreamWriter	实现一个 Textwriter，使其以一种特定的编码向流中写入字符
StringReader	实现从字符串进行读取的 TextReader

续表

类	说　明
StringWriter	实现一个用于将信息写入字符串的 TextWriter。该信息存储在基础 StringBuilder 中
TextReader	表示可读取连续字符系列的读取器
TextWriter	表示可以编写一个有序字符系列的编写器。该类为抽象类
UnmanagedMemoryStream	提供从托管代码访问非托管内存块的能力

26.2　文件的基本操作

26.2.1　文件操作类 File 和 FileInfo

File 类和 FileInfo 类都可以对文件进行创建、复制、删除、移动、打开、读取，以及获取文件的基本信息操作，下面对这两个类进行简单介绍。

1．File 类

File 类支持对文件的基本操作，包括提供用于创建、复制、删除、移动和打开文件的静态方法，并协助创建 FileStream 对象。由于所有的 File 类的方法都是静态的，所以如果只想执行一个操作，那么使用 File 方法的效率比使用相应的 FileInfo 实例方法可能更高。所有的 File 方法都要求当前所操作的文件的路径。也可将 File 类用于获取和设置文件属性或有关文件创建、访问及写入操作的 DateTime（时间）信息。File 类可以被实例化，但不能被其他类继承。

File 类的常用方法及说明如表 26.2 所示。

表 26.2　File 类的常用方法及说明

方　法	说　明
AppendAllText	将指定的字符串追加到文件中，如果文件还不存在则创建该文件
AppendText	创建一个 StreamWriter，它将 UTF-8 编码文本追加到现有文件
Create	在指定路径中创建文件
Copy	将现有文件复制到新文件
CreateText	创建或打开一个文件用于写入 UTF-8 编码的文本
Decrypt	解密由当前帐户使用 Encrypt 方法加密的文件
Delete	删除指定的文件。如果指定的文件不存在，则不引发异常
Encrypt	将某个文件加密，使得只有加密该文件的账户才能将其解密
Exists	确定指定的文件是否存在
GetCreationTime	返回指定文件或目录的创建日期和时间
GetCreationTimeUtc	返回指定的文件或目录的创建日期及时间，其格式为协调通用时间（UTC）
GetLastAccessTime	返回上次访问指定文件或目录的日期和时间
GetLastAccessTimeUtc	返回上次访问指定的文件或目录的日期及时间，其格式为协调通用时间（UTC）
GetLastWriteTime	返回上次写入指定文件或目录的日期和时间
GetLastWriteTimeUtc	返回上次写入指定的文件或目录的日期和时间，其格式为协调通用时间（UTC）
Move	将指定文件移到新位置，并提供指定新文件名的选项
Open	打开指定路径上的 FileStream
OpenRead	打开现有文件以进行读取
OpenText	打开现有 UTF-8 编码文本文件以进行读取
OpenWrite	打开现有文件以进行写入
ReadAllBytes	打开一个文件，将文件的内容读入一个字符串，然后关闭该文件

续表

方　法	说　明
ReadAllLines	打开一个文本文件，将文件的所有行都读入一个字符串数组，然后关闭该文件
ReadAllText	打开一个文本文件，将文件的所有行读入一个字符串，然后关闭该文件
Replace	使用其他文件的内容替换指定文件的内容，这一过程将删除原始文件，并创建被替换文件的备份
SetCreationTime	设置创建该文件的日期和时间
SetCreationTimeUtc	设置文件创建的日期和时间，其格式为协调通用时间（UTC）
SetLastAccessTime	设置上次访问指定文件的日期和时间
SetLastAccessTimeUtc	设置上次访问指定的文件的日期和时间，其格式为协调通用时间（UTC）
SetLastWriteTime	设置上次写入指定文件的日期和时间
SetLastWriteTimeUtc	设置上次写入指定的文件的日期和时间，其格式为协调通用时间（UTC）
WriteAllBytes	创建一个新文件，在其中写入指定的字节数组，然后关闭该文件。如果目标文件已存在，则覆盖该文件
WriteAllLines	创建一个新文件，在其中写入指定的字符串，然后关闭文件。如果目标文件已存在，则覆盖该文件
WriteAllText	创建一个新文件，在文件中写入内容，然后关闭文件。如果目标文件已存在，则覆盖该文件

2．FileInfo 类

FileInfo 类和 File 类之间的许多方法调用都是相同的，但是 FileInfo 类没有静态方法，仅可以用于实例化对象。File 类是静态类，它的调用需要字符串参数为每一个方法调用规定文件位置，因此如果要在对象上进行单一方法调用，则可以使用静态 File 类，反之则使用 FileInfo 类。

FileInfo 类与 File 类所调用的方法基本相同，只是 File 类在调用方法时必须使用文件路径，而 FileInfo 类在使用前已经通过文件路径进行了实例化对象，所以在调用方法时不需要再使用文件路径。

FileInfo 类的常用方法及说明如表 26.3 所示。

表 26.3　　FileInfo 类的常用属性及说明

属　性	说　明
Attributes	获取或设置当前 Filesysteminfo 的 Fileattributes
CreationTime	获取或设置当前 FileSystemInfo 对象的创建时间
DirectoryName	获取表示目录的完整路径的字符串
Exists	获取指示文件是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
Length	获取当前文件的大小
Name	获取文件名

交叉链接：FileInfo 类所使用的相关方法请参见表 26.2。

26.2.2　创建文件

用户在创建文件时，有两种格式的文件，一是 UTF-8 编码文本的文件，二是非 UTF-8 编码文本的文件。在文本中包含“<”（<）、“>”（>）、“&”（&）和“"”（”）语法时，或创建的是 XML 文件，应该使用 UTF-8 编码文本的格式。可以用 File 类和 FileInfo 类实现文件的创建操作，本节主要用 File 类进行讲解。

1．通过 Create 方法创建非 UTF-8 编码文本

Create 方法在指定路径中创建或覆盖文件。

语法：

```
public static FileStream Create(string path,int bufferSize,FileOptions options)
```

参数说明如下。

path：要创建的文件的路径及名称。

bufferSize：用于读取和写入文件的已放入缓冲区的字节数。

options：FileOptions 值之一，它描述如何创建或覆盖该文件。FileOptions 值的成员及说明如表 26.4 所示。

表 26.4　　FileOptions 值的成员及说明

成　员	说　明
None	指示无其他参数
WriteThrough	指示系统应通过任何中间缓存、直接写入磁盘
Asynchronous	指示文件可用于异步读取和写入
RandomAccess	指示随机访问文件。系统可将此选项用作优化文件缓存的提示
DeleteOnClose	指示当不再使用某个文件时，自动删除该文件
SequentialScan	指示按从头到尾的顺序访问文件。系统可将此选项用作优化文件缓存的提示。如果应用程序移动用于随机访问的文件指针，可能不发生优化缓存，但仍然保证操作的正确性
Encrypted	指示文件是加密的，只能通过用于加密的同一用户账户来解密

返回值：Filestream 对象，它提供对 path 中指定的文件的读/写访问。

例如，在 D 盘下创建一个纯文本格式的文本文件，如果有同名文件则覆盖。代码如下

```
FileStream fs = File.Create("D:\hb.txt");
```

例如，创建一个具有指定缓冲区大小的文件，并向该文件中写入文本。代码如下：

```
string path = "hb.txt";
using (FileStream fs = File.Create(path, 1024))
{
    Byte[] info = new UTF8Encoding(true).GetBytes("This is some text in the file.");   //将字符串转换成字节
    fs.Write(info, 0, info.Length);                                                     //将字节写入到文本文件中
}
```

例如，创建一个可异步读取和写入的文本文件。代码如下：

```
File.Create("hb.txt", 1024,FileOptions.Asynchronous);
```

2．通过 CreateText 方法创建 UTF-8 编码的文本

使用 CreateText 方法创建或打开一个文件用于写入 UTF-8 编码的文本。

语法：

```
public static StreamWriter CreateText(string path)
```

参数说明如下。

path：要创建的文件的路径及名称。

返回值：StreamWriter 对象，使用 UTF-8 编码写入指定的文件。

例如，创建一个 XML 文件，并向该文件中写入文本。代码如下：

```
string path = @"D:\MyTest.xml";                    //设置 XML 文件的路径
using (StreamWriter sw = File.CreateText(path)) //创建一个可写入的 XML 文件
{                                                   //向 XML 文件中写入内容
    sw.WriteLine("<!-- Display special characters that require special encoding: < > & ”  -->");
    sw.WriteLine("<TextBlock>");
    sw.WriteLine("&lt;      <!-- 小于号 -->");
    sw.WriteLine("&gt;      <!-- 大于号 -->");
    sw.WriteLine("&amp;     <!-- “and”符 -->");
    sw.WriteLine("&quot;    <!-- 双引号 -->");
    sw.WriteLine("</TextBlock>");
}
```

例如，创建一个位图文件。代码如下：

```
string path = @"D:\MyTest.bmp";                    //设置一个位图文件的路径
File.CreateText(path);                              //创建一个位图文件
```

说明：File 类不但可以创建文本文件，还可以创建 Word、Excel、位图等其他文件。

注意：在创建文件时，也可以使用 FileInfo 类实现，其调用的方法基本相同，下面用 FileInfo 类实现文件的创建。代码如下：

```
FileInfo fileInfo = new FileInfo("hb.text");        //实例化 FileInfo 类
StreamWriter sw = fileInfo.CreateText();            //创建文件
sw.Write("This is some text in the file.");         //向文件中写入信息
sw.Dispose();                                       //释放资源
```

26.2.3 打开文件

用户在打开文件时可以有 4 种方式：一是以读/写方式打开文件，二是以只读方式打开文件，三是以写入方式打开文件，四是以 UTF-8 编码文本打开文件。可以用 File 类和 FileInfo 类实现文件的读取操作，本节主要用 File 类进行讲解。

1．以读/写方式打开文件

以读/写方式打开文件是用 File 类的 Open 方法实现的。该方法用于打开指定路径上的 FileStream，并具有读/写访问权限。

语法：

```
public static FileStream Open(string path,FileMode mode)
```

参数说明如下：

path：string 型，要打开的文件路径。

mode：FileMode 值，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 值的成员及说明如表 26.5 所示。

表 26.5　　FileMode 值的成员及说明

成　员	说　明
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

返回值：FileStream 类，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。

例如，打开一个可读写的文件。代码如下：

```
FileStream fs = File.Open(path, FileMode.Open);          //path 变量为打开文件的路径
```

例如，在打开一个不存的文件时，以读写的方式创建文件并打开。代码如下：

```
FileStream fs = File.Open(path, FileMode. OpenOrCreate);
```

例如，在打开文件时，清空文件中的内容，然后进行读写操作。代码如下：

```
FileStream fs = File.Open(path, FileMode.Truncate);
```

例如，打开文件后将光标移动到文件尾，然后在文件尾进行读写操作。代码如下：

```
FileStream fs = File.Open(path, FileMode.Append);
```

注意：在打开文件时，也可以使用 FileInfo 类实现，其调用的方法基本相同，下面用 FileInfo 类对文件进行写入操作。代码如下：

```
FileInfo fileInfo = new FileInfo("hb.txt");
using (Stream stream = fileInfo.Open(FileMode.Open))        //以读写方式打开文件
{
```

```
        Byte[] info = new UTF8Encoding(true).GetBytes("民以食为天");        //以字符编码 UTF8 设置文本格式
        stream.Write(info, 0, info.Length);                                 //写入文本
    }
```

示例 26-01　以读/写方式打开文件

本示例用 File 类的 Open 方法以不同的方式打开文件，其中包含“读写方式打开”、“追加方式打开”、“清空后打开”和“覆盖方式打开”，然后对其进行写入和读取的操作。

程序代码如下：

代码 26-01　光盘位置：光盘\mingrisoft\sl\26\26_01

在 radioButton1 控件的 CheckedChanged 事件中通过该控件的 Tag 属性判断单选项的选中情况。代码如下：

```
    private void radioButton1_CheckedChanged(object sender, EventArgs e)
    {
        if (((RadioButton)sender).Checked == true)                          //如果单选按钮被选中
        {
            switch (Convert.ToInt32(((RadioButton)sender).Tag.ToString()))  //判断单选项的选中情况
            {
                //记录文件的打开方式
                case 0: fileM = FileMode.Open; break;                       //以读/写方式打开文件
                case 1: fileM = FileMode.Append; break;                     //以追加方式打开文件
                case 2: fileM = FileMode.Truncate; break;                   //打开文件后清空文件内容
                case 3: fileM = FileMode.Create; break;                     //以覆盖方式打开文件
            }
        }
    }
```

说明：在添加 4 个 RadioButton 控件后，将该类控件的 Tag 属性分别设置为 0、1、2、3，用于判断文件的打开方式，然后在 radioButton1 控件的 CheckedChanged 事件获取文件的打开方式，为了节少代码量，将其他 3 个 RadioButton 控件的 CheckedChanged 事件指向 radioButton1 控件的 CheckedChanged 事件。

在“读写操作”按钮的单击事件中，通过用户选择的打开方式打开指定文件，然后对该文件进行读/写操作。代码如下：

```
    FileMode fileM = FileMode.Open;
    private void button1_Click(object sender, EventArgs e)
    {
        string path = textBox1.Text;                                        //获取打开文件的路径
        try
        {
            using (FileStream fs = File.Open(path, fileM))                  //以指定的方式打开文件
            {
                if (fileM != FileMode.Truncate)                             //如果在打开文件后不清空文件
                {
                    Byte[] info = new UTF8Encoding(true).GetBytes(textBox2.Text);//将要添加的内容转换成字节
                    fs.Write(info, 0, info.Length);                         //向文件中写入内容
                }
            }
            using (FileStream fs = File.Open(path, FileMode.Open))          //以读/写方式打开文件
            {
                byte[] b = new byte[1024];                          //定义一个字节数组
                UTF8Encoding temp = new UTF8Encoding(true);                 //实现 UTF-8 编码
                string pp = "";
                while (fs.Read(b, 0, b.Length) > 0)                         //读取文本中的内容
                {
                    pp += temp.GetString(b);                                //累加读取的结果
                }
                MessageBox.Show(pp);                                        //显示文本中的内容
            }
        }
```

```
    catch                                                    //如果文件不存在，则发生异常
    {
        if (MessageBox.Show("该文件不存在，是否创建文件。", "提示", MessageBoxButtons.YesNo) ==
DialogResult.Yes)                                            //显示提示框，判断是否创建文件
        {
            FileStream fs = File.Open(path, FileMode.CreateNew);      //在指定的路径下创建文件
            fs.Dispose();
        }
    }
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 26.1 所示。

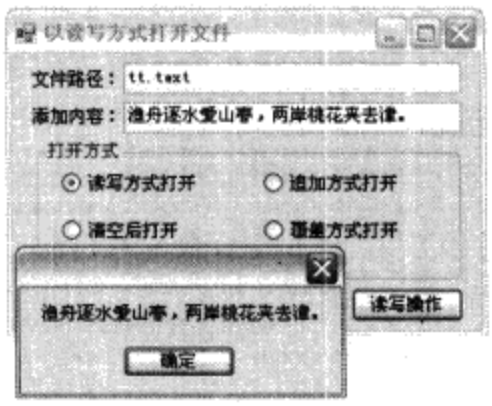


图 26.1　以读/写方式打开文件

注意：在用 File 类和 FileInfo 类创建文本文件时，其默认的字符编码为 UTF-8，而在 Windows 环境中手动创建文本文件时，其字符编码是 ANSI。

2．以只读方式打开文件

以只读方式打开文件是用 File 类的 OpenRead 方法实现的。该方法用于打开现有文件并对其读取。

语法：

```
public static FileStream OpenRead(string path)
```

参数说明如下。

path：string，要打开的文件路径。

返回值：FileStream 类，以只读方式打开指定路径上的 FileStream。

例如，以只读方式打开文件。代码如下：

```
FileStream fs = File. OpenRead(path);
```

在这里要说明的是，如果只想读取 UTF-8 编码文本文件，可以使用 OpenText 方法。

语法：

```
public static StreamReader OpenText(string path)
```

参数说明如下。

path：string，要打开的文件路径。

返回值：StreamReader 类。

例如，读取指定路径下字符编码为 UTF-8 的文本文件的内容。代码如下：

```
using (StreamReader sr = File.OpenText(path))//实例化 StreamReader 类
{
    string s = "";
    while ((s = sr.ReadLine()) != null)                  //如果从当前流中读取的字符串不为空
    {
        MessageBox.Show(s);                              //显示读取的文本内容
    }
}
```

3．以写入方式打开文件

以写入方式打开文件是用 File 类的 OpenWrite 方法实现的。该方法用于打开现有文件以进行写入。

语法：

```
public static FileStream OpenWrite(string path)
```

参数说明如下。

path：string，要打开的文件路径。

返回值：FileStream 类，以只读方式打开指定路径上的 FileStream。

例如，以写入方式打开文件。代码如下：

```
FileStream fs = File. OpenWrite(path);
```

26.2.4 判断文件是否存在

在对文件进行创建、复制、移除、打开和删除操作前，判断文件是否存在是非常必要的，它可以避免对一个不存在的文件进行操作。

1．File 类的 Exists 方法

该方法用于确定指定的文件是否存在。

语法：

```
public static bool Exists(string path)
```

参数说明如下。

path：string，要检查的文件。

返回值：布尔型，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。

例如，根据指定的文件路径，判断文件是否存在。代码如下：

```
string Path = "hb.txt";
if (File.Exists(Path))                          //如果文件存在
{
    MessageBox.Show("文件存在");
}
else//文件不存在
{
    MessageBox.Show("文件不存在");
}
```

注意：如果不指明路径，则默认为应用程序当前路径。

2．FileInfo 类的 Exists 属性

该方法用于确定指定的文件是否存在。

语法：

```
public static bool Exists(string path)
```

参数说明如下。

path：string，要检查的文件。

返回值：布尔型，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。

例如，在删除文件前判断文件是否存在，如果存在则删除当前文件。代码如下：

```
string Path = "E:\工作记录.rar";
FileInfo fileInfo=new FileInfo(Path);                   //实例化 fileInfo 类
if (fileInfo.Exists)                                    //如果文件存在
{
    fileInfo.Delete();                                  //删除当前文件
}
else
{
    MessageBox.Show("文件不存在");
}
```

注意：在使用 Delete 方法删除文件后，不可以使用 Exists 属性（方法）判断当前文件是否存在，因为在删除大文件时，会有一段删除文件时间，所以在用 Exists 判断时，文件虽然存在，但是正在被删除。

26.2.5 复制或移动文件

在日常工作中，文件的复制与移动是经常被操作的，例如制作一个文档管理系统，当执行文件的导入/导出操作，或对文件进行备份时，都要进行文件的复制操作。

1．文件的复制

文件的复制可以用 File 类的 Copy 方法和 FileInfo 类的 CopyTo 方法实现。

（1）File 类的 Copy 方法，将现有文件复制到新的目录下。

语法：

```
public static void Copy(string sourceFileName,string destFileName)
public static void Copy(string sourceFileName,string destFileName,bool overwrite)
```

参数说明如下。

sourceFileName：string，要复制的文件。

destFileName：string，目标文件的名称。不能是目录。

overwrite：bool，如果可以覆盖目标文件，则为 true；否则为 false。

例如，将 C 盘下的 hb.txt 文件复制到 D 盘下。代码如下：

```
File.Copy("C:\\hb.txt","D:\\hb.txt");
```

（2）FileInfo 类的 CopyTo 方法，将现有文件复制到新的目录下。

语法：

```
public static void CopyTo(string destFileName)
public static void CopyTo(string destFileName,bool overwrite)
```

参数说明如下。

destFileName：string，目标文件的名称。不能是目录。

overwrite：bool，如果可以覆盖目标文件，则为 true；否则为 false。

例如，将 C 盘下的 hb.txt 文件复制到 D 盘下，并覆盖已有文件。代码如下：

```
FileInfo fileInfo = new FileInfo("C:\\hb.txt");
fileInfo.CopyTo("D:\\hb.txt",true);
```

2．文件的移动

文件的移动可以用 File 类的 Move 方法和 FileInfo 类的 MoveTo 方法实现。

（1）File 类的 Move 方法，将现有文件移动到新的目录下。

语法：

```
public static void Move(string sourceFileName, string destFileName);
```

参数说明如下。

sourceFileName：要移动的文件。

destFileName：目标文件的名称。不能是目录。

例如，将 C 盘下的 hb.txt 文件移动到 D 盘下。代码如下：

```
File. Move("C:\\hb.txt","D:\\hb.txt");
```

例如，将 C 盘下 hb.txt 文件的名称改为 pp.txt。代码如下：

```
File. Move("C:\\hb.txt","C:\\pp.txt");
```

（2）FileInfo 类的 MoveTo 方法，将现有文件移动到新的目录下。

语法：

```
public static void MoveTo(string destFileName);
```

参数说明如下。

destFileName：目标文件的名称。不能是目录。

例如，将 C 盘下的 hb.txt 文件移动到 D 盘下。代码如下：

```
FileInfo fileInfo = new FileInfo("C:\\hb.txt");
fileInfo. MoveTo("D:\\hb.txt");
```

注意：当用户使用 Move(MoveTo)方法改写现有文件时（原文件和目的文件是同一个文件），将产生 IOException 异常。Move(MoveTo)方法和 Copy(CopyTo)方法都支持相对路径。

示例 26-02　批量文件的复制与移动

在对文件进行复制和移动操作时，一般都是对多个文件进行操作。本示例将不同路径下的

多个文件复制或移动到指定的目录下。运行结果如图 26.2 所示。

图 26.2 批量文件的复制与移动

程序代码如下：

代码 26-02 光盘位置：光盘\mingrisoft\sl\26\26_02

在窗体的加载事件中，设置记录要进行操作的 ListView 控件，在该控件中添加两个列，列名分别为“文件路径”和“文件名”。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    listView1.GridLines = true;                                          //在各数据之间形成网格线
    listView1.View = View.Details;                                       //显示列名称
    listView1.FullRowSelect = true;                                      //在单击某项时，对其进行选中
    listView1.HeaderStyle = ColumnHeaderStyle.Nonclickable;              //隐藏列标题
    listView1.Columns.Add("文件路径", listView1.Width - 100, HorizontalAlignment.Left);  //添加列标题
    listView1.Columns.Add("文件名", 95, HorizontalAlignment.Left);          //添加列标题
    radioButton1.Checked = true;                                         //指定当前是复制操作
}
```

在“添加”按钮的单击事件中，调用“打开”对话框，将选中文件的路径和文件名添加到 ListView 控件上。代码如下：

```
string FileN = "";                          //记录要操作的文件路径
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog openFile = new OpenFileDialog();              //实例化 OpenFileDialog 类
    if (openFile.ShowDialog() == DialogResult.OK)                //打开”对话框
    {
        FileN = openFile.FileName;                               //获取选中文件的路径
        ListViewItem item = new ListViewItem(FileN);             //实例化 ListViewItem 类
        //将文件名添加到指定的列上
        item.SubItems.Add(FileN.Substring(FileN.LastIndexOf('\\') + 1, FileN.Length – FileN.LastIndexOf('\\') - 1));
        listView1.Items.Add(item);                               //添加行信息
    }
}
```

在“..”按钮的单击事件中，调用“浏览文件夹”对话框，选择要进行复制或移动的目的路径。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    FolderBrowserDialog folderDialog = new FolderBrowserDialog();      //实例化 FolderBrowserDialog 类
    if (folderDialog.ShowDialog() == DialogResult.OK)                  //打开“浏览文件夹”对话框
    {
        textBox1.Text = folderDialog.SelectedPath;                     //获取指定的目录
    }
}
```

在“执行”按钮的单击事件中，根据单选项“复制”和“移动”，将文件进行批量操作。代码如下：

```
private void button3_Click(object sender, EventArgs e)
{
    int n = 0;                                                    //标识，判断是复制，还是移动
    if (radioButton1.Checked)                                     //如果当前是复制操作
        n = 1;
    if (radioButton2.Checked)                                     //如果当前是移动操作
        n = 2;
    string pp = radioButton1.Checked ? radioButton1.Text : radioButton2.Text;     //记录操作类型
    for (int i = 0; i < listView1.Items.Count; i++)               //遍历 listView1 控件中所有记录
    {
        //执行文件的复制或移动操作
        FileCopyAndMove(listView1.Items[i].SubItems[0].Text, textBox1.Text, n);
    }
    MessageBox.Show("文件" + pp + "完成");
}
```

自定义方法 FileCopyAndMove 对单个文件进行复制或移动的操作。参数 sourceFileName 表示要复制或移动的文路径；参数 destFileName 表示目的文件的路径；参数 n 表示当前操作是复制还是移动。代码如下：

```
public void FileCopyAndMove(string sourceFileName, string destFileName, int n)
{
    FileInfo fileInfo = new FileInfo(sourceFileName);             //实例化 FileInfo 类
    if (n == 1)                                                   //如果是复制操作
        fileInfo.CopyTo(destFileName + "\\" + fileInfo.Name);
    if (n == 2)                                                   //如果是移动操作
        fileInfo.MoveTo(destFileName + "\\" + fileInfo.Name);
}
```

注意：以上示例也可以用 File 类完成文件的批量复制与移动。

26.2.6 删除文件

文件的删除可以用 File 类或 FileInfo 类的 Delete 方法来实现。下面主要对 File 类进行讲解。

File 类的 Delete 方法用于删除指定的文件。如果文件不存在，则不引发异常。

语法：

```
public static void Delete(string path)
```

参数说明如下。

path string，要删除的文件的名称。

例如，删除 D 盘下的 hb.txt 文件。代码如下：

```
File.Delete("D:\\hb.txt");
```

在这里要说明的是 FileInfo 类的 Delete 方法是一个无参数方法，删除 D 盘下 hb.txt 文件的代码如下：

```
FileInfo fileInfo = new FileInfo("D:\\hb.txt");          //实例化 FileInfo 类
fileInfo.Delete();
```

26.2.7 获取文件基本信息

获取文件的基本信息主要是用 File 类或 FileInfo 类来实现的，本节主要讲解用 FileInfo 类获取文件的基本信息，如扩展名、创建日期、大小等。

1．获取文件的名称及扩展名

（1）Name 属性，获取文件名。

语法：

```
public override string Name { get; }
```

属性值：string，文件名。

例如，获取指定文件的扩展名。代码如下：

```
FileInfo aFile = new FileInfo ("D:\\hb.txt");
string tem_Name = aFile.Name;
```

以上代码中 tem_Ext 变量的值为 hb.txt

（2）Extension 属性，获取表示文件扩展名部分的字符串。

语法：

```
public string Extension { get; }
```

属性值：string，包含 FileSystemInfo 扩展名的字符串。

例如，获取指定文件的扩展名。代码如下：

```
FileInfo aFile = new FileInfo ("D:\\hb.txt");
string tem_Ext = aFile. Extension;
```

以上代码中 tem_Ext 变量的值为.txt

2．获取文件的大小

Length 属性用于获取当前文件的大小（字节）。

语法：

```
public long Length { get; }
```

属性值：Int64，当前文件的大小（字节）。

例如，获取文件的大小。代码如下：

```
FileInfo aFile = new FileInfo("D:\\hb.txt");
Int64 tem_Length = aFile.Length;
```

3．获取文件的创建时间和修改时间

（1）CreationTime 属性，获取或设置当前 FileSystemInfo 对象的创建时间。

语法：

```
public DateTime CreationTime { get; set; }
```

属性值：DateTime，当前 FileSystemInfo 对象的创建日期和时间。

例如，获取文件的创建日期。代码如下：

```
FileInfo aFile = new FileInfo("D:\\hb.txt");
DateTime tem_CreationTime = aFile.CreationTime;
```

（2）LastAccessTime 属性，获取或设置上次访问当前文件或目录的时间。

语法：

```
public DateTime LastWriteTime { get; set; }
```

属性值：DateTime，上次写入当前文件的时间。

例如，获取文件的修改日期。代码如下：

```
FileInfo aFile = new FileInfo("D:\\hb.txt");
DateTime tem_WriteTime = aFile.LastWriteTime;
```

4．获取文件的完整路径

FullName 属性用于获取文件的完整路径。

语法：

```
public virtual string FullName { get; }
```

属性值：string，包含完整目录的字符串。

例如，获取文件的完整路径（包括文件名及扩展名）。代码如下：

```
FileInfo aFile = new FileInfo("D:\\hb.txt");
DateTime tem_Path = aFile.FullName;
```

文件的基本信息在网络通信中是非常有用的，例如，制作一个局域网的文档审核系统，可以根据当前文件的扩展名及修改时间判断是否要提交文档，然后通过文件的大小，对当前文件以流的方式进行网络传递。

26.3 文件夹的基本操作

26.3.1 文件夹操作类 Directory 和 DirectoryInfo

Directory 类和 DirectoryInfo 类都可以对文件进行创建、移动、浏览目录及其子目录，下面对这两个类进行简单介绍。

1．Directory 类

Directory 类用于典型操作，如复制、移动、重命名、创建和删除目录。也可将 Directory 类用于获取和设置与目录的创建、访问及写入操作相关的 DateTime 信息。

Directory 类的常用方法及说明如表 26.6 所示。

表 26.6　　Directory 类的常用方法及说明

方　法	说　明
CreateDirectory	创建指定路径中的所有目录
Delete	删除指定的目录
Exists	确定给定路径是否引用磁盘上的现有目录
GetCreationTime	获取目录的创建日期和时间
GetCurrentDirectory	获取应用程序的当前工作目录
GetDirectories	获取指定目录中子目录的名称
GetFiles	返回指定目录中的文件的名称
GetFileSystemEntries	返回指定目录中所有文件和子目录的名称
GetLastAccessTime	返回上次访问指定文件或目录的日期和时间
GetLastWriteTime	返回上次写入指定文件或目录的日期和时间
GetLogicalDrives	检索此计算机上格式为“<驱动器号>:\”的逻辑驱动器的名称
GetParent	检索指定路径的父目录，包括绝对路径和相对路径
Move	将文件或目录及其内容移到新位置
SetCreationTime	为指定的文件或目录设置创建日期和时间
SetCurrentDirectory	将应用程序的当前工作目录设置为指定的目录
SetLastAccessTime	设置上次访问指定文件或目录的日期和时间
SetLastWriteTime	设置上次写入目录的日期和时间

2．DirectoryInfo 类

DirectoryInfo 类和 Directory 类之间的关系与 FileInfo 类和 File 类之间的关系十分类似。这里不再赘述。下面介绍 DirectoryInfo 类的常用属性，如表 26.7 所示。

表 26.7　　DirectoryInfo 类的常用属性及说明

属　性	说　明
Attributes	获取或设置当前 Filesysteminfo 的 Fileattributes
CreationTime	获取或设置当前 FileSystemInfo 对象的创建时间
Exists	获取指示目录是否存在的值
FullName	获取目录或文件的完整目录
Parent	获取指定子目录的父目录
Name	获取此 DirectoryInfo 实例的名称

交叉链接：DirectoryInfo 类所使用的相关方法请参见表 26.6 所示。

26.3.2 创建文件夹

1．用 Directory 类

CreateDirectory 方法用于创建指定目录下的文件夹和子文件夹。

语法：

```
public static DirectoryInfo CreateDirectory(string path)
```

参数说明如下。

path：string，要创建的目录路径。

返回值：path 指定的 DirectoryInfo。

例如，在 E 盘的根目录下创建 soft 文件夹。代码如下：

```
Directory. CreateDirectory ("E:\\soft ");
```

2．用 DirectoryInfo 类

Create 方法用于创建目录。

语法：

```
public void Create()
```

注意：如果目录已经存在，则此方法不执行任何操作。

在用 DirectoryInfo 类的 Create 方法创建文件夹前，首先通过要创建文件夹的路径实例化 DirectoryInfo 类，然后再通过 Create 方法进行创建。

例如，在 E 盘的根目录下创建 soft 文件夹。代码如下：

```
DirectoryInfo dirInfo = new DirectoryInfo("E:\\soft");
dirInfo.Create();
```

26.3.3 判断文件夹是否存在

Exists 方法确定给定路径是否引用磁盘上的现有目录。

语法：

```
public static bool Exists(string path)
```

参数说明如下。

path：string，要测试的路径。

返回值：bool，如果 path 引用现有目录，则为 true；否则为 false。

例如，判断 E 盘下是否有文件夹 soft，如果没有则创建。代码如下：

```
if (Directory.Exists("E:\\soft"))                    //如果文件夹存在
{
    MessageBox.Show("文件夹存在");
}
else
{
    Directory.CreateDirectory ("E:\\soft ");         //新建一个文件夹
}
```

这里要说明的是通过 DirectoryInfo 类的 Exists 属性也可以判断文件夹是否存在。例如：

```
DirectoryInfo dirInfo = new DirectoryInfo("E:\\soft");
if (dirInfo.Exists)                                  //如果文件夹存在
{
    MessageBox.Show("文件夹存在");
}
else
{
    dirInfo.Create();                                //新建一个文件夹
}
```

26.3.4 移动文件夹

Move 方法用于将文件或目录及其内容移到新位置。

语法：

```
public static void Move(string sourceDirName,string destDirName)
```

参数说明如下。

sourceDirName：string，要移动的文件或目录的路径。

DestDirName：string，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。

例如，将 D 盘根目录下的 soft 文件夹移到 D:\mr\sl 下。代码如下：

```
Directory.Move("D:\\soft", "D:\\mr\\sl");
```

例如，将 C 盘根目录下 soft 文件夹的名称改为 mr。代码如下：

```
Directory.Move("C:\\soft", "C:\\mr");
```

下面是 Move 方法的错误用法，请读者注意。

例如，在不同的驱动器下进行移动。代码如下：

```
Directory.Move("C:\\soft", "D:\\mr");
```

例如，源文件夹和目的文件夹的路径和名称不能相同。代码如下：

```
Directory.Move("C:\\soft", " C:\\soft");
```

这里要说明的是通过 DirectoryInfo 类的 MoveTo 方法也可以对文件夹进行移动。例如：

```
DirectoryInfo dirInfo = new DirectoryInfo("D:\\soft");
dirInfo.MoveTo( "D:\\mr\\sl");
```

26.3.5 删除文件夹

Delete 方法用于删除指定的文件夹。它有两种形式：一是删除空文件夹，二是删除文件夹下的所有子文件夹和文件。

语法 1:

```
public static void Delete(string path)
```

语法 2

```
public static void Delete(string path,bool recursive)
```

参数说明如下。

path：string，要移除的目录的名称。

recursive：bool，若要移除 path 中的目录、子目录和文件，则为 true；否则为 false。

注意：如要用语法 1 删除一个非空的文件夹，会出现异常。

例如，删除 D 盘下的空文件夹 soft。代码如下：

```
Directory.Delete("D:\\soft") ;
```

例如，删除 D 盘下的文件夹 mr，并删除该文件夹下的所有子文件夹及文件。代码如下：

```
Directory.Delete("D:\\mr",True) ;
```

这里要说明的是通过 DirectoryInfo 类的 Delete 方法也可以对文件夹进行删除。其语法格式与 Directory 类的 Delete 方法基本相同。

例如，删除 D 盘下的空文件夹 soft。代码如下：

```
DirectoryInfo dirInfo = new DirectoryInfo("D:\\soft");
dirInfo.Delete();
```

例如，删除 D 盘下的文件夹 mr，并删除该文件夹下的所有子文件夹及文件。代码如下：

```
DirectoryInfo dirInfo = new DirectoryInfo("D:\\soft");
dirInfo.Delete(true);
```

26.3.6 遍历文件夹中的子文件夹及文件

1．遍历文件夹中的子文件夹

（1）Directory 类的 GetDirectories 方法。

该方法用于获取指定目录中所有文件夹的名称，也可以通过条件搜索匹配的文件夹。

语法:

```
public static string[] GetDirectories(string path)
public static string[] GetDirectories(string path,string searchPattern)
```

参数说明如下。

path：string，为其返回子目录名称的数组的路径。

searchPattern：string，要与 path 中的文件夹名称匹配的搜索字符串。此参数不能以两个句点（“..”）结束，不能在 DirectorySeparatorChar 或 AltDirectorySeparatorChar 的前面包含两个句点（“..”），也不能包含 InvalidPathChars 中的任何字符。

交叉链接：关于 searchPattern 参数的应用，可以参见下面 DirectoryInfo 类 GetDirectories 方法。

返回值：一个 String 类型的数组，它包含 path 中子目录的名称。

例如，获取指定路径下的文件夹名称。代码如下：

```
string DirAllName = "";                                    //记录所有文件夹的名称
string DirName = "";                                       //记录单个文件夹的名称
string[] DirEntries = Directory.GetDirectories(textBox1.Text);      //获取指定文件夹下的所有子文件夹路径
foreach (string DName in DirEntries)                       //遍历子文件夹
{
    //获取子文件夹的名称
    DirName = DName.Substring(DName.LastIndexOf('\\') + 1, DName.Length - DName.LastIndexOf('\\') - 1);
    DirAllName += DirName + "\n";                          //以换行的形式记录子文件夹的名称
}
```

在以上代码的 DirAllName 变量中记录了指定目录下的所有文件夹名称。

（2）DirectoryInfo 类 GetDirectories 方法。

该方法不但可以获取指定目录下的所有文件夹，还可以通过条件搜索匹配的文件夹。

语法 1：

```
public DirectoryInfo[] GetDirectories()
```

返回值：DirectoryInfo 类型的数组。

语法 2：

```
public DirectoryInfo[] GetDirectories(string searchPattern)
```

参数说明如下。

searchPattern：string，搜索字符串。如，搜索所有以单词“2007”开头的目录，其条件设置为“2007 * ”。

返回值：与 searchPattern 匹配的 DirectoryInfo 类型的数组。

例如，获取指定路径下的文件夹名称。代码如下：

```
string DirAllName = "";                                    //记录所有文件夹的名称
DirectoryInfo DirInfo = new DirectoryInfo(textBox1.Text);          //实例化 DirectoryInfo 类
DirectoryInfo[] DirInfos = DirInfo.GetDirectories();               //获取指定目录下的文件夹
foreach (DirectoryInfo Dir in DirInfos)                            //遍历文件夹数组
{
    DirAllName += Dir.Name + "\n";                                 //以换行的形式记录子文件夹的名称
}
```

在以上代码的 DirAllName 变量中记录了指定目录下的所有文件夹名称。

如果想要在指定的目录下根据指定的条件搜索匹配文件夹，可以将以上代码的加粗部分进行修改。如，查询文件夹的名称前缀带有“2007”的字符串。代码如下：

```
DirectoryInfo[] DirInfos = DirInfo.GetDirectories("2007*");      //获取指定目录下的文件夹
```

根据指定条件搜索文件夹在日常工作中是十分重要的，例如，在制作一个工作反馈系统时，员工的工作记录都存储在以当天的年月日命名的文件夹下，如果要对某一月工作记录或某一年工作记录进行审核，就要根据指定的日期提取符合条件的文件夹。

2．遍历文件夹中的文件

（1）Directory 类的 GetFile 方法。

该方法用于获取指定目录中所有文件的名称，也可以通过条件搜索匹配的文件。

语法：

```
public static string[] GetFiles(string path)
public static string[] GetFiles(string path,string searchPattern)
```

参数说明如下。

path：string，要搜索的目录。

searchPattern：string，要与 path 中的文件名匹配的搜索字符串。此参数不能以两个句点（“..”）结束，不能在 DirectorySeparatorChar 或 AltDirectorySeparatorChar 的前面包含两个句点（“..”），也不能包含 InvalidPathChars 中的任何字符。

交叉链接：关于 searchPattern 参数的应用，可以参见下面的 DirectoryInfo 类 GetFile 方法。

返回值：一个 String 类型的数组，它包含 path 中文件的名称。

例如，获取指定目录下的文件名称。代码如下：

```
string FileNames = "";                                          //记录所有文件的名称
string fileName = "";                                           //记录单个文件的名称
string[] fileEntries = Directory.GetFiles(textBox1.Text);       //获取指定目录下所有文件的完整目录
foreach (string FName in fileEntries)                           //遍历文件
{
    //获取文件的名称
    fileName = FName.Substring(FName.LastIndexOf('\\') + 1, FName.Length - FName.LastIndexOf('\\') - 1);
    FileNames += fileName + "\n";                               //以换行的形式记录文件的名称
}
```

在以上代码的 FileNames 变量中记录了指定目录下的所有文件名称。

（2）DirectoryInfo 类的 GetFile 方法。

该方法不但可以获取指定目录下的所有文件，还可以通过条件搜索匹配的文件。

语法 1：

```
public FileInfo[] GetFiles()
```

返回值：FileInfo 类型的数组。

语法 2：

```
public FileInfo[] GetFiles(string searchPattern)
```

参数说明如下。

searchPattern：string，搜索字符串。如，搜索指定目录下的 TXT 文件，其条件设置为“*.txt”。

返回值：FileInfo 类型的数组。

例如，获取指定目录下的文件名称。代码如下：

```
string FileAllName = "";                                        //记录所有文件的名称
DirectoryInfo FileInfo = new DirectoryInfo(textBox1.Text);      //实例化 DirectoryInfo 类
FileInfo[] fileInfos = FileInfo.GetFiles();                     //获取指定目录下的文件
foreach (FileInfo file in fileInfos)                            //遍历文件数组
{
    FileAllName += file.Name + "\n";                            //以换行的形式记录文件的名称
}
```

如果想要在指定的目录下根据指定的条件搜索匹配文件，可以将以上代码的加粗部分进行修改。如，搜索指定目录下的 Word 文档。代码如下：

```
FileInfo[] fileInfos = FileInfo.GetFiles("*.DOC");              //获取指定目录下的文件
```

注意：在文件夹中遍历子文件夹和文件时，笔者建议读者使用 DirectoryInfo 类。

实例位置：光盘\mingrisoft\dxyy\26\26_01

本实例主要遍历指定驱动器下的所有文件夹及文件的名称。运行结果如图 26.3 所示。

程序开发步骤如下。

（1）在 VS2008 中新建一个 Windows 应用程序项目，并命名为 CatalogShow。

（2）打开 Windows 窗体设计器，从工具箱中为窗体添加一个 Label 控件、一个 ComboBox

控件和一个 TreeView 控件。

图 26.3　遍历驱动器中的文件及文件夹

（3）在窗体的加载事件中，将计算机上的逻辑驱动器的名称添加到 comboBox1 控件中。代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    string[] dirs = Directory.GetLogicalDrives();                    //获取计算上的逻辑驱动器的名称
    if (dirs.Length > 0)                                             //如果有驱动器
    {
        for (int i = 0; i < dirs.Length; i++)                        //遍历驱动器
        {
            comboBox1.Items.Add(dirs[i]);                            //将驱动名称添加到下拉项中
        }
    }
}
```

（4）在 comboBox1 控件的 SelectedValueChanged 事件中，当改变 comboBox1 控件的选项值时，将调用自定义方法 Folder_List 获取指定驱动器下的所有文件夹及文件。代码如下：

```
private void comboBox1_SelectedValueChanged(object sender, EventArgs e)
{
    if (((ComboBox)sender).Text.Length > 0)                          //如果在下拉项中选择了值
    {
        treeView1.Nodes.Clear();                                     //清空 treeView1 控件
        TreeNode TNode = new TreeNode();                             //实例化 TreeNode
        //将驱动器下的文件夹及文件名称添加到 treeView1 控件上
        Folder_List(treeView1, ((ComboBox)sender).Text, TNode, 0);
    }
}
```

（5）自定义方法 Folder_List 用于显示指定驱动器或文件夹下的所有子文件夹及文件的名称。参数 TV 表示 TreeView 控件，用于显示驱动器下的文件夹及文件；参数 Sdir 表示要遍历的目录；参数 n 表示标识，可以用该标识判断当前操作的是文件夹，还是文件。代码如下：

```
private void Folder_List(TreeView TV, string Sdir, TreeNode TNode, int n)
{
    if (TNode.Nodes.Count > 0)                                       //如果当前节点下有子节点
        if (TNode.Nodes[0].Text != "")                               //如果第一个子节点的文本为空
            return;                                                  //退出本次操作
    if (TNode.Text == "")                                            //如果当前节点的文本为空
        Sdir += "\\";                                                //设置驱动器的根路径
    DirectoryInfo dir = new DirectoryInfo(Sdir);                     //实例化 DirectoryInfo 类
    try
    {
        if (!dir.Exists)                                             //判断文件夹是否存在
        {
            return;
        }
        DirectoryInfo dirD = dir as DirectoryInfo;                   //如果给定参数不是文件夹则退出
```

```
if (dirD == null)                                   //如果文件夹是否为空
{
    TNode.Nodes.Clear();                            //清空当前节点
    return;
}
else
{
    if (n == 0)                                     //如果当前是文件夹
    {
        if (TNode.Text == "")                       //如果当前节点为空
            TNode = TV.Nodes.Add(dirD.Name);            //添加文件夹的名称
        else
        {
            TNode.Nodes.Clear();                    //清空当前节点
        }
        TNode.Tag = 0;                              //设置文件夹的标识
    }
}
FileSystemInfo[] files = dirD.GetFileSystemInfos();         //获取文件夹中所有文件和文件夹
//对单个 FileSystemInfo 进行判断,遍历文件和文件夹
foreach (FileSystemInfo FSys in files)
{
    FileInfo file = FSys as FileInfo;               //实例化 FileInfo 类
    if (file != null)                               //如果是文件的话，将文件名添加到节点下
    {
        FileInfo SFInfo = new FileInfo(file.DirectoryName + "\\" + file.Name);      //获取文件所在路径
        TNode.Nodes.Add(file.Name);                 //添加文件名
        TNode.Tag = 0;                              //设置文件标识
    }
    else                                            //如果是文件夹
    {
        TreeNode TemNode = TNode.Nodes.Add(FSys.Name);                  //添加文件夹名称
        TNode.Tag = 1;                              //设置文件夹标识
        //在该文件夹的节点下清加一个空文件夹，表示文件夹下有子文件夹或文件
        TemNode.Nodes.Add("");
    }
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
    return;
}
}
```

（6）在 treeView1 控件的 NodeMouseDoubleClick 事件中，当用户双击文件夹节点时，调用自定义方法 Folder_List 添加该文件夹下的所有子文件夹及文件。代码如下：

```
private void treeView1_NodeMouseDoubleClick(object sender, TreeNodeMouseClickEventArgs e)
{
    if (((TreeView)sender).SelectedNode == null)            //如当前节点为空
        return;
    Folder_List(treeView1, ((TreeView)sender).SelectedNode.FullPath.Replace("\\\\", "\\"),
((TreeView)sender).SelectedNode, 0);  //将指定目录下的文件夹及文件名称清加到 treeView1 控件的指定节点下
}
```

26.4 文件流的操作

26.4.1 文本文件的读取

文件的读操作是文件最重要的操作之一，System.IO 命名空间提供了许多文件读操作类，对文件进行操作的常见方式有两种：文本模式和二进制模式。

1．文件模式

以文本模式对文件进行读操作一般用 StreamWriter 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。

（1）实例化 StreamReader 类。

在使用 StreamReader 类前，首先要对 StreamReader 类进行实例化，然后才能用该类进行读取操作。下面对 StreamReader 类的实例化语法进行说明。

语法 1:

```
public StreamReader(string path)
```

参数说明如下。

path：string，要向其中写入的完整文件路径。path 可以是文件名。

例如，根据文本文件的路径实例化 StreamReader 类。代码如下：

```
StreamReader sw = new StreamReader("C:\hb.txt")
```

语法 2:

```
public StreamReader(Stream stream,Encoding encoding)
```

参数说明如下。

stream：string，要读取的流。

Encoding：Encoding 类，要使用的字符编码。如果省略该参数，则用 UTF-8 编码读取流。Encoding 类的常用字符编码及说明如表 26.8 所示。

表 26.8　　Encoding 类的常用字符编码及说明

字符编码	说　明
Default	操作系统的当前 ANSI 代码页的编码
BigEndianUnicode	使用 Big-Endian 字节顺序的 UTF-16 格式的编码
Unicode	使用 Little-Endian 字节顺序的 UTF-16 格式的编码
UTF8	UTF-8 格式的编码

例如，用指定的编码及默认缓冲区大小，为指定的流初始化 StreamWriter 类的新实例。代码如下：

```
StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead("C:\hb.txt"), var_Encoding);
```

（2）StreamReader 类的常用属性。

EndOfStream 属性用于获取一个值，该值表示当前的流位置是否在流的末尾。

语法:

```
public bool EndOfStream { get; }
```

属性值：bool，如果当前的流位置在流的末尾，则为 true，否则为 false。

例如，在读取文件时，判断流的位置是否在流的尾部。代码如下：

```
using (StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead(textBox1.Text),
System.Text.Encoding.Default))                    //以只读方式打开字符编码为 ANSI 的文本文件
{
    string pp = "";
    while (!strRead.EndOfStream)                  //如果没有读取到文本尾
    {
        pp += strRead.ReadLine();                 //累加读取的结果
    }
    richTextBox1.Text = pp;                       //显示读取后的信息
}
```

（3）　StreamReader 类的常用方法。

StreamReader 类的常用方法及说明如表 26.9 所示。

表 26.9 **StreamReader 类的常用方法及说明**

方　法	说　明
Close	已重写。关闭 StringReader
CreateObjRef	创建一个对象，该对象包含生成用于与远程对象进行通信的代理所需的全部相关信息（从 Marshal ByRefObject 继承）
Dispose	已重载
Equals	确定两个 Object 实例是否相等
GetHashCode	用作特定类型的哈希函数。GetHashCode 适合在哈希算法和数据结构（如哈希表）中使用
GetLifetimeService	检索控制此实例的生存期策略的当前生存期服务对象
GetType	获取当前实例的 Type
InitializeLifetimeService	获取控制此实例的生存期策略的生存期服务对象
Peek	返回下一个可用的字符，但不使用它
Read	读取输入字符串中的下一个字符或下一组字符
ReadBlock	从当前流中读取最大 count 的字符并从 index 开始将该数据写入 buffer(从 TextReader 继承)
ReadLine	从基础字符串中读取一行
ReadToEnd	将整个流或从流的当前位置到流的结尾作为字符串读取
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
Synchronized	在指定 TextReader 周围创建线程安全包装 （从 TextReader 继承）
ToString	返回表示当前 Object 的 String

下面对 StreamReader 类中比较重要的方法进行详细介绍。

① Close 方法。关闭 StreamReader 对象和基础流，并释放与读取器关联的所有系统资源。

语法：

```
public override void Close ()
```

在创建和使用完一个 StreamReader 后一定要将其及时关闭。

② ReadLine 方法。从当前流中读取一行字符并将数据作为字符串返回。

语法：

```
public override string ReadLine ()
```

返回值：输入流中的下一行；如果到达了输入流的末尾，则为空引用。

StreamReader 类对象有两个重要的方法 Read 和 ReadLine，分别用来从文本文件中读取字符串和读取一行字符串。下面的示例实现了通过 StreamReader 对象从文本文件 TestFile.txt 中读取字符串的功能。程序代码如下：

```
using (StreamReader sr = new StreamReader("TestFile.txt"))
{
    String line;
     while ((line = sr.ReadLine()) != null)
     {
         Console.WriteLine(line);
     }
}
```

注意：使用 StreamReader 时，应在使用完毕时调用 Close 方法将其关闭。在实例化 StreamReader 类对象时使用 using 语句可以使对象在使用完毕后自动释放，因此可以省去调用 Close 方法。

③ Peek 方法，返回下一个可用的字符，但不使用它。

语法：

```
public override int Peek()
```

返回值：Int32，一个表示下一个要读取的字符的整数。如果没有更多可读取的字符或该流不支持查找，则为-1。

示例 26-03 以只读方式读取文件内容

在读取具有中文信息的文本文件时，有时会出现乱码，这是因为该文件的字符编码为 ANSI。本示例将用 StreamReader 类读取字符编码为 ANSI 的文本文件，这样可以读取任意字符编码的文本文件。

程序代码如下：

代码 26-03 光盘位置：光盘\mingrisoft\sl\26\26_03

```
private void button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Length == 0)                    //如果没有要打开的文件路径
        return;                                       //退出当前操作
    using (StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead(textBox1.Text),
System.Text.Encoding.Default))                        //以只读方式打开字符编码为 ANSI 的文本文件
    {
        string pp = "";
        while (strRead.Peek() > -1)                   //读取文本中的内容
        {
            pp += strRead.ReadLine();                 //累加读取的结果
        }
        richTextBox1.Text = pp;
    }
}
```

注意：在以上代码中，也可以将加粗代码替换成以下语句来实读取编码类型为 ANSI 的文本文件。

```
StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead(textBox1.Text),
System.Text.Encoding.GetEncoding("gb2312"))
```

按<Ctrl+F5>组合键运行程序，运行结果如图 26.4 所示。

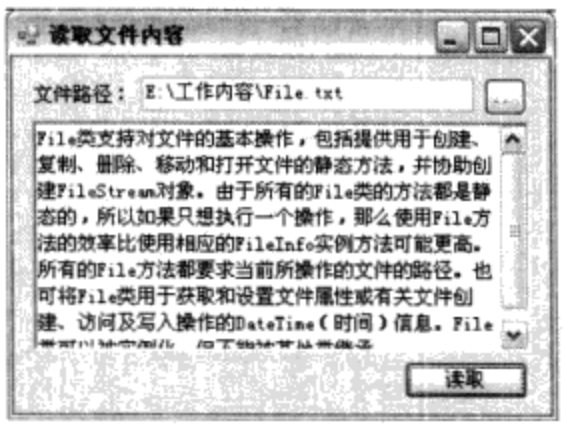


图 26.4 以只读方式读取文件内容

2．二进制模式

System.IO 还提供了 BinaryReader 类用于按二进制来读取文件。它提供的一些方法和 BinaryWriter 是对称的。BinaryReader 用特定的编码将基元数据类型读作二进制值。

下面通过一个示例说明 BinaryReader 的用法，首先调用 26.6.2 节中 BinaryWeiter 的示例将字符串写入一个二进制文件，然后调用下面的方法将内容读出。代码如下：

```
public void ReadText(string path)
{
    BinaryReader binReader = new BinaryReader(File.Open(path, FileMode.Open));   //以二进制读写文件
    char[] c = new char[10];                                                     //定义字符数组
    c = binReader.ReadChars(10);                                                 //定义数组的大小
    for (int i = 0; i < 10; i++)                                                 //读取数组中的信息
    {
        MessageBox.Show(c[i].ToString());                                        //显示单个字符
    }
}
```

26.4.2 文本文件的写入

文件的写操作是文件最重要的操作之一，System.IO 命名空间提供了许多文件写操作类，对文件进行操作的常见方式有两种：文本模式和二进制模式。

1．文本模式

以文本模式对文件进行写操作一般用 StreamWriter 类实现，该类是专门用来处理文本文件的类，可以方便地向文本文件中写入字符串。同时也负责重要的转换和处理向 FileStream 对象

写入的工作。

（1）实例化 StreamWriter 类。

在使用 StreamWriter 类前，首先要对 StreamWriter 类进行实例化，然后对能用该类进行写入操作。下面对 StreamWriter 类的实例化语法进行说明。

语法 1:

```
public StreamWriter(string path)
```

参数说明如下。

path：string，要向其中写入的完整文件路径。path 可以是文件名。

例如，根据文本文件的路径实例化 StreamWriter 类。代码如下：

```
StreamWriter sw = new StreamWriter("C:\hb.txt")
```

语法 2:

```
public StreamWriter(Stream stream,Encoding encoding)
```

参数说明如下。

stream：string，要写入的流。

Encoding：Encoding 类，要使用的字符编码。如果省略该参数，则用 UTF-8 编码写入流。

例如，用指定的编码及默认缓冲区大小，为指定的流初始化 StreamWriter 类的新实例。代码如下：

```
StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead("C:\hb.txt"), var_Encoding);
```

（2）StreamWriter 类的常用属性。

StreamWriter 类的常用属性及说明如表 26.10 所示。

表 26.10　　StreamWriter 类的常用属性及说明

属　性	说　明
Encoding	获取将输出写入到其中的 Encoding
FormatProvider	获取控制格式设置的对象（从 TextWriter 继承）
NewLine	获取或设置由当前 TextWriter 使用的行结束符字符串（从 TEXTWRITER 继承）

Encoding 属性用于获取将输出写入到其中的 Encoding。

语法:

```
public override Encoding Encoding { get; }
```

属性值：用来写入输出的 Encoding。

（3）StreamWriter 类的常用方法。

StreamWriter 类的常用方法及说明如表 26.11 所示。

表 26.11　　StreamWriter 类的常用方法及说明

方　法	说　明
Close	关闭当前的 StringWriter 和基础流
CreateObjRef	创建一个对象，该对象包含生成用于与远程对象进行通信的代理所需的全部相关信息（从 MarshalByRefObject 继承）
Dispose	已重载
Equals	确定两个 Object 实例是否相等
Flush	清理当前编写器的所有缓冲区，使所有缓冲数据写入基础设备（从 TextWriter 继承）
GetHashCode	用作特定类型的哈希函数。GetHashCode 适合在哈希算法和数据结构（如哈希表）中使用
GetLifetimeService	检索控制此实例的生存期策略的当前生存期服务对象
GetStringBuilder	返回基础 StringBuilder
GetType	获取当前实例的 Type

续表

方　法	说　明
InitializeLifetimeService	获取控制此实例的生存期策略的生存期服务对象
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
Synchronized	在指定 TextWriter 周围创建线程安全包装 （从 TextWriter 继承）
ToString	返回包含迄今为止写入到当前 StringWriter 中的字符的字符串
Write	写入到 StringWriter 的此实例中
WriteLine	写入重载参数指定的某些数据，后跟行结束符 （从 TextWriter 继承）

下面对 StreamWriter 类中比较重要的方法进行详细介绍。

① Close 方法。关闭当前的 StringWriter 和基础流。

语法：

```
public override void Close ()
```

在创建和使用完一个 StringWriter 后，一定要将其及时关闭。

② Flush 方法。清理当前编写器的所有缓冲区，使所有缓冲数据写入基础设备。

语法：

```
public virtual void Flush ()
```

在调用Close方法之前调用Flush方法，可以将以前写入缓冲区的任何数据都复制到文件中。

③ WriteLine 方法。将行结束符写入文本流。

语法：

```
public virtual void WriteLine ()
```

返回值：默认行结束符是后跟换行符的回车符（“\r\n”），但使用 NewLine 属性可以更改此值。

StreamWriter 类有两个最重要、最常用的方法，一个是 Write 方法，另一个是 WriteLine 方法。这两个方法都是用来向文本文件中写入字符串的，但二者也有区别。WriteLine 只用于字符串，并且会自动追加一个换行符（回车\换行）。Write 不追加换行符，可以向文本流写入字符串，也可以写任何基本数据类型（int32、single 等）的文本形式。下面的示例实现了通过 StreamWriter 对象向文本文件“hb.txt”中写入字符串的功能。

```
using (StreamWriter sw = new StreamWriter("hb.txt"))
{
   sw.Write("This is the ");
   sw.WriteLine("header for the file.");
   sw.WriteLine("--------------------");
   sw.Write("The date is: ");
   sw.WriteLine(DateTime.Now);
}
```

注意：使用 StreamWriter 时，应在使用完毕时调用 Close 方法将其关闭。在实例化 StreamWriter 类对象时使用 using 语句实例化 StreamWriter 类对象可以使对象在使用完毕后自动释放。因此可以省去调用 Close 方法。

在使用 StreamReader 类时，必须添加命名空间 using System.IO，如果不想添加命名空间，可以用 System.IO 下的 StreamWriter 类实现。

例如，向指定的文件中写入信息。代码如下：

```
public void WriteText(string s)
{
    System.IO.StreamWriter sw = new StreamWriter("hb.txt", True);
    sw.WriteLine(s);
    sw.Close();
}
```

示例 26-04　向文本文件中写入内容

在向文本文件中写入内容时，首先要获取该文件的字符编码，然后根据字符编码向文件中

写入指定格式的文本信息，如果写入的文本和字符编码不匹配，会出现乱码。本示例在打开文件时，首先获取该文件的字符编码，然后通过字符编码打开文件并读取，用户根据文件内容对其进行修改，最后根据字符编码将改后的文本写入到文件中。

程序代码如下：

代码 26-04　光盘位置：光盘\mingrisoft\sl\26\26_04

自定义方法 GetEncoding 用编码类型 ANSI 将 FileStream 类中的文件读作二进制值，通过前 3 位字节判断当前文件的字符编码，并返回相应的字符编码。代码如下：

```
public static System.Text.Encoding GetEncoding(FileStream fs)
{
    //用特定的编码将基元数据类型读作二进制值
    BinaryReader tem_Binary = new BinaryReader(fs, System.Text.Encoding.Default);
    byte[] tem_Byte = tem_Binary.ReadBytes(3);                    //获取前 3 个字节
    if (tem_Byte[1] >= 0xEF)                                      //如果字符编码不是 ANSI
    {
        if (tem_Byte[0] == 0xEF && tem_Byte[1] == 0xFF && tem_Byte[2] == 0xBF)    //如果字符编码是 UTF8
        {
            return System.Text.Encoding.UTF8;
        }
        else if (tem_Byte[0] == 0xFF && tem_Byte[1] == 0xEF)          //如果字符编码是 Unicodebigendian
        {
            return System.Text.Encoding.BigEndianUnicode;
        }
        else if (tem_Byte[0] == 0xEF && tem_Byte[1] == 0xFF)          //如果字符编码是 Unicode
        {
            return System.Text.Encoding.Unicode;
        }
        else                                  //字符编码为 ANSI
        {
            return System.Text.Encoding.Default;
        }
    }
    else                                      //字符编码为 ANSI
    {
        return System.Text.Encoding.Default;
    }
}
```

在“..”按钮的单击事件中，打开“打开”对话框，在该对话框中选择要写入信息的文件，然后通过 GetEncoding 方法获取当前文件的字符编码，并根据字符编码读取文件中的内容。代码如下：

```
System.Text.Encoding var_Encoding;                                    //定义一个字符编码的变量
private void button1_Click(object sender, EventArgs e)
{
    OpenFileDialog openFile = new OpenFileDialog();                   //实例化 OpenFileDialog 类
    if (openFile.ShowDialog() == DialogResult.OK)                     //打开"打开"对话框
    {
        textBox1.Text = openFile.FileName;                            //获取选中文件的路径
        using (FileStream fs = File.Open(textBox1.Text, FileMode.Open))   //以指定的方式打开文件
        {
            var_Encoding = GetEncoding(fs);                           //获取文件的字符编码
        }
        using (StreamReader strRead = new StreamReader((System.IO.Stream)File.OpenRead(textBox1.Text),
var_Encoding))                    //用指定的字符编码为指定的流初始化 StreamReader 类
        {
            string pp = "";
            richTextBox1.Clear();                                     //清空控件中的文本内容
            while (strRead.Peek() > -1)                               //读取流文件
            {
                pp += strRead.ReadLine();                             //记录文本内容
            }
```

```
            richTextBox1.Text = pp;                                    //显示文本内容
        }
    }
}
```

在“写入”按钮的单击事件中，将修改后的文本信息以流的形式写入到指定文件中。代码如下：

```
private void button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Length == 0)                                     //如果没有选择要写入的文件
        return;                                                        //退出当前操作
    FileStream fs = File.Open(textBox1.Text, FileMode.Truncate);       //打开文件后清空文件中的内容
    fs.Dispose();                                                      //释放当前对象
    using (StreamWriter strWriter = new StreamWriter((System.IO.Stream)File.OpenWrite(textBox1.Text),
var_Encoding))                                                         //用指定的字符编码为指定的流初始化 StreamWriter 类
    {
        strWriter.Write(richTextBox1.Text);                            //写入文本信息
    }
}
```

注意：在向文件中写入信息时，如果修改的信息大小小于原文件的大小，只按修改信息的大小进行替换。这样，在写入信息前为了确保文件尾没有原文件的信息，可以在写入信息前用 Open 方法清空文件中的内容，为了使代码更加简捷，可以直接用 Open 方法完成文件的清空与写入，代码如下：

```
using (StreamWriter strWriter = new StreamWriter((System.IO.Stream)File.Open(textBox1.Text,FileMode.Truncate),
var_Encoding))                    //用指定的字符编码为指定的流初始化 StreamWriter 类，并清空文件内容
{
    strWriter.Write(richTextBox1.Text);
}
```

按<Ctrl+F5>组合键运行程序，运行结果如图 26.5 所示。

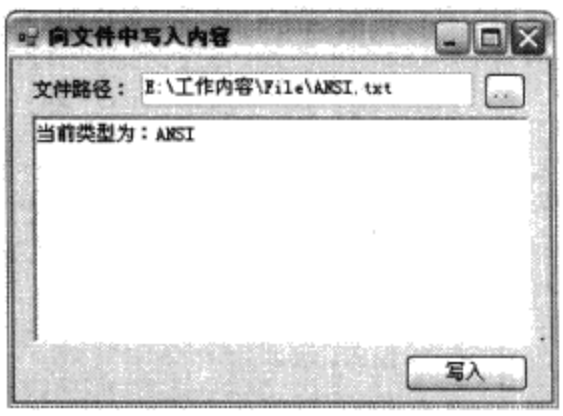


图 26.5　向文件中写入内容

2．二进制模式

System.IO 还提供了 BinaryWriter 类，用于按二进制模式写入文件。BinaryWriter 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。

下面的示例通过 BinaryWriter 类将“Helllo Word”写入到文件“hb.bin”中。代码如下：

```
BinaryWriter binWriter = new BinaryWriter(File.Open("hb.bin", FileMode.Create));
binWriter.Write("Helllo Word");
binWriter.Close();
```

26.4.3　大文件的复制

FileStream 对象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 方法对文件进行随机访问，也同时包含了流的标准输入、标准输出和标准错误等。FileStream 默认对文件的打开方式是同步的，但它同样可以支持异步操作。

FileStream 对象支持使用 Seek 方法对文件进行随机访问。Seek 允许将读取/写入位置移动到文件中的任意位置，这是通过字节偏移参考点参数完成的。字节偏移量是相对于查找参考点

而言的，该参考点可以是基础文件的开始、当前位置或结尾，分别由 Seekorigin 类的 3 个属性表示。

1．FileStream 类的常用属性

FileStream 类的常用属性及说明如表 26.12 所示。

表 26.12　　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
Handle	获取当前 FileStream 对象所封装文件的操作系统文件句柄
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
Safefilehandle	获取 SafeFileHandle 对象，该对象表示当前 FileStream 对象封装的文件的操作系统文件句柄
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

下面对 FileStream 类中比较重要的属性进行详细介绍。

（1）Length 属性。获取用字节表示的流长度。

语法：

```
public override long Length { get; }
```

属性值：用字节表示流长度的值。

例如，获取指定文件的大小。代码如下：

```
FileStream fileStream = new FileStream(textBox1.Text, FileMode.Open);
MessageBox.Show(fileStream.Length.ToString());
fileStream.Close();
```

（2）Position 属性。获取或设置此流的当前位置。

语法：

```
public override long Position { get; set; }
```

属性值：此流的当前位置。

例如，设置流的当前位置为文件的最后。代码如下：

```
FileStream fileStream = new FileStream(textBox1.Text, FileMode.Open);
fileStream.Position = fileStream.Length;
```

2．FileStream 类的常用方法

FileStream 类的常用方法及说明如表 26.13 所示。

表 26.13　　FileStream 类的常用方法及说明

方　法	说　明
BeginRead	开始异步读
BeginWrite	开始异步写
Close	关闭当前流并释放与之关联的所有资源（如套接字和文件句柄）
CreateObjRef	创建一个对象，该对象包含生成用于与远程对象进行通信的代理所需的全部相关信息
Dispose	已重载
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

续表

方　法	说　明
Equals	确定两个 Object 实例是否相等
Flush	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
GetAccessControl	获取 FileSecurity 对象，该对象封装当前 FileStream 对象所描述的文件的访问控制列表（ACL）项
GetHashCode	用作特定类型的哈希函数。GetHashCode 适合在哈希算法和数据结构（如哈希表）中使用
GetLifetimeService	检索控制此实例的生存期策略的当前生存期服务对象
GetType	获取当前实例的 Type
InitializeLifetimeService	获取控制此实例的生存期策略的生存期服务对象
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
ReferenceEquals	确定指定的 Object 实例是否是相同的实例
Seek	将该流的当前位置设置为给定值
SetAccessControl	将 FileSecurity 对象所描述的访问控制列表（ACL）项应用于当前 FileStream 对象所描述的文件
SetLength	将该流的长度设置为给定值
Synchronized	在指定的 Stream 对象周围创建线程安全（同步）包装
ToString	返回表示当前 Object 的 String
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

下面对 FileStream 类中比较重要的方法进行详细介绍。

（1）Close 方法。关闭当前流并释放与之关联的所有资源（如套接字和文件句柄）。

语法：

```
public virtual void Close ()
```

在创建和使用完一个流后一定要将其及时关闭。

（2）Flush 方法。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：

```
public override void Flush ()
```

在调用 Close 方法方法之前调用 Flush 方法，可以将以前写入缓冲区的任何数据都复制到文件中，并且缓冲区被清除（其编码器状态除外）。

（3）Seek 方法，将该流的当前位置设置为给定值。

语法：

```
public override long Seek(long offset,SeekOrigin origin)
```

参数说明如下。

offset：Int64，相对于 origin 的点，从此处开始查找。

origin：SeekOrigin，使用 SeekOrigin 类型的值，将开始位置、结束位置或当前位置指定为 origin 的参考点。

返回值：Int64，流中的新位置。

当打开文件时，FileStream 类对象会维护一个内部文件指针，这个指针将会指向文件中的位置，在多数情况下，默认指向文件的开始位置，但是通过修改指针允许程序将指针定位在文件的任何地方。例如，以 8KB 为单位的移动距离。代码如下：

```
aFile.Seek(8,SeekOrigin.Begin)
```

SeekOrigin 有 3 个枚举值 Begin、Current 和 End，分别是开始、当前和结束。上面的代码将文件指针移到以开始为起始位置的 8KB 后。

（4）Write 方法。使用从缓冲区读取的数据将字节块写入该流。

语法：

```
public override void Write (byte[] array,int offset,int count)
```

参数说明如下。

array：指包含要写入该流的数据的缓冲区。

offset：指数组中的从零开始的字节偏移量，从此处开始将字节复制到当前流。

count：指要写入当前流的最大字节数。

例如，向指定的文件写入 1024 字节。代码如下：

```
int SectSize = 1024;
byte[] buffer = new byte[SectSize];
FileStream fileStream = new FileStream(textBox1.Text, FileMode.Open);
fileStream.Write(buffer, 0, SectSize);
```

说明：Read 方法的语法及应用与 Write 方法基本相同，只是该方法用于读取操作。

（5）BeginRead 方法。开始异步读。

语法：

```
public override IAsyncResult BeginRead(byte[] array,int offset,int numBytes,AsyncCallback userCallback,Object
stateObject)
```

参数说明如下。

array：byte[]，要读入数据的缓冲区。

offset：Int32，array 中的字节偏移量，从此处开始读取。

numBytes：Int32，最多读取的字节数。

userCallback：AsyncCallback，异步读操作完成后调用的方法。

stateObject：Object，一个用户提供的对象，它将该特定的异步读取请求与其他请求区别开来。

返回值：IasyncResult，引用异步读的 IAsyncResult。

说明：BeginWrite 方法的语法及应用与 BeginRead 方法基本相同，只是该方法用于异步写入操作。

（6）EndRead 方法。等待挂起的异步读取完成。

语法：

```
public override int EndRead(IAsyncResult asyncResult)
```

参数说明如下。

asyncResult：IasyncResult，对所等待的挂起异步请求的引用。

返回值：Int32，从流中读取的字节数，介于 0 和所请求的字节数之间。流仅在流的末尾返回 0；否则应一直阻塞至少有 1 个字节可用为止。

说明：EndWrite 方法的语法及应用与 EndRead 方法基本相同，只是该方法用于等待挂起异步写入的完成。

实例位置：光盘\mingrisoft\dxyy\26\26_02

在对大文件进行复制时，为了能更好的让用户检测到文件的复制情况，本实例用 FileStream 类以流的形式对文件的异步读取，并在读取时显示复制的进度条。运行结果如图 26.6 所示。

程序开发步骤如下。

（1）在 VS2008 中新建一个 Windows 应用程序项目，并命名为 AsynchronismFile。

（2）打开 Windows 窗体设计器，从工具箱中为窗体添加两个 Label 控件、两个 TextBox 控件和三个 Button 控件。

（3）在当前项目中添加一个窗体，将其名命为 Frm_Plan，从工具箱中为窗体添加一个 Label 控件、一个 ProgressBar 控件和一个 Timer 控件。

（4）在主窗体中设置源文件的路径和目的文件的路径，然后单击“复制‘按钮，调用“复制”对话框窗体（Frm_Plan），在该窗体中进行文件的复制。

图 26.6　文件的异步复制

（5）在 Frm_Plan 窗体中设置变量。代码如下：

```
public string filePath = "";                          //源文件路径
public string ToFile = "";                            //目的文件的路径
int totalSize;                                        //源文件的大小
int position=0;                                       //记录文件已复制的大小
const int BUFFER_SIZE = 4096;                         //流的传输大小
byte[] buffer;                                        //记录复制的字节
Stream stream;                                        //定义 Stream 类
FileStream fs;                                        //定义 FileStream 类
```

（6）在“复制”窗体的第一次显示事件中，通过源文件和目的文件的路径对文件进行异步读取。代码如下：

```
private void Frm_Plan_Shown(object sender, EventArgs e)
{
    if (filePath.Length == 0 || ToFile.Length == 0)          //如果没有设置了源文件和目的文件的路径
        Close();                                             //关闭“复制”对话框
    //通过源文件路径以只读方式实例化 FileStream 类
    fs = new FileStream(filePath, FileMode.Open, FileAccess.Read);
    totalSize = (int)fs.Length;                              //获取源文件的大小
    this.progressBar1.Maximum = totalSize;                   //设置进行度控件的最大值
    stream = fs;                                             //记录源文件的文件流
    FileInfo fileInfo = new FileInfo(filePath);              //实例化 FileInfo 类
    ToFile += "\\" + fileInfo.Name;                          //设置目的文件
    if (File.Exists(ToFile))                                 //如果目的文件存在
        File.Delete(ToFile);                                 //删除目的文件
    if (totalSize > BUFFER_SIZE)                             //如果源文件的大小大于流的的传输大小
    {
        buffer = new byte[BUFFER_SIZE];                      //根据传大小设置字节
        timer1.Start();                                      //开启计时器
        //开始文件的异步读取
        stream.BeginRead(buffer, 0, BUFFER_SIZE, new AsyncCallback(AsyncCopyFile), null);
    }
    else                                                     //小文件
    {
        fileInfo.CopyTo(ToFile, true);                       //对文件直接进行复制
        position = totalSize;                                //设置进度栏的进度
        fs.Close();                                          //关闭当前流
        timer1.Start();                                      //启动计时器
    }
}
```

AsyncCopyFile 是异步读取调用完成后所调用的方法，该方法主要是通过文件流的传输大小，对文件进行异步传输。代码如下：

```
private void AsyncCopyFile(IAsyncResult ar)
{
    int readedLength;                                        //记录传输流的字节数
    lock (stream)                                            //在读取访问的同时防止其他进程更改
    {
        readedLength = stream.EndRead(ar);                   //等待挂起的异步读取完成
    }
    //通过目的文件的路径以追加方式实例化 FileStream 类
    FileStream fsWriter = new FileStream(ToFile, FileMode.Append, FileAccess.Write);
    fsWriter.Write(buffer, 0, buffer.Length);                //将源文件中读取的流写入到目的文件中
```

```
        fsWriter.Close();                                   //关闭写操作
        position += readedLength;                           //记录传递流的字节数
        if (position >= totalSize)                          //如果写入完成
        {
            stream.Close();                                 //关闭当前流
            return;
        }
        lock (stream)                                       //在读取访问的同时防止其他进程更改
        {
            int leftSize = totalSize - position;            //获取没有复制的文件大小
            if (leftSize < BUFFER_SIZE)                     //如果小于流的传输大小
                buffer = new byte[leftSize];                //设置字节大小
            //开始文件的异步读取
            stream.BeginRead(buffer, 0, buffer.Length, new AsyncCallback(AsyncCopyFile), null);
        }
    }
```

在 timer1 组件的 Tick 事件中，通过流的传输大小设置进度条。代码如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
    this.progressBar1.Value = position;                     //设置进度栏的当前位置
    if (progressBar1.Value >= progressBar1.Maximum)         //如果文件复制完成
    {
        fs.Close();                                         //关闭当前流
        timer1.Stop();                                      //关闭计时器
        Close();                                            //关闭当前窗体
    }
}
```

注意：本实例为了用进度栏显示文件的复制情况，并可以在复制过程中任意移动“复制”窗口，用 lock 方法对文件的读取部分进行锁定，然后用 timer1 组件控件进度栏。

第 27 章

帮助文件

- WinHelp 帮助文件
- HTML 帮助文件

帮助文件能够为应用程序提供简单明确的程序说明，因此制作帮助文件是程序设计工作中的一个重要步骤。在帮助文件中可以包含图形、经过格式化编排的文本、声音和动画等。本章将对帮助文件的制作和使用进行详细介绍。

27.1 WinHelp 帮助文件

带有帮助文件的应用程序无论对程序员还是用户都是非常有用的，它能节省印刷大量用户手册的开支，而且还可以方便程序员编辑修改程序，还可以方便用户手册的印制。在 Windows 操作系统中，帮助文件可分为 hlp（help）格式与 chm（compiled html file）格式两种，这两种格式的帮助文件分别可以使用 Microsoft Help Workshop(.hlp)和 Microsoft HTML Help Workshop（.chm）两种不同的软件制作。

27.1.1 安装 Microsoft Help Workshop

“Microsoft Help Workshop”是一个用于制作.hlp 帮助文件的软件，该软件需要单独安装。安装程序位置：光盘\mingrisoft\sl\27\27_01。安装“Microsoft Help Workshop”软件的步骤如下。

（1）在本书光盘“mingrisoft\sl\27\27_01\Help Workshop（制作帮助的软件）”目录下运行 SETUP.EXE 文件，弹出如图 27.1 所示的安装初始界面。

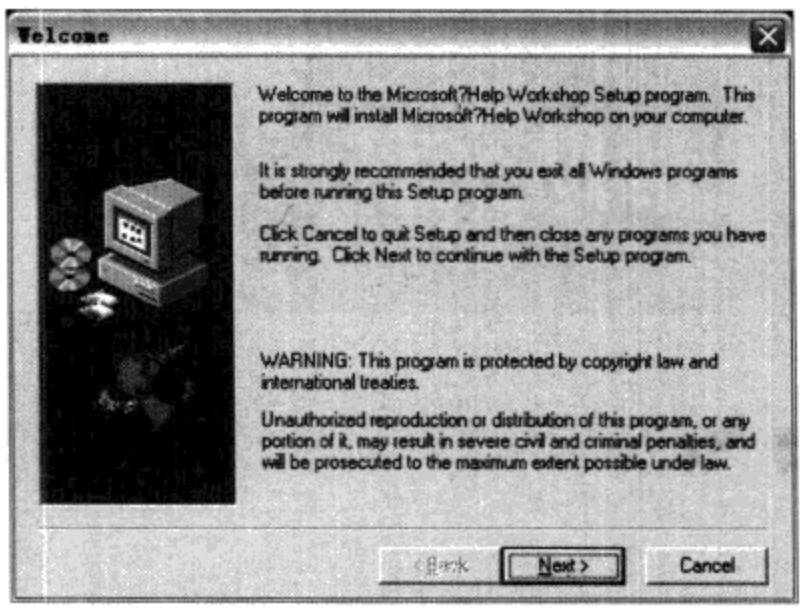


图 27.1 安装初始界面

（2）单击“Next”按钮，进入图 27.2 所示的选择安装路径界面。

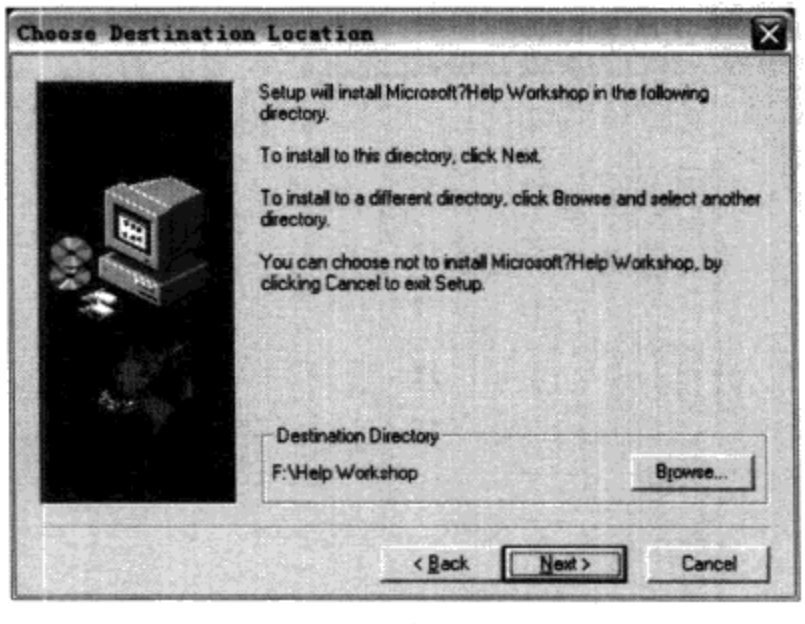


图 27.2 选择安装路径

（3）单击“Browse”按钮，选择软件安装路径，单击“Next”按钮，在弹出的安装类型界面中选中“Typical”单选按钮，如图 27.3 所示。

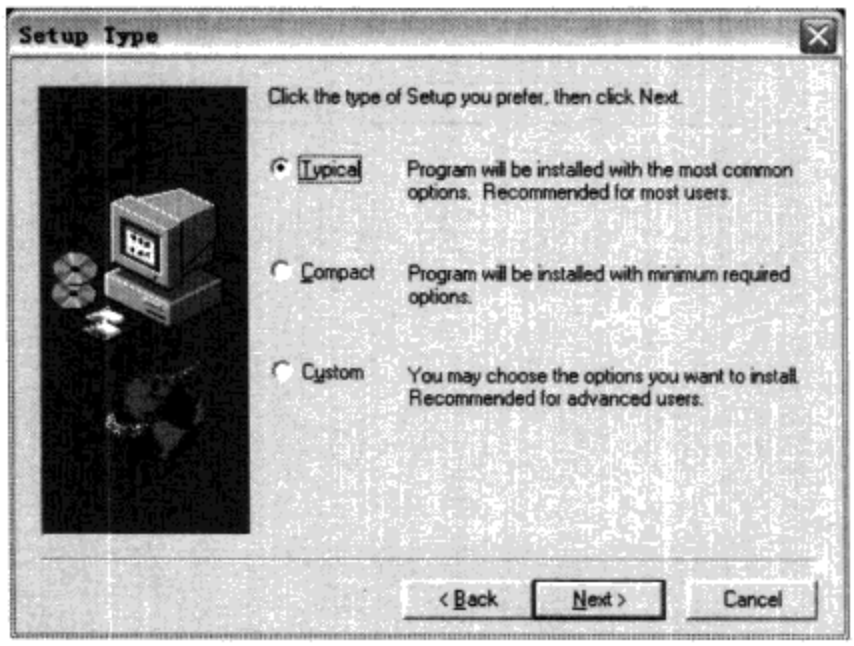


图 27.3　选择安装类型

（4）单击“Next”按钮，开始安装，如图 27.4 所示。安装完成后，在弹出的对话框中单击“确定”按钮，即可完成“Microsoft Help Workshop”软件的安装过程。

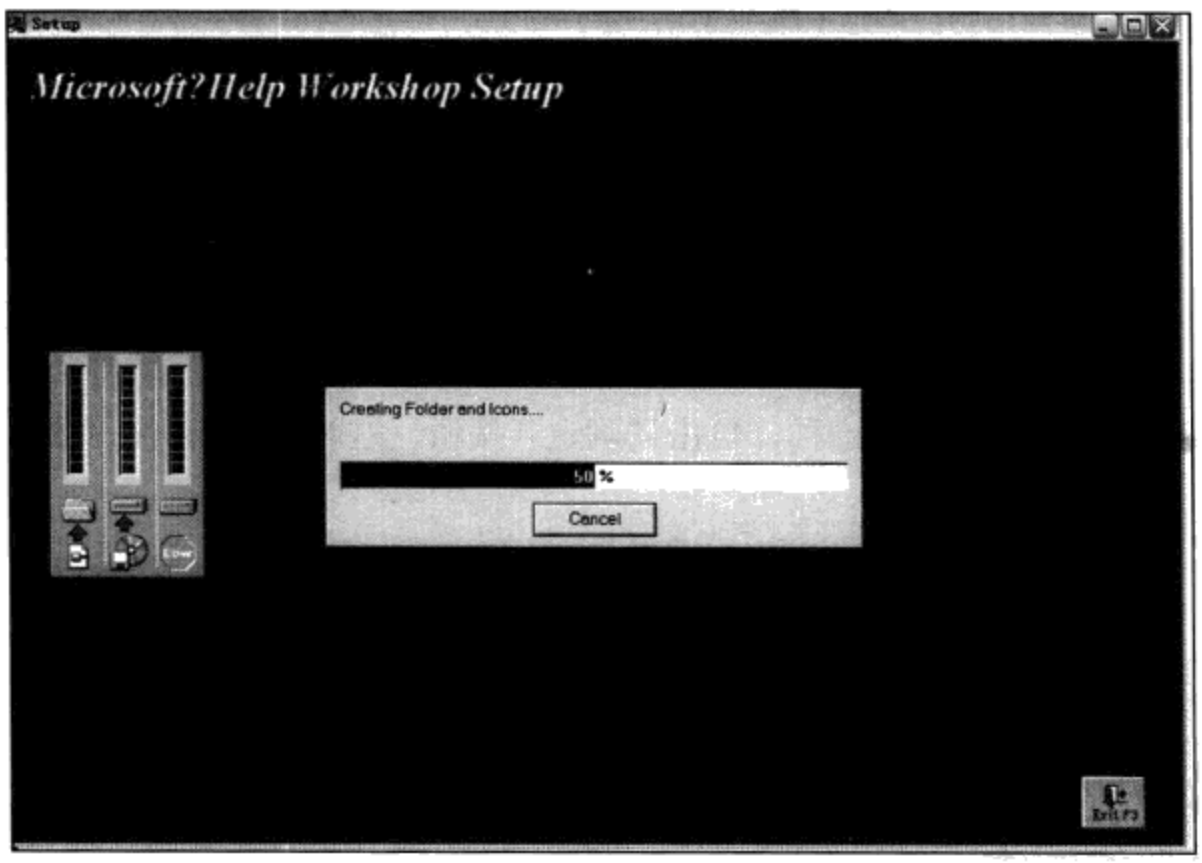


图 27.4　安装过程

27.1.2　编辑 RTF 文件

帮助文件主要用来存储软件的帮助信息，这种文件的格式为 RTF（Rich Text Format）格式。帮助文件像一本书，由目录及许多章节组成，各章节之间的跳转是由自定义的热点来完成的。程序开发人员可以通过 Word 等编辑器直接编辑 RTF 格式文件。下面介绍（由目录等 5 条信息组成）如何通过 Word 编辑 RTF 格式文件。

通过 Word 编辑 RTF 格式文件的步骤如下。

1．创建 RTF 格式的文档

打开 Word 格式的文档。在“文件”菜单中选择“另存为”选项，弹出“另存为”对话框，

在“保存类型”下拉列表框中选择“RTF 格式（*.rtf）”选项，然后选择文件保存路径保存 RTF 格式的文件，“另存为”对话框如图 27.5 所示。

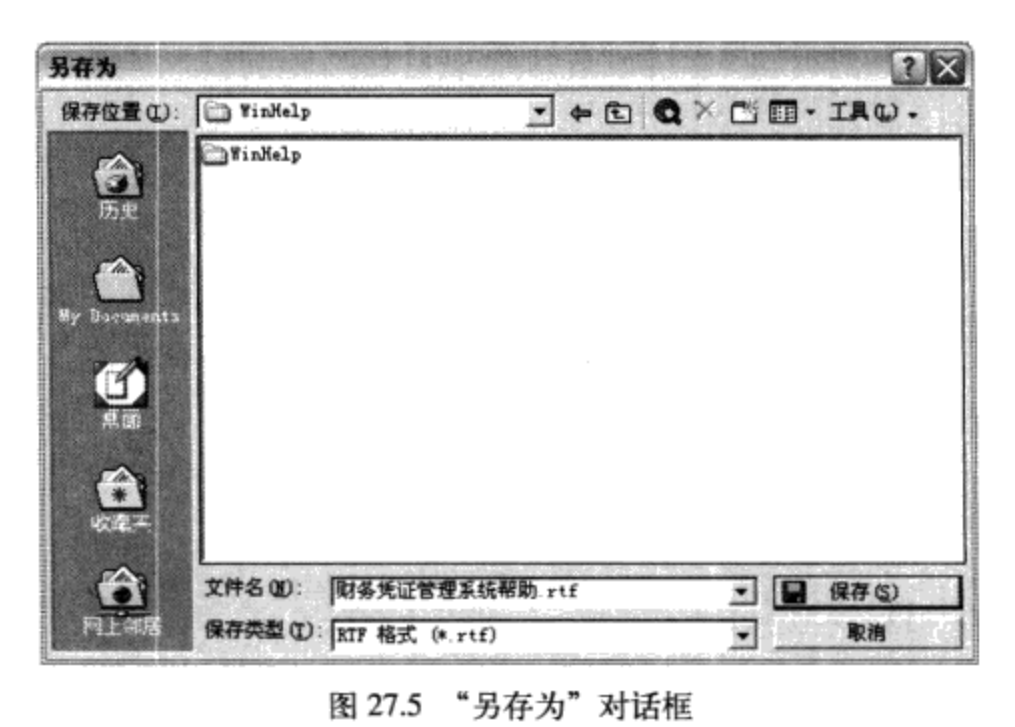


图 27.5 “另存为”对话框

2．编辑文本标题

按照 Word 排版格式给创建的 RTF 文挡进行排版，如设置文档的一级、二级标题，字体的大小和颜色等。

3．设置链接文本

链接文本是指实现从一个主题到另一个主题的链接。使用带链接文本的帮助文件时，不必使用菜单来选择帮助主题，可直接通过链接文本来查看相应的帮助主题。设置链接文本由设置分页、添加脚注和设置连接字符 3 个步骤组成。

● 设置分页

将光标移到需要设置分页处的那一行，在菜单栏中选择“插入”/“分隔符”选项，打开“分隔符”对话框，选择“分页符”选项，单击“确定”按钮，这时在光标所在行处将产生一条虚线，表示分页操作已经完成。

● 添加脚注

RTF 文件中插入脚注时，需要在每页的页首插入，脚注编号方式选“自定义方式”，自定义标记为“#”、“$”、“K”、“+”和“!”，它们的含义分别如下。

#脚注：惟一标识每个帮助主题的变量定义，也称跳转名。本书示例中，给每一节分别命名为 one、two、three、four 和 five 等。

$ 脚注：指定每个帮助主题的标题信息，可用不超过 64 个的汉字表示。本书示例中，给每一节分别命名为概述、系统管理模块、会计科目模块、凭证管理模块和账目查询模块。

K 脚注：用来指定与每个帮助主题相关联的查询搜索关键字。

+脚注：用来定义帮助文档中各帮助主题的浏览顺序。

! 脚注：用于单击后执行一条宏命令的定义，需要用隐藏字体。

由于 RTF 格式文件的页码较多，下面只演示在第一页“概述”页插入“脚注#”和“脚注$”，步骤如下。

（1）将光标定位到第一节标题“概述”前面，如图 27.6 所示。

（2）在菜单中选择“插入”/“脚注和尾注”命令，在弹出的“脚注和尾注”对话框中，选择“脚注”和“自定义标记”选项，如图 27.7 所示。

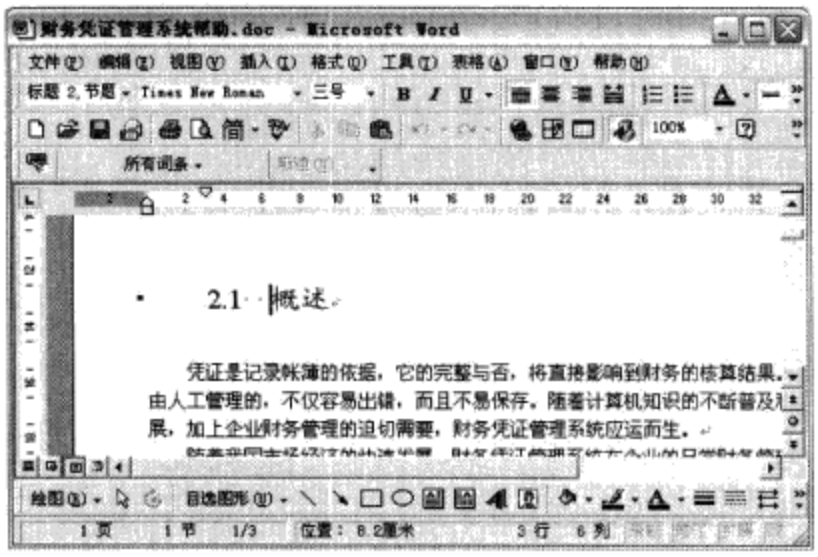


图 27.6　定位光标

图 27.7　“脚注和尾注”对话框

(3) 单击“符号”按钮，弹出“符号”对话框，如图 27.8 所示。选择“$”符号，单击“确定”按钮，返回“脚注和尾注”对话框，如图 27.9 所示。

图 27.8　“符号”对话框

图 27.9　$符号

(4) 在图 27.9 中单击“确定”按钮，这时文档下方出现脚注编辑窗口，光标也自动落在文档下方的脚注编辑窗口中，在光标处输入这一节的标题：“概述”，如图 27.10 所示。如果看不到文档下方的脚注窗口，选中系统菜单“视图”中的“脚注”选项即可。

图 27.10　$符号效果

(5) 设置跳转名“#”，重复步骤 1，步骤 2，在图 27.8 符号对话框中选择“#”符号，单击“确定”按钮，返回“脚注和尾注”对话框，单击“确定”按钮，这时文档下方出现脚注编辑窗口。光标自动落在文档下方的脚注编辑窗口中，在光标处输入这一节的跳转名：“one”（也可以用数字编写），效果如图 27.11 所示。如果看不到文档下方的脚注窗口，选中系统菜单“视

图”中的“脚注”选项即可。

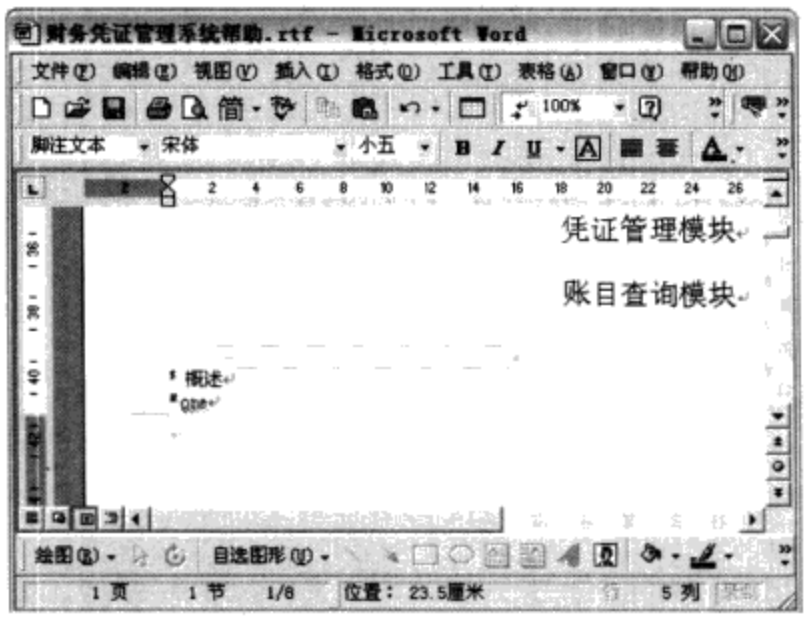


图 27.11　#符号效果

设置链接字符，使用光标选中需要链接的字符串，单击鼠标右键，在弹出的快捷菜单中选择“字体”选项，打开“字体”对话框，在“下划线”下拉列表框中选择“双下划线”类型，单击“确定”按钮。另外在链接的字符串后添加一个空格和链接到脚注页的脚注编号“#”设置的值，用光标选中空格和链接到脚注页的脚注编号，然后在“字体”对话框中选择“隐藏文字”复选框，单击“确定”按钮，将其设置为隐藏文字。本示例中#的值分别是 one、two、three、four 和 five，它们分别标识 5 个页面。本示例设置链接字符后的效果如图 27.12 所示。

图 27.12　连接字符

27.1.3　制作帮助索引文件

帮助索引文件能快速直观地帮助用户找到所需要的内容，制作帮助索引文件的步骤如下。

（1）运行“Microsoft Help Workshop”安装目录下的 hcrtf.exe 文件，在弹出的窗体中选择“File”/“New”选项，弹出如图 27.13 所示的“New”对话框。

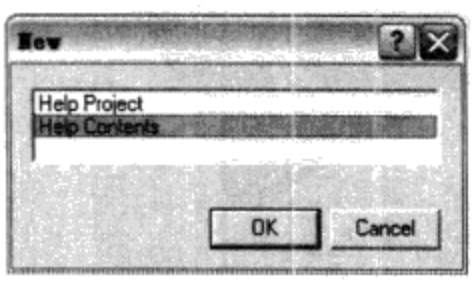


图 27.13　“New”对话框

（2）在“New”对话框中，选择“Help Contents”选项，单击“OK”按钮，弹出如图 27.14 所示的窗口。在该窗口的“Default filename”文本框中输入“MRMONEYHELP.HLP”，同

时在“Default title”文本框中输入“财务凭证管理”。

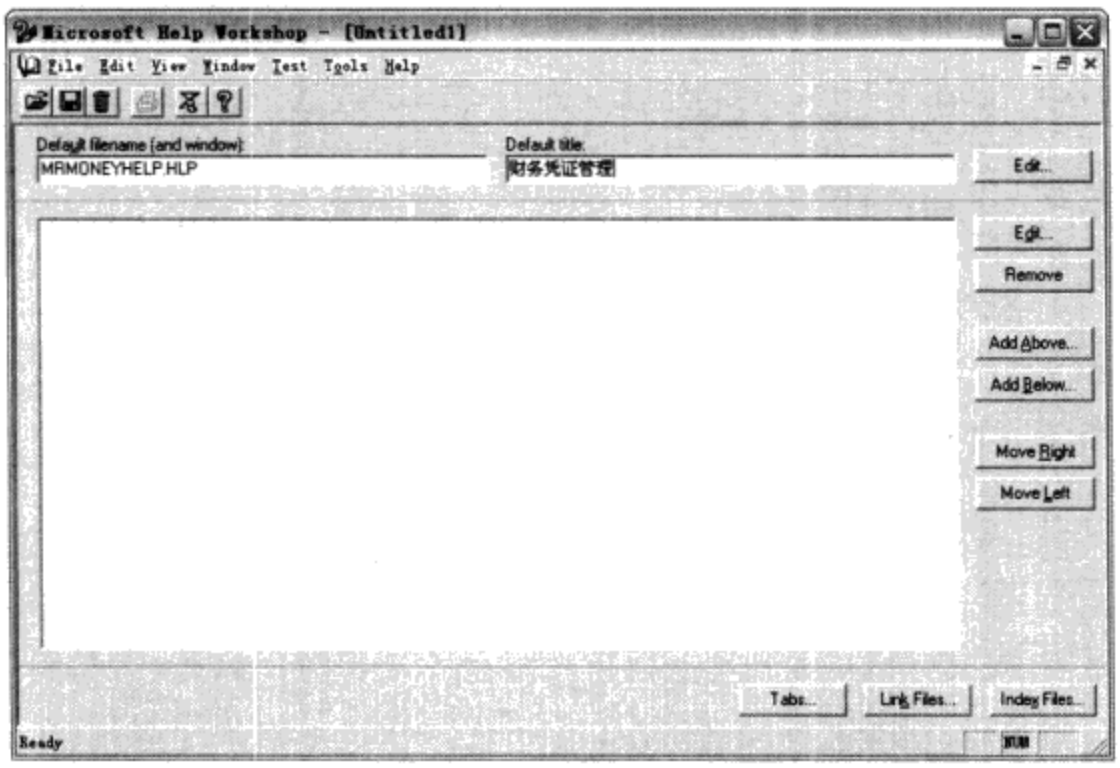


图 27.14　设置文件的标题

(3) 单击“Add Below”按钮，弹出如图 27.15 所示的对话框，该对话框中选中“Heading”单选按钮，在“Title”文本框中输入“概述”。

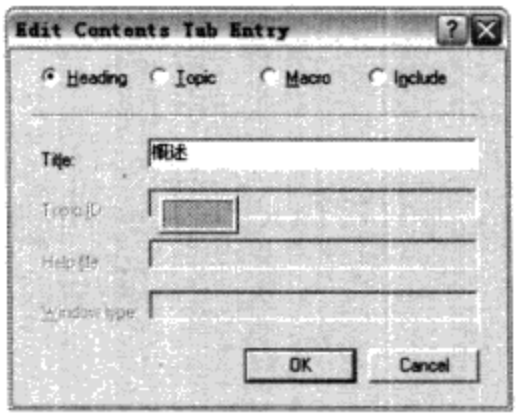


图 27.15　设置 Title 标题

(4) 单击“OK”按钮，如图 27.16 所示，然后单击“Add Below”按钮，弹出如图 27.17 所示的对话框。在该对话框中选中“Topic”单选按钮，在“Title”文本框中输入索引名称“概述信息”，在“Topic ID”文本框中输入索引所对应 RTF 文件中的脚注标号，也就是#的值（本示例 RTF 文件中概述内容的脚注标号是 one），单击“OK“按钮，如图 27.18 所示。

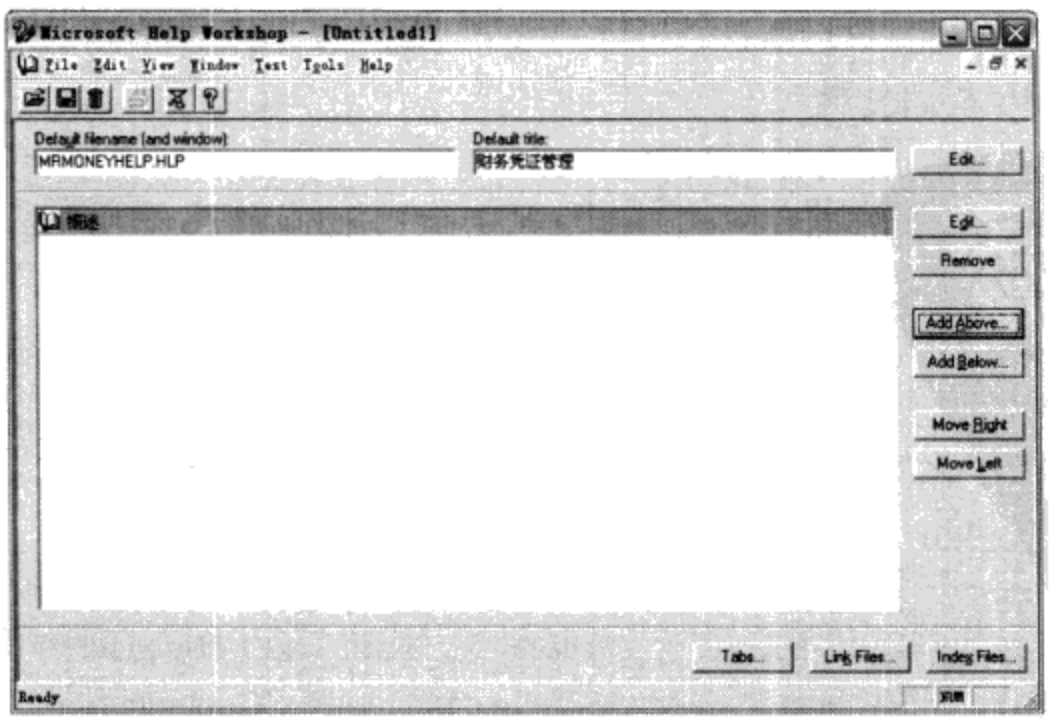


图 27.16　Title

(5) 按照以上步骤依次完成 Title 和索引项的制作，如图 27.19 所示，然后选择“File”/“Save As”选项，将设置好的文件存储为索引文件 mrMoneyHelp.cnt。

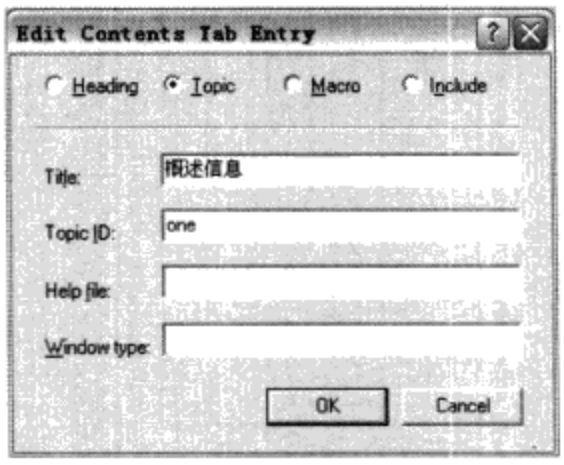


图 27.17　设置索引项

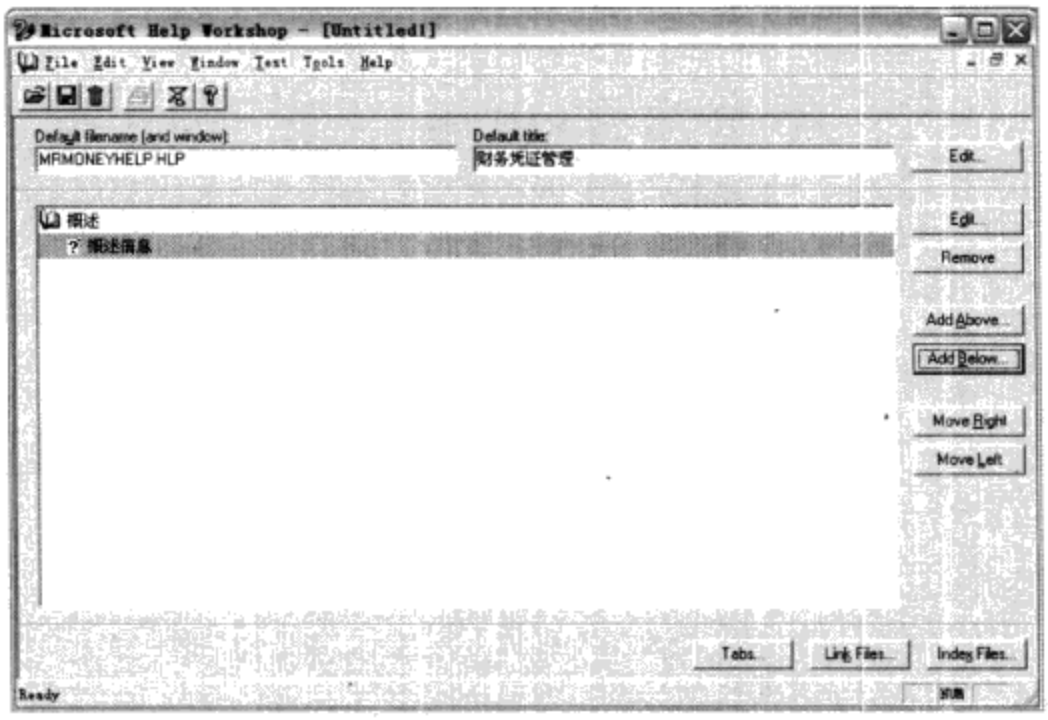


图 27.18　索引项名称

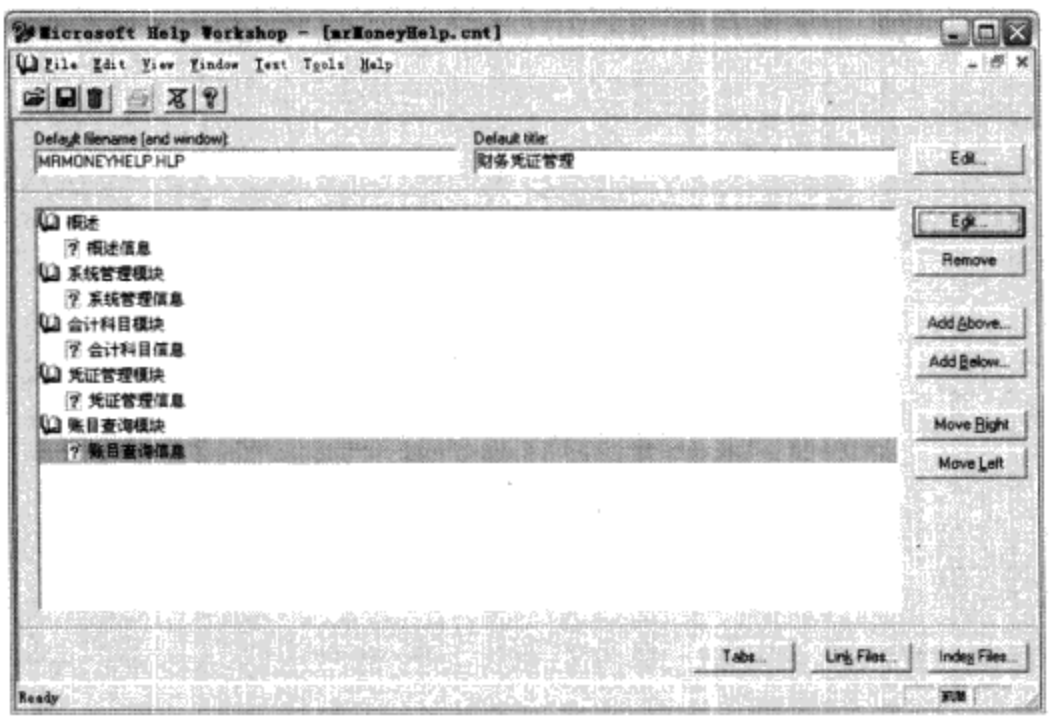


图 27.19　帮助索引文件

27.1.4　制作.Hlp 帮助文件

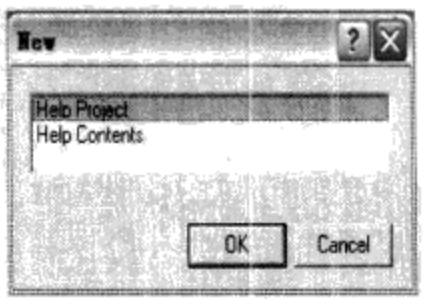


图 27.20　“New”对话框

（1）运行“Microsoft Help Workshop”安装目录下的 hcrtf.exe 文件，在弹出的窗体中选择“File”/“New”选项，弹出如图 27.20 所示的“New”对话框。

（2）选择“Help Project”选项，单击“OK”按钮，在弹出的对话框中输入.hpj 文件的文件名(这里以 mrMoney.hpj 文件为例)，同时保存文件位置，单击“保存”按钮，弹出“Microsoft Help Workshop-[mrMoney.hpj]”对话框，如图 27.21 所示。

（3）单击“Options”按钮，弹出如图 27.22 所示对话框，选择“Files”选项卡，单击“Change”按钮，将“财务凭证管理系统帮助.rtf ”文件添加到“Rich Text Format(RTF) files”下拉列表中。单击“Browse”按钮，将主题文件 mrMoneyHelp.cnt 添加到“Coptents file”文本框中。

（4）单击“确定”按钮，返回到“Microsoft Help Workshop-[mrMoney.hpj]”对话框，如图 27.23 所示，单击“Save and Compile”按钮，编译 mrMoneyHelp.hlp 文件。.Hlp 帮助文件结果如图 27.24 所示。

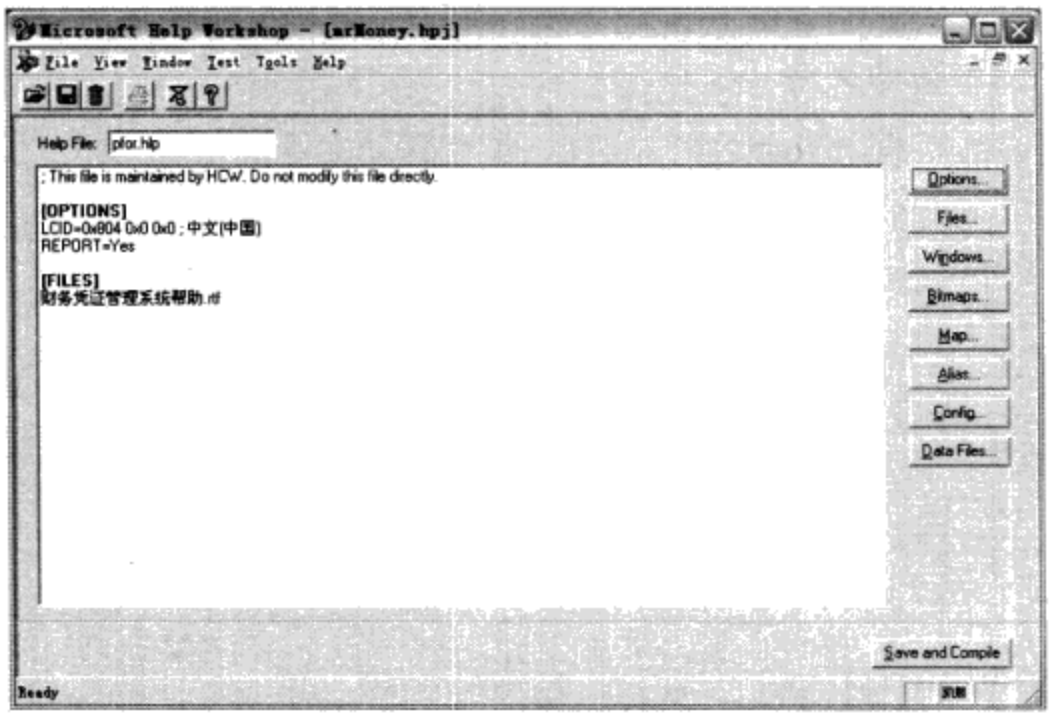


图 27.21 “Microsoft Help Workshop-[mrMoney.hpj]”对话框

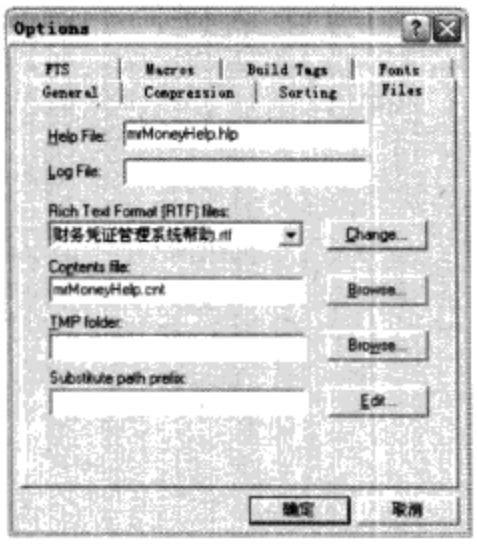


图 27.22 添加.rtf 文件和帮助主题文件

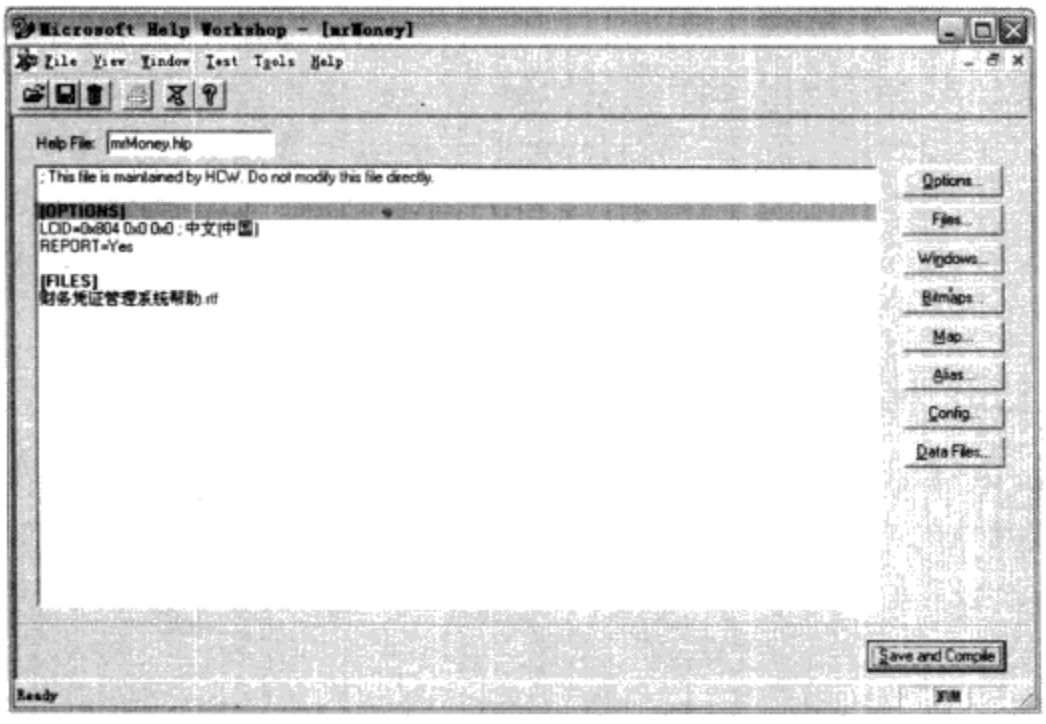


图 27.23 单击“Save and Compile”按钮

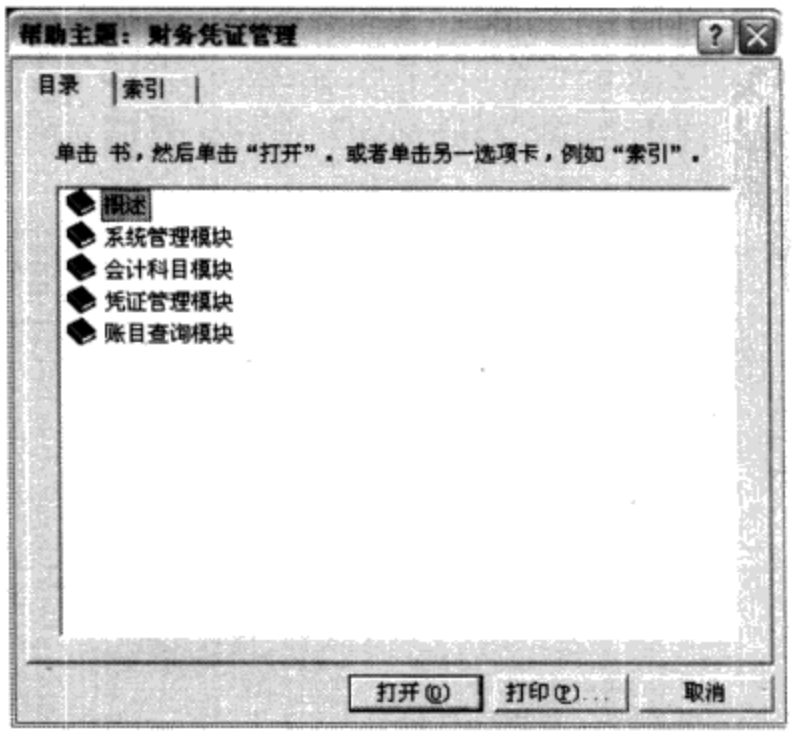


图 27.24 帮助文件

27.1.5 在 C#中使用帮助文件

示例 27-01 在 C#中使用帮助文件

本示例中，当程序运行时，单击“启动帮助文件”按钮，调用帮助文件。

程序主要代码如下：

代码 27-01 光盘位置：光盘\mingrisoft\lx\27\27_01

```
private void button1_Click(object sender, EventArgs e)
{
    string reportPath = Application.StartupPath.Substring(0, Application.StartupPath.Substring(0,
Application.StartupPath.LastIndexOf("\\")).LastIndexOf("\\"));          //获取帮助文件所在的目路
    reportPath += @"\MRMONEYHELP.HLP";                                   //设置帮助文件的路径
    System.Diagnostics.Process.Start(reportPath);                        //打开帮助文件
}
```

27.2 HTML 帮助文件

27.2.1 HTML 帮助文件概述

HTML Help WorkShop 是制作 chm 文件的最佳工具，chm 文件是微软公司 1998 年推出的基于 HTML 文件特性的帮助文件系统，Windows 98 中把 chm 类型文件称作“已编译的 HTML 帮助文件”。chm 同样支持被 Internet Explorer 支持的 JavaScript、VBScript、ActiveX、Java Applet、Flash、HTML 图像文件（GIF、JPEG、PNG）和音频视频（AU、MIDI、WAV、AVI）等文件，并可以通过 URL 地址与因特网联系在一起。

27.2.2 安装 HTML Help Workshop

HTML Help Workshop 是一个用于制作.chm 帮助文件的软件，其安装步骤如下。

（1）在本书附带光盘“mingrisoft\sl\27\02\HTMLHELP”目录下运行 HTMLHELP.exe 文件。

（2）在 License Agreement 窗口中单击“Accept”按钮接受协议，进行下一步操作。

（3）单击 Welcome 窗口中的“Next”按钮，进行下一步操作。

（4）单击“Select Destination Folder”窗口中的“Browse”按钮，在打开的对话框窗口中选择 Microsoft HTML Help Workshop 所要安装的路径，单击“OK”按钮，然后单击“Next”按钮，进行下一步操作。“Select Destination Folder”窗口如图 27.25 所示。

（5）选择“Choose Type of Setup”窗口中的 Complete（完全安装）、Custom（自定义安装）等单选项来选择安装类型。这里选择“Complete”选项，单击“Next”按钮，进行下一步操作。“Choose Type of Setup”窗口如图 27.26 所示。

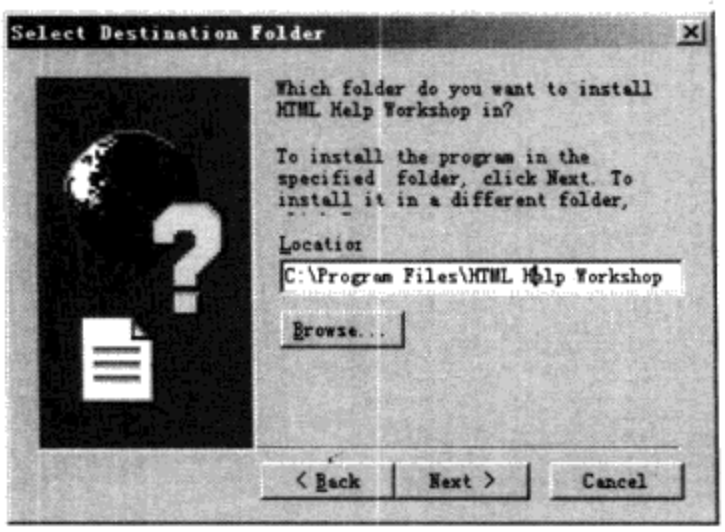


图 27.25 “Select Destination Folde”窗口

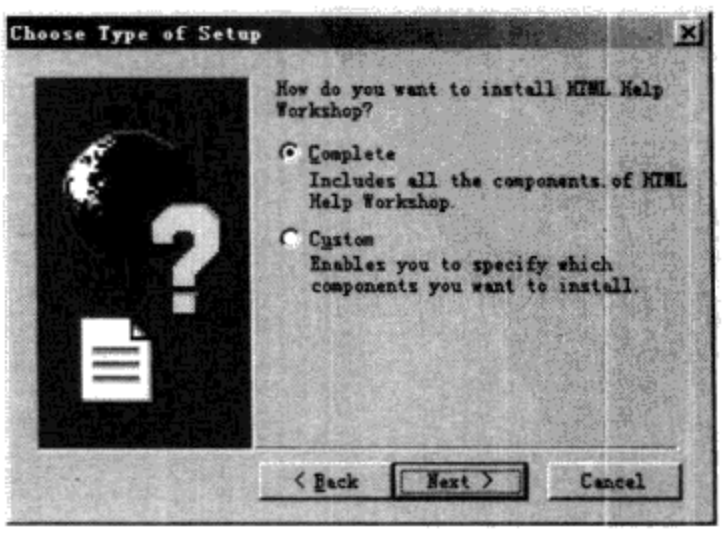


图 27.26 “Choose Type of Setup”窗口

（6）通过在“Select Program Item Group”窗口中选择“Create a new submenu call”（在“开始”菜单中创建一个新的子项）与“Place Commands on this submenu”（将子项添加到“开始”菜单中已有的子项中）单选项，选择创建子项列表，单击“Next”按钮，进行下一步操作。Select Program Item Group 窗口如图 27.27 所示。

（7）在“Personalize HTML Help Workshop”窗口中输入名称后，单击“Next”按钮，开始安装 Microsoft HTML Help Workshop。安装完成后会弹出对话框，如图 27.28 所示，单击“OK”按钮即可完成安装。

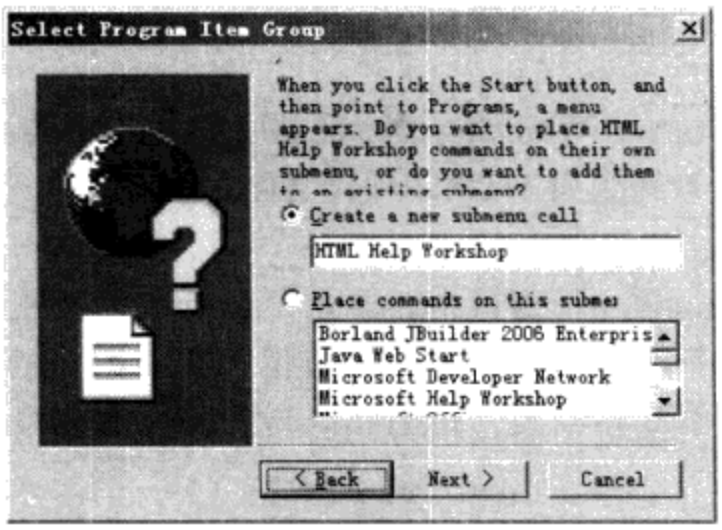


图 27.27 “Select Program Item Group”窗口

图 27.28 安装完毕

27.2.3 HTML 帮助主题文件

帮助主题文件可以通过在 Word 文档中插入超级链接与书签（使用 Word 编写帮助主题，需要将其另存为.html 或.htm 文件）实现，也可以使用 HTML 语言通过正文链接实现。本示例一共有 5 个帮助主题文件，分别为财务凭证管理系统帮助.html、FileHead.html、WarrInfo.htm、System.html 和 subject.htm。

本书光盘位置：光盘\mingrisoft\mingrisoft\sl\27\27_02\help。

27.2.4 HTML 帮助项目文件

帮助项目文件是将帮助文件中应用的所有元素都联系在一起，如帮助主题文件、图像文件、索引文件和内容文件等。帮助项目文件是一个纯文本文件，任何字符处理软件都能将其打开并编辑。项目文件经过 HTML Help Workshop 的编译，将成为一个单一的.chm（帮助文件）。创建 HTML 帮助项目文件的步骤如下。

（1）单击“开始”按钮，选择“所有程序”/“HTML Help Workshop”/“HTML Help Workshop”选项，启动 HTML Help Workshop，如图 27.29 所示。

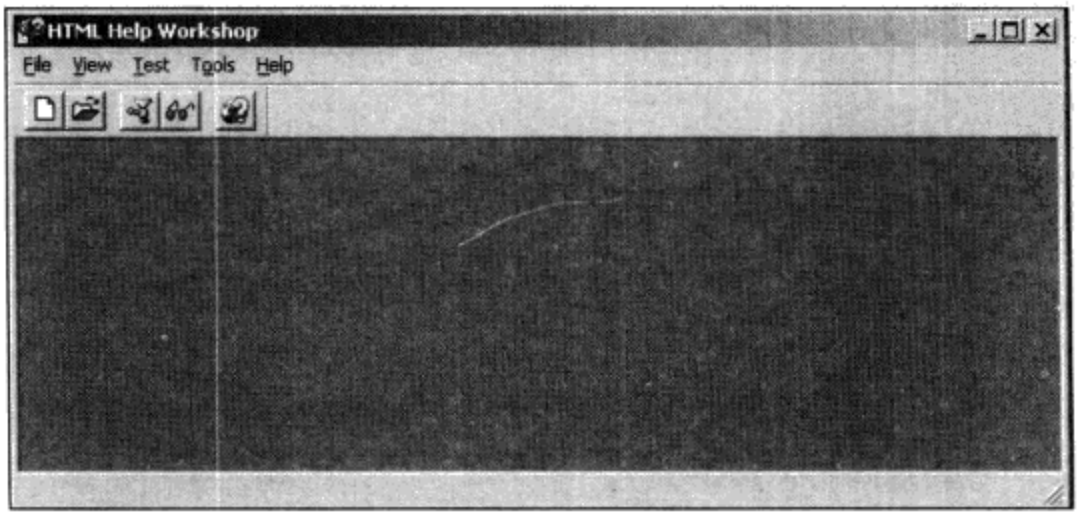


图 27.29 运行界面

（2）选择“HTML Help Workshop”窗口的“File”/“New”选项或直接选取工具条上的新建工具，打开“New”对话框，如图 27.30 所示。选择“Project”选项，单击“OK”按钮，HTML 项目文件向导即可启动。

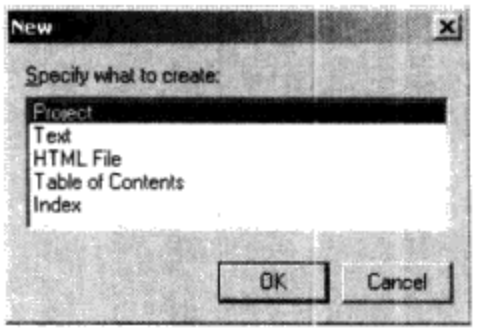


图 27.30 “New”对话框

“New”对话框中各选项的类型说明如下。

- Projeet 工程文件
- Text：文本文件。
- HTML File：HTML 文件。
- Table of Contents：目录文件。
- Index：索引文件。

（3）在“New Project”窗口中有一个名为“Convert Winhelp project”的复选框，将其选中，即可将一个已经存在的 Winhelp 项目文件（.HJP）转变为 HTML 项目文件。单击“下一步”按钮，如图 27.31 所示。

（4）在图 27.32 所示的“New Project--Destination”窗口中，单击“Browse”按钮，弹出“打开”对话框，如图 27.33 所示。在“打开”对话框中指定要创建工程文件的路径及文件名，工程文件的扩展名为.hhp，在文件名文本框中写入“mrHelp”，然后单击“打开”按钮。

（5）在图 27.32 所示界面中，选择路径，单击“下一步”按钮，弹出如图 27.34 所示的“New Project -- Existing Files”对话框，该对话框中设置是否将现有的目录文件（.hhc）、索引文件（.hhk）和主题文件（.htm）添加到工程中。如果选择这些复选框，则需要指定文件的名称和位置。本示例中直接单击“下一步”按钮，弹出如图 27.35 所示界面。

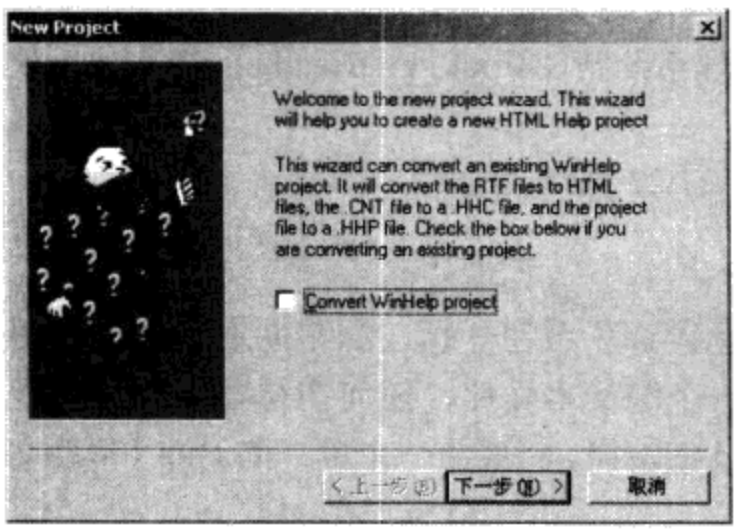


图 27.31 “New Project”对话框

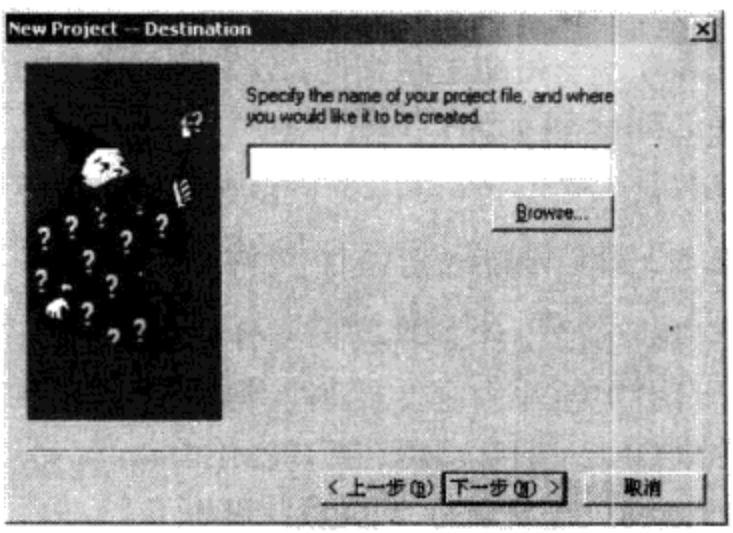


图 27.32 “New Project - - Destination”对话框

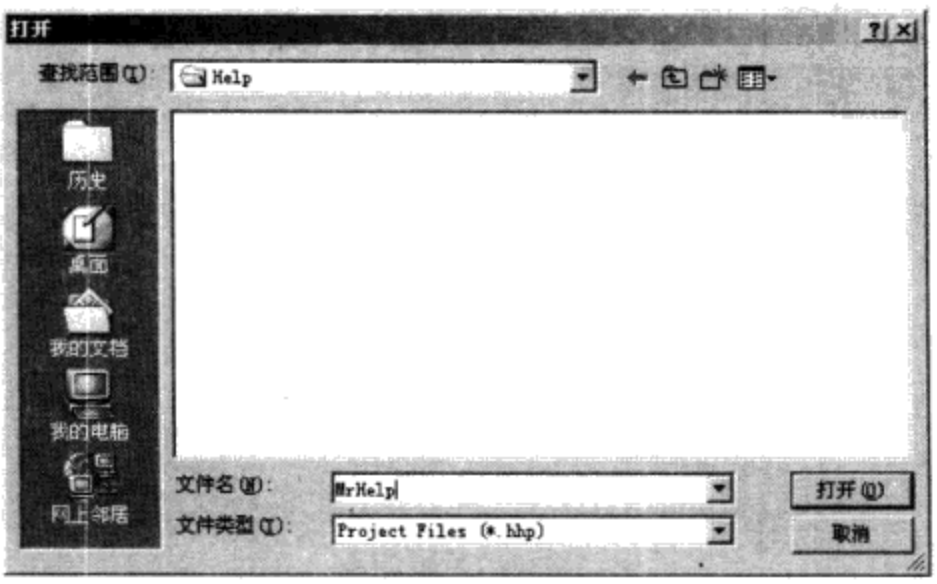


图 27.33 “打开”对话框

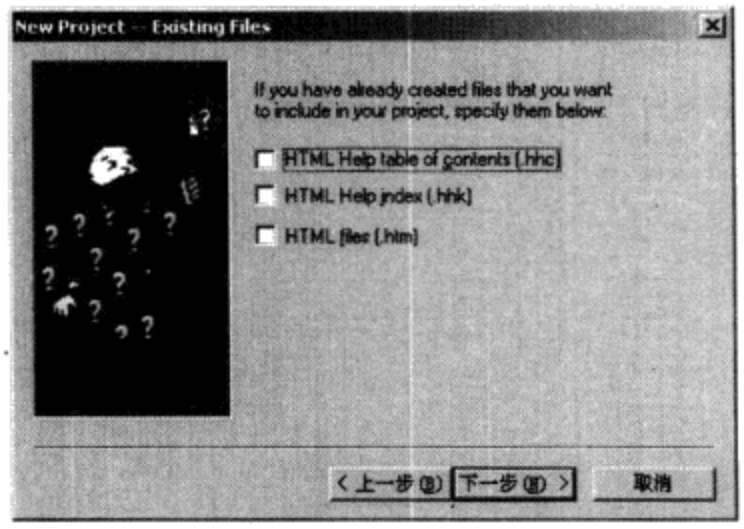


图 27.34 “New Project -- Existing Files”对话框

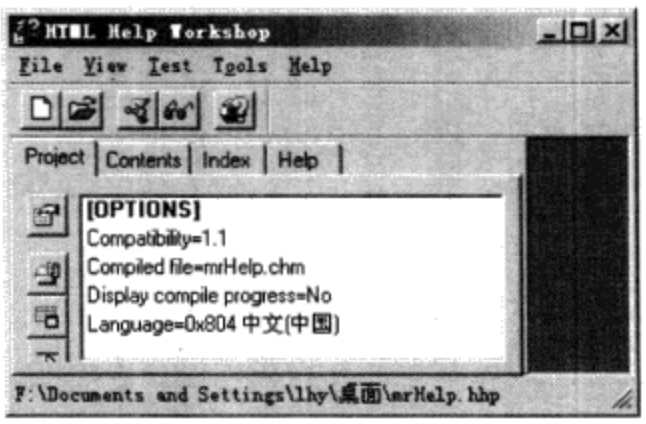


图 27.35 “HTML Help Workshop”中的工程窗口

(6) 在图 27.35 的左侧有 7 个按钮，它们的功能说明如表 27.1 所示。

表 27.1 工程窗口中的功能按钮

图　标	名　称	说　明
	Change project options	改变工程选项
	Add/Remove topic files	添加/删除主题文件
	Add/Modify window definitions	添加/修改、定义窗口
	HtmlHelp API information	HtmlHelp API 信息
	View HTML source	查看 HTML 源文件
	Save project，contents and index files	保存工程、目录和索引文件
	Save all project file and compile	保存和编译所有文件

(7) 在图 27.35 中，单击按钮，出现工程选项对话框，如图 27.36 所示，在“Title”文本框中输入“财务凭证管理系统”，单击“确定”按钮。

(8) 在图 27.35 中，单击按钮，弹出“Topic Files”对话框，如图 27.37 所示。单击“Add”按钮，弹出“打开”对话框，如图 27.38 所示，将作为封面页的 HTML 文件添加到工程中。封面页文件必须存在，否则编译的帮助文件运行后，系统会报错。至此，HTML 帮助项目文件的制作过程即可完成，其结果如图 27.39 所示。

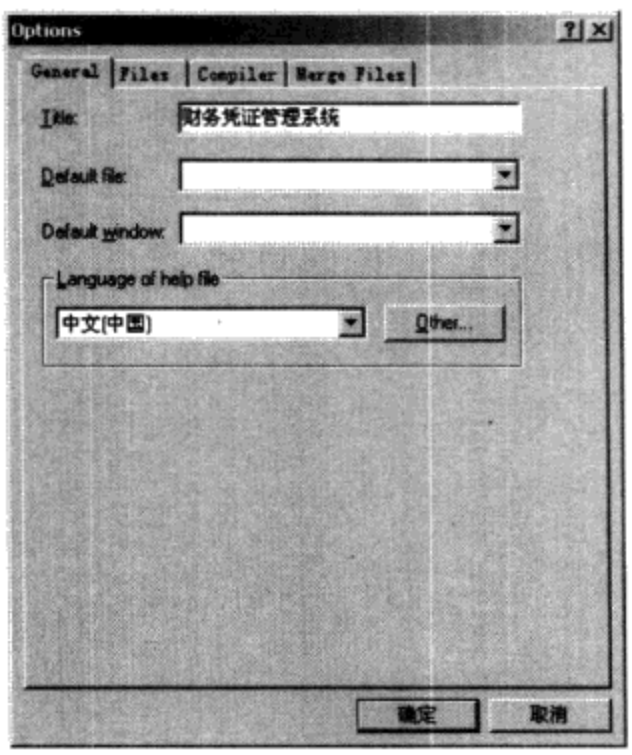


图 27.36 工程选项对话框

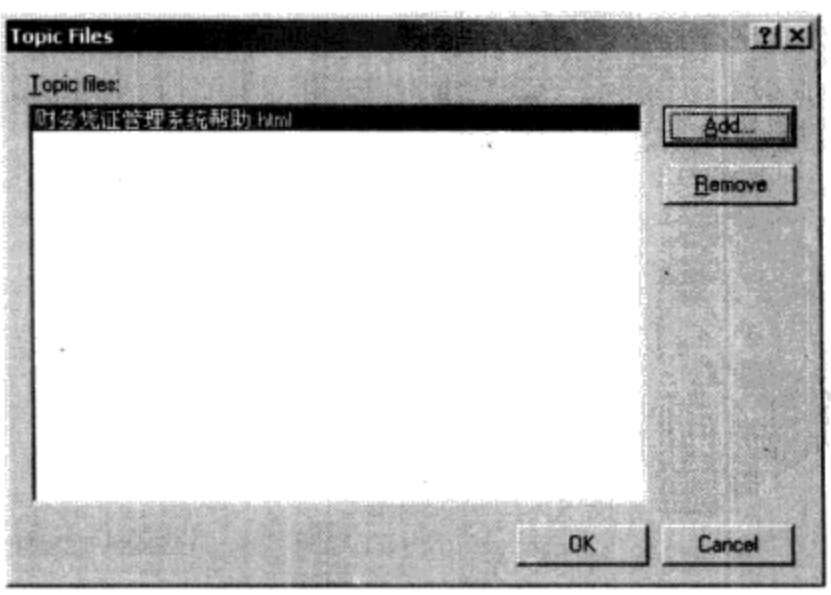


图 27.37 “Topic Files”对话框

图 27.38　添加封面文件

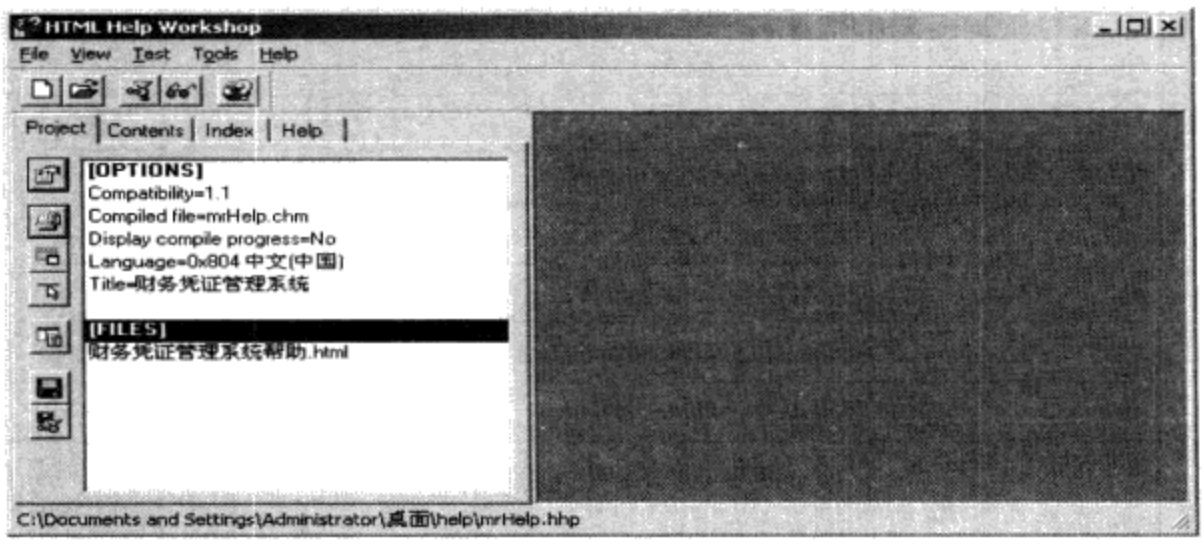


图 27.39　HTML 帮助项目文件

27.2.5　目录和目录文件

目录文件是包含帮助文件中所有目录的一个 HTML 格式文件，这些目录包括条目名称（也称为条目标题）、跳转主题的路径及条目属性等。创建 HTML 帮助目录文件的步骤如下。

（1）在图 27.39 所示对话框，选择“Contens”选项卡，弹出“Table of Contents Not Specified”对话框，选择“Create a new contents file”选项，单击“OK”按钮，创建目录文件，如图 27.40 所示。将目录文件保存到工程目录下，如图 27.41 所示，创建目录文件后的 Contens 选项卡如图 27.42 所示。

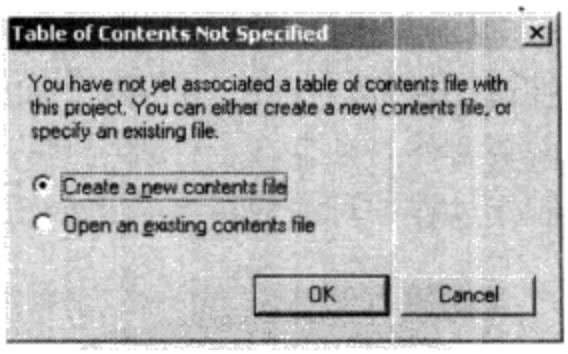


图 27.40 “Table of Contents Not Specified”对话框

图 27.41　目录文件保存对话框

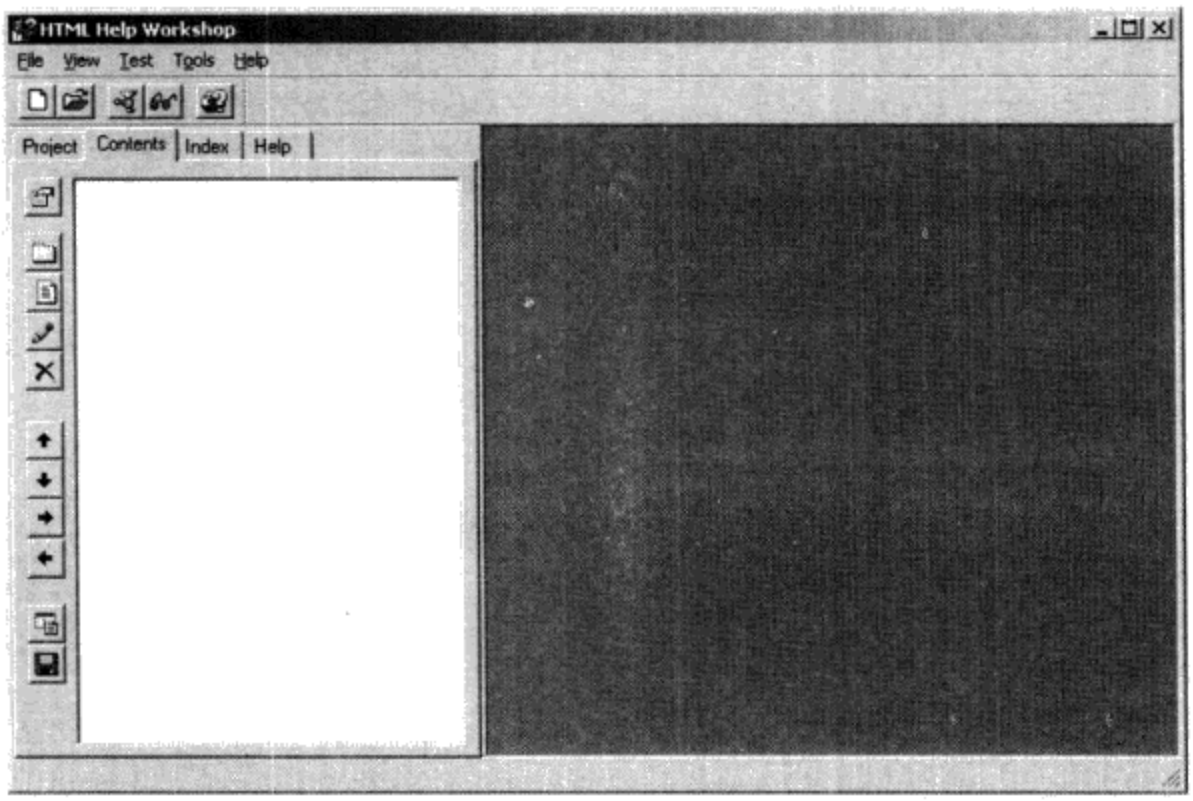


图 27.42　创建目录文件后的 Contens 选项卡

(2) 图 27.42 所示的 Contens 选项卡窗口左侧有 11 个按钮，它们的功能说明如表 27.2 所示。

表 27.2　　Contens 选项卡中的功能按钮说明

图　标	名　称	说　明
	Contens properties	目录属性
	Insert a heading	插入标题
	Insert a page	插入主题
	Edit selection	编辑
	Delete selection	删除
	Move selection up	上移
	Move selection down	下移
	Move selection right	右移
	Move selection left	左移
	View HTML source	查看 HTML 源文件
	Save file	保存文件

(3) 目录文件可以通过按钮和按钮将帮助文件左侧的文件目录和右侧的文件内容链接起来。当单击这两个按钮时，弹出“Table of Contents Entry”对话框。在“Entry title”文本框中输入主题名，单击“Add”按钮，添加相应的主题文件，单击“Add/Edit”按钮进行链接，如图 27.43 所示。

(4) 单击按钮，弹出“Table of Contents Entry”对话框，在“Entry title”文本框中输入“概述”，然后单击“Add/Edit”按钮，弹出“Path or URL”对话框，如图 27.44 所示。

(5) 在图 27.44 所示对话框中，单击“Browse”按钮，弹出“打开”对话框，找到要链接的文件，如图 27.45 所示，单击“打开”按钮。

(6) 返回如图 27.44 所示的“Path or Url”对话框，单击“OK”按钮，返回如图 27.46 所示“Table of Contents Entry”对话框，单击“确定”按钮。返回 Contens 选项卡，如图 27.47 所示。

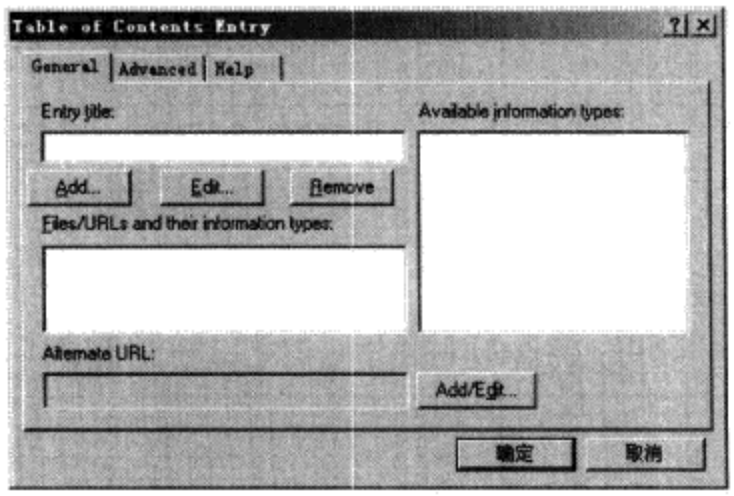


图 27.43 “Table of Contents Entry”对话框

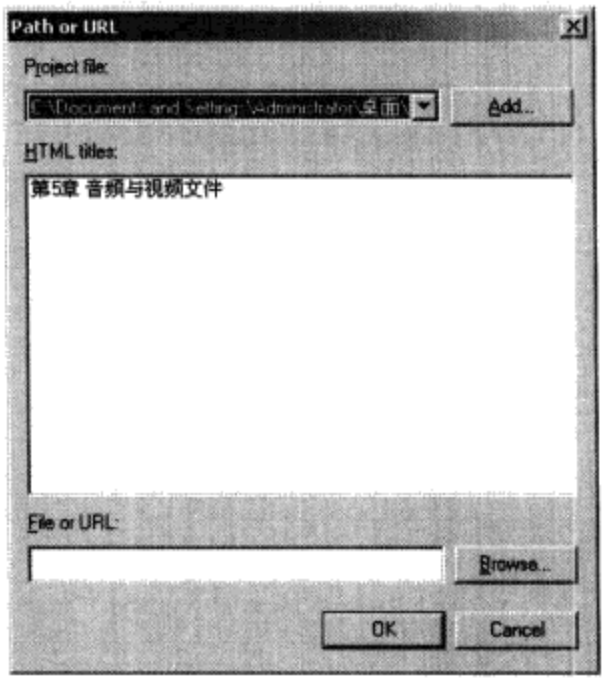


图 27.44 “Path or URL”对话框

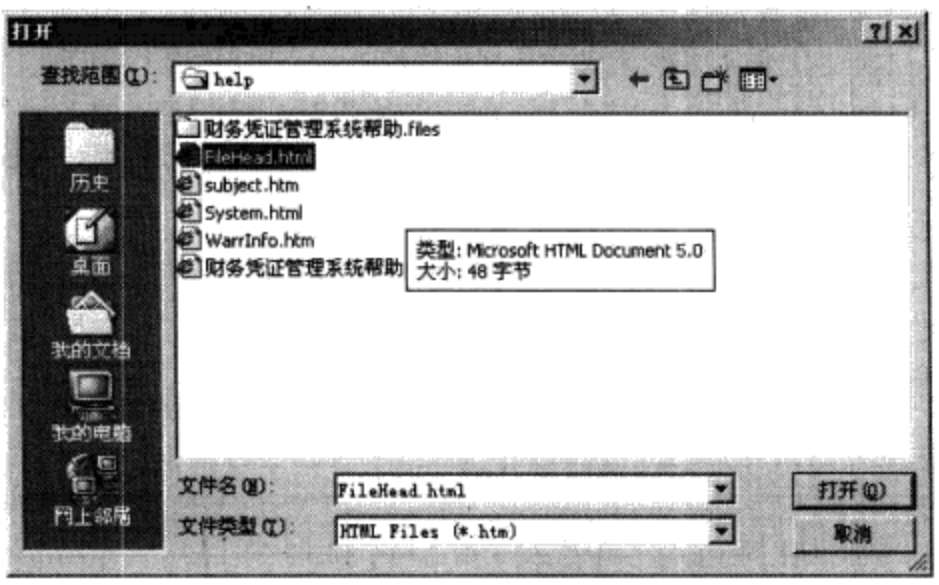


图 27.45 “打开”对话框

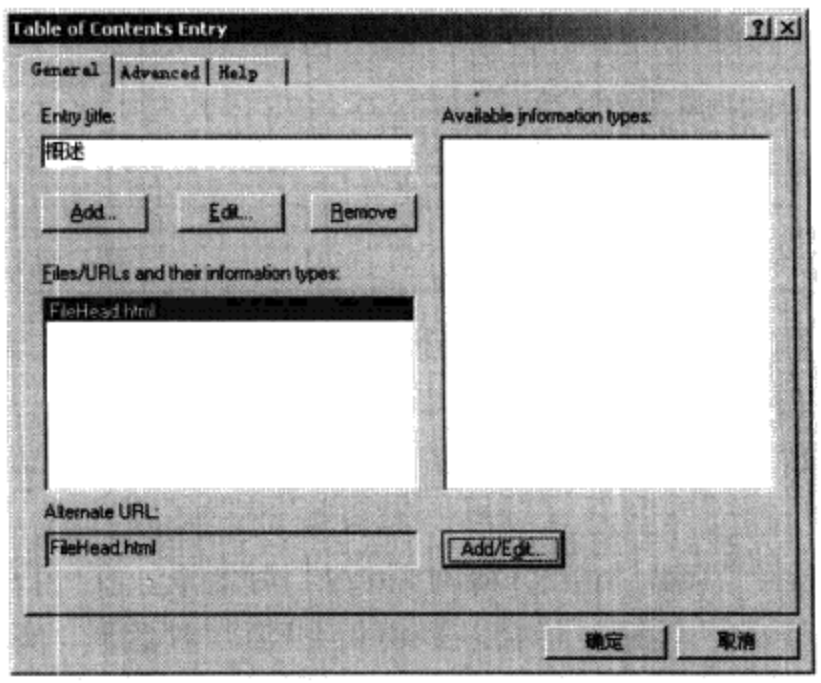


图 27.46 “Table of Contents Entry”对话框

(7) 在图 27.47 所示对话框中，单击按钮，弹出“HTML Help Workshop”对话框，如图 27.48 所示。单击“是”按钮，重复步骤 3，依次添加所有文件目录。添加完成后的目录文件如图 27.49 所示。

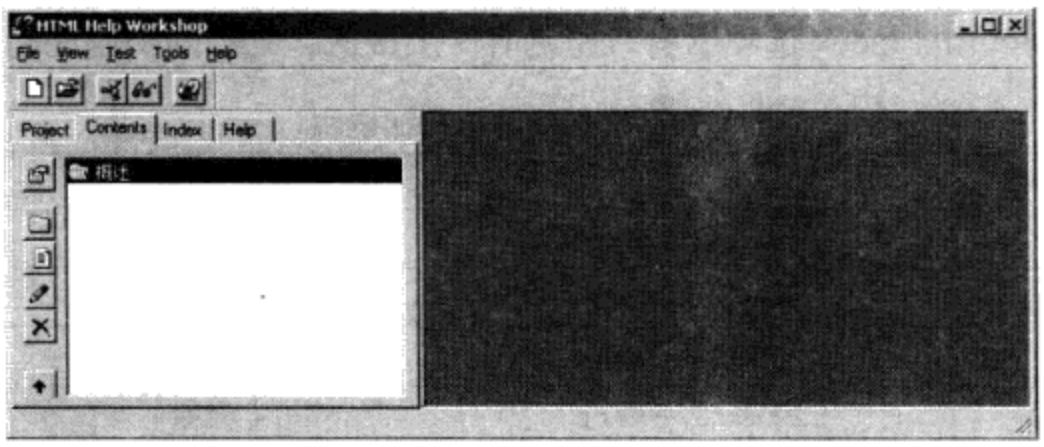


图 27.47　Contens 选项卡

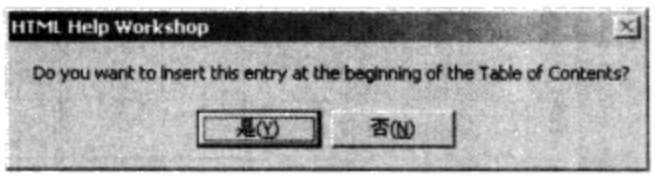


图 27.48　“HTML Help Workshop”对话框

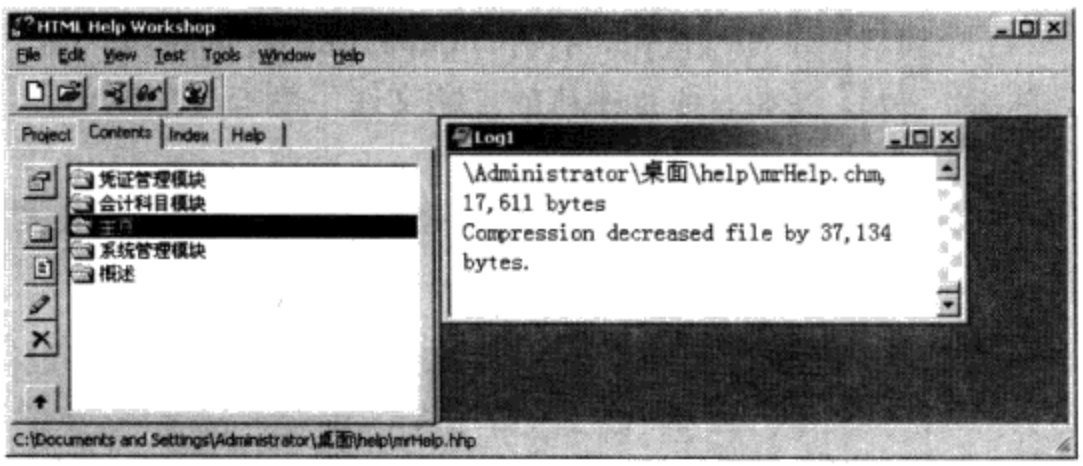


图 27.49　添加所有的文件目录

27.2.6　关键字和索引文件

关键字是将一个或多个帮助主题文件关联在一起的字、词和短语等，包含若干关键字的文件称为索引文件。索引文件是一个以.hhk 为扩展名的 HTML 文件。创建 HTML 帮助关键字和索引文件的步骤如下。

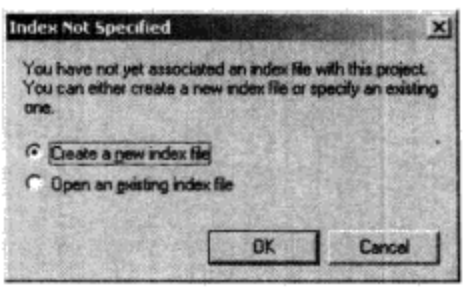


图 27.50　“Index Not Specified”对话框

（1）在图 27.49 所示窗口中，单击“Index”选项卡，弹出“Index Not Specified”对话框，选择“Create a new index file”选项，单击“OK”按钮，创建索引文件，如图 27.50 所示。将索引文件保存到工程目录下，如图 27.51 所示，创建索引文件后的 Index 选项卡如图 27.52 所示。

图 27.51　索引文件保存对话框

图 27.52　Index 选项

（2）图 27.52 所示的 Index 选项卡窗口左侧有 9 个按钮，它们的功能说明如表 27.3 所示。

表 27.3　　索引文件的功能按钮说明

图　标	名　称	说　明
	Index properties	目录属性
	Insert a keyword	插入关键字
	Edit selection	编辑
	Delete selection	删除
	Move selection up	上移
	Move selection down	下移
	Move selection right	右移
	Move selection left	左移
	Sort every keyword at same level as selection	按字母排序

（3）在图 27.52 所示窗口中单击按钮，打开“Index Entry”对话框，在“Keyword”文本框中输入关键字，单击“Add”按钮，添加相应的主题文件，单击“Add/Edit”按钮，将关键字与相关联的主题文件进行链接，如图 27.53 所示。

（4）在图 27.52 中，单击按钮，在“Index Entry”对话框的“Keyword”文本框中输入“one”，单击“Add”按钮，弹出“Path or URL”对话框，如图 27.54 所示。

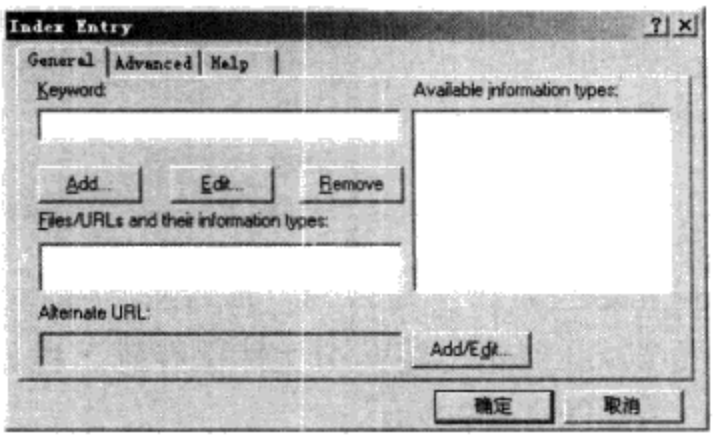


图 27.53　“Index Entry”对话框

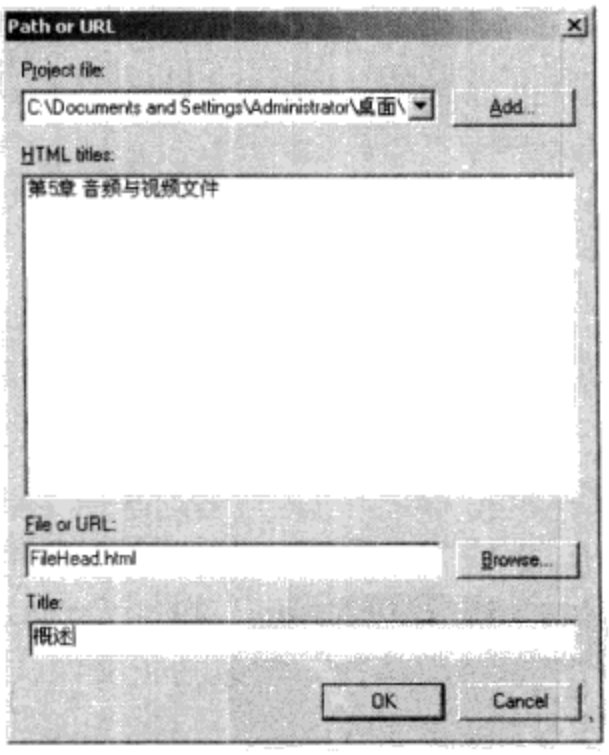


图 27.54　“Path or URL”对话框

（5）在图 27.54 中单击“Browse”按钮，弹出“打开”对话框，找到要链接的文件，如图 27.55 所示。单击“打开”按钮，返回如图 27.54 所示“Path or URL”对话框，在“Title”文本框输入“概述”，单击“OK”按钮，返回如图 27.56 所示的“Index Entry”对话框，然后单击“Add/Edit”按钮，弹出“Path or URL”对话框，添加链接文件，单击“OK”按钮，返回“Index Entry”对话框，如图 27.56 所示，单击“确定”按钮，返回 Index 选项卡，如图 27.57 所示。

（6）在图 27.57 中单击按钮，弹出“HTML Help Workshop”对话框，如图 27.58 所示，单击“是”按钮，重复步骤 3，依次添加所有索引文件。添加完成后的索引文件如图 27.59 所示。

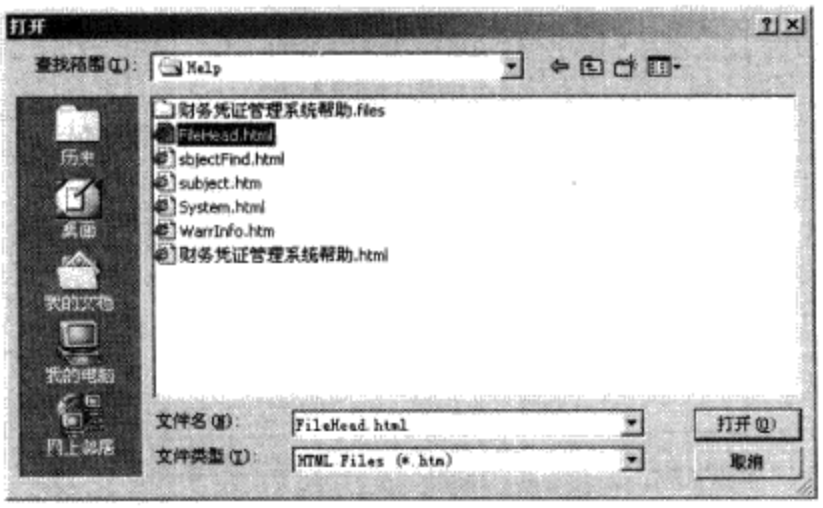


图 27.55　打开文件对话框

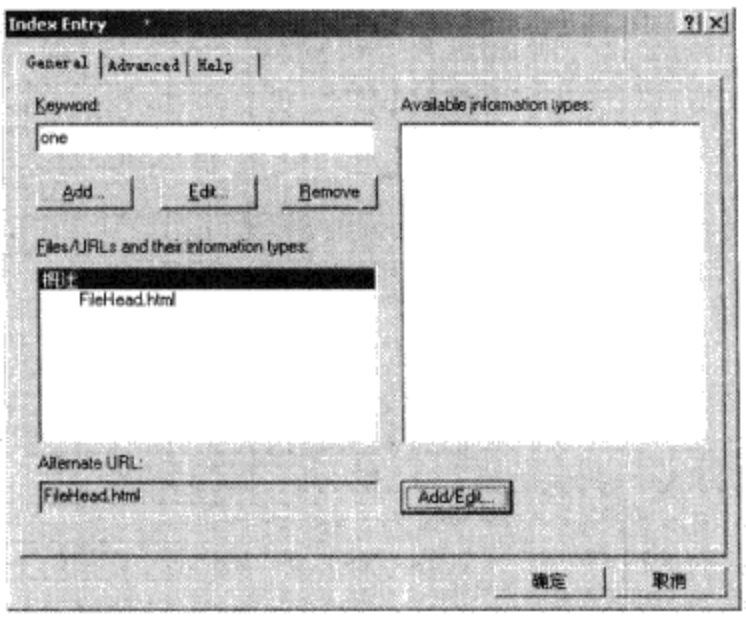


图 27.56　“Index Entry”对话框

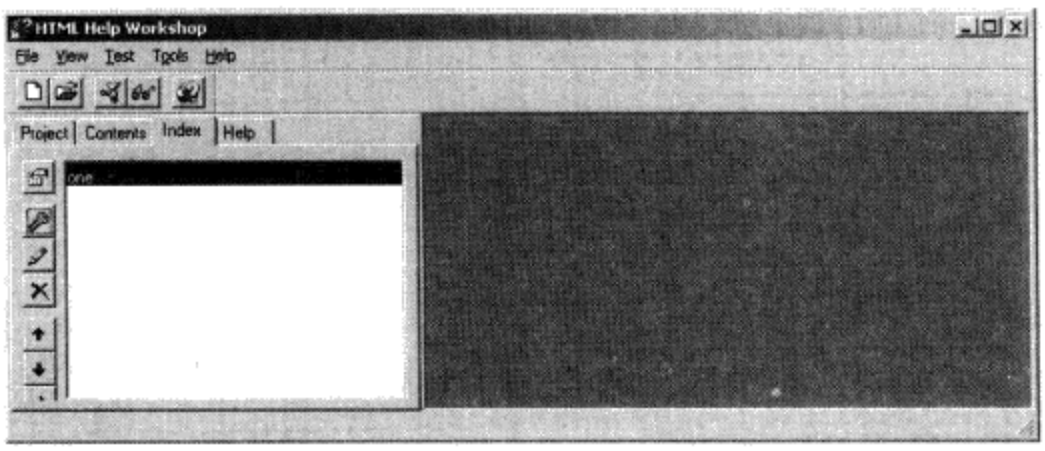


图 27.57　Index 选项卡

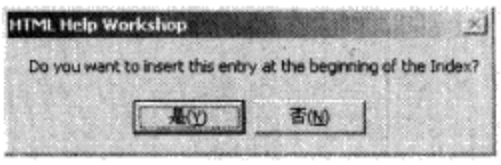


图 27.58　“HTML Help Workshop”对话框

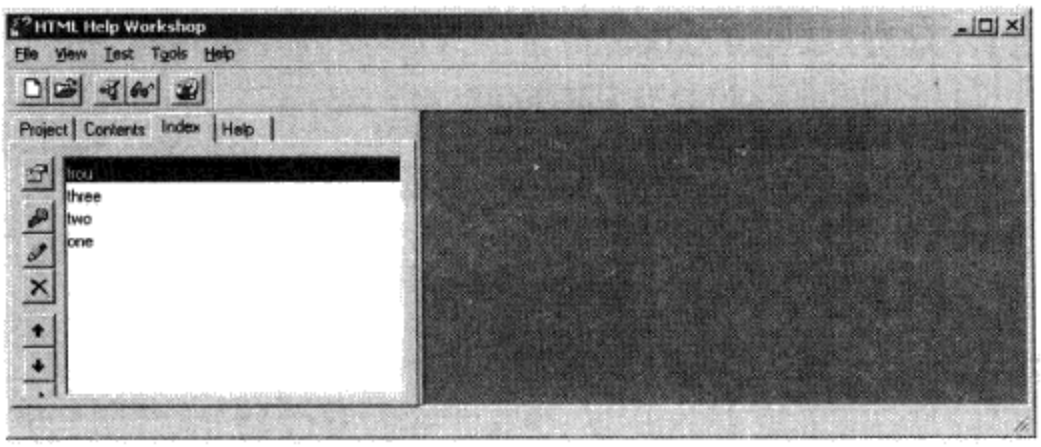


图 27.59　Index 选项

27.2.7　制作.chm 文件

使用 HTML Help Workshop 软件将 HTML 帮助主题文件、HTML 帮助项目文件、目录和目录文件、关键字和索引文件等编译成.chm（帮助系统）文件，步骤如下。

（1）在图 27.59 中单击“Project”选项卡，单击“Add/Modify window definition”按钮，在弹出的“Window Types”对话框中设置帮助文件的风格，如图 27.60 所示。

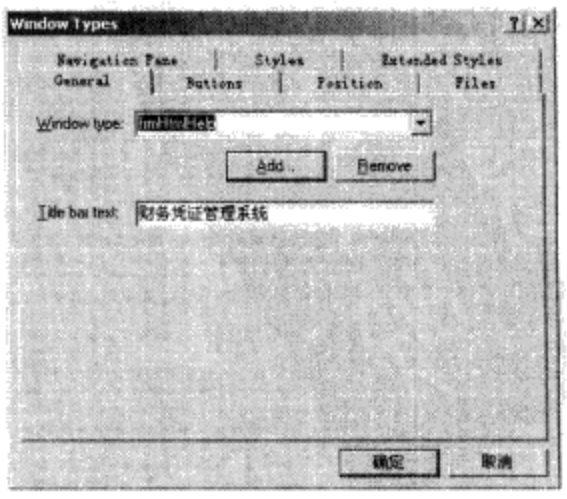


图 27.60　“Window Types”对话框

（2）在菜单栏中选择“File”/“Compile”选项，如图 27.61 所示，弹出“HTML Help Workshop”对话框，如图 27.62 所示。提示是否要编译工程，单击“是”按钮，弹出“Create a Compiled file”对话框，选择编译文件路径，如图 27.63 所示，默认路径为工程路径下。单击“Compile”按钮，开始编译文件。

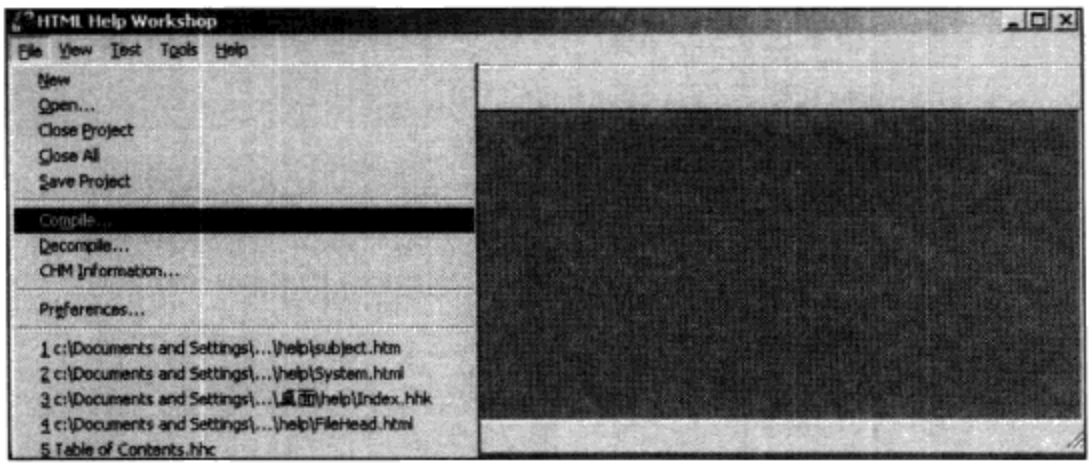


图 27.61　选择“File”/“Compile”选项

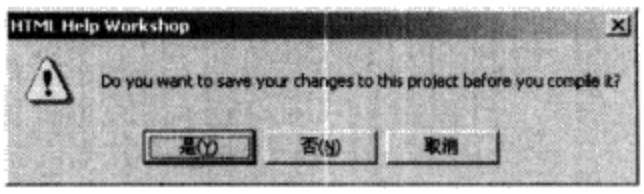


图 27.62　“HTML Help Workshop”对话框

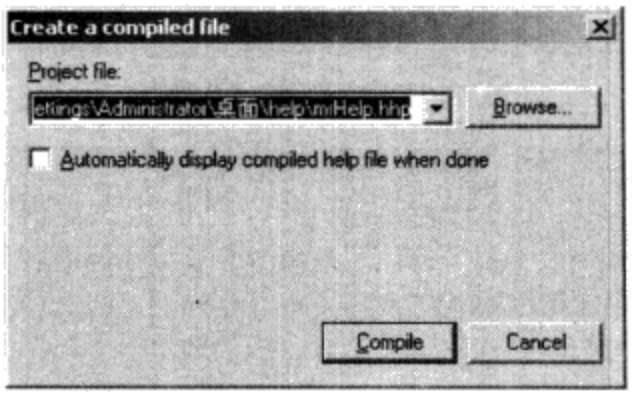


图 27.63　“Create a compiled file”对话框

（3）文件编译成功，如图 27.64 所示，在工程的根目录下找到经过编译的帮助文件 mrHelp.chm，双击打开该文件，效果如图 27.65 所示。

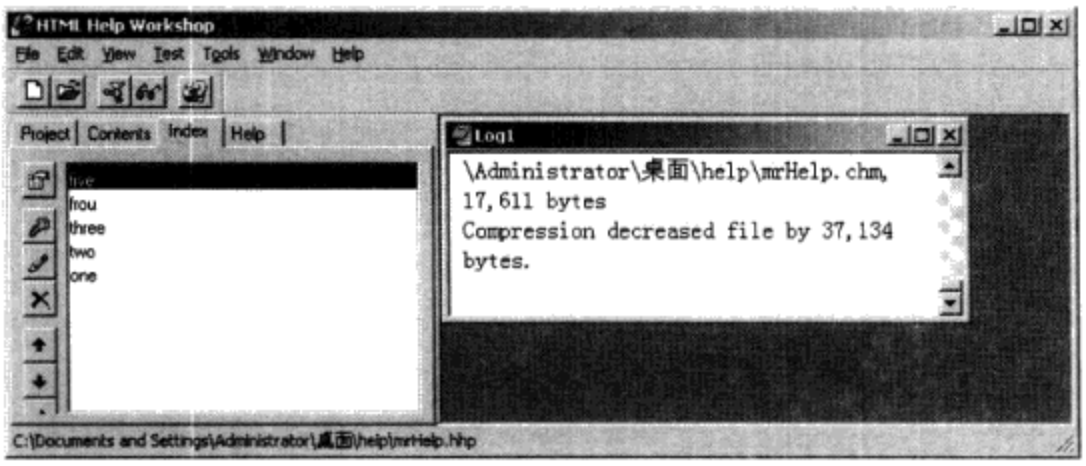


图 27.64　文件编译成功

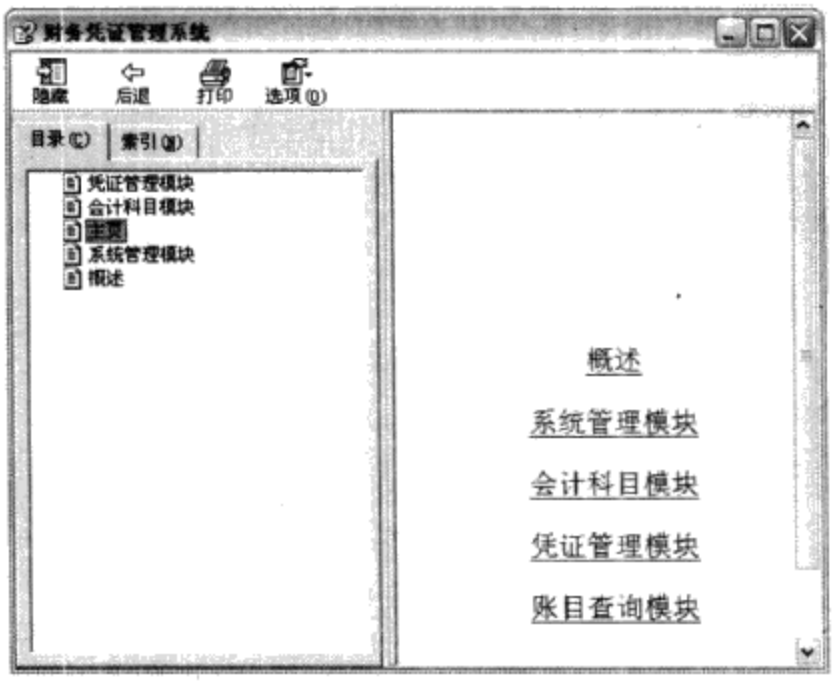


图 27.65　mrHelp.chm 文件效果

27.2.8 在 C#中使用.chm 帮助文件

将编译好的.chm 帮助文件添加到 C# 编写的程序中时，需要调用 Help 类的 ShowHelp 和 ShowHelpIndex 静态方法。下面对这两个方法进行详细介绍。

（1）ShowHelp 方法。显示帮助文件的内容。该方法有 2 种重载形式，它们的语法形式分别如下。

语法 1:

```
public static void ShowHelp (Control parent,string url)
```

参数说明如下。

parent：标识“帮助”对话框的父级的 Control。

url：帮助文件的路径和名称。

返回值：显示指定 URL 处的帮助文件内容。

例如，显示 C 盘下的 charmap.chm 帮助文件。代码如下：

```
Help.ShowHelp(TextBox1, "c:\\charmap.chm");
```

语法 2:

```
public static void ShowHelp (Control parent,string url,HelpNavigato rnavigator)
```

参数说明如下。

parent：标识“帮助”对话框的父级的 Control。

url：帮助文件的路径和名称。

navigator：HelpNavigator 值之一。HelpNavigator 值及说明如表 27.4 所示。

表 27.4　HelpNavigator 值及说明

值	说　明
TopicId	帮助文件将打开到由数字主题标识符指示的主题
AssociateIndex	帮助文件将打开到特定主题首字母的索引项
Find	帮助文件将打开到搜索页
Index	帮助文件将打开到索引
KeywordIndex	如果存在具有指定索引项的主题，则帮助文件打开到该主题；否则将显示与指定关键字最接近的索引项
TableOfContents	帮助文件将打开到目
Topic	如果指定的主题存在，则帮助文件将打开到指定的主题

例如，显示 C 盘下的 charmap.chm 帮助文件，并切换到“索引”选项卡。代码如下：

```
Help.ShowHelp(TextBox1, "c:\\charmap.chm", HelpNavigator..Index);
```

（2）ShowHelpIndex 方法。显示指定帮助文件的索引。

语法:

```
public static void ShowHelpIndex (Control parent,string url)
```

例如，显示帮助文件的索引。代码如下：

```
Help.ShowHelpIndex(this, c:\\charmap.chm);
```

示例 27-02　在 Windows 应用程序中调用.chm 帮助文件

本示例实现的是，当程序运行时，单击“help”按钮，在程序中调用.chm 帮助文件。程序主要代码。

代码 27-02　光盘位置：光盘\mingrisoft\sl\27\27_02

```
private void button1_Click (object sender, EventArgs e)
{
    string helpfile = Application.StartupPath.Substring(0,Application.StartupPath.Substring(0,Application.Startup
Path.LastIndexOf("\\")).LastIndexOf("\\"));                    //获取帮助文件的所在目录
    helpfile+=@"\help\mrHelp.chm";                              //设置帮助文件的路径
    Help.ShowHelp(this, helpfile);                              //显示帮助文件
    Help.ShowHelpIndex(this, helpfile);                         //显示指定帮助的索引
}
```

第 6 篇

图形图像与多媒体

C#

第 28 章

GDI+图形图像

- GDI+概述
- 直线、曲线和图形
- 图像、位图和图元文件
- 坐标系和坐标变换
- GDI+绘制图表的应用

无论 Windows 应用程序，还是 ASP.NET Web 网站，都经常需要使用图形或图像来更形象、更直观地描绘一些内容。例如，进销存管理系统中使用柱形图来分析每月的商品销售情况，或者使用折线图分析每月的网站流量等。本章将详细介绍如何在 Visual Studio 2008 开发环境中使用 GDI+绘制图形图像。

28.1 GDI+概述

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）。

一般来说，有 3 种基本类型的绘图界面，分别为屏幕上的窗口和控件、要发送给打印机的页面以及内存中的位图和图像。Graphics 类封装了一个 GDI+绘图界面，因此该类提供了可以在以上 3 种绘图界面上绘图的功能。另外，程序开发人员还可以使用该类绘制文本、线条、矩形、曲线、多边形、椭圆、圆弧和贝塞尔样条等。

GDI+基类的主要命名空间及说明如表 28.1 所示。

表 28.1　　GDI + 基类的主要命名空间及说明

命 名 空 间	说　　明
System.Drawing	包含与基本绘图功能有关的大多数类、结构、枚举和委托
System.Drawing.Drawing2D	为大多数高级 2D 和矢量绘图操作提供了支持，包括消除锯齿、几何转换和图形路径
System.Drawing.Imaging	帮助处理图像（位图和 GIF 文件等）的各种类
System.Drawing.Printing	把打印机或打印预览窗口作为输出设备时使用的类
System.Drawing.Design	一些预定义的对话框、属性表和其他用户界面元素，与在设计期间扩展用户界面相关
System.Drawing.Text	对字体和字体系列执行更高级操作的类

28.2 直线、曲线和图形

28.2.1 向量图形概述

程序开发过程中，GDI+中的一些基本构造块可用于绘制二维图形，GDI+支持的基本构造块有：直线、矩形、椭圆、弧线、多边形、基数样条和贝塞尔样条等。

Graphics 类提供了绘制 GDI+基本构造块的各种方法，包括 DrawLine 方法、DrawRectangle 方法、DrawEllipse 方法、DrawPolygon 方法、DrawArc 方法、DrawCurve 方法（针对基数样条）和 DrawBezier 方法等。这些方法中的每一种都可以重载，即每种方法都支持几个不同的参数列表。例如，DrawLine 方法的一种形式接收一个 Pen 对象和 4 个整数，而它的另一种形式则接收一个 Pen 对象和两个 Point 对象。

绘制直线、矩形和贝塞尔样条的方法具有多个伴随方法，可在一个调用中绘制如下项：DrawLines 、 DrawRectangles 和 DrawBeziers 。 DrawCurve 方法也有一个伴随方法 DrawClosedCurve，该伴随方法能够通过连接曲线起点和终点的方式来闭合曲线。

Graphics 类的所有绘制方法都应与 Pen 对象一起使用。若要进行图形绘制，至少必须创建两个对象：Graphics 对象和 Pen 对象。

28.2.2 GDI+中的直线和矩形

1．绘制直线

绘制直线时，可以调用 Graphics 类中的 DrawLine 方法，该方法为可重载方法，它主要用

来绘制一条连接由坐标对指定的两个点的线条，其常用格式有以下两种。

（1）绘制一条连接两个 Point 结构的线。

语法：

```
public void DrawLine (Pen pen,Point pt1,Point pt2)
```

参数说明如下。

pen：Pen 对象，它确定线条的颜色、宽度和样式。

pt1：Point 结构，它表示要连接的第一个点。

pt2：Point 结构，它表示要连接的第二个点。

（2）绘制一条连接由坐标对指定的两个点的线条。

语法：

```
public void DrawLine (Pen pen,int x1,int y1,int x2,int y2)
```

DrawLine 方法中各参数及说明如表 28.2 所示。

表 28.2　DrawLine 方法参数及说明

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

示例 28-01　绘制直线

本示例实现的是：当程序运行时，单击“绘制”按钮，在窗体中的指定位置绘制一条直线。

代码 28-01　光盘位置：光盘\mingrisoft\sl\28\01

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个实例对象，然后调用 Graphics 对象的 DrawLine 方法在窗体中绘制一条直线。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();          //实例化一个封装 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 2);                 //实例化一个用于绘制直线和曲线的对象
    graphics.DrawLine(myPen, 50, 30, 170, 30);          //绘制一条连接由坐标对指定的两个点的线条
}
```

运行本示例，效果如图 28.1 所示。

图 28.1　绘制直线

2．绘制矩形

绘制矩形时，可以调用 Graphics 类中的 DrawRectangle 方法，该方法为可重载方法，它主要用来绘制由坐标对、宽度和高度指定的矩形，其常用格式有以下两种。

（1）绘制由 Rectangle 结构指定的矩形。

语法：

```
public void DrawRectangle (Pen pen,Rectangle rect)
```

参数说明如下。

pen：Pen 对象，它确定矩形的颜色、宽度和样式。

rect：表示要绘制矩形的 Rectangle 结构。

例如，声明一个 Rectangle 结构，代码如下：

```
Rectangle rect = new Rectangle(0, 0, 80, 50);
```

（2）绘制由坐标对、宽度和高度指定的矩形。

语法：

```
public void DrawRectangle (Pen pen,int x,int y,int width,int height)
```

DrawRectangle 方法中各参数及说明如表 28.3 所示。

表 28.3　DrawRectangle 方法参数及说明

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

示例 28-02　绘制矩形

本示例实现的是：当程序运行时，单击“绘制”按钮，在窗体中的指定位置绘制一个指定大小的矩形。

代码 28-02　光盘位置：光盘\mingrisoft\sl\28\02

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个实例对象，然后调用 Graphics 对象的 DrawRectangle 方法。在窗体中绘制一个指定大小的矩形。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();          //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 2);                 //实例化一个用于绘制直线和曲线的对象
    graphics.DrawRectangle(myPen, 70, 20, 80, 50);      //绘制由坐标对、宽度和高度指定的矩形
}
```

运行本示例，效果如图 28.2 所示。

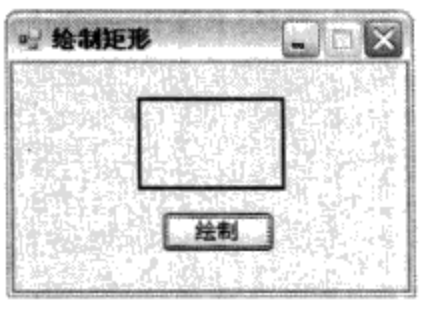


图 28.2　绘制矩形

28.2.3　GDI+中的椭圆和弧

1．绘制椭圆

绘制椭圆时，可以调用 Graphics 类中的 DrawEllipse 方法，该方法为可重载方法，它主要用来绘制边界由 Rectangle 结构指定的椭圆，其常用格式有以下两种。

（1）绘制边界由 Rectangle 结构指定的椭圆。

语法：

```
public void DrawEllipse (Pen pen,Rectangle rect)
```

参数说明如下。

pen：Pen 对象，它确定曲线的颜色、宽度和样式。

rect：Rectangle 结构，它定义椭圆的边界。

（2）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：

```
public void DrawEllipse (Pen pen,int x,int y,int width,int height)
```

DrawEllipse 方法中各参数及说明如表 28.4 所示。

表 28.4　DrawEllipse 方法参数及说明

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

示例 28-03　绘制椭圆

本示例实现的是：当程序运行时，单击“绘制”按钮，在窗体中的指定位置绘制一个指定边框大小的椭圆。

代码 28-03　光盘位置：光盘\mingrisoft\sl\28\03

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个实例对象，然后定义一个 Rectangle 结构，以用来控制椭圆的边框大小，最后调用 Graphics 对象的 DrawEllipse 方法在窗体中绘制一个指定边框大小的椭圆。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 3);                           //实例化一个用于绘制直线和曲线的对象
    Rectangle myRectangle = new Rectangle(70, 20, 100, 60);       //实例化一个用于表示一个矩形的位置和大小的对象
    graphics.DrawEllipse(myPen, myRectangle);                     //绘制由指定边界限定的椭圆
}
```

运行本示例，效果如图 28.3 所示。

图 28.3　绘制椭圆

2．绘制圆弧

绘制圆弧时，可以调用 Graphics 类中的 DrawArc 方法，该方法为可重载方法，它主要用来绘制一段弧线，其常用格式有以下两种。

（1）绘制一段弧线，它表示由 Rectangle 结构指定的椭圆的一部分。

语法：

```
public void DrawArc (Pen pen,Rectangle rect,float startAngle,float sweepAngle)
```

DrawArc 方法中各参数及说明如表 28.5 所示。

表 28.5　DrawArc 方法参数及说明

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）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：

```
public void DrawArc (Pen pen,int x,int y,int width,int height,int startAngle,int sweepAngle)
```

DrawArc 方法中各参数及说明如表 28.6 所示。

表 28.6　　DrawArc 方法参数及说明

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）

示例 28-04　绘制圆弧

本示例实现的是：当程序运行时，单击“绘制”按钮，在窗体中的指定位置绘制一段指定长度的圆弧。

代码 28-04　光盘位置：光盘\mingrisoft\sl\28\04

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个实例对象，然后定义一个 Rectangle 结构，用来控制椭圆的边框大小，最后调用 Graphics 对象的 DrawArc 方法在指定的边框区域内绘制一段圆弧。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 5);                           //实例化一个用于绘制直线和曲线的对象
    Rectangle myRectangle = new Rectangle(70, 20, 100, 60);       //实例化一个用于表示矩形的位置和大小的对象
    graphics.DrawArc(myPen, myRectangle,0,90);                    //绘制一段圆弧
}
```

运行本示例，效果如图 28.4 所示。

图 28.4　绘制圆弧

28.2.4　GDI+中的多边形

绘制多边形，需要 Graphics 对象、Pen 对象和 Point（或 PointF）对象数组。Graphics 对象提供 DrawPolygon 方法，Pen 对象存储用于呈现多边形的线条属性，例如，宽度和颜色等，Point（或 PointF）对象数组存储多边形的各个顶点。Pen 对象和 Point（或 PointF）对象数组作为参数传递给 DrawPolygon 方法，该方法为可重载方法，其常用格式有以下两种。

（1）绘制由一组 Point 结构定义的多边形。

语法：

```
public void DrawPolygon (Pen pen,Point[] points)
```

参数说明如下。

pen：Pen 对象，它确定多边形的颜色、宽度和样式。

points：Point 结构数组，这些结构表示多边形的顶点。

（2）绘制由一组 PointF 结构定义的多边形。

语法：

```
public void DrawPolygon (Pen pen,PointF[] points)
```

参数说明如下。

pen：Pen 对象，它确定多边形的颜色、宽度和样式。

points：PointF 结构数组，这些结构表示多边形的顶点。

示例 28-05　绘制多边形

本示例实现的是：当程序运行时，单击“绘制”按钮，在窗体中绘制一个多边形。

代码 28-05　光盘位置：光盘\mingrisoft\sl\28\05

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个对象，然后实例化 6 个 Point 对象，用来作为多边形的顶点，声明一个 Point 结构数组，并将已经实例化的 6 个 Point 对象赋值给该数组，最后调用 Graphics 对象的 DrawPolygon 方法绘制一个多边形。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                          //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Red, 3);                                  //实例化一个用于绘制直线和曲线的对象
    Point point1 = new Point(80, 20);                                   //实例化一个坐标点 point1
    Point point2 = new Point(40, 50);                                   //实例化一个坐标点 point2
    Point point3 = new Point(80, 80);                                   //实例化一个坐标点 point3
    Point point4 = new Point(160, 80);                                  //实例化一个坐标点 point4
    Point point5 = new Point(200, 50);                                  //实例化一个坐标点 point5
    Point point6 = new Point(160, 20);                                  //实例化一个坐标点 point6
    Point[] myPoints = { point1, point2, point3, point4, point5, point6 };  //实例化一个坐标点数组
    graphics.DrawPolygon(myPen, myPoints);                              //绘制多边形
}
```

运行本示例，效果如图 28.5 所示。

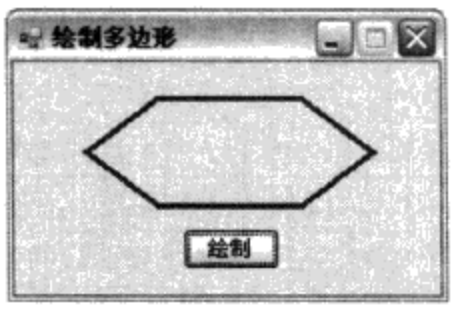


图 28.5　绘制多边形

28.2.5　GDI+中的基数样条

基数样条是一连串单独的曲线，这些曲线连接起来形成一条较大的曲线。基数样条由点的数组和张力参数指定，样条平滑地经过数组中的每个点，曲线的陡度上没有尖角和突然的变化。

绘制基数样条，需要 Graphics 对象、Pen 对象和 Point（或 PointF）对象数组。Graphics 类的实例提供了 DrawCurve 方法用于绘制基数样条，Pen 对象存储基数样条的属性（如线宽和颜色等），Point（或 PointF）对象数组存储曲线将要经过的点。DrawCurve 方法为可重载方法，其常用格式有以下 4 种。

（1）绘制经过一组指定 Point 结构的基数样条。

语法：

```
public void DrawCurve (Pen pen,Point[] points)
```

参数说明如下。

pen：Pen 对象，它确定曲线的颜色、宽度和高度。

points：Point 结构数组，这些结构定义样条。

（2）使用指定的张力，绘制经过一组指定 Point 结构的基数样条。

语法：

```
public void DrawCurve (Pen pen,Point[] points,float tension)
```

参数说明如下。

pen：Pen 对象，它确定曲线的颜色、宽度和高度。

points：Point 结构数组，这些结构定义样条。

tension：大于或等于 0.0F 的值，该值指定曲线的张力。

（3）从相对于数组开始位置的偏移量开始，绘制经过一组指定 PointF 结构的基数样条。

语法：

```
public void DrawCurve (Pen pen,PointF[] points,int offset,int numberOfSegments)
```

DrawCurve 方法中各参数及说明如表 28.7 所示。

表 28.7　　DrawCurve 方法参数及说明

参　数	说　明
pen	Pen 对象，它确定曲线的颜色、宽度和高度
points	PointF 结构数组，这些结构定义样条
offset	从 points 参数数组中的第一个元素到曲线中起始点的偏移量
numberOfSegments	起始点之后要包含在曲线中的段数

（4）使用指定的张力，绘制经过一组指定 Point 结构的基数样条。

语法：

```
public void DrawCurve (Pen pen,Point[] points,Int offset,int numberOfSegments,float tension)
```

DrawCurve 方法中各参数及说明如表 28.8 所示。

表 28.8　　DrawCurve 方法参数及说明

参　数	说　明
pen	Pen 对象，它确定曲线的颜色、宽度和高度
points	Point 结构数组，这些结构定义样条
offset	从 points 参数数组中的第一个元素到曲线中起始点的偏移量
numberOfSegments	起始点之后要包含在曲线中的段数
tension	大于或等于 0.0F 的值，该值指定曲线的张力

示例 28-06　绘制基数样条

本示例中，当程序运行时，单击“绘制”按钮，在窗体中使用指定的张力，绘制一条经过指定 Point 结构的基数样条。

代码 28-06　光盘位置：光盘\mingrisoft\sl\28\06

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个对象，然后实例化 6 个 Point 对象，用来作为基数样条的各连接点，声明一个 Point 结构数组，并将已经实例化的 6 个 Point 对象赋值给该数组，最后调用 Graphics 对象的 DrawCurve 方法绘制一条经过 Point 结构的基数样条。“绘制”按钮的 Click 事件代码如下：

```
private void DrawClick(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 3);                           //实例化一个用于绘制直线和曲线的对象
    Point point1 = new Point(50, 20);                             //定义一个名字为 point1 的坐标点
    Point point2 = new Point(60, 30);                             //定义一个名字为 point2 的坐标点
    Point point3 = new Point(70, 25);                             //定义一个名字为 point3 的坐标点
    Point point4 = new Point(100, 50);                            //定义一个名字为 point4 的坐标点
```

```
    Point point5 = new Point(130, 30);                                    //定义一个名字为 point5 的坐标点
    Point point6 = new Point(150, 45);                                    //定义一个名字为 point6 的坐标点
    Point[] myPoints = { point1, point2, point3, point4, point5, point6 };  //定义一个名字为 myPoints 的坐标点数组
    graphics.DrawCurve(myPen, myPoints, 1.0F);                            //使用指定的张力绘制平滑曲线
}
```

运行本示例，效果如图 28.6 所示。

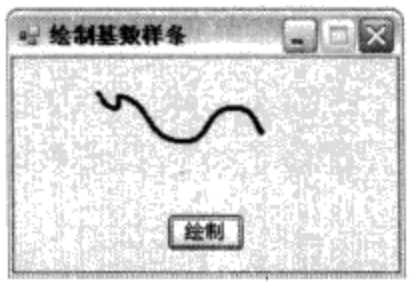


图 28.6　绘制基数样条

28.2.6　GDI+中的贝塞尔样条

贝塞尔样条是由 4 个点指定的曲线：两个端点（p1 和 p2）和两个控制点（c1 和 c2）。曲线开始于 p1，结束于 p2。该曲线不经过控制点，但是控制点的作用像磁铁一样，在某些方向上拉伸曲线并影响曲线弯曲的方式。

绘制贝塞尔样条时，可以调用 Graphics 类中的 DrawBezier 方法，该方法为可重载方法，它主要用来绘制贝塞尔样条，其常用格式有以下两种。

（1）绘制由 4 个 Point 结构定义的贝塞尔样条。

语法：

```
public void DrawBezier (Pen pen,Point pt1,Point pt2,Point pt3,Point pt4)
```

DrawBezier 方法中各参数及说明如表 28.9 所示。

表 28.9　　DrawBezier 方法参数及说明

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）绘制由 4 个点的有序坐标对定义的贝塞尔样条。

语法：

```
public void DrawBezier (Pen pen,float x1,float y1,float x2,float y2,float x3,float y3,float x4,float y4)
```

DrawBezier 方法中各参数及说明如表 28.10 所示。

表 28.10　　DrawBezier 方法参数及说明

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
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

示例 28-07　绘制贝塞尔样条

本示例中，当程序运行时，单击“绘制”按钮，在窗体中的指定位置绘制一段贝塞尔样条。

代码 28-07　光盘位置：光盘\mingrisoft\sl\28\07

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类和 Pen 类的两个实例对象，然后定义 6 个 float 类型的变量，分别用来表示贝塞尔样条起始点的 *x* 坐标、*y* 坐标，第一个控制点的 *x* 坐标、*y* 坐标，第二个点的 *x* 坐标、*y* 坐标，结束点的 *x* 坐标和 *y* 坐标，最后调用 Graphics 对象的 DrawBezier 方法在窗体中绘制贝塞尔样条。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();          //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 3);                 //实例化一个用于绘制直线和曲线的对象
    float startX = 50.0F;                               //实例化起始点的 x 坐标
    float startY = 80.0F;                               //实例化起始点的 y 坐标
    float controlX1 = 150.0F;                           //实例化第一个控制点的 x 坐标
    float controlY1 = 20.0F;                            //实例化第一个控制点的 y 坐标
    float controlX2 = 230.0F;                           //实例化第二个控制点的 x 坐标
    float controlY2 = 50.0F;                            //实例化第二个控制点的 y 坐标
    float endX = 190.0F;                                //实例化结束点的 x 坐标
    float endY = 80.0F;                                 //实例化结束点的 y 坐标
    //绘制由四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);
}
```

运行本示例，效果如图 28.7 所示。

图 28.7　绘制贝塞尔样条

28.2.7　GDI+中的图形路径

路径是通过组合直线、矩形和简单的曲线而形成的。在 GDI+中，GraphicsPath 对象允许将基本构造块收集到一个单元中，调用一次 Graphics 类的 DrawPath 方法，就可以绘制出整个单元的直线、矩形、多边形和曲线。

GraphicsPath 类提供了下列方法来创建要绘制的图形路径：AddLine 方法、AddRectangle 方法、AddEllipse 方法、AddArc 方法、AddPolygon 方法、AddCurve（针对基数样条）方法和 AddBezier 方法。这些方法中的任何一种都是可重载的，即每种方法都支持几个不同的参数列表。例如，AddLine 方法的一种格式接收 4 个整数，而其另一种格式则接收两个 Point 对象。

将直线、矩形和贝塞尔样条添加到路径的方法具有多个伴随方法，这些伴随方法在一次调用中将多个项添加到路径。伴随方法包括 AddLines 方法、AddRectangles 方法和 AddBeziers 方法。同样，AddCurve 方法和 AddArc 方法也有将闭合曲线或扇形添加到路径的伴随方法，例如，AddClosedCurve 方法和 AddPie 方法。

若要绘制路径，需要 Graphics 对象、Pen 对象和 GraphicsPath 对象。Graphics 对象提供了 DrawPath 方法，Pen 对象存储用于呈现路径的线条属性，如宽度和颜色，而 GraphicsPath 对象

存储构成路径的直线和曲线序列。Pen 对象和 GraphicsPath 对象作为参数传递给 DrawPath 方法。

DrawPath 方法用来绘制 GraphicsPath 图形路径，其语法格式如下：

```
public void DrawPath (Pen pen,GraphicsPath path)
```

参数说明如下。

pen：Pen 对象，它确定路径的颜色、宽度和样式。

path：要绘制的 GraphicsPath 图形路径。

示例 28-08　绘制图形路径

本示例中，当程序运行时，单击“绘制”按钮，在窗体中根据条件绘制指定的图形路径。

代码 28-08　光盘位置：光盘\mingrisoft\sl\28\08

Form1 窗体中，在“绘制”按钮的 Click 事件中分别声明 Graphics 类、GraphicsPath 类和 Pen 类的 3 个实例对象，然后声明一个 Point 结构数组，用来定义基数样条的各顶点，同时调用 GraphicsPath 对象的 AddArc 方法、AddCurve 方法、AddString 方法和 AddPie 方法，增加图形路径，最后调用 Graphics 对象的 DrawPath 方法在窗体中绘制指定的图形路径。“绘制”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();              //实例化一个 GDI+绘图图面类对象
    //实例化一个表示一系列相互连接的直线和曲线的类的实例
    GraphicsPath myGraphicsPath = new GraphicsPath();
    Pen myPen = new Pen(Color.Blue, 1);                     //实例化一个用于绘制直线和曲线的对象
    Point[] myPoints = { new Point(15, 30), new Point(30, 40), new Point(50, 30) };//实例化一个坐标点数组
    myGraphicsPath.AddArc(15, 20, 80, 50, 210, 120);   //向当前图形中追加一段椭圆弧
    myGraphicsPath.StartFigure();                           //不闭合当前图形直接开始另一图形的绘制
    myGraphicsPath.AddCurve(myPoints);                      //向当前图形中添加一段样条曲线
    myGraphicsPath.AddString("图形路径", new FontFamily("华文行楷"), (int)FontStyle.Underline, 50, new PointF(20, 50),
new StringFormat());                                        //向此路径中添加文本字符串
    myGraphicsPath.AddPie(180, 20, 80, 50, 210, 120);  //向当前路径添加一个扇形轮廓
    graphics.DrawPath(myPen, myGraphicsPath);               //以指定的格式绘制图形
}
```

注意：使用 GraphicsPath 类时，需要添加 System.Drawing.Drawing2D 命名空间。

运行本示例，效果如图 28.8 所示。

图 28.8　绘制图形路径

28.2.8　GDI+中的画笔

闭合的形状（例如，矩形或椭圆）由轮廓和内部组成。用钢笔绘制出轮廓，并用画笔填充其内部。GDI+提供了几种填充闭合形状内部的画笔类：SolidBrush 类、HatchBrush 类、TextureBrush 类、LinearGradientBrush 类和 PathGradientBrush 类。所有这些类都是从 Brush 类继承的。

1．实心画笔

使用实心画笔时，需要使用 SolidBrush 类，该类主要用来使用定义的单色画笔填充图形形

状，如矩形、椭圆、扇形、多边形和封闭路径等。

示例 28-09　利用实心画笔绘制椭圆

本示例中，当程序运行时，单击“应用”按钮，在窗体中绘制一个使用指定颜色填充的椭圆。

代码 28-09　光盘位置：光盘\mingrisoft\sl\28\09

Form1 窗体中，在“应用”按钮的 Click 事件中分别声明 Graphics 类和 SolidBrush 类的两个实例对象，调用 Graphics 对象的 FillEllipse 方法在窗体中绘制一个使用指定颜色填充的椭圆。“应用”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    SolidBrush mySolidBrush = new SolidBrush(Color.Green);        //实例化一个单色画笔类对象
    graphics.FillEllipse(mySolidBrush, 70, 20, 100, 50);          //以指定的颜色填充边框内部的椭圆
}
```

运行本示例，效果如图 28.9 所示。

2．阴影画笔

使用阴影画笔时，需要使用 HatchBrush 类，该类位于 System.Drawing.Drawing2D 命名空间内，它主要使用阴影样式、前景色和背景色定义矩形画笔。

示例 28-10　利用阴影画笔绘制矩形

本示例中，当程序运行时，单击“应用”按钮，在窗体中绘制一个带有阴影的矩形。

代码 28-10　光盘位置：光盘\mingrisoft\sl\28\10

Form1 窗体中，在“应用”按钮的 Click 事件中分别声明 Graphics 类和 HatchBrush 类的两个实例对象，然后调用 Graphics 对象的 FillRectangle 方法在窗体中绘制一个带有阴影的矩形。“应用”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();            //实例化一个 GDI+绘图图面类对象
    //实例化一个用阴影样式、前景色和背景色定义矩形画笔的类的实例
    HatchBrush myhatchBrush = new HatchBrush(HatchStyle.Horizontal, Color.Green);
    graphics.FillRectangle(myhatchBrush, 70, 20, 100, 50);   //用指定的颜色填充矩形的内部
}
```

运行本示例，效果如图 28.10 所示。

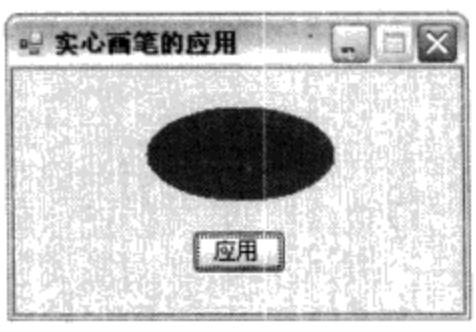


图 28.9　利用实心画笔绘制椭圆

图 28.10　利用阴影画笔绘制矩形

3．纹理画笔

使用纹理画笔时，需要使用 TextureBrush 类，该类的每个属性都是 Brush 对象，这种对象使用图像来填充形状的内部。

示例 28-11　利用纹理画笔绘制椭圆

本示例中，当程序运行时，单击“应用”按钮，在窗体中绘制一个使用指定图像填充的椭圆。

代码 28-11　光盘位置：光盘\mingrisoft\sl\28\11

Form1 窗体中，在“应用”按钮的 Click 事件中分别声明 Graphics 类和 TextureBrush 类的两个实例对象，然后声明一个 Image 对象，用来指定填充图形所用的图像，最后调用 Graphics 对象的 FillEllipse 方法在窗体中绘制一个使用指定图像填充的椭圆。“应用”按钮的 Click 事件代码如下：

```
private void Draw_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Image myImage = (Image)Properties.Resources.example;          //实例化一个图像类对象
    TextureBrush myTBrush = new TextureBrush(myImage);            //实例化一个 TextureBrush 类对象
    graphics.FillEllipse(myTBrush, 70, 20, 100, 50);              //用指定的格式填充椭圆的内部
}
```

运行本示例，效果如图 28.11 所示。

4．渐变画笔

使用渐变画笔时，需要使用 LinearGradientBrush 类和 PathGradientBrush 类，这两个类都位于 System.Drawing.Drawing2D 命名空间内，其中 LinearGradientBrush 类使用线性渐变封装 Brush 对象，而 PathGradientBrush 类则通过路径渐变封装 Brush 对象。

示例 28-12　利用渐变画笔绘制圆形

本示例中，当程序运行时，单击“线性渐变”按钮，在窗体中根据条件绘制指定的图形路径。当单击“路径渐变”按钮时，在窗体中根据指定的条件绘制图形。

代码 28-12　光盘位置：光盘\mingrisoft\sl\28\12

Form1 窗体中，单击“线性渐变”按钮，程序调用 Graphics 对象的 FillEllipse 方法，根据声明的 LinearGradientBrush 类对象，绘制一个线性渐变图形。“线性渐变”按钮的 Click 事件代码如下：

```
private void LinearGradient_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Rectangle myRectangle = new Rectangle(40, 20, 90, 70);        //实例化一个矩形区域
    LinearGradientBrush LGBrush = new LinearGradientBrush(myRectangle, Color.Yellow, Color.YellowGreen,
LinearGradientMode.Horizontal);                                   //实例化一个线性渐变类的对象
    graphics.FillEllipse(LGBrush, myRectangle);                   //以指定的格式填充椭圆的内部
}
```

单击“路径渐变”按钮，程序调用 Graphics 对象的 FillPath 方法，根据已经声明的 PathGradientBrush 类对象，绘制一个路径渐变图形。“路径渐变”按钮的 Click 事件代码如下：

```
private void PathGradient_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Point centerPoint = new Point(200, 60);                       //实例化一个坐标点 centerPoint 对象
    GraphicsPath path = new GraphicsPath();                       //实例化一个表示一系列相互连接的直线和曲线的对象
    int R = 20;                                                   //定义一个整型变量 R
    path.AddEllipse(centerPoint.X - R, centerPoint.Y - R, 2 * R, 2 * R);      //以指定的格式填充一个椭圆
    path.AddEllipse(centerPoint.X - 2 * R, centerPoint.Y - 2 * R, 4 * R, 4 * R);
    path.AddEllipse(centerPoint.X - 3 * R, centerPoint.Y - 3 * R, 6 * R, 6 * R);
    PathGradientBrush brush = new PathGradientBrush(path);        //实例化一个路径渐变类对象
    brush.CenterPoint = centerPoint;                              //设置路径渐变的中心点
    brush.CenterColor = Color.Gray;                               //设置路径渐变中心点的颜色
    //使用 Color 类型的数组指定与路径上每个顶点对应的颜色
    brush.SurroundColors = new Color[] { Color.Black, Color.Blue, Color.GreenYellow };
    graphics.FillPath(brush, path);                               //以指定的格式填充图形的内部
}
```

运行本示例，效果如图 28.12 所示。

图 28.11　利用纹理画笔绘制椭圆

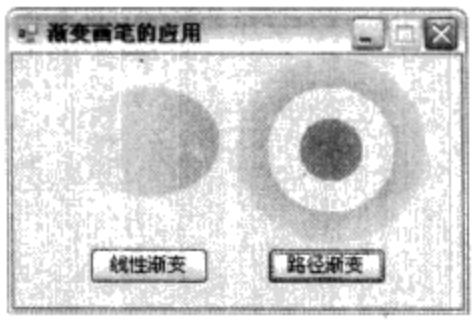


图 28.12　利用渐变画笔绘制圆形

28.2.9　GDI+中曲线的填充

GDI+中，可以使用画笔来填充曲线，Graphics 类填充曲线的方法有 FillRectangle 方法、FillEllipse 方法、FillPie 方法、FillPolygon 方法、FillClosedCurve 方法、FillPath 方法和 FillRegion 方法。只要调用其中的任意一个方法，并使用一种指定的画笔类型作为其参数，就可以填充曲线。

示例 28-13　填充曲线的应用

本示例中，当程序运行时，单击“填充曲线”按钮，在窗体中绘制一段图形路径，并使用指定画笔对该图形路径进行填充。

代码 28-13　光盘位置：光盘\mingrisoft\sl\28\13

Form1 窗体中，单击“填充曲线”按钮，程序首先调用 Graphics 对象的 FillPath 方法，将要绘制的图形路径使用指定画笔填充好，然后再调用 Graphics 对象的 DrawPath 方法将图形路径绘制出来。“填充曲线”按钮的 Click 事件代码如下：

```
private void FillCurve_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                          //实例化一个 GDI+绘图图面类对象
    SolidBrush mySolidBrush = new SolidBrush(Color.GreenYellow);        //实例化一个单色画笔类对象
    GraphicsPath myGraphicsPath = new GraphicsPath();        //实例化一个表示一系列相互连接的直线和曲线的对象
    Pen myPen = new Pen(Color.Blue, 1);                      //实例化一个用于绘制直线和曲线的对象
    Point[] myPoints = { new Point(15, 30), new Point(30, 40), new Point(50, 30) };   //实例化一个坐标点数组
    myGraphicsPath.AddArc(15, 20, 80, 50, 210, 120);         //向当前图形中追加一段椭圆弧
    myGraphicsPath.StartFigure();                            //不闭合当前图形即开始绘制另一个图形
    myGraphicsPath.AddCurve(myPoints);                       //向当前图形添加一段样条曲线
    myGraphicsPath.AddString("曲线填充", new FontFamily("华文行楷"), (int)FontStyle.Regular, 50, new PointF(20,
50), new StringFormat());                                    //向当前路径下添加文本字符串
    myGraphicsPath.AddPie(180, 20, 80, 50, 210, 120);        //向当前路径添加一个扇形轮廓
    graphics.FillPath(mySolidBrush, myGraphicsPath);         //填充指定形状的内部
    graphics.DrawPath(myPen, myGraphicsPath);                //绘制图形
}
```

运行本示例，效果如图 28.13 所示。

图 28.13　填充曲线的应用

28.2.10　GDI+中的区域

区域是输出设备显示区域的一部分，它可以是简单的（例如，单个矩形），也可以是复杂的

（例如，多边形和闭合曲线的组合）。

区域常用于剪辑和命中检测。剪辑需要将要绘制的图形限制到显示区域的一个特定部分，通常是需要更新的部分，而命中检测则需要通过检查来确定按下鼠标时，光标是否在屏幕的特定区域中。

可以从矩形或路径中构造区域，也可以通过合并现有的区域来创建复杂区域。Region 类提供以下区域合并方法：Intersect 方法、Union 方法、Xor 方法、Exclude 方法和 Complement 方法。其中，Intersect 方法用来将 Region 对象更新为其自身与指定 Region 对象的交集，Union 方法用来将 Region 对象更新为其自身与指定 GraphicsPath（或 Region）对象的并集，Xor 方法用来将 Region 对象更新为其自身与指定 GraphicsPath（或 Region）对象的并集减去两者的交集，Exclude 方法用来将 Region 对象更新为其自身与指定 GraphicsPath（或 Region）对象不相交的部分，Complement 方法用来将 Region 对象更新为指定 GraphicsPath（或 Region）对象与此 Region 对象相交的部分。

两个区域的交集是同时属于这两个区域的所有点的集合。并集是属于一个或另一个或两个区域的所有点的集合。区域的补集是不在该区域的所有点的集合。

若要填充区域，需要有 Graphics 对象、Brush 对象和 Region 对象。Graphics 对象提供 FillRegion 方法，该方法用来填充 Region 区域的内部，其语法格式如下：

```
public void FillRegion (Brush brush,Region region)
```

参数说明如下。

bursh：表示确定填充特性的 Brush 对象。

region：Region 对象，表示要填充的区域。

示例 28-14　区域截取的应用

本示例中，当程序运行时，单击“区域截取”按钮，截取窗体中的一组指定文本并显示。

代码 28-14　光盘位置：光盘\mingrisoft\sl\28\14

Form1 窗体中，单击“区域截取”按钮，首先在窗体中绘制一个多边形，然后声明 Region 类的一个对象，用来指定截取区域的大小；同时调用 Graphics 对象的 SetClip 方法，将该对象的剪辑区域设置为当前剪辑区域与指定 Region 对象的组合结果；最后调用 Graphics 对象的 DrawString 方法在窗体中绘制一组文本（该文本的一部分内容包含在截取区域中，另一部分不在其中）。“区域截取”按钮的 Click 事件代码如下：

```
private void ZoneIntercept_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    //实例化一个坐标点数组
    Point[] mypoints = { new Point(50, 20), new Point(120, 20), new Point(160, 80), new Point(50, 100) };
    GraphicsPath mygraphicsPath = new GraphicsPath();     //实例化一个表示一系列相互连接的直线和曲线的对象
    mygraphicsPath.AddPolygon(mypoints);                          //向此路径添加多边形
    Region myregion = new Region(mygraphicsPath);//实例化一个由矩形和由路径构成的图形形状的内部的对象
    Pen mypen = new Pen(Color.Blue, 2);                           //实例化一个用于绘制直线和曲线的对象
    graphics.DrawPath(mypen, mygraphicsPath);                     //以指定的格式绘制图像
    //将剪辑区域设置为当前剪辑区域与指定 Region 对象的组合结果
    graphics.SetClip(myregion, CombineMode.Replace);
    graphics.DrawString("GDI+区域", new Font(new FontFamily("楷体_GB2312"), 36, FontStyle.Regular,
GraphicsUnit.Pixel), new SolidBrush(Color.Red), new PointF(40, 50));      //绘制指定的文本字符串
```

注意：使用 GraphicsPath 类和 CombineMode 枚举时，需要添加 System.Drawing.Drawing2D 命名空间。

运行本示例，效果如图 28.14 所示。

图 28.14　区域截取的应用

28.2.11　用直线和曲线消除锯齿

1．用直线消除锯齿

用基本的 DrawLine 方法绘制出来的直线看上去带有锯齿，有点像楼梯，这种用楼梯状来表示直线的技术被称为锯齿化，楼梯是理论直线的一个别名。

一项更为复杂的呈现直线的技术需要使用部分透明的像素和不透明的像素，而这种呈现方式被称为消除锯齿，它可以生成视觉上更平滑的直线。消除锯齿需要使用 Graphics 类的 SmoothingMode 属性，该属性属于 System.Drawing.Drawing2D 命名空间，它主要用来获取或设置 Graphics 图像的呈现质量，其属性值及说明如表 28.11 所示。

表 28.11　　SmoothingMode 属性值及说明

属　性　值	说　　明
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

示例 28-15　直线消除锯齿的应用

本示例中，当程序运行时，单击“原图”按钮，调用 DrawLine 方法绘制一条直线，然后单击“直线消除锯齿”按钮，在窗体中绘制一条经过消除锯齿的直线，比较它们的呈现效果。代码如下：

代码 28-15　光盘位置：光盘\mingrisoft\sl\28\15

```
private void Smooth_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                  //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 2);                         //实例化一个用于绘制直线和曲线的对象
    graphics.SmoothingMode = SmoothingMode.AntiAlias;           //设置图形的呈现质量
    graphics.DrawLine(myPen, 140, 40, 220, 40);                 //绘制一条连接由坐标对指定的两个点的线条
}
private void Sawtooth_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                  //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 2);                         //实例化一个用于绘制直线和曲线的对象
    graphics.DrawLine(myPen, 40, 40, 120, 40);                  //绘制一条连接由坐标对指定的两个点的线条
}
```

运行本示例，效果如图 28.15 所示。

2．用曲线消除锯齿

用曲线消除锯齿操作的实现方法同用直线消除锯齿操作的方法类似，也需要用到 Graphics 类的 SmoothingMode 属性。

交叉链接：SmoothingMode 属性的详细介绍请参见上节“用直线消除锯齿”。

示例 28-16　曲线消除锯齿的应用

本示例中，当程序运行时，单击“原图‘按钮，调用 DrawEllipse 方法绘制一个椭圆，然后单击“曲线消除锯齿”按钮，在窗体中绘制一个经过消除锯齿的椭圆，比较它们的呈现效果。代码如下：

代码 28-16　光盘位置：光盘\mingrisoft\sl\28\16

```
private void Sawtooth_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                  //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 4);                         //实例化一个用于绘制直线和曲线的对象
    Rectangle myRectangle = new Rectangle(40, 20, 80, 40);      //实例化一个矩形工作区域对象
    graphics.DrawEllipse(myPen, myRectangle);                   //在矩形结构内绘制椭圆
}
private void EllipseSawtooth_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                  //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 4);                         //实例化一个用于绘制直线和曲线的对象
    Rectangle myRectangle = new Rectangle(140, 20, 80, 40);     //实例化一个矩形工作区域对象
    graphics.SmoothingMode = SmoothingMode.AntiAlias;           //设置图像的呈现质量
    graphics.DrawEllipse(myPen, myRectangle);                   //在矩形结构内绘制椭圆
}
```

运行本示例，效果如图 28.16 所示。

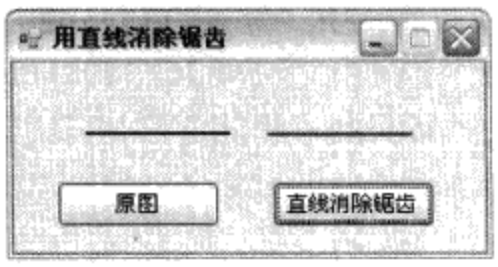


图 28.15　直线消除锯齿的应用

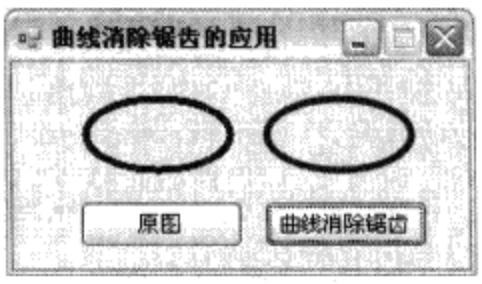


图 28.16　曲线消除锯齿的应用

28.3　图像、位图和图元文件

Image 类是抽象基类，它提供了处理位图和图元文件的方法。Bitmap 类和 Metafile 类都继承自 Image 类，其中 Bitmap 类提供了用于加载、保存和处理位图的方法，而 Metafile 类提供了用于记录和检查图元文件的方法。本节将详细介绍图像、位图和图元文件的相关内容。

28.3.1　位图类型

位图是位的数组，它指定了像素矩阵中各像素的颜色。单个像素的位数决定了可分配到该像素的颜色数。例如，如果用 4 位来呈现每个像素，那么一个给定的像素就可以分配到 16（2^4=16）种颜色中的一种。表 28.12 列出了指定位数时，一个像素可分配到的颜色数量。

表 28.12　　指定位数时，一个像素可分配到的颜色数量

每像素的位数	一个像素可分配到的颜色数量
1	$2^1=2$
2	$2^2=4$
4	$2^4=16$
8	$2^8=256$

将位图存储到磁盘文件的标准格式有多种，GDI+支持下面 6 种图形文件格式。

（1）BMP 格式。BMP 格式是 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 文件通常不能被压缩，因此，它不太适合通过 Internet 传输。

（2）可交换图像文件格式（GIF）。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。

（3）联合摄影专家组（JPEG）。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 文件不支持透明和动画效果。

JPEG 图像中的压缩级别是可以控制的，但是较高的压缩级别（较小的文件）会导致丢失更多的信息，JPEG 压缩不适用于线条图形、纯色块和有清晰边界的图片。

JPEG 是一种压缩方案，不是一种文件格式。“JPEG 文件交换格式（JFIF）”是一种文件格式，它常用于存储和传输已根据 JPEG 方案压缩的图像。Web 浏览器显示的 JFIF 文件使用.jpg 扩展名。

（4）可交换图像文件（EXIF）。EXIF 是一种适用于数码相机拍摄的照片的文件格式。EXIF 文件中含有根据 JPEG 规格压缩的图像。另外，它还包含了有关照片的信息（拍摄日期、快门速度和曝光时间等）和相机的信息（制造商和型号等）。

（5）可移植网络图形（PNG）。PNG 格式不但保留了许多 GIF 格式的优点，还提供了超出 GIF 的功能。像 GIF 文件一样，PNG 文件在压缩时也不损失信息。PNG 文件能以每像素 8、24 或 48 位来存储颜色，并以每像素 1、2、4、8 或 16 位来存储灰度，相比之下，GIF 文件只能使用每像素 1、2、4 或 8 位。PNG 文件还可为每个像素存储一个 alpha 值，该值指定了该像素颜色与背景颜色混合的程度。

PNG 优于 GIF 之处在于，它能渐进地显示一幅图像（也就是说，在图像通过网络连接传递的过程中，显示的图像将越来越完整）。PNG 文件可包含灰度校正和颜色校正信息，以便图像可在各种各样的显示设备上精确地呈现。

（6）标签图像文件格式（TIFF）。TIFF 是一种灵活的和可扩展的格式，各种平台和图像处理应用程序都支持这种格式。TIFF 文件能以每像素任意位来存储图像，并可以使用各种各样的压缩算法。

28.3.2　GDI+中的元文件

GDI+提供了 Metafile 类，以便能够记录和显示图元文件。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。Metafile 类记录的命令和设置可存储在内存中，也可以保存到文件或流。

GDI+能够显示用以下格式存储的图元文件。

- Windows 图元文件（WMF）。
- 增强性图元文件（EMF）。

● EMF+。

GDI+能够用 EMF 和 EMF+格式记录图元文件，但不能使用 WMF 格式。

EMF+是 EMF 的扩展，它可存储 GDI+记录。EMF+格式有两种变体："EMF+惟一"和"EMF+双重"。"EMF+惟一"图元文件只包含 GDI+记录，这样的图元文件可以用 GDI+来显示，但不能使用 GDI 显示；"EMF+双重"图元文件包含 GDI+和 GDI 记录，该图元文件中的每个 GDI+记录都与一个备用的 GDI 记录相对应，这样的图元文件可用 GDI+或 GDI 来显示。

注意：Metafile 类属于 System.Drawing.Imaging 命名空间。

28.3.3 在 GDI+中绘制和克隆图像

1．绘制图像

在绘制图像时，可以使用 Bitmap 类对象指定 Image 图像，然后调用 Graphics 类中的 DrawImage 方法进行绘制。DrawImage 方法为可重载方法，它主要用来在指定位置绘制指定的 Image 图像，其常用格式有以下 3 种。

（1）在指定的位置按原始大小绘制指定的 Image 图像。

语法：

```
public void DrawImage (Image image,Point point)
```

参数说明如下。

image：要绘制的 Image 图像。

point：Point 结构，它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：

```
public void DrawImage (Image image,Rectangle rect)
```

参数说明如下。

image：要绘制的 Image 图像。

rect：Rectangle 结构，它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：

```
public void DrawImage (Image image,Point[] destPoints,Rectangle srcRect,GraphicsUnit srcUnit)
```

DrawImage 方法中各参数及说明如表 28.13 所示。

表 28.13　DrawImage 方法各参数及说明

参　数	说　明
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，它指定 image 图像中要绘制的部分
srcUnit	GraphicsUnit 枚举成员，它指定 srcRect 参数所用的度量单位

示例 28-17　绘制图像

本示例中，当程序运行时，单击"绘制"按钮，在窗体中的指定位置根据已创建的 Bitmap 对象绘制一幅指定大小的图像。

代码 28-17　光盘位置：光盘\mingrisoft\sl\28\17

Form1 窗体中，在"绘制"按钮的 Click 事件中分别声明 Graphics 类和 Bitmap 类的两个实例对象，然后调用 Graphics 对象的 DrawImage 方法，在窗体中根据已创建的 Bitmap 对象绘制一幅指定大小的图像。"绘制"按钮的 Click 事件代码如下：

```
private void DrawPicture_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();          //实例化一个 GDI+绘图图面类对象
    Bitmap bitmap = Properties.Resources.image;         //实例化一个 GDI+位图对象
    graphics.DrawImage(bitmap, 20, 20, 180, 130);       //在指定位置按指定大小绘制指定图像
}
```

运行本示例，效果如图 28.17 所示。

2．克隆图像

克隆图像时，可以使用 Bitmap 类提供的 Clone 方法，该方法为可重载方法，主要用来创建现有 Bitmap 对象的副本，从而克隆图像，其常用格式有如下两种。

（1）创建 Image 图像的一个精确副本。

语法：

```
public Object Clone ()
```

返回值：该方法创建的、强制转换为对象的 Image 图像。

（2）创建 Bitmap 对象（它由 Rectangle 结构和指定的 PixelFormat 枚举定义）的某个部分的副本。

语法：

```
public Bitmap Clone (Rectangle rect,PixelFormat format)
```

参数说明如下。

rect：定义 Bitmap 对象中要复制的部分，坐标相对于该 Bitmap 对象。

format：为目标 Bitmap 对象指定 PixelFormat 枚举。

返回值：该方法创建的新 Bitmap 对象。

示例 28-18　克隆图像

本示例中，当程序运行时，单击“克隆”按钮，在窗体的左边将原图绘制出来，然后使用 Bitmap 类的 Clone 方法指定克隆原图的哪一部分，并将要克隆的部分绘制在窗体的右边。

代码 28-18　光盘位置：光盘\mingrisoft\sl\28\18

Form1 窗体中，单击“克隆”按钮，程序调用 Bitmap 类的 Clone 方法指定要克隆的图像，然后使用 DrawImage 方法将要克隆的图像绘制在窗体中的指定位置。“克隆”按钮的 Click 事件代码如下：

```
private void ClonePicture_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                          //实例化一个 GDI+绘图图面类对象
    Bitmap bitmap = new Bitmap("image.jpg");                            //实例化一个 GDI+位图对象
    Rectangle rectangle = new Rectangle(50, 50, 180, 80);               //实例化一个矩形工作区域
    Bitmap cloneBitmap = bitmap.Clone(rectangle, PixelFormat.DontCare);     //实例化一个 GDI+位图对象
    graphics.DrawImage(bitmap, 20, 20, 180, 130);                       //在指定位置按指定大小绘制指定图像
    graphics.DrawImage(cloneBitmap, 220, 60, 100, 50);                  //在指定位置按指定大小绘制克隆后的指定图像
}
```

运行本示例，效果如图 28.18 所示。

图 28.17　绘制图像

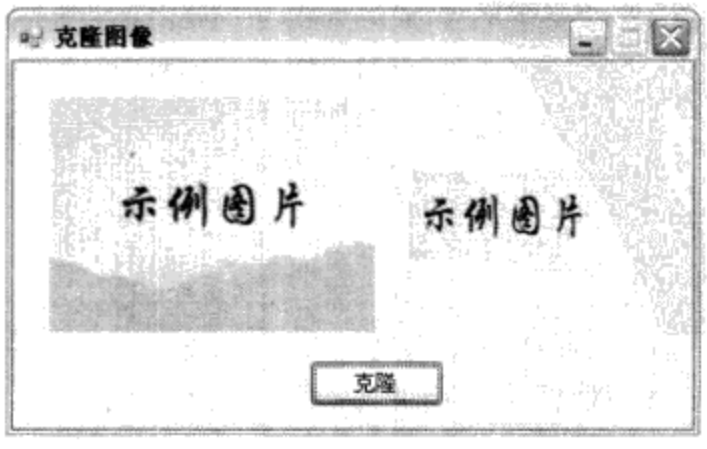


图 28.18　克隆图像

注意：在使用 PixelFormat 枚举时，需要添加 System.Drawing.Imaging 命名空间。

28.3.4 在 GDI+中缩放图像

DrawImage 为可重载方法，它的一种形式为接收一个 Bitmap 对象和一个 Rectangle 结构。Rectangle 矩形结构指定了绘图操作的目标，如果目标矩形的大小与原始图像的大小不同，原始图像将进行缩放，以适应目标矩形。

交叉链接：DrawImage 方法在前面已经介绍过，详请参见 28.3.3 节。

示例 28-19 缩放图像的应用

本示例中，当程序运行时，单击“缩放”按钮，在窗体中从左向右依次绘制 3 幅图像，其中第 1 幅图为原始图像的缩小形式，第 2 幅图与原始图像大小相同，第 3 幅图为原始图像的放大形式。

代码 28-19 光盘位置：光盘\mingrisoft\sl\28\19

Form1 窗体中，在“缩放”按钮的 Click 事件中定义 3 个 Rectangle 矩形结构，然后调用 Graphics 类的 DrawImage 方法，在窗体中根据已经定义的 3 个 Rectangle 矩形结构依次绘制 3 幅图像。“缩放”按钮的 Click 事件代码如下：

```
private void ZoomPicture_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Bitmap bitmap = new Bitmap("image.jpg");                      //实例化一个 GDI+位图图像
    Rectangle rectangle1 = new Rectangle(10, 80, 100, 60);        //实例化一个矩形工作区域 rectangle1
    Rectangle rectangle2 = new Rectangle(130, 30, 268, 190);      //实例化一个矩形工作区域 rectangle2
    Rectangle rectangle3 = new Rectangle(420, 20, 288, 210);      //实例化一个矩形工作区域 rectangle3
    graphics.DrawImage(bitmap, rectangle1);      //在指定的位置按 rectangle1 的大小绘制指定图像
    graphics.DrawImage(bitmap, rectangle2);      //在指定的位置按 rectangle2 的大小绘制指定图像
    graphics.DrawImage(bitmap, rectangle3);      //在指定的位置按 rectangle3 的大小绘制指定图像
}
```

运行本示例，效果如图 28.19 所示。

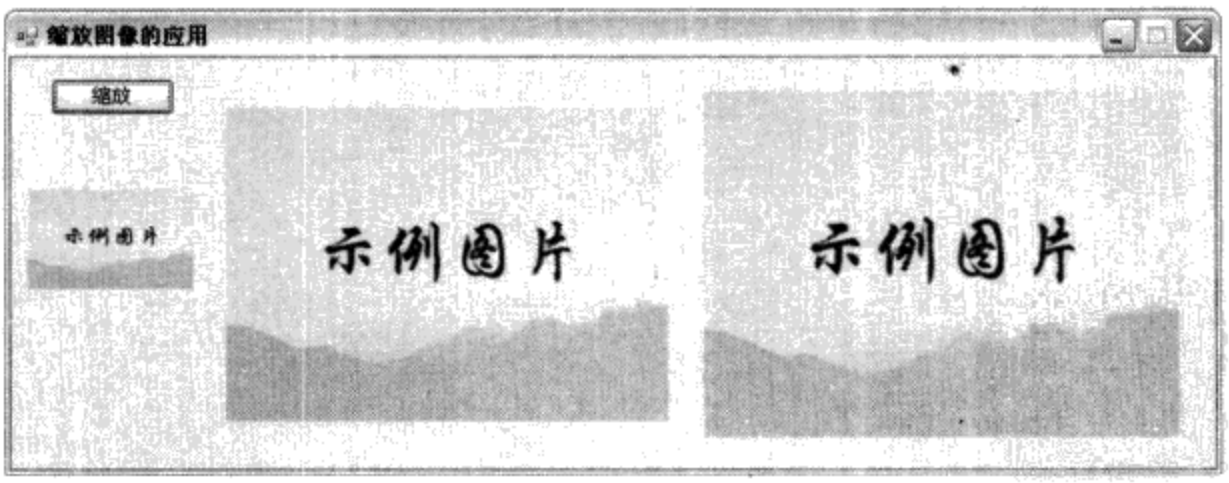


图 28.19 缩放图像的应用

28.4 坐标系和坐标变换

GDI+提供了世界变换和页面变换两种坐标变换，以便可以变换（旋转、缩放和平移等）所绘制的图形，而这两种坐标变换还允许用户使用多种坐标系。本节将详细介绍坐标系和坐标变换的相关内容。

28.4.1 坐标系类型

GDI+使用 3 个坐标空间：世界、页面和设备。世界坐标系是用于建立特殊图形（世界模型）

的坐标系，页面坐标系是指绘图图面（如窗体或控件）使用的坐标系，设备坐标系是在其上进行绘制的物理设备（如屏幕或纸张）所使用的坐标系。一种称为“世界变换”的变换可将世界坐标系转换为页面坐标系，而另一种称为“页面变换”的变换可将页面坐标系转换为设备坐标系。

用于将世界坐标系映射到页面坐标系的“世界变换”保存在 Graphics 类的 Transform 属性中，而“页面变换”用于将页面坐标系映射到设备坐标系，Graphics 类提供了用于操作“页面变换”的 PageUnit 属性和 PageScale 属性。另外，Graphics 类还提供了两个只读属性：DpiX 和 DpiY，分别用于检查显示设备每英寸的水平点数和垂直点数。

28.4.2 变形的矩阵表示形式

m×*n* 矩阵是排列成 *m* 行和 *n* 列的一系列数，例如，图 28.20 为几个矩阵表示形式。

用户可以通过将单个元素相加来合并两个尺寸相同的矩阵，例如，图 28.21 表示两个矩阵相加。

$$\underset{2\ 3}{\begin{bmatrix} 3 & 1 & 4 \\ 2 & 5 & 0 \end{bmatrix}} \quad \underset{4\ 2}{\begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 0 & 4 \\ 5 & 6 \end{bmatrix}} \quad \underset{1\ 3}{\begin{bmatrix} 1.6 & 0.2 & 1.0 \end{bmatrix}}$$

$$\underset{2\ 2}{\begin{bmatrix} 2 & 0 \\ 0 & 3.5 \end{bmatrix}} \quad \underset{2\ 1}{\begin{bmatrix} 5 \\ 3 \end{bmatrix}} \quad \underset{3\ 3}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 40 & 20 & 1 \end{bmatrix}}$$

图 28.20　矩阵

$$\begin{bmatrix} 5 & 4 \end{bmatrix} + \begin{bmatrix} 20 & 30 \end{bmatrix} = \begin{bmatrix} 25 & 34 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 4 \\ 1 & 5 \\ 0 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 1 & 7 \\ 1 & 9 \end{bmatrix}$$

图 28.21　矩阵相加

m×*n* 矩阵可与一个 *n*×*p* 矩阵相乘，结果为一个 *m*×*p* 矩阵，但如果两个矩阵相乘，第一个矩阵的列数必须与第二个矩阵的行数相同。例如，图 28.22 表示两个矩阵相乘。

$$\underset{1\times2}{\begin{bmatrix} 2 & 3 \end{bmatrix}} \underset{2\times1}{\begin{bmatrix} 2 \\ 4 \end{bmatrix}} = \underset{1\times1}{\begin{bmatrix} 16 \end{bmatrix}}$$

$$\underset{1\times3}{\begin{bmatrix} 1 & 3 & 2 \end{bmatrix}} \underset{3\times2}{\begin{bmatrix} 1 & 0 \\ 2 & 4 \\ 5 & 1 \end{bmatrix}} = \underset{1\times2}{\begin{bmatrix} 17 & 14 \end{bmatrix}}$$

$$\underset{2\times3}{\begin{bmatrix} 2 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix}} \underset{3\times3}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 3 & 1 \end{bmatrix}} = \underset{2\times3}{\begin{bmatrix} 4 & 13 & 1 \\ 6 & 9 & 1 \end{bmatrix}}$$

图 28.22　矩阵相乘

如果将平面中的点视为 1×2 矩阵，则可通过将该点乘以一个 2×2 矩阵来将该点变换，例如，图 28.23 显示了应用于点（2,1）的几个变换。

图 28.23 中显示的所有变换都是线性变换，如果变换（如平移）不是线性的，则不能表示为与 2×2 矩阵相乘的形式。例如，从点（2,1）开始，将其旋转 90°，然后在 *x* 方向将其平移 3 个单位，在 *y* 方向将其平移 4 个单位，这时可先使用矩阵乘法，再使用矩阵加法来完成此操作，如图 28.24 所示。

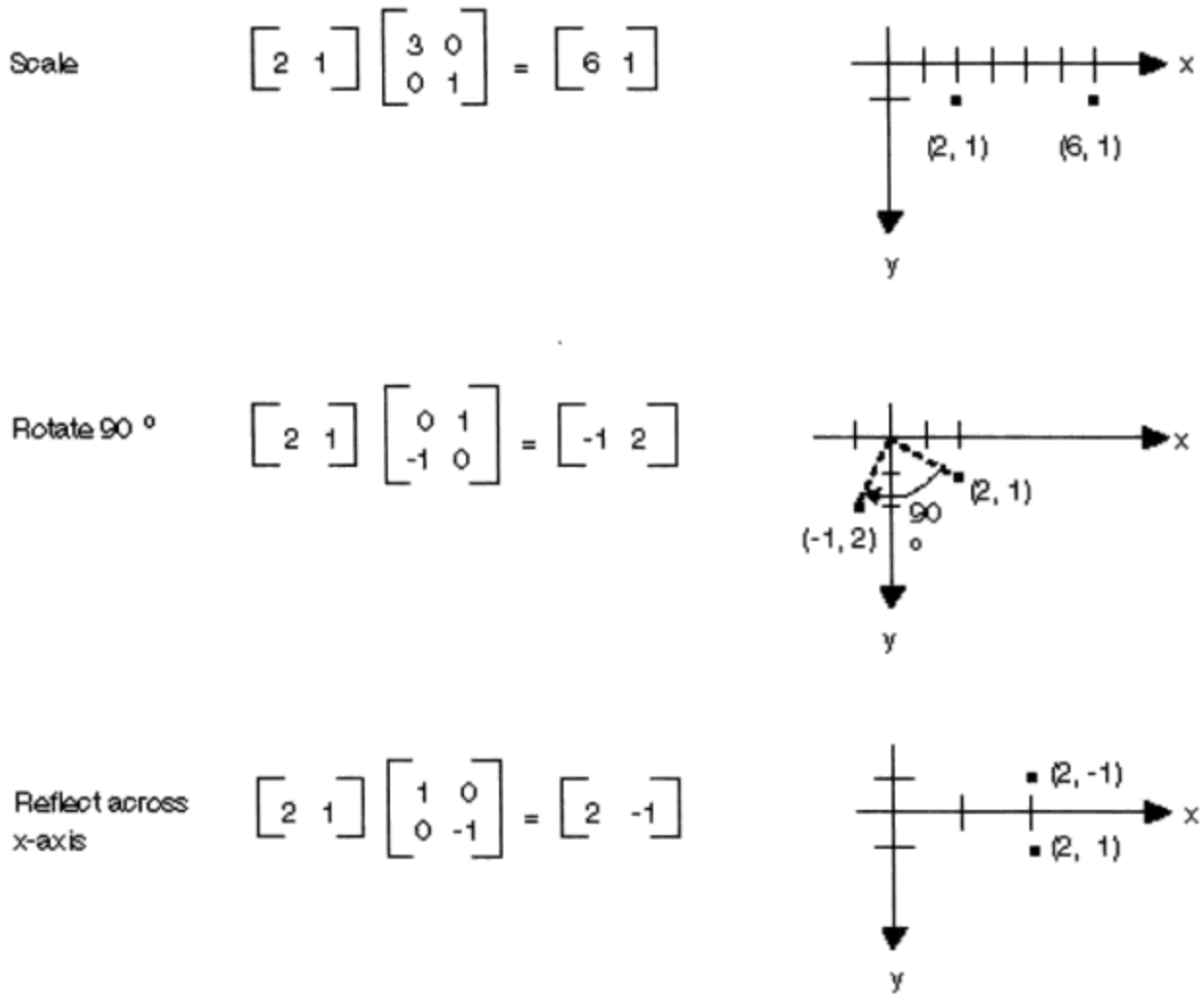


图 28.23　点的变换

后面跟一平移的线性变换称为仿射变换。将仿射变换存储于一对矩阵（一个用于线性部分，一个用于平移）的替换方案是将整个变换存储于 3×3 矩阵。如果要使 3×3 矩阵起作用，平面上的点必须存储于具有虚拟第 3 坐标的 1×3 矩阵中。通常的方法是使所有的第 3 坐标等于 1，例如，矩阵[2 1 1]表示点（2，1）。将图 28.24 中“点的平移”转换为仿射变换的结果如图 28.25 所示。

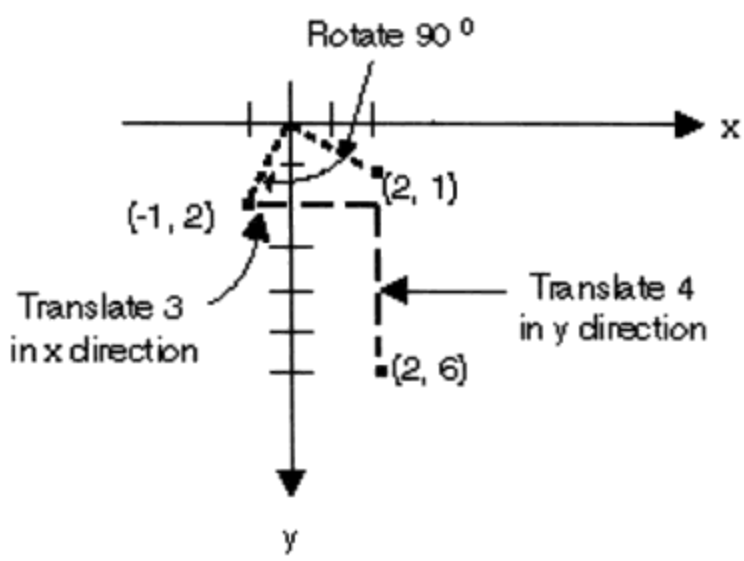


图 28.24　点的平移

$$\begin{bmatrix} 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 3 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 1 \end{bmatrix}$$

图 28.25　仿射变换

在前面的示例中，点（2，1）映射到了点（2，6），请注意，3×3 矩阵的第 3 列数字为：0、0、1，对于仿射变换的 3×3 矩阵而言，重要的数字是列 1 和列 2 中的 6 个数字，如图 28.26 所示，矩阵左上角的 2×2 部分表示变换的线性部分，第 3 行中的前两项表示变换的平移部分。

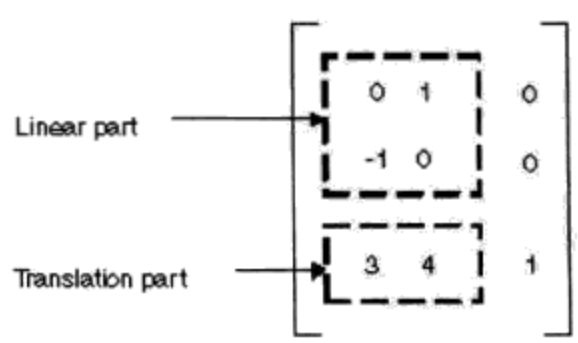


图 28.26　仿射变换分解图

在 GDI+中，可以在 Matrix 对象中存储仿射变换。由于表示仿射变换的矩阵的第 3 列总是（0，0，1），因此在构造 Matrix 对象时，只需指定前两列中的 6 个数即可。例如构造上面显示的矩阵代码如下：

```
Matrix myMatrix =new Matrix(0,1, -1, 0, 3, 4)
```

GDI+中，还有一种变换叫复合变换。复合变换是一个接一个的变换序列，它的矩阵表示形式可通过将几个单独的变换矩阵相乘而得到。下面以如表 28.14 所示的矩阵和变换来进行说明。

表 28.14 矩阵及变换说明

矩　阵	说　明
矩阵 A	旋转 90°
矩阵 B	在 *x* 方向上缩放 2 倍
矩阵 C	在 *y* 方向上平移 3 个单位

如果从由矩阵[2 1 1]表示的点(2, 1)开始，并先后乘以 A、B、C，则点(2, 1)将按列出的顺序经历 3 种变换。

[2 1 1]ABC=[-2 5 1]

可以不将复合变换的 3 部分存储于 3 个独立的矩阵，而是一起乘以 A、B 和 C 来得到存储整个复合变换的单个的 3×3 矩阵。假定 ABC=D，则一个点乘以 D 得出的结果与一个点先后乘以 A、B、C 的结果相同。

[2 1 1]D=[-2 5 1]

如图 28.27 显示了矩阵 A、B、C 和 D。

复合变换的矩阵可通过将几个单独的变换矩阵相乘而得到，这就意味着任何仿射变换的序列均可存储于单个的 Matrix 对象中。

Matrix 类提供构建复合变换的方法包括 Multiply 方法、Rotate 方法、RotateAt 方法、Scale 方法、Shear 方法和 Translate 方法。例如，创建一个复合变换的矩阵，方式为先旋转 30°，再在 *y* 方向上缩放 2 倍，然后在 *x* 方向上平移 5 个单位。代码如下：

```
Matrix myMatrix = new Matrix();
myMatrix.Rotate(30);
myMatrix.Scale(1, 2, MatrixOrder.Append);
myMatrix.Translate(5, 0, MatrixOrder.Append);
```

如图 28.28 所示为复合变换后的矩阵。

$$\underset{A}{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{B}{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{C}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{bmatrix}} \underset{=}{=} \underset{D}{\begin{bmatrix} 0 & 1 & 0 \\ -2 & 0 & 0 \\ 0 & 3 & 1 \end{bmatrix}}$$

图 28.27　矩阵 A、B、C、D

$$\begin{bmatrix} \cos 30° & 2\sin 30° & 0 \\ -\sin 30° & 2\cos 30° & 0 \\ 5 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 0.866 & 1.0 & 0 \\ -0.5 & 1.73 & 0 \\ 5 & 0 & 1 \end{bmatrix}$$

图 28.28　复合变换后的矩阵

注意：

① 复合变换的顺序非常重要，一般说来，先旋转、再缩放、然后平移，与先缩放、再旋转、然后平移是不同的。同样，矩阵相乘的顺序也是非常重要的，一般说来，ABC 与 BAC 不同。

② Matrix 类属于 System.Drawing.Drawing2D 命名空间。

28.4.3　全局变换和局部变换

1．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。

示例 28-20　图像的全局变换

本示例实现的是，当程序运行时，单击“变换”按钮，在窗体中的相同位置绘制一个椭圆和一个经过全局变换处理的椭圆。

代码 28-20　光盘位置：光盘\mingrisoft\sl\28\20

Form1 窗体中，在“变换”按钮的 Click 事件中，首先通过调用 Graphics 对象的 DrawEllipse 方法绘制一个普通椭圆，然后使用 Graphics 对象创建一个全局变换（该变换首先在 *y* 轴方向上缩放 0.5 倍，再在 *x* 轴方向平移 50 个单位，然后旋转 30°），最后再次调用 Graphics 对象的 DrawEllipse 方法绘制椭圆。“变换”按钮的 Click 事件代码如下：

```
private void WholeTransform_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    Pen myPen = new Pen(Color.Blue, 2);                           //实例化一个用于绘制直线和曲线的对象
    graphics.DrawEllipse(myPen, 10, 30, 100, 50);                 //绘制一个由边框定义的椭圆
    graphics.ScaleTransform(1, 0.5f);                             //将指定的缩放操作应用于此矩阵
    graphics.TranslateTransform(50, 0, MatrixOrder.Append);       //通过以指定顺序将指定平移应用于此变换矩阵
    graphics.RotateTransform(30, MatrixOrder.Append);             //以指定顺序将指定旋转应用到此变换矩阵
    graphics.DrawEllipse(myPen, 10, 30, 100, 50);                 //绘制一个由边框定义的椭圆
}
```

运行本示例，效果如图 28.29 所示。

如图 28.30 显示了变换中涉及的矩阵。

图 28.29　图像的全局变换

$$\underset{\text{Scale}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \underset{\text{Translate}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 50 & 0 & 1 \end{bmatrix}} \underset{\text{Rotate}}{\begin{bmatrix} \cos 30° & \sin 30° & 0 \\ -\sin 30° & \cos 30° & 0 \\ 0 & 0 & 1 \end{bmatrix}} = \begin{bmatrix} \cos 30° & \sin 30° & 0 \\ -0.5\sin 30° & 0.5\cos 30° & 0 \\ 50\cos 30° & 50\sin 30° & 1 \end{bmatrix}$$

图 28.30　和示例相关的矩阵

知识扩展：在前面的示例中，椭圆围绕坐标系的原点旋转。原点位于工作区的左上角。与椭圆围绕其自身中心旋转相比，这样会产生不同的结果。

2．局部变换

局部变换是应用于要绘制的特定项目的变换，例如，GraphicsPath 对象的 Transform 方法可用来对该路径的数据点进行变换。

例如，下面的示例绘制了一个没有变换的矩形以及一个有旋转变换的路径。代码如下：

```
Matrix myMatrix = new Matrix();
myMatrix.Rotate(45);
myGraphicsPath.Transform(myMatrix);
myGraphics.DrawRectangle(myPen, 10, 10, 100, 50);
myGraphics.DrawPath(myPen, myGraphicsPath);
```

另外，局部变换还可与世界变换合并，得到多种结果。例如，可以使用世界变换来修正坐标系统，而局部变换则可用来旋转和缩放在坐标系统上绘制的对象。

假定您需要原点距工作区左边缘 200 像素、距工作区顶部 150 像素的坐标系统。此外，假定您需要的度量单位是像素，且 *x* 轴指向右方，*y* 轴指向上方。默认的坐标系统是 *y* 轴指向下方，因此您需要执行绕水平坐标轴的反射。如图 28.31 显示了这样的矩阵反射。

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

图 28.31　矩阵的反射结果

下一步，假定您需要执行一个向右 200 个单位、向下 150 个单位的平移。

例如，通过设置 Graphics 对象的世界变换，建立前面刚刚描述过的坐标系统。代码如下：

```
Matrix myMatrix = new Matrix(1, 0, 0, -1, 0, 0);
myGraphics.Transform = myMatrix;
myGraphics.TranslateTransform(200, 150, MatrixOrder.Append);
```

示例 28-21　图像的局部变换

本示例中，当程序运行时，单击“变换”按钮，在窗体中的指定位置绘制一个矩形和一个

经过局部变换处理的矩形。

代码 28-21　光盘位置：光盘\mingrisoft\sl\28\21

Form1 窗体中，在“变换”按钮的 Click 事件中，首先通过调用 Graphics 对象的 FillPath 方法绘制一个普通矩形，然后声明 Matrix 类的一个实例对象，并通过调用该对象的 Scale 方法和 Rotate 方法，创建一个在水平方向上缩放两倍，然后旋转 30° 的局部变换，同时通过 Transform 属性将该局部变换应用到 GraphicsPath 对象，最后再次调用 Graphics 对象的 FillPath 方法绘制矩形。“变换”按钮的 Click 事件代码如下：

```
private void LocalTransform_Click(object sender, EventArgs e)
{
    Graphics graphics = this.CreateGraphics();                    //实例化一个 GDI+绘图图面类对象
    GraphicsPath myGraphicsPath = new GraphicsPath();             //实例化一个表示一系列相互连接的直线和曲线的对象
    Rectangle myRectangle = new Rectangle(20, 20, 60, 40);        //实例化一个矩形工作区域的对象
    myGraphicsPath.AddRectangle(myRectangle);                     //向指定路径中添加一个矩形
    SolidBrush mySolidBrush1 = new SolidBrush(Color.LightGreen); //实例化一个定义单色画笔类对象
    graphics.FillPath(mySolidBrush1, myGraphicsPath);             //用指定的颜色 mySolidBrush1 填充指定结构的内部
    Matrix myPathMatrix = new Matrix();                           //实例化一个表示几何变换的 3*3 矩阵
    myPathMatrix.Scale(2, 1);        //通过预先计算比例向量，将指定的比例向量应用到当前矩阵中
    myPathMatrix.Rotate(30, MatrixOrder.Append);                  //按照顺时针旋转该矩阵
    myGraphicsPath.Transform(myPathMatrix);                       //将变换矩阵应用到当前矩阵中
    SolidBrush mySolidBrush2 = new SolidBrush(Color.YellowGreen);//实例化一个定义单色画笔类对象
    graphics.FillPath(mySolidBrush2, myGraphicsPath);             //用指定的颜色 mySolidBrush2 填充指定结构的内部
}
```

运行本示例，效果如图 28.32 所示。

图 28.32　图像的局部变换

注意：使用 Matrix 类时，需要添加 System.Drawing.Drawing2D 命名空间。

28.5　GDI+绘制图表的应用

随着编程技术的快速发展，图表技术已经越来越受到人们的欢迎。下面通过几个典型实例介绍不同类型图表在程序开发中的应用。

28.5.1　绘制柱型图分析商品月销售情况

实例位置：光盘\mingrisoft\dxyy\28\01

柱型图也称为条形图，它是网站开发中最常用的一种图表技术之一。本节通过一个实例来介绍使用柱型图分析商品的月销售情况。

程序开发步骤如下。

(1) 新建一个网站，命名为 DrawColumnWebApplication，其主页默认为 Default.aspx。

(2) 在该网站中添加一个 Info.aspx 页面，该页面主要用来使用柱型图显示各月份的商品销售情况。

(3) 程序主要代码如下。

Info.aspx 页面中定义了两个方法，分别为 sumNum 方法和 CreateImage 方法，其中 sumNum

方法用来计算各月份的商品销售量总和，而 CreateImage 方法则用来调用 Graphics 对象的相关方法绘制柱型图。sumNum 方法实现代码如下：

```
    private int sumNum(int P_int_year)
    {
        string P_str_sum = "SELECT
SUM(month1+month2+month3+month4+month5+month6+month7+month8+month9+month10+month11+month12) AS
number FROM tb_01 WHERE yearID=" + P_int_year + "";                    //实例化一个数据库查询语句
        //实例化一个数据库连接对象
        SqlConnection sqlcon = new SqlConnection(ConfigurationManager.AppSettings["connectionString"]);
        SqlDataAdapter myda = new SqlDataAdapter(P_str_sum, sqlcon);      //实例化一个更新数据库的实例
        DataSet myds = new DataSet();                                     //实例化一个数据集 myds
        myda.Fill(myds);                                                  //向数据集 myds 中填充数据
        return Convert.ToInt32(myds.Tables[0].Rows[0][0].ToString());     //返回表中第 0 行第 0 列的内容
    }
```

CreateImage 方法实现代码如下：

```
    private void CreateImage(int P_int_year)
    {
        int height = 400, width = 600;                          //实例化两个整型变量 height、width
        Bitmap image = new Bitmap(width, height);               //实例化一个 GDI+绘图图面类对象
        Graphics graphics = Graphics.FromImage(image);          //创建一个 Graphics 类对象
        try
        {
            graphics.Clear(Color.White);                        //清空图片背景色
            Font font = new Font("Arial", 9, FontStyle.Regular);    //实例化一个用于定义特定的文本格式的实例 font
            Font font1 = new Font("宋体", 20, FontStyle.Regular); //实例化一个用于定义特定的文本格式的实例 font1
            LinearGradientBrush brush = new LinearGradientBrush(new Rectangle(0, 0, image.Width, image.Height),
Color.Blue, Color.Blue, 1.2f, true);                            //实例化一个线性渐变对象
            //向指定的矩形工作区域内部填充指定颜色
            graphics.FillRectangle(Brushes.WhiteSmoke, 0, 0, width, height);
            Brush brush1 = new SolidBrush(Color.Blue);              //实例化一个用于填充图形形状的内部的对象
            //在指定的位置绘制指定的文本内容
            graphics.DrawString("" + P_int_year + "年商品月销售情况", font1, brush1, new PointF(150, 30));
            graphics.DrawRectangle(new Pen(Color.Blue), 0, 0, image.Width - 1, image.Height - 1);//画图片的边框线
            Pen mypen = new Pen(brush, 1);                          //实例化一个用于绘制直线和曲线的对象 mypen
            //绘制横向线条
            int x = 100;                                            //实例化一个整型变量
            for (int i = 0; i < 11; i++)                            //从 0 循环遍历至 10
            {
                graphics.DrawLine(mypen, x, 80, x, 366);            //用指定的对象 mypen 绘制指定的线条
                x = x + 40;                                         //递增变量 x 的值
            }
            Pen mypen1 = new Pen(Color.Blue, 2);                    //实例化一个用于绘制直线和曲线的对象 mypen1
            graphics.DrawLine(mypen1, x - 480, 80, x - 480, 366); //用指定的对象 mypen1 绘制指定的线条
            //绘制纵向线条
            int y = 106;                                            //实例化一个整型变量
            for (int i = 0; i < 10; i++)                            //从 0 循环遍历至 9
            {
                graphics.DrawLine(mypen, 60, y, 540, y);            //用指定的对象 mypen 绘制指定的线条
                y = y + 26;                                         //递增变量 y 的值
            }
            graphics.DrawLine(mypen1, 60, y, 540, y);               //以指定的对象 mypen1 绘制指定的线条
            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" };                                      //实例化一个字符串数组 n
            x = 60;                                                 //重新为变量 x 赋值
            for (int i = 0; i < 12; i++)                            //从 0 循环遍历至 11
            {
                graphics.DrawString(n[i].ToString(), font, Brushes.Red, x, 374);     //设置文字内容及输出位置
                x = x + 40;                                         //递增变量 x 的值
            }
            //实例化一个字符串数组 m
            String[] m = { "100%", " 90%", " 80%", " 70%", " 60%", " 50%", " 40%", " 30%", " 20%", " 10%", "  0%" };
```

```
y = 98;                                                          //重新为变量 y 赋值
for (int i = 0; i < 11; i++)                                     //从 0 循环遍历至 10
{
    graphics.DrawString(m[i].ToString(), font, Brushes.Red, 25, y);   //设置文字内容及输出位置
    y = y + 26;                                                  //递增变量 y 的值
}
int[] Count = new int[12];                                       //实例化一个整型数组
//实例化一个数据库查询语句
string P_str_yearID = "SELECT * FROM tb_01 WHERE yearID=" + P_int_year + "";
//实例化一个数据库连接对象
SqlConnection sqlcon = new SqlConnection(ConfigurationManager.AppSettings["connectionString"]);
//实例化一个用于操作数据库的对象
SqlDataAdapter myda = new SqlDataAdapter(P_str_yearID, sqlcon);
DataSet myds = new DataSet();                                    //实例化一个数据集 myds
myda.Fill(myds);                                                 //填充数据集 myds
int P_int_num = sumNum(P_int_year);          //实例化一个整型变量 P_int_num 用于保存一年的总销售量
for (int j = 0; j < 12; j++)                 //从 0 循环遍历至 12
{
    //计算当前的销售量占总销售量的百分比
    Count[j] = Convert.ToInt32(myds.Tables[0].Rows[0][j + 1].ToString()) * 100 / P_int_num;
}
//显示柱状效果
x = 70;                                      //重新为变量 x 赋值
for (int i = 0; i < 12; i++)                 //从 0 循环遍历至 11
{
    SolidBrush mybrush = new SolidBrush(Color.Red);  //实例化一个用于定义单色画笔类对象
    //用指定的对象填充矩形区域的内部
    graphics.FillRectangle(mybrush, x, 366 - Count[i] * 26 / 10, 20, Count[i] * 26 / 10);
    x = x + 40;                              //递增变量 x 的值
}
System.IO.MemoryStream MStream = new System.IO.MemoryStream();  //实例化一个内存流对象
//将当前图像以指定的格式保存在指定的流中
image.Save(MStream, System.Drawing.Imaging.ImageFormat.Gif);
Response.ClearContent();                     //清除缓冲区流中的所有内容
Response.ContentType = "image/Gif";          //设置输出流的 HTTP MIME 类型
Response.BinaryWrite(MStream.ToArray());     //将一个二进制字符串写入输出流中
}
finally
{
    graphics.Dispose();     //释放对象 graphics 占用的系统资源
    image.Dispose();        //释放对象 image 占用的系统资源
}
}
```

运行本示例，效果如图 28.33 所示。

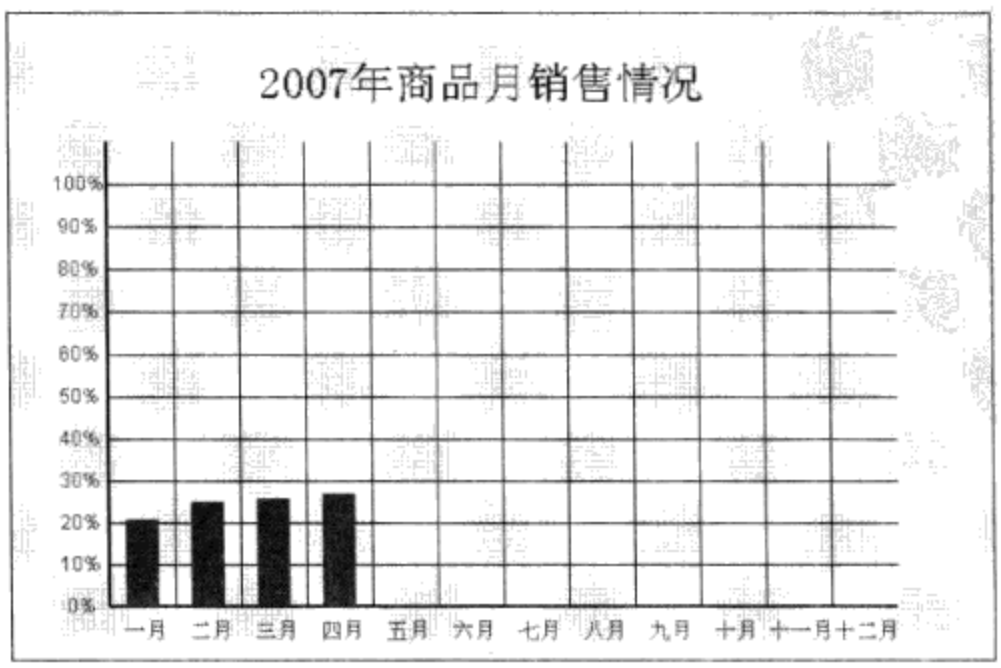


图 28.33　柱型图分析商品月销售情况

28.5.2 绘制饼图分析男女比例

实例位置：光盘\mingrisoft\dxyy\28\02

饼图是一种常用来显示数据比例的图表技术，常用来突出某一重要数据。本节将通过一个实例来介绍使用饼图显示公司员工的男女比例情况。

程序开发步骤如下。

（1）新建一个 Windows 应用程序，命名为 DrawCakeShapeApplication，其主窗体默认为 Form1.cs。

（2）Form1.cs 窗体中，添加一个 Button 控件，用来执行画图操作。

（3）程序主要代码如下。

单击“分析男女比例”按钮，程序调用 Graphics 对象的 FillPie 方法，根据数据库中记录，绘制一张体现公司员工男女比例的饼图。“分析男女比例”按钮的 Click 事件代码如下：

```
private void DrawCakeShape_Click(object sender, EventArgs e)
{
    //实例化一个数据库连接对象
    SqlConnection sqlcon = new SqlConnection("Data Source=(local);Database=db_28;Uid=sa;Pwd=");
    string P_str_sel = "select *  from tb_02";                //实例化一个数据库连接字符串
    SqlDataAdapter myda1 = new SqlDataAdapter(P_str_sel, sqlcon);  //实例化一个操作数据库对象 myda1
    DataSet myds1 = new DataSet();                            //实例化一个数据集对象 myds1
    myda1.Fill(myds1);                                        //向数据集 myds1 中填充内容
    string P_str_sumNum = "select sum(boyNum+girlNum) as sumNum FROM tb_02";  //计算男女同事的总数
    SqlDataAdapter myda2 = new SqlDataAdapter(P_str_sumNum, sqlcon);  //实例化一个操作数据库对象 myda2
    DataSet myds2 = new DataSet();                            //实例化一个数据集对象 myds2
    myda2.Fill(myds2);                                        //向数据集 myds2 中填充内容
    int P_int_sum = Convert.ToInt32(myds2.Tables[0].Rows[0][0].ToString());//获取表 myds2 中第 0 行第 0 列的元素
    int P_int_boy = Convert.ToInt32(myds1.Tables[0].Rows[0][0].ToString());//获取表 myds1 中第 0 行第 0 列的元素
    int P_int_girl = Convert.ToInt32(myds1.Tables[0].Rows[0][1].ToString());//获取表 myds1 中第 0 行第 1 列的元素
    Graphics graphics = this.CreateGraphics();                //实例化一个 GDI+绘图图面类对象
    try
    {
        graphics.Clear(Color.White);                          //清除整个绘图面并以白色填充
        Pen pen1 = new Pen(Color.Red);                        //实例化一个用于绘制直线和曲线的对象
        Brush brush1 = new SolidBrush(Color.YellowGreen);     //实例化一个用于填充图形形状内部的对象 brush1
        Brush brush2 = new SolidBrush(Color.Blue);            //实例化一个用于填充图形形状内部的对象 brush2
        Brush brush3 = new SolidBrush(Color.Brown);           //实例化一个用于填充图形形状内部的对象 brush3
        Font font1 = new Font("Courier New", 16, FontStyle.Bold);//实例化一个定义特定的文本格式的对象 font1
        Font font2 = new Font("Courier New", 8);              //实例化一个定义特定的文本格式的对象 font2
        graphics.FillRectangle(brush1, 0, 0, 370, 350);       //用指定的格式填充指定的矩形区域
        graphics.DrawString("公司员工男女比例分析", font1, brush2, new Point(60, 20));//书写标题
        //男员工在圆中分配的角度
        float angle1 = Convert.ToSingle((360 * (Convert.ToSingle(P_int_boy) / Convert.ToSingle(P_int_sum))));
        //女员工在圆中分配的角度
        float angle2 = Convert.ToSingle((360 * (Convert.ToSingle(P_int_girl) / Convert.ToSingle(P_int_sum))));
        graphics.FillPie(brush2, 100, 60, 180, 180, 0, angle1);       //绘制男员工所占比例
        graphics.FillPie(brush3, 100, 60, 180, 180, angle1, angle2);  //绘制女员工所占比例
        graphics.DrawRectangle(pen1, 50, 255, 260, 50);               //绘制范围框
        graphics.FillRectangle(brush2, 85, 265, 20, 10);              //绘制小矩形
        //在指定位置按指定格式绘制指定的内容(男员工占公司总人数比例:)
        graphics.DrawString("男员工占公司总人数比例:" + Convert.ToInt32(P_int_boy) * 100 /
Convert.ToInt32(P_int_sum) + "%", font2, brush2, 120, 265);
        graphics.FillRectangle(brush3, 85, 285, 20, 10);              //用指定的格式填充矩形的内部
        //在指定位置按指定格式绘制指定的内容(女员工占公司总人数比例:)
        graphics.DrawString("女员工占公司总人数比例:" + Convert.ToInt32(P_int_girl) * 100 /
Convert.ToInt32(P_int_sum) + "%", font2, brush3, 120, 285);
    }
    catch (Exception ex)                                      //捕获异常
```

```
        {
            //弹出错误提示信息
            MessageBox.Show(ex.Message, "信息", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        graphics.Dispose();                          //释放 graphics 所占用的资源
    }
```

运行本示例，效果如图 28.34 所示。

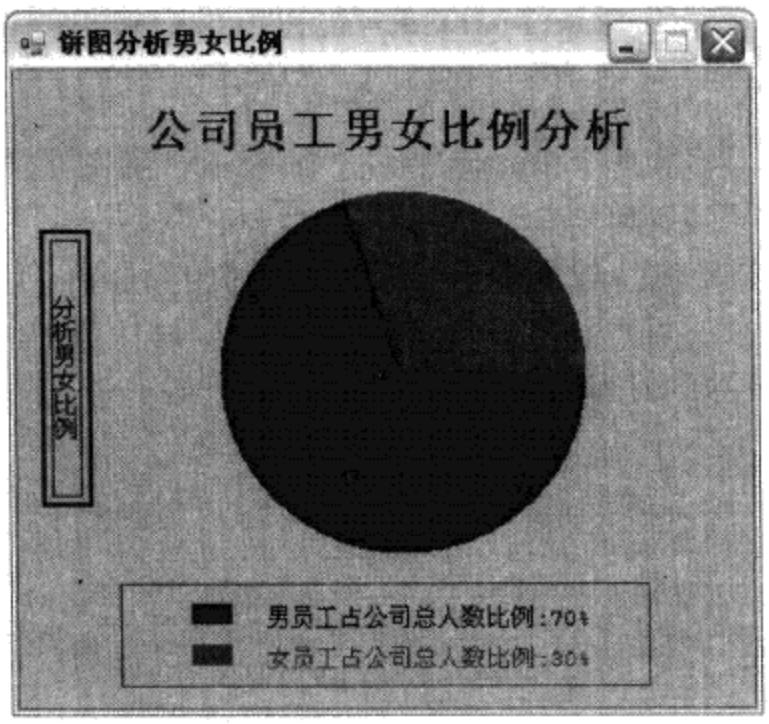


图 28.34　饼图分析男女比例

28.5.3　绘制折线图分析网站流量

实例位置：光盘\mingrisoft\dxyy\28\03

折线图是网站开发中经常用到的一种图表技术，折线图可以很直观地反映出相关数据的变化趋势。本节通过一个实例来介绍如何使用折线图分析网站的流量。

程序开发步骤如下。

（1）新建一个网站，命名为 NetFluxAnalysis，其主页默认为 Default.aspx。

（2）在该网站中添加一个 Info.aspx 页面，该页面主要用来使用折线图显示各月份的网站访问人数。

（3）程序主要代码如下。

Info.aspx 页面中定义了一个方法 CreateImage，该方法主要用来从数据库中提取数据，并根据提取的数据，调用 Graphics 对象的相关方法绘制一张反映网站流量情况的折线图。CreateImage 方法实现代码如下：

```
    private void CreateImage(int P_int_year)
    {
        int height = 400, width = 600;                      //实例化两个整型变量用来限定 GDI+位图对象的高和宽
        Bitmap image = new Bitmap(width, height);           //实例化一个 GDI+位图对象
        Graphics graphics = Graphics.FromImage(image);      //实例化一个 GDI+绘图图面类对象
        try
        {
            graphics.Clear(Color.White);                    //清除整个绘图面并以指定的颜色填充
            Font font = new Font("Arial", 9, FontStyle.Regular);     //实例化一个用于指定文本格式的对象 font
            Font font1 = new Font("宋体", 20, FontStyle.Regular);  //实例化一个用于指定文本格式的对象 font1
            LinearGradientBrush brush = new LinearGradientBrush(new Rectangle(0, 0, image.Width, image.Height),
Color.Blue, Color.Blue, 1.2f, true);                        //实例化一个线性渐变类对象
            graphics.FillRectangle(Brushes.WhiteSmoke, 0, 0, width, height);//以指定的格式填充指定的矩形工作区域
            Brush brush1 = new SolidBrush(Color.Blue);          //实例化一个用于填充图形形状内部的对象
            //实例化一个数据库查询字符串 P_str_sum
```

```
string P_str_sum = "SELECT * FROM tb_03 WHERE yearID=" + P_int_year + "";
//实例化一个数据库连接对象
SqlConnection sqlcon = new SqlConnection(ConfigurationManager.AppSettings["connectionString"]);
sqlcon.Open();                                          //打开数据库
SqlCommand sqlcom = new SqlCommand(P_str_sum, sqlcon);  //实例化一个用于指定 SQL 语句的对象
//实例化一个数据库读取器
SqlDataReader sqlread = sqlcom.ExecuteReader(CommandBehavior.CloseConnection);
sqlread.Read();                                         //读取数据库中的数据
if (sqlread.HasRows)                                    //当数据库仍存在数据时
{
    //在指定的位置用指定的格式绘制字符串
    graphics.DrawString("" + P_int_year + "年各月份网站访问人数", font1, brush1, new PointF(130, 30));
}
sqlread.Close();                                        //关闭数据读取器对象
graphics.DrawRectangle(new Pen(Color.Blue), 0, 0, image.Width - 1, image.Height - 1);//画图片的边框线
Pen mypen = new Pen(brush, 1);                          //实例化一个用于绘制直线和曲线的对象 mypen
//绘制横向线条
int x = 100;                                            //实例化一个整型变量 x
for (int i = 0; i < 11; i++)                            //从 0 循环至 10
{
    graphics.DrawLine(mypen, x, 80, x, 340);            //在指定的位置绘制直线
    x = x + 40;                                         //递增变量 x 的值
}
Pen mypen1 = new Pen(Color.Blue, 2);                    //实例化一个用于绘制直线和曲线的对象 mypen1
Pen mypen2 = new Pen(Color.Red, 2);                     //实例化一个用于绘制直线和曲线的对象 mypen2
graphics.DrawLine(mypen1, x - 480, 80, x - 480, 340);   //在指定的位置以指定格式 mypen1 绘制线条
//绘制纵向线条
int y = 106;                                            //实例化一个整型变量 y
for (int i = 0; i < 9; i++)                             //从 0 循环遍历至 8
{
    graphics.DrawLine(mypen, 60, y, 540, y);            //在指定的位置以指定格式 mypen 绘制线条
    y = y + 26;                                         //递增变量 y 的值
}
graphics.DrawLine(mypen1, 60, y, 540, y);               //在指定的位置以指定格式 mypen1 绘制线条
//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"};                              //实例化一个字符串数组 n
x = 35;                                                 //重新为变量 x 赋值
for (int i = 0; i < 12; i++)                            //从 0 循环遍历至 11
{
    graphics.DrawString(n[i].ToString(), font, Brushes.Red, x, 348);   //设置文字内容及输出位置
    x = x + 40;                                         //递增变量 x 的值
}
//y 轴
String[] m = { " 900 人", " 800 人", " 700 人", " 600 人", " 500 人", " 400 人", " 300 人", " 200 人", " 100 人", "
0 人" };                                                //实例化一个字符串数组 m
y = 98;                                                 //重新为变量 y 赋值
for (int i = 0; i < 10; i++)                            //从 0 循环遍历至 9
{
    graphics.DrawString(m[i].ToString(), font, Brushes.Red, 20, y);   //设置文字内容及输出位置
    y = y + 26;                                         //递增变量 y 的值
}
int[] Count = new int[12];                              //实例化一个整型数组
//实例化一个数据库查询字符串
string P_str_yearID = "SELECT * FROM tb_03 WHERE yearID=" + P_int_year + "";
SqlDataAdapter myda = new SqlDataAdapter(P_str_yearID, sqlcon);//实例化一个用于操作数据库的对象
DataSet myds = new DataSet();                           //实例化一个数据集对象 myds
myda.Fill(myds);                                        //向数据集 myds 中填充内容
for (int j = 0; j < 12; j++)                            //从 0 循环遍历至 11
{
    //计算每个月的流量所占的百分比
    Count[j] = Convert.ToInt32(myds.Tables[0].Rows[0][j + 1].ToString()) * 26 / 100;
```

```
        }
        SolidBrush mybrush = new SolidBrush(Color.Red);     //实例化一个单色画笔类对象 mybrush
        Point[] myPoint = new Point[12];                    //实例化一个坐标点数组 myPoint
        myPoint[0].X = 60; myPoint[0].Y = 340 - Count[0];   //设置数组中每个元素的 X 坐标与 Y 坐标
        myPoint[1].X = 100; myPoint[1].Y = 340 - Count[1];
        myPoint[2].X = 140; myPoint[2].Y = 340 - Count[2];
        myPoint[3].X = 180; myPoint[3].Y = 340 - Count[3];
        myPoint[4].X = 220; myPoint[4].Y = 340 - Count[4];
        myPoint[5].X = 260; myPoint[5].Y = 340 - Count[5];
        myPoint[6].X = 300; myPoint[6].Y = 340 - Count[6];
        myPoint[7].X = 340; myPoint[7].Y = 340 - Count[7];
        myPoint[8].X = 380; myPoint[8].Y = 340 - Count[8];
        myPoint[9].X = 420; myPoint[9].Y = 340 - Count[9];
        myPoint[10].X = 460; myPoint[10].Y = 340 - Count[10];
        myPoint[11].X = 500; myPoint[11].Y = 340 - Count[11];
        graphics.DrawLines(mypen2, myPoint);                //绘制折线
        System.IO.MemoryStream MStream = new System.IO.MemoryStream();  //实例化一个内存流对象
        //将当前的图像以指定的格式保存到流中
        image.Save(MStream, System.Drawing.Imaging.ImageFormat.Gif);
        Response.ClearContent();                            //清除缓冲区流中的所有内容
        Response.ContentType = "image/Gif";                 //设置输出流的 HTTP MIME 类型
        Response.BinaryWrite(MStream.ToArray());            //将一个二进制字符串输入 HTTP 输出流
    }
    finally
    {
        graphics.Dispose();                                 //释放对象 graphics 所占用的资源
        image.Dispose();                                    //释放对象 image 所占用的资源
    }
}
```

运行本示例，效果如图 28.35 所示。

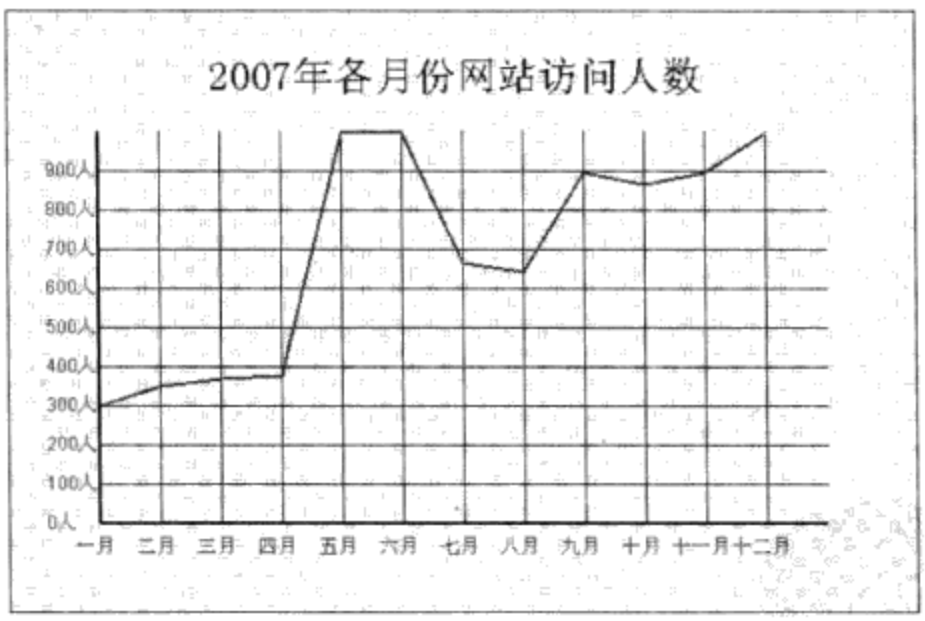


图 28.35　折线图分析网站流量

28.5.4　绘制验证码图片

实例位置：光盘\mingrisoft\dxyy\28\04

在网站开发过程中，验证码技术是保护网站安全的最基本环节，它可以防止非法人员利用注册工具或登录工具来攻击网站。本节通过一个实例来介绍如何在网站中绘制验证码图片。

程序开发步骤如下。

(1) 新建一个网站，将其命名为 DrawIdentifyingCode，默认主页名为 Default.aspx。

(2) 在该网站中添加一个 code.aspx 页面，用于生成验证码图片。

(3) 程序主要代码如下。

code.aspx 页面中定义了两个方法，分别为 GenerateCheckCode 方法和 CreateCheckCodeImage

方法，其中 GenerateCheckCode 方法用来生成随机字符串，而 CreateCheckCodeImage 方法则用来将随机生成的字符串转换成验证码图片。GenerateCheckCode 方法实现代码如下：

```
protected string GenerateCheckCode(int number)
{
    string checkCode = String.Empty;                    //实例化一个字符串 checkCode
    //实例化一个字符串 Vchar
    string Vchar =
"0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z";
    string[] VcArray = Vchar.Split(',');                //实例化一个字符串数组 VcArray
    int temp = -1;                                      //记录上次随机数值，尽量避免产生几个一样的随机数
    Random rand = new Random();                         //实例化一个随机类的对象
    for (int i = 1; i < number + 1; i++)                //从 1 循环遍历至 number
    {
        if (temp != -1)                                 //当 number 的值不为 0 时
        {
            rand = new Random(i * temp * unchecked((int)DateTime.Now.Ticks));  //重新实例化对象 rand
        }
        int t = rand.Next(VcArray.Length);              //返回一个小于最大值的非负随机数
        if (temp != -1 && temp == t)                    //当 number 的值不为 0 且 temp 的值为 t 时
        {
            return GenerateCheckCode(number);//递归调用 GenerateCheckCode 方法
        }
        temp = t;                                       //设置 temp 的值为 t
        checkCode += VcArray[t];                        //为字符串 checkCode 赋值
    }
    Response.Cookies.Add(new HttpCookie("CheckCode", checkCode));  //将 checkCode 的值添加至 Cooki 集合中
    return checkCode;                                   //返回 checkCode 的值
}
```

CreateCheckCodeImage 方法实现代码如下：

```
private void CreateCheckCodeImage(string checkCode)
{
    if (checkCode == null || checkCode.Trim() == String.Empty)   //当 checkCode 的值为空时
        return;                                         //跳出该方法
    //实例化一个 GDI+位图对象
    System.Drawing.Bitmap image = new System.Drawing.Bitmap((int)Math.Ceiling((checkCode.Length * 12.5)), 22);
    Graphics g = Graphics.FromImage(image);             //实例化一个 GDI+绘图图面类对象
    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
        {
            int x1 = random.Next(image.Width);          //实例化第一个点的 x 坐标
            int x2 = random.Next(image.Width);          //实例化第二个点的 x 坐标
            int y1 = random.Next(image.Height);         //实例化第一个点的 y 坐标
            int y2 = random.Next(image.Height);         //实例化第二个点的 y 坐标
            g.DrawLine(new Pen(Color.Black), x1, y1, x2, y2);  //绘制一条连接指定两点的直线
        }
        Font font = new System.Drawing.Font("Arial", 12, (System.Drawing.FontStyle.Bold |
System.Drawing.FontStyle.Italic));                      //实例化一个文本格式对象 font
        System.Drawing.Drawing2D.LinearGradientBrush brush = new
System.Drawing.Drawing2D.LinearGradientBrush(new Rectangle(0, 0, image.Width, image.Height), Color.Blue,
Color.DarkRed, 1.2f, true);                             //实例化一个线性渐变类对象
        g.DrawString(checkCode, font, brush, 2, 2);     //在指定的位置以指定的格式绘制指定的内容
        for (int i = 0; i < 100; i++)                   //画图片的前景噪音点
        {
            int x = random.Next(image.Width);           //设置像素点的 x 坐标
            int y = random.Next(image.Height);          //设置像素点的 y 坐标
            image.SetPixel(x, y, Color.FromArgb(random.Next()));  //获取位图图像中指定像素点的颜色
        }
        //以指定的颜色绘制矩形工作区域
```

```
            g.DrawRectangle(new Pen(Color.Silver), 0, 0, image.Width - 1, image.Height - 1);
            System.IO.MemoryStream ms = new System.IO.MemoryStream();//实例化一个内存流对象
            image.Save(ms, System.Drawing.Imaging.ImageFormat.Gif);        //将此图像以指定的格式存储在内存流中
            Response.ClearContent();                                       //清除缓冲区流中的所有内容
            Response.ContentType = "image/Gif";                            //设置输出流的的 HTTP MIME 类型
            Response.BinaryWrite(ms.ToArray());                            //将一个二进制字符串写入 HTTP 输出流
        }
        finally
        {
            g.Dispose();                                                   //释放对象 g 所占用的资源
            image.Dispose();                                               //释放对象 image 所占用的资源
        }
    }
```

将 Default.aspx 页设置为首页，在该页中添加一个 Image 控件，用来显示验证码图片，并将其 ImageUrl 属性设置为“code.aspx”，实现代码如下：

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/code.aspx"/>
```

运行本示例，效果如图 28.36 所示。

图 28.36　验证码图片

第 29 章

多媒体技术

- 多媒体概述
- 文字处理
- 音频与视频处理
- 动画设计

多媒体技术是在计算机、电子、通信等技术的基础上建立和发展起来的。它的出现标志着信息社会的到来。在.NET 中主要通过两种设计方式实现多媒体程序，一种是利用多媒体控件，另一种是利用 DirectShow 技术。

29.1 多媒体概述

多媒体技术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。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、操作和控制的多媒体信息集成环境和交互系统。

在 Windows 平台上，多媒体元素主要包括音频、数字视频、模拟视频、动画、图像和文本 6 种基本的多媒体类型。其中，图形和图像技术在前面的章节中已经进行了介绍，在本章就不再赘述，本章仅向读者介绍文本、音频、视频、动画这 4 种多媒体类型的处理和应用。

1．文本

文本一般分为普通文本和丰富文本，普通文本又称为非格式化文本，丰富文本又称为格式化文本。

2．音频

音频可分为 3 种类型，即波形音频、CD Audio 和 MIDI。

- 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。
- 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。
- 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。

音频文件常见的格式有 WAV、MOD、MP3、RA、CMF、CDA、MID 等。

3．视频

视频分为数字视频和模拟视频。

- 数字视频是经过视频采集卡收集到的数字化文件。
- 模拟视频与数字视频的信息来源不同，模拟视频通常来自于摄像机、VCD 和激光影碟等，模拟视频可以使应用程序在一个可缩放的窗口中不丢帧并以 30 帧/秒的速率播放视频。

4．动画

所谓动画，就是通过以 15～20 帧/秒的速度顺序地播放一系列连续动作的图形图像，并可以带有与其同步的音频。

29.2 文 字 处 理

在 C#中可以通过 Label 控件、TextBox 控件、窗体和 PictureBox 控件来显示文字。窗体和 PictureBox 控件主要是通过 DrawString 方法实现，而 TextBox 控件和 Label 控件是专门用来显示文字信息的。要想实现文字的滚动，可以通过将 Label 控件的 Left、Top 属性和 Timer 控件相结合，使 Label 控件每隔一段时间就移动一次位置，如果时间间隔合适，就可以实现 Label 控件中显示的文字滚动的效果。

示例 29-01　由左向右滚动的文字

本示例利用 Label 控件来实现文字的滚动。要实现文字由左向右滚动，可以设置 Label 控件的 Left 属性，例如，“label1.Left = label1.Left + 25；”。

程序开发步骤如下。

（1）创建一个项目，命名为 RollCharacterApplication，设置默认窗体的 Text 属性为“滚动文字”。

（2）在窗体上添加一个 Label 控件和一个 Timer 控件，设置 Timer 控件的 Interval 属性为 200。

（3）程序代码如下。

代码 29-01　光盘位置：光盘\mingrisoft\sl\29\01

```
private void timer1_Tick(object sender, EventArgs e)
{
    this.label1.ForeColor = Color.Red;      //设置 label1 控件的前景色
    if (label1.Left < this.Width)//设置控件左边缘与其容器的工作区左边缘之间的距离小于当前窗体的宽度时
    {
        label1.Left = label1.Left + 25;     //设置控件左边缘与其容器的工作区左边缘之间的距离递增 25
    }
    //当控件左边缘与其容器的工作区左边缘之间的距离大于当前窗体宽度的相反数时
    else if (label1.Left > -this.Width)
    {
        //设置控件左边缘与其容器的工作区左边缘之间的距离为当前控件宽度的相反数
        label1.Left = -label1.Width;
    }
}
```

运行本示例，效果如图 29.1 所示。

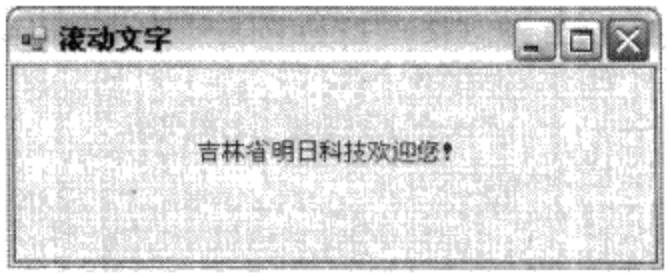


图 29.1　滚动文字

29.3　音频与视频处理

29.3.1　Windows Media Player 控件

Windows Media Player 是一种媒体播放器，可以播放当前最流行的音频、视频文件和大多数混合型的多媒体文件。为了便于程序的开发，Visual Studio 2008 集成开发环境提供了 Windows Media Player 控件，并且提供了相关的属性、方法，开发者根据提供的属性、方法完全可以实现 Windows Media Player 播放器的所有功能。

在使用 Windows Media Player 控件进行程序开发前，必须将 Windows Media Player 控件添加到工具箱中，步骤如下。

（1）选择工具箱，并单击鼠标右键，在弹出的快捷菜单中选择“选择项”。

（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，选择名称为“Windows Media Player”，单击“确定”按钮，Windows Media Player 控件添加成功，如图 29.2 所示。

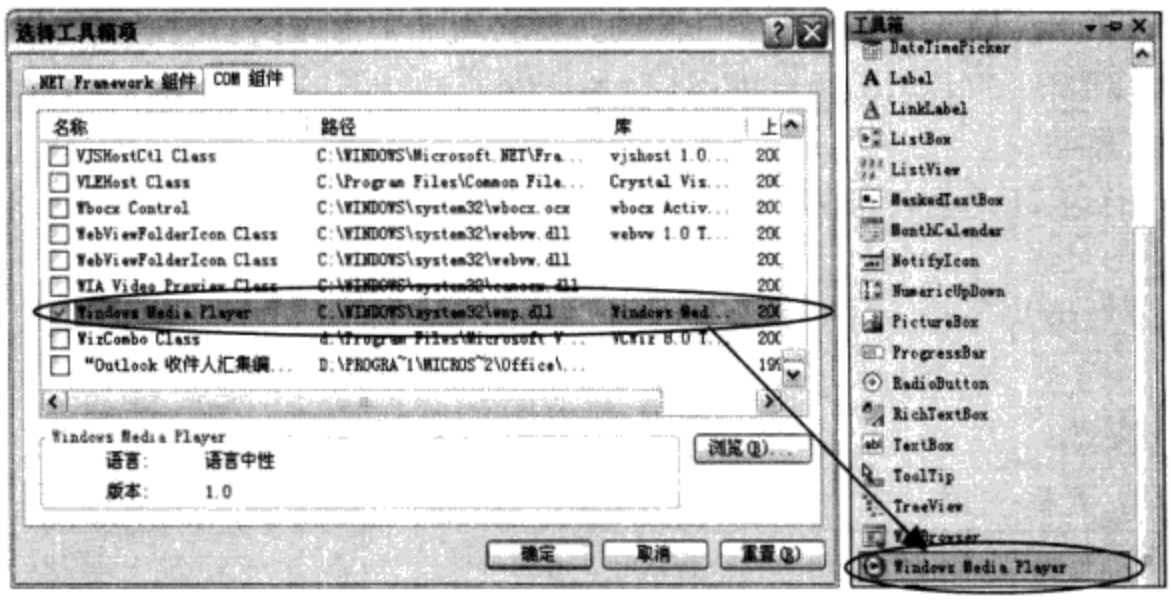


图 29.2　添加 Windows Media Player 控件

表 29.1 和表 29.21 所示为 Windows Media Player 控件提供的主要属性和方法。

表 29.1　　Windows Media Player 控件主要属性及说明

属　性	说　明
cdromCollection	获取一个 IWMPCdromCollection 接口
closedCaption	获取一个 IWMPClosedCaption 接口
Ctlcontrols	获取一个 IWMPControls 接口
Ctlenabled	获取或设置一个值用来表示 Windows Media Player 控件是否可用
currentMedia	获取或设置一个和当前媒体对应的 IWMPMedia 接口
currentPlaylist	获取或设置当前播放列表 IWMPPlaylist 接口
dvd	获取一个 IWMPDVD 接口
enableContextMenu	获取或设置一个值用来表示是否显示右键菜单
Error	获取一个 IWMPError 接口
fullScreen	获取或设置一个值用来表示 Windows Media Player 控件是否处于完全模式
isOnline	获取或设置一个值用来表示用户是否联网
mediaCollection	获取一个 IWMPMediaCollection 接口
network	获取一个 IWMPNetwork 接口
openState	获取一个用来标识目录源的状态
playlistCollection	获取一个 IWMPPlaylistCollection 接口
playState	获取一个值用来表示 Windows Media Player 的操作状态
settings	获取一个 IWMPSettings 接口
status	获取一个值用来标识 Windows Media Player 控件的当前状态
stretchToFit	获取或设置一个值用来指示媒体文件是否自动适应 Windows Media Player 控件的大小
uiMode	获取或设置当 Windows Media Player 控件嵌入网页时在用户界面中显示的控件
URL	获取或设置播放的文件名
versionInfo	获取一个指明 Windows Media Player 版本的值
windowlessVideo	获取或设置一个值用来表示 Windows Media Player 控件是否处于无窗模式

下面对 Windows Media Player 控件比较重要的属性进行详细说明。

1．cdromCollection 属性。

该属性表示获取一个 IWMPCdromCollection 接口。

语法：

```
public virtual IWMPCdromCollection cdromCollection {get;}
```

从 IWMPCdromCollection 接口可以获得一个 IWMPCdromCollection 对象。IWMPCdrom Collection 对象提供一种途径组织和访问 CD 或者 DVD 光驱。下面对 IWMPCdromCollection 对象比较重要的属性 count 进行介绍。

该属性表示获取系统中可用的 CD 和 DVD 光驱数。

语法：

```
int count {get;}
```

属性值：表示系统中可用的光驱数。

例如，获取系统中可用的 CD 和 DVD 光驱个数。代码如下：

```
int numDrives = player.cdromCollection.count;
MessageBox.Show("可用的 CD 和 DVD 光驱个数为:  " + numDrives.ToString());
```

下面对 IWMPCdromCollection 对象比较重要的方法进行介绍。

（1）GetByDriveSpecifier 方法

该方法表示获取具有特定说明文字的光驱。

语法：

```
IWMPCdrom getByDriveSpecifier (string bstrDriveSpecifier);
```

参数说明如下。

bstrDriveSpecifier：表示驱动器的描述文字，在末尾需要有一个冒号。

返回值：表示一个 IWMPCdrom 对象。

例如，获取文本框中描述的指定光驱并弹出该光驱。代码如下：

```
string driveLetter = myText.Text;          //获取指定光驱的描述文字
driveLetter += ":";                        //在描述文字的末尾添加一个冒号
//实例化一个 IWMPCdrom 对象用来接收获取的光驱
WMPLib.IWMPCdrom drive = player.cdromCollection.getByDriveSpecifier(driveLetter);
drive.eject();                             //打开光驱
```

（2）Item 方法

该方法表示获取指定索引下的光驱。

语法：

```
IWMPCdrom Item (int lIndex);
```

参数说明如下。

lIndex：一个整型数值用来表示光驱的索引。

返回值：表示一个 IWMPCdrom 对象。

例如，获取本地计算机上的所有可用光驱及光盘中的内容名称，将它们显示在 ListBox 控件中。代码如下：

```
int numDrives = player.cdromCollection.count;                    //获取本地计算机上的所有可用光驱总数
for (int i = 0; i < numDrives; i++)                              //循环遍历每一个光驱对象
{
    string pl = player.cdromCollection.Item(i).driveSpecifier + " ";   //获取光驱的驱动器号
    pl += player.cdromCollection.Item(i).Playlist.name;          //向字符串中添加 CD 或者 DVD 中的内容
    myPlaylists.Items.Add(pl);                                   //向 ListBox 控件中添加内容
}
```

在上面知识讲解的过程中多次用到 IWMPCdrom 对象，下面对 IWMPCdrom 对象进行详细讲解。IWMPCdrom 对象可以用来访问驱动器中的 CD 或者 DVD。下面对 IWMPCdrom 对象比较重要的属性进行讲解。

（1）DriveSpecifier 属性

该属性表示获取当前计算机上的 CD 或者 DVD 的驱动器号。

语法：

```
string driveSpecifier {get;}
```

属性值：表示光驱的驱动器号。

例如，获取可用的 CD 或者 DVD 驱动器号显示在消息框中。代码如下：

```
string MyDriveSpecifiers = "找到的光驱驱动器号有: ";          //实例化一个提示性字符串
int numDrives = player.cdromCollection.count;                  //实例化一个整型变量用来存储可用的光驱总数
for (int i = 0; i < numDrives; i++)                            //循环遍历光驱集合中的每一个记录
{
    //向字符串 MyDriveSpecifiers 中添加驱动器号
    MyDriveSpecifiers += player.cdromCollection.Item(i).driveSpecifier;
    MyDriveSpecifiers += " ";                                  //追加一个空格
}
System.Windows.Forms.MessageBox.Show(MyDriveSpecifiers);       //弹出信息框
```

（2）Playlist 属性

该属性表示获取一个用来表示 CD 中的目录内容或者 DVD 目录内容的 IWMPPlaylist 对象。

语法：

```
IWMPPlaylist Playlist {get;}
```

属性值：表示一个 IWMPPlaylist 对象。

例如，获取第一个 CD 中的目录内容显示在多行文本框中。代码如下：

```
WMPLib.IWMPPlaylist playlist = player.cdromCollection.Item(0).Playlist;   //实例化一个 IWMPPlaylist 对象
String[] trackList = new String[playlist.count];                          //实例化一个字符串数组
for (int i = 0; i < playlist.count; i++)                                  //循环遍历 playlist 中的每一项
{
    trackList[i] = playlist.get_Item(i).name;                             //为数组赋值
}
myText.Lines = trackList;                                                 //将数组中的内容显示在多行文本框中
```

下面对 IWMPCdrom 对象比较重要的 eject 方法进行详细介绍。

该方法表示弹出光驱。

语法：

```
void eject ();
```

例如，在 Button 的单击事件中弹出 CD 或者 DVD 光驱集合中索引为 0 的光驱。代码如下：

```
private void ejectButton_Click(object o, System.EventArgs args)
{
    player.cdromCollection.Item(0).eject();
}
```

在上面知识的讲解过程中，多次提到 IWMPPlaylist 对象。下面对 IWMPPlaylist 对象进行详细讲解。如表 29.2 所示为 IWMPPlaylist 对象的常用属性。

表 29.2　　IWMPPlaylist 对象的常用属性及说明

属　性	说　明
attributeCount	获取和播放列表关联的属性个数
count	获取播放列表中媒体文件的个数
name	获取或设置播放列表的名称

下面对 IWMPPlaylist 对象的属性进行详细的说明。

（1）attributeCount 属性

该属性表示获取和播放列表关联的属性个数。

语法：

```
int attributeCount {get;}
```

属性值：表示属性个数。

（2）count 属性

该属性表示获取播放列表中媒体文件的个数。

语法：

```
int count {get;}
```

属性值：表示媒体文件的个数。

（3）name 属性

该属性表示获取或设置播放列表的名称。

语法：

```
string name {get; set;}
```

属性值：表示播放列表的名称。

下面对 IWMPPlaylist 对象的常用方法进行讲解。如表 29.3 所示为 IWMPPlaylist 对象的常用方法。

表 29.3　　IWMPPlaylist 对象的常用方法及说明

方　法	说　明
appendItem	向播放列表的末尾添加媒体文件
clear	清空播放列表中的内容
get_attributeName	获取播放列表中指定索引下的详细属性说明
get_isIdentical	获取一个值用来表示指定的播放列表是否是当前播放列表
get_Item	表示获取指定索引下的 IWMPMedia 对象
getItemInfo	获取播放列表中指定属性名下的属性值
insertItem	向播放列表中的指定位置插入媒体文件
moveItem	改变播放列表中媒体文件的位置
removeItem	移除指定的媒体文件
setItemInfo	设置当前播放列表下指定属性的属性值

下面对 IWMPPlaylist 对象的方法进行详细讲解。

（1）appendItem 方法

该方法表示向播放列表的末尾添加媒体文件。

语法：

```
void appendItem(IWMPMedia pIWMPMedia);
```

参数说明如下。

pIWMPMedia：表示即将添加的媒体文件。

例如，向指定的播放列表中添加当前播放的媒体文件。代码如下：

```
WMPLib.IWMPMedia currMedia = player.currentMedia; //实例化一个 IWMPMedia 对象
//实例化一个 IWMPPlaylist 对象
WMPLib.IWMPPlaylist plThreeList = player.playlistCollection.getByName("ThreeList").Item(0);
plThreeList.appendItem(currMedia);                    //向 IWMPPlaylist 对象中添加当前播放的媒体文件
```

（2）clear 方法

该方法表示清空播放列表中的内容。

语法：

```
void clear ();
```

（3）get_attributeName 方法

该方法表示获取指定索引下的属性名称。

语法：

```
string get_attributeName(int lIndex);
```

参数说明如下。

lIndex：表示播放列表属性的索引。

返回值：表示指定索引下的属性名称。

（4）get_isIdentical 方法

该方法表示判断指定的播放列表是否是当前播放列表。

语法：

```
bool get_isIdentical (IWMPPlaylist pIWMPPlaylist);
```

参数说明如下。

pIWMPPlaylist：表示播放列表对象。

返回值：若指定的播放列表与当前的播放列表一致，则该值为 true；否则，该值为 false。

（5）get_Item 方法

该方法表示获取指定索引下的媒体文件。

语法：

```
IWMPMedia get_Item (int lIndex);
```

参数说明如下。

lIndex：表示播放列表中媒体文件的索引。

返回值：表示指定索引下的媒体文件。

（6）insertItem 方法

该方法表示向播放列表中的指定位置插入媒体文件。

语法：

```
void insertItem(int lIndex,IWMPMedia pIWMPMedia);
```

参数说明如下。

lIndex：表示即将向播放列表中添加的媒体文件所在的位置。

pIWMPMedia：表示向播放列表中添加的媒体文件。

（7）moveItem 方法

该方法表示改变媒体文件在播放列表中的位置。

语法：

```
void moveItem (int lIndexOld,int lIndexNew);
```

参数说明如下。

lindexOld：表示媒体文件的原始索引。

lIndexNew：表示媒体文件的新索引。

（8）removeItem 方法

该方法表示移除指定的媒体文件。

语法：

```
void removeItem (IWMPMedia pIWMPMedia)
```

参数说明如下。

pIWMPMedia：表示即将移除的媒体文件。

（9）setItemInfo 方法

该方法表示设置当前播放列表下指定属性的属性值。

语法：

```
void setItemInfo (string bstrName,string bstrValue)
```

参数列表如下。

bstrName：表示要进行设置的属性名。

bstrValue：表示要进行设置的属性值。

2．closedCaption 属性

该属性表示获取一个 IWMPClosedCaption 接口。

语法：

```
public virtual IWMPClosedCaption closedCaption {get;}
```

从 IWMPClosedCaption 接口可以获得一个 IWMPClosedCaption 对象，IWMPClosedCaption 对象提供一种方式用来同步显示媒体文件的字幕。如表 29.4 所示是 IWMPClosedCaption 对象的常

用属性。

表 29.4 IWMPClosedCaption 对象的常用属性及说明

属　性	说　明
captioningId	获取或设置 HTML 元素显示字幕的名称
SAMIFileName	获取或设置包含需要字幕信息的文件名称
SAMILang	获取或设置字幕信息的显示语言
SAMIStyle	获取或设置字幕信息的格式

下面对 IWMPClosedCaption 对象的属性进行详细的说明。

（1）captioningId 属性

该属性表示获取或设置 HTML 元素显示字幕的名称。

语法：

```
string captioningId {get; set;}
```

属性值：表示 HTML 元素的 ID。

（2）SAMIFileName 属性

该属性表示获取或设置包含需要字幕信息的文件名称。

语法：

```
string SAMIFileName {get; set;}
```

属性值：表示 SAMI 文件的文件名。

（3）SAMILang 属性

该属性表示获取或设置字幕信息的显示语言。

语法：

```
string SAMILang {get; set;}
```

属性值：表示 SAMI 文件中语言标识符的指定名称。

（4）SAMIStyle 属性

该属性表示获取或设置字幕信息的格式。

语法：

```
string SAMIStyle {get; set;}
```

属性值：表示 SAMI 文件中格式标识符的指定名称。

3．Ctlcontrols 属性

该属性表示获取一个 IWMPControls 接口。

语法：

```
public virtual IWMPControls Ctlcontrols {get;}
```

从 IWMPControls 接口中可以得到一个 IWMPControls 对象。IWMPControls 对象可以提供控制 Windows Media Player 控件播放的功能。如表 29.5 所示为 IWMPControls 对象常用的属性。

表 29.5 IWMPControls 对象的常用属性及说明

属　性	说　明
currentItem	获取或设置播放列表中正处于播放状态的媒体文件
currentMarker	获取或设置当前标记数字
currentPosition	获取或设置从媒体文件开始播放到当前位置所需要的时间
currentPositionString	获取当前媒体文件的播放时间

下面对 IWMPControls 对象的属性进行详细的说明。

（1）currentItem 属性

该属性表示获取或设置播放列表中正处于播放状态的媒体文件。

语法：

```
IWMPMedia currentItem {get; set;}
```

属性值：表示一个媒体文件。

例如，选定 ListBox 控件中的指定文件进行播放。代码如下：

```
private void playItem_OnSelectedIndexChanged(object sender, System.EventArgs e)
{
    //实例化一个整型变量用来保存当前选定项的索引
    int selectedItem = ((System.Windows.Forms.ListBox)sender).SelectedIndex;
    player.Ctlcontrols.stop();                                                    //停止播放
    player.Ctlcontrols.currentItem = player.currentPlaylist.get_Item(selectedItem); //重新设置当前播放文件
    player.Ctlcontrols.play();                                                    //播放设置后的媒体文件
}
```

（2）currentMarker 属性

该属性表示获取或设置当前标记数字。

语法：

```
int currentMarker {get; set;}
```

属性值：表示一个标记数字。

例如，选定 ListBox 控件中的文件开始播放。代码如下：

```
markers.Items.Add("特务");                                                    //向 ListBox 控件中添加内容
markers.Items.Add("日不落");
markers.Items.Add("加速度");
markers.Items.Add("快乐崇拜");
private void markers_OnSelectedIndexChanged(object sender, System.EventArgs e)
{
    int selectedMarker = ((System.Windows.Forms.ListBox)sender).SelectedIndex;//获取当前选定项的索引
    player.Ctlcontrols.currentMarker = selectedMarker;                        //设置播放器的当前标记数字
}
```

（3）currentPosition 属性

该属性表示获取或设置媒体文件从开始播放到当前位置所需要的时间。

语法：

```
double currentPosition {get; set;}
```

属性值：表示当前播放位置。

例如，设置当前媒体文件的播放位置。代码如下：

```
private void setPosition_Click(object sender, System.EventArgs e)
{
    player.Ctlcontrols.currentPosition = Convert.ToDouble(newPosition.Text);      //设置媒体文件的播放位置
}
```

（4）currentPositionString 属性

该属性表示获取当前媒体文件的播放时间。

语法：

```
string currentPositionString {get;}
```

属性值：表示一个以字符串形式显示的播放时间。

例如，在标签控件中显示当前媒体文件的播放时间。代码如下：

```
timer.Interval = 1000;                                                        //设置 Timer 控件发生 Tick 事件的时间间隔
timer.Start();                                                                //启动计时器
private void TimerEventProcessor(object sender, System.EventArgs e)
{
    currentPositionLabel.Text = player.Ctlcontrols.currentPositionString;//在标签控件中显示当前文件的播放时间
}
```

下面对 IWMPControls 对象的常用方法进行说明，如表 29.6 所示。

表 29.6 IWMPControls 对象的常用方法及说明

方　法	说　明
fastForward	表示加快媒体文件的播放进度
fastReverse	表示向相反的方向加快媒体文件的播放进度
get_isAvailable	表示获取一个值用来表示指定的类型是否可用
next	表示播放播放列表中处于当前文件的下一个文件
pause	表示暂停媒体文件的播放
play	表示播放媒体文件
playItem	表示播放指定的媒体文件
previous	表示播放播放列表中当前文件的上一个文件
stop	表示停止文件的播放

下面对 IWMPControls 对象的方法进行详细的讲解。

（1）fastForward 方法

该方法表示加快媒体文件的播放进度。

语法：

```
void fastForward ();
```

例如，假定窗体中有一个“快进“按钮，当单击“快进”按钮时，加快媒体文件的当前播放进度。代码如下：

```
private void fastForwardButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("fastForward"))  //当快进功能处于可用状态时
    {
        controls.fastForward();                   //加快当前媒体文件的播放进度
    }
}
```

（2）fastReverse 方法

该方法表示向相反的方向加快媒体文件的播放。

语法：

```
void fastReverse ();
```

例如，假定窗体中有一个“快退“按钮，当单击“快退”按钮时，向相反的方向加快媒体文件的当前播放进度。代码如下：

```
private void fastReverseButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("fastReverse"))  //当快退功能处于可用状态时
    {
        controls.fastReverse();                   //向相反的方向加快媒体文件的播放进度
    }
}
```

（3）get_isAvailable 方法

该方法表示获取一个值用来表示指定的类型是否可用。

语法：

```
bool get_isAvailable (string bstrItem);
```

参数说明如下。

bstrItem：表示指定的类型，常见的参数如表 29.7 所示。

返回值：如果当前类型处于可用状态，则该值为 true；否则，该值为 false。

表 29.7　　get_isAvailable 方法的常用参数及说明

参　数	说　明
CurrentItem	判断用户是否可以使用 currentItem 属性
currentMarker	判断用户是否可以定位到指定的标记
currentPosition	判断用户是否可以定位到文件中的指定位置
FastForward	判断当前文件是否支持快进功能以及能否调用该方法
FastReverse	判断当前文件是否支持快退功能以及能否调用该方法
Next	判断用户是否可以定位到当前播放列表中的下一个媒体文件
Pause	判断当前状态下 Pause 方法是否处于可用状态
Play	判断当前状态下 Play 方法是否处于可用状态
Previous	判断用户是否可以定位到当前播放列表中的上一个媒体文件
Step	判断在播放状态下 Step 方法是否处于可用状态
Stop	判断当前状态下 Stop 方法是否处于可用状态

例如，判断当前媒体文件是否支持 currentPosition 属性。代码如下：

```
if (player.Ctlcontrols.get_isAvailable("currentPosition"))
{
    player.Ctlcontrols.currentPosition = 0;
}
```

（4）next 方法

该方法表示播放播放列表中处于当前文件的下一个文件。

语法：

```
void next ();
```

例如，假定窗体中有一个“下一个“按钮，当单击“下一个”按钮时，播放当前播放列表中的下一个文件。代码如下：

```
private void nextButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("next"))          //当 next 方法处于可用状态时
    {
        controls.next();                           //播放下一个媒体文件
    }
}
```

（5）pause 方法

该方法表示暂停媒体文件的播放。

语法：

```
void pause ();
```

例如，假定窗体中有一个“暂停“按钮，当单击“暂停”按钮时，暂停当前文件的播放。代码如下：

```
private void pauseButton_Click(object o, System.EventArgs args)
    }
//实例化一个 IWMPControls 对象
WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
if (controls.get_isAvailable("pause"))             //当暂停功能处于可用状态时
{
    controls.pause();                              //暂停当前媒体文件的播放
}
}
```

（6）play 方法

该方法表示播放媒体文件。

语法：

```
void play ();
```

例如，假定窗体中有一个“播放“按钮，当单击“播放”按钮时，播放当前的媒体文件。代码如下：

```
private void playButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("play"))          //当播放功能处于可用状态时
    {
        controls.play();                           //播放当前的媒体文件
    }
}
```

（7）playItem 方法

该方法表示播放指定的媒体文件。

语法：

```
void playItem (IWMPMedia pIWMPMedia);
```

参数说明如下。

pIWMPMedia：表示即将播放的媒体文件。

例如，播放播放列表中指定索引下的媒体文件。代码如下：

```
int index = 3;                                                    //实例化一个整型变量
WMPLib.IWMPMedia media = player.currentPlaylist.get_Item(index); //实例化一个 IWMPMedia 对象
player.Ctlcontrols.playItem(media);                               //播放媒体文件 media
```

（8）previous 方法

该方法表示播放播放列表中当前文件的上一个文件。

语法：

```
void previous ();
```

例如，假定窗体中有一个“上一个”按钮，当单击“上一个”按钮时，播放播放列表中的上一个文件。代码如下：

```
private void previousButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("previous"))      //当上一个功能处于可用状态时
    {
        controls.previous();                       //播放播放列表中的上一个媒体文件
    }
}
```

（9）stop 方法

该方法表示停止文件的播放。

语法：

```
void stop ();
```

例如，假定窗体中有一个“停止“按钮，当单击“停止”按钮时，停止媒体文件的播放。代码如下：

```
private void stopButton_Click(object o, System.EventArgs args)
{
    //实例化一个 IWMPControls 对象
    WMPLib.IWMPControls3 controls = (WMPLib.IWMPControls3)player.Ctlcontrols;
    if (controls.get_isAvailable("stop"))      //当停止功能处于可用状态时
    {
        controls.stop();                       //停止当前媒体文件的播放
    }
}
```

4．currentMedia 属性

该属性表示获取或设置一个和当前媒体对应的 IWMPMedia 接口。

语法：

```
public virtual IWMPMedia currentMedia {get; set;}
```

属性值：表示一个 IWMPMedia 对象。

例如，获取当前媒体文件的文件名。代码如下：

```
WMPLib.IWMPMedia3 firstMedia = (WMPLib.IWMPMedia3)player.mediaCollection.getAll().get_Item(0);
player.currentMedia = firstMedia;//设置获取的文件为当前播放文件
currentMediaLabel.Text = ("第一个媒体文件的名称 = " + player.currentMedia.name); //在文本框中显示文件的名称
```

从 IWMPMedia 接口中可以得到一个 IWMPMedia 对象。IWMPMedia 对象可以提供一种获取或者设置媒体文件属性的方式。如表 29.8 所示为 IWMPMedia 对象常见的属性。

表 29.8　　IWMPMedia 对象的常用属性及说明

属　性	说　明
attributeCount	获取和媒体文件相关的属性个数
Duration	获取当前媒体文件的播放时间（以秒为单位）
durationString	获取当前媒体文件的播放时间（以时分秒的形式）
imageSourceHeight	获取当前媒体文件的高度（以像素为单位）
imageSourceWidth	获取当前媒体文件的宽度（以像素为单位）
markerCount	获取当前媒体文件中的标记
Name	获取或设置当前媒体文件的文件名
sourceURL	获取当前媒体文件的 URL

下面对 IWMPMedia 对象的属性进行详细说明。

（1）attributeCount 属性

该属性表示和媒体文件相关的属性个数。

语法：

```
int attributeCount {get;}
```

属性值：当前媒体文件可用的属性个数。

例如，获取当前媒体文件的可用属性，然后让其显示在多行文本框中。代码如下：

```
WMPLib.IWMPMedia3 cm = (WMPLib.IWMPMedia3)player.currentMedia; //实例化一个 IWMPMedia 对象
int numAttributes = cm.attributeCount;          //保存媒体文件中可用的属性总个数
string atName,atValue;                          //实例化两个字符串变量
string[] atList = new string[numAttributes];    //初始化一个字符串数组用来保存属性及属性值
for (int i = 0; i < numAttributes; i++)         //循环遍历属性列表中的每一个属性
{
    atName = cm.getAttributeName(i);            //保存属性名
    atValue = cm.getItemInfo(atName);           //保存属性值
    atList[i] = (atName + ": " + atValue);      //实例化数组中的每一个元素
}
attributeList.Lines = atList;                   //将数组中的内容显示在文本框中
```

（2）duration 属性

该属性表示获取当前媒体文件的播放时间（以秒为单位）。

语法：

```
double duration {get;}
```

属性值：表示当前媒体文件的播放时间。

例如，显示当前媒体文件离播放结束还有多少时间。代码如下：

```
timer.Interval = 1000;                  //设置 Timer 控件发生 Tick 事件的时间间隔
timer.Start();                          //启动计时器
private void TimerEventProcessor(object sender, System.EventArgs e)
{
```

```
    double t = Math.Floor(player.currentMedia.duration - player.Ctlcontrols.currentPosition);//计算剩余时间
    timeRemaining.Text = ("剩余时间: " + t.ToString());//在 Label 控件中显示剩余时间
}
```

（3）durationString 属性

该属性表示获取当前媒体文件的播放时间（以时分秒的形式）。

语法：

```
string durationString {get;}
```

属性值：表示当前媒体文件的播放时间（以时分秒的形式显示）。

例如，获取指定媒体文件的播放时间显示在 Label 控件中。代码如下：

```
private void player_OpenStateChange(object sender, AxWMPLib._WMPOCXEvents_OpenStateChangeEvent e)
{
    if (e.newState == (int)WMPLib.WMPOpenState.wmposMediaOpen)//当当前媒体文件处于打开状态时
    {
        mediaDuration.Text = player.currentMedia.durationString;//在 Label 控件中显示当前媒体文件的播放时间
    }
}
```

（4）imageSourceHeight 属性

该属性表示获取当前媒体文件的高度（以像素为单位）。

语法：

```
int imageSourceHeight {get;}
```

属性值：当前媒体文件的高度。

例如，在文本框中显示当前媒体文件的高度和宽度。代码如下：

```
private void player_OpenStateChange(object sender, AxWMPLib._WMPOCXEvents_OpenStateChangeEvent e)
{
    if (e.newState == (int)WMPLib.WMPOpenState.wmposMediaOpen)      //当当前媒体文件处于打开状态时
    {
        int height = player.currentMedia.imageSourceHeight;          //获取媒体文件的高度
        int width = player.currentMedia.imageSourceWidth;            //获取媒体文件的宽度
        videoSize.Clear();                                           //清空文本框中的内容
        if (height != 0 && width != 0)//当获取的媒体文件的高度和宽度不为 0 时
        {
            //在文本框中显示媒体文件的宽度和高度
            videoSize.Text = (width.ToString() + " x " + height.ToString());
        }
    }
}
```

（5）imageSourceWidth 属性

该属性表示获取当前媒体文件的宽度（以像素为单位）。

语法：

```
int imageSourceWidth {get;}
```

属性值：表示当前媒体文件的宽度值。

（6）markerCount 属性

该属性表示获取当前媒体文件中的标记。

语法：

```
int markerCount {get;}
```

属性值：表示当前媒体文件中的标记个数。

例如，获取当前媒体文件中的所有标记名称让其显示在多行文本框中。代码如下：

```
int mcount = player.currentMedia.markerCount;   //实例化一个整型变量用来保存当前媒体文件的标记个数
string[] markerNames = new string[mcount];      //初始化一个字符串数组
if (mcount > 0)                                 //当标记个数存在时
{
    for (int i = 1; i < mcount + 1; i++)        //循环遍历每一个标记
    {
        markerNames[i] = player.currentMedia.getMarkerName(i);   //实例化数组中的每一个元素
```

```
    }
    markerList.Lines = markerNames;                    //将获取的标记名称显示在多行文本框中
}
```

（7）name 属性

该属性表示获取或设置当前媒体文件的文件名。

语法：

```
string name {get; set;}
```

属性值：表示当前媒体文件的文件名。

例如，设置当前的媒体文件名为指定的文件名。代码如下：

```
private void setNewName_Click(object sender, System.EventArgs e)
{
    WMPLib.IWMPMedia3 cm = (WMPLib.IWMPMedia3)player.currentMedia;  //实例化一个 IWMPMedia 对象
    cm.name = nameText.Text;                    //改变当前媒体文件的文本名为文本框中的指定文件名
}
```

（8）sourceURL 属性

该属性表示获取当前媒体文件的 URL。

语法：

```
string sourceURL {get;}
```

属性值：表示媒体文件的 URL。

例如，播放“All Music”播放列表集合中的第一个播放列表中的第一个媒体文件。代码如下：

```
//获取 All Music 播放列表集合中的第一个播放列表
WMPLib.IWMPPlaylist pl = player.playlistCollection.getByName("All Music").Item(0);
WMPLib.IWMPMedia3 media = (WMPLib.IWMPMedia3)pl.get_Item(0);  //获取播放列表中的第一个媒体文件
player.URL = media.sourceURL;        //设置当前播放器的 URL 为获取的媒体文件
player.Ctlcontrols.play();           //播放指定的媒体文件
```

IWMPMedia 对象的常用方法，如表 29.9 所示。

表 29.9　　IWMPMedia 对象的常用方法及说明

方　法	说　明
get_isIdentical	获取一个值表明指定的媒体文件是否和当前播放的媒体文件同名
getAttributeName	获取和指定索引匹配的属性名称
getItemInfo	获取媒体文件指定属性的属性值
getItemInfoByAtom	获取指定索引下的属性值
getMarkerName	获取指定索引下标记的名称
getMarkerTime	获取指定索引下标记所处的时间
isMemberOf	获取一个值表示是否指定的媒体文件处于指定的播放列表中
isReadOnlyItem	获取一个值表示指定的媒体文件的属性是否可编辑
setItemInfo	设置媒体文件的指定属性值

下面对 IWMPMedia 对象的方法进行详细说明。

（1）get_isIdentical 方法

该方法表示获取一个值表明指定的媒体文件是否和当前播放的媒体文件同名。

语法：

```
bool get_isIdentical (IWMPMedia pIWMPMedia);
```

参数说明如下。

pIWMPMedia：表示一个媒体文件。

返回值：如果指定的媒体文件和当前播放的媒体文件同名，则该值为 true；否则，该值为 false。

例如，当指定的媒体文件和当前播放的媒体文件不同时，播放指定的媒体文件。代码如下：

```
if (newMedia.get_isIdentical(player.currentMedia) != true)        //当指定的媒体文件和当前播放的媒体文件不同时
{
    player.currentMedia = newMedia;                               //设置指定的媒体文件为当前播放文件
    player.Ctlcontrols.play();                                    //播放指定的媒体文件
}
```

（2）getAttributeName 方法

该方法表示获取和指定索引匹配的属性名称。

语法：

```
string getAttributeName (int lIndex);
```

参数说明如下。

lIndex：表示一个索引。

返回值：表示属性名称。

例如，在多行文本框中显示当前媒体文件的索引和属性名称。代码如下：

```
WMPLib.IWMPMedia3 cm = (WMPLib.IWMPMedia3)player.currentMedia; //获取播放器当前播放的媒体文件
int attCount = cm.attributeCount;                              //获取当前播放的媒体文件的属性个数
string[] attInfo = new string[attCount];                       //实例化一个长度和属性个数相同的字符串数组
for (int i = 0; i < attCount; i++)                             //从 0 循环遍历至 attCount-1
{
    attInfo[i] = ("属性" + i + ": " + cm.getAttributeName(i));  //实例化数组中的每一个元素
}
attributeNames.Lines = attInfo;                                //将数组中的内容显示在多行文本框中
```

（3）getItemInfo 方法

该方法表示获取媒体文件指定属性的属性值。

语法：

```
string getItemInfo (string bstrItemName);
```

参数说明如下。

bstrItemName：表示属性的名称。

返回值：表示属性值。

（4）getItemInfoByAtom 方法

该方法表示获取指定索引下的属性值。

语法：

```
string getItemInfoByAtom (int lAtom);
```

参数说明如下。

lAtom：表示可用属性集中指定的属性索引。

返回值：表示属性值。

（5）getMarkerName 方法

该方法表示获取指定索引下标记的名称。

语法：

```
string getMarkerName (int MarkerNum);
```

参数说明如下。

MarkerNum：表示标记的索引。

返回值：表示标记的名称。

例如，获取当前媒体文件的标记名称显示在多行文本框中。代码如下：

```
int mcount = player.currentMedia.markerCount;                  //获取当前媒体文件的标记个数
string[] markers = new string[mcount];                         //实例化一个和标记个数相同的字符串数组
if (mcount > 0)                                                //当当前媒体文件的标记个数大于 0 时
{
    for (int i = 1; i < mcount + 1; i++)                       //从 1 循环遍历至 mcount
    {
```

```
        markers[i] = player.currentMedia.getMarkerName(i);  //实例化字符串数组中的每一个元素
    }
    markerNames.Lines = markers;                            //将数组中的内容显示在多行文本框中
}
```

（6）getMarkerTime 方法

该方法表示获取指定索引下标记所处的时间。

语法：

```
double getMarkerTime (int MarkerNum);
```

参数说明如下。

MarkerNum：表示标记的索引。

返回值：表示标记所处位置的时间。

例如，获取每一个标记的位置显示在多行文本框中。代码如下：

```
int mcount = player.currentMedia.markerCount;                    //获取当前媒体文件的标记个数
string[] markers = new string[mcount];                           //实例化一个长度与标记个数一致的字符串数组
if (mcount > 0)                                                  //当媒体文件中存在标记时
{
    for (int i = 1; i < mcount + 1; i++)                         //从 1 循环遍历至 mcount
    {
        markers[i] = "标记数字" + i + " 出现在 ";                  //实例化字符串数组中的每一个元素
        markers[i] += player.currentMedia.getMarkerTime(i).ToString();  //向当前元素中追加标记所在的时间
        markers[i] += " 秒.";                                    //向当前元素中追加标记时间的单位
    }
    markerTimes.Lines = markers;                                 //在多行文本框中显示字符串数组中的内容
}
```

（7）isMemberOf 方法

该方法用于判断指定的媒体文件是否处于指定的播放列表中。

语法：

```
bool isMemberOf (IWMPPlaylist pPlaylist);
```

参数说明如下。

pPlaylist：表示一个播放列表。

返回值：如果当前媒体文件位于指定的播放列表中，那么该值为 true；否则，该值为 false。

例如，判断当前媒体文件是否属于指定的播放列表，如果不属于，那么将当前播放文件追加到指定播放列表末尾。代码如下：

```
WMPLib.IWMPPlaylist sPlaylist = player.playlistCollection.getByName("All Music").Item(0);//获取一个播放列表
if (player.currentMedia.isMemberOf(sPlaylist) == false)          //当播放的媒体文件不属于指定的播放列表时
{
    sPlaylist.appendItem(player.currentMedia);                   //将播放文件追加到指定的播放列表末尾
}
```

（8）isReadOnlyItem 方法

该方法用于判断指定媒体文件的属性是否可编辑。

语法：

```
bool isReadOnlyItem (string bstrItemName);
```

参数说明如下。

bstrItemName：表示一个媒体文件。

返回值：如果当前媒体文件的属性可编辑，那么该值为 true；否则，该值为 false。

例如，将当前媒体文件的属性及其读写性显示在多行文本框中。代码如下：

```
WMPLib.IWMPMedia3 cm = (WMPLib.IWMPMedia3)player.currentMedia; //获取当前的媒体文件
int attCount = player.currentMedia.attributeCount;              //获取当前媒体文件的属性个数
string[] atInfo = new string[attCount];                         //初始化一个字符串数组
string atName;                                                  //定义一个字符串用来保存属性名
for (int i = 0; i < attCount; i++)                              //从 0 循环遍历至 attCount-1
{
```

```
    atName = cm.getAttributeName(i);                                    //获取指定索引下的属性名
    string test = ((cm.isReadOnlyItem(atName)) ? "只读" : "可读写");     //获取媒体文件属性的读写性
    atInfo[i] = (atName + " 是 " + test);                               //实例化数组中的每一个元素
}
rwText.Lines = atInfo;                                                  //将数组的内容显示在多行文本框中
```

（9）setItemInfo 方法

该方法表示设置媒体文件的指定属性值。

语法：

```
void setItemInfo (string bstrItemName,string bstrVal);
```

参数说明如下。

bstrItemName：表示一个属性名。

bstrVal：表示一个新的属性值。

例如，设置指定属性值为一个新的值。代码如下：

```
private void setNewGenre_Click(object sender, System.EventArgs e)
{
    WMPLib.IWMPMedia3 cm = (WMPLib.IWMPMedia3)player.currentMedia;  //获取当前的媒体文件
    string atValue = genText.Text;                                  //设置指定属性的属性值
    if( cm.isReadOnlyItem("Genre") == false )                       //当指定属性可编辑时
    {
        cm.setItemInfo("Genre", atValue);                           //为该属性设定新值
    }
}
```

5．currentPlaylist 属性

该属性表示获取或设置当前播放列表接口 IWMPPlaylist。

语法：

```
public virtual IWMPPlaylist currentPlaylist {get; set;}
```

例如，获取第一个播放列表的名称。代码如下：

```
WMPLib.IWMPPlaylist firstPlaylist = player.playlistCollection.getAll().Item(0);        //实例化一个播放列表对象
player.currentPlaylist = firstPlaylist;                                                 //设置当前的播放列表
currentPlaylistLabel.Text = ("第一个播放列表的名称 = " + player.currentPlaylist.name);  //显示播放列表的名称
```

6．dvd 属性

该属性表示获取一个 IWMPDVD 接口。

语法：

```
public virtual IWMPDVD dvd {get;}
```

例如，停止录音重放和显示标题菜单。代码如下：

```
player.dvd.titleMenu();
```

从 IWMPDVD 接口可以得到一个 IWMPDVD 对象，IWMPDVD 对象可以提供操作 DVD 的属性和方法。下面对 IWMPDVD 对象比较重要的属性 domain 进行说明。

该属性表示获取 DVD 的当前范围。

语法：

```
string domain {get;}
```

属性值：常见的属性值如表 29.10 所示。

表 29.10　　domain 的常用属性值及说明

属　性　值	说　　明
FirstPlay	显示默认的光盘唱片
videoManagerMenu	显示全部唱片的菜单
videoTitleSetMenu	显示当前字幕设置菜单
Title	通常情况下，显示当前字幕
Stop	表示 DVD 导航器处于停止范围内
Undefined	表示 Windows Media Player 播放器不在 DVD 范围内

IWMPDVD 对象的常用方法如表 29.11 所示。

表 29.11　　IWMPDVD 对象的常用方法及说明

方　法	说　明
back	从子菜单返回至父菜单
get_isAvailable	获取一个值表示指定的操作是否可以执行
resume	当菜单被调用时从相同的位置开始播放
titleMenu	停止播放和显示字幕菜单
topMenu	停止播放和显示原始菜单

下面对 IWMPDVD 对象的方法进行详细介绍。

（1）back 方法

该方法表示从子菜单返回到父菜单。

语法：

```
void back ();
```

（2）get_isAvailable 方法

该方法表示获取一个值表示指定的操作是否可以执行。

语法：

```
bool get_isAvailable (string bstrItem)
```

参数说明如下。

bstrItem：常见的参数值如表 29.12 所示。

返回值：若指定的操作可以执行，则该值为 true；否则，该值为 false。

表 29.12　　get_isAvailable 方法的常用参数及说明

参　数	说　明
Back	判断 back 方法是否处于可用状态
Dvd	判断 DVD 是否已经加载
dvdDecoder	判断 DVD 解码器是否已经安装
Resume	判断 resume 方法是否处于可用状态
titleMenu	判断 titleMenu 方法是否处于可用状态
topMenu	判断 topMenu 方法是否处于可用状态(通常情况下，该方法为 rootMenu)

（3）resume 方法

该方法表示当菜单被调用时从相同的位置开始播放。

语法：

```
void resume ();
```

（4）titleMenu 方法

该方法表示停止播放和显示字幕菜单。

语法：

```
void titleMenu ();
```

（5）topMenu 方法

该方法表示停止播放和显示原始菜单。

语法：

```
void topMenu ();
```

7．Error 属性

该属性表示获取一个 IWMPError 接口。

语法：

```
public virtual IWMPError Error {get;}
```

从 IWMPError 接口可以得到一个 IWMPError 对象。IWMPError 对象提供一系列访问 IWMPErrorItem 对象的属性和方法。下面详细讲解 IWMPError 对象的属性 errorCount。

该属性表示获取消息队列中错误的个数。

语法：

```
int errorCount {get;}
```

属性值：表示错误的个数。

例如，以消息框的形式显示消息队列中刚出现的错误的描述信息。代码如下：

```
private void player_ErrorEvent_errorCount(object sender, System.EventArgs e)
{
    int max = player.Error.errorCount;                          //获取错误的总数
    string errDesc = player.Error.get_Item(max-1).errorDescription;  //获取刚发生的错误的描述
    System.Windows.Forms.MessageBox.Show(errDesc);              //显示错误信息
}
```

IWMPError 对象的常见方法如表 29.13 所示。

表 29.13　　IWMPError 对象的常用方法及说明

方　法	说　明
ClearErrorQueue	从错误队列中清空错误消息
get_Item	获取错误队列中指定索引下的 IWMPErrorItem 对象
WebHelp	显示第一条错误信息的远程帮助页面

下面对 IWMPError 对象的常见方法进行详细讲解。

（1）clearErrorQueue 方法

该方法表示从错误队列中清空消息。

语法：

```
void clearErrorQueue ();
```

例如，显示全部的错误信息后清空错误队列中的全部错误。代码如下：

```
private void player_ErrorEvent_clearErrorQueue(object sender, System.EventArgs e)
{
    int max = player.Error.errorCount;                          //获取错误队列中的错误个数
    for (int i = 0; i < max; i++)                               //从 0 循环遍历至 max-1
    {
        string errDesc = player.Error.get_Item(i).errorDescription;  //获取指定索引下的错误描述
        System.Windows.Forms.MessageBox.Show(errDesc);          //在消息框中显示错误描述
    }
    player.Error.clearErrorQueue();                             //清空消息队列中的内容
}
```

（2）get_Item 方法

该方法表示获取错误队列中指定索引下的 IWMPErrorItem 对象。

语法：

```
IWMPErrorItem get_Item (int dwIndex);
```

参数说明如下。

dwIndex：表示错误队列中从 0 开始的索引。

返回值：表示一个 IWMPErrorItem 对象。

例如，获取错误队列中刚发生的错误代码及描述。代码如下：

```
private void player_ErrorEvent_get_Item(object sender, System.EventArgs e)
{
    int max = (player.Error.errorCount - 1);          //获取刚发生的错误的索引
    WMPLib.IWMPErrorItem2 errItem = (WMPLib.IWMPErrorItem2)player.Error.get_Item(max);//获取错误对象
    int errNum = errItem.errorCode;                   //获取错误的代码
    string errDesc = errItem.errorDescription;        //获取错误的描述
    //以消息框的形式显示错误信息
```

```
        System.Windows.Forms.MessageBox.Show(errNum.ToString() + ":    " + errDesc);
    }
```

（3）webHelp 方法

该方法表示显示第一条错误信息的远程帮助页面。

语法：

```
void webHelp ();
```

例如，在浏览器中加载 Microsoft Windows Media Player 的远程帮助页。代码如下：

```
private void webHelpButton_Click(object sender, System.EventArgs e)
{
    player.Error.webHelp();                    //显示 Windows Media Player 控件的远程帮助页
}
```

8．mediaCollection 属性

该属性表示获取一个 IWMPMediaCollection 接口。

语法：

```
public virtual IWMPMediaCollection mediaCollection {get;}
```

从 IWMPMediaCollection 接口可以得到一个 IWMPMediaCollection 对象。IWMPMediaCollection 对象提供大量组织媒体文件的方法。如表 29.14 所示为 IWMPMediaCollection 对象的常用方法。

表 29.14　　IWMPMediaCollection 对象的常用方法及说明

方　法	说　明
Add	向媒体库中添加一个新的媒体文件或者一个播放列表
GetAll	返回一个包含媒体库中所有媒体文件的播放列表
getAttributeStringCollection	获取一种媒体文件类型下的指定属性的所有属性值集合 IWMPStringCollection
getByAlbum	从指定唱片集中获取一个播放列表
getByAttribute	获取一个具有指定属性和属性值的播放列表
getByAuthor	获取由指定作者的媒体文件组成的播放列表
getByGenre	获取有指定流派的媒体文件组成的播放列表
getByName	获取具有指定名称的播放列表
getMediaAtom	获取指定属性的索引
Remove	从媒体集合中移除指定的媒体文件
setDeleted	将指定的媒体文件移动到已删除文件夹中

下面对 IWMPMediaCollection 对象的方法进行详细讲解。

（1）add 方法

该方法表示向媒体库中添加一个新的媒体文件或者一个播放列表。

语法：

```
IWMPMedia add (string bstrURL);
```

参数说明如下。

bstrURL：表示指定位置的媒体文件或者播放列表。

返回值：表示将添加的媒体文件或者播放列表。

例如，向媒体集合中添加 3 个媒体文件。代码如下：

```
player.mediaCollection.add("http://www.proseware.com/Media/天下无双.wma");    //向媒体集中添加媒体文件
player.mediaCollection.add(@"\\yourservername\Public\笔记.wma");
player.mediaCollection.add(@"C:\WMSDK\WMPSDK\samples\media\谁动了我的琴弦.wma");
```

（2）getAll 方法

该方法表示返回一个包含媒体库中所有媒体文件的播放列表。

语法：

```
IWMPPlaylist getAll ();
```

例如，在媒体集合中随机播放媒体文件。代码如下：

```
System.Random randGenerator = new System.Random();                    //实例化一个随机类的对象
int count = player.mediaCollection.getAll().count;                    //获取媒体集合中的媒体文件个数
int rand = randGenerator.Next(count);                                 //获取有效值范围内的随机值
player.currentMedia = player.mediaCollection.getAll().get_Item(rand); //设置当前播放文件
player.Ctlcontrols.play();                                            //播放指定的媒体文件
```

（3）getAttributeStringCollection 方法

该方法表示获取一种媒体文件类型下指定属性的所有属性值组成的对象 IWMPString Collection。

语法：

```
IWMPStringCollection getAttributeStringCollection (string bstrAttribute,string bstrMediaType);
```

参数说明如下。

bstrAttribute：表示重新获得值的属性。

bstrMediaType：表示重新获得值的媒体类型。

返回值：表示重新获得值的 IWMPStringCollection 对象。

例如，显示媒体集合中指定属性的音频文件。在 ListBox 控件中显示属性列表，在多行文本框中显示结果。代码如下：

```
private void audioAttributes_OnSelectedIndexChanged(object sender, System.EventArgs e)
{
    //从 ListBox 控件中获得属性类型
    string attributeType = (string)((System.Windows.Forms.ListBox)sender).SelectedItem;
    //获取一个媒体集合对象
    WMPLib.IWMPMediaCollection2 library = (WMPLib.IWMPMediaCollection2)player.mediaCollection;
    WMPLib.IWMPStringCollection2 all = (WMPLib.IWMPStringCollection2)library.getAttributeStringCollection
(attributeType, "Audio");          //获取用户选择的指定属性类型的音频文件集合对象
    attributeValues.Clear();       //清空文本框中的原有内容
    string[] attributeArray = new string[all.count]; //实例化一个字符串数组
    for (int i = 0; i < all.count; i++)              //循环遍历集合中的每一个元素
    {
        attributeArray[i] = all.Item(i);             //为字符串数组赋值
    }
    attributeValues.Lines = attributeArray;          //将内容显示在多行文本框中
}
```

（4）getByAlbum 方法

该方法表示从指定唱片集中获取一个播放列表。

语法：

```
IWMPPlaylist getByAlbum (string bstrAlbum);
```

参数说明如下。

bstrAlbum：表示唱片集的标题。

返回值：表示一个播放列表。

例如，创建一个指定唱片集下的播放列表。代码如下：

```
private void playAlbum_Click(object sender, System.EventArgs e)
{
    string album = getAlbum.Text;                                          //获取唱片集的名称
    WMPLib.IWMPPlaylist pl = player.mediaCollection.getByAlbum(album);     //获取指定唱片集下的播放列表
    player.currentPlaylist = pl;                                           //设置当前的播放列表
    player.Ctlcontrols.play();                                             //播放媒体文件
}
```

（5）getByAttribute 方法

该方法表示获取一个具有指定属性和属性值的播放列表。

语法：

```
IWMPPlaylist getByAttribute (string bstrAttribute,string bstrValue);
```

参数说明如下。

bstrAttribute：表示指定的属性。

bstrValue：表示指定的属性值。

返回值：表示一个播放列表。

例如，播放媒体库中具有“artist”属性和“Triode 48”属性值的所有媒体文件。代码如下：

```
//获取一个具有指定属性和属性值的播放列表
WMPLib.IWMPPlaylist pl = player.mediaCollection.getByAttribute("Artist", "Triode 48");
player.currentPlaylist = pl; //设置当前的播放列表
player.Ctlcontrols.play();   //播放媒体文件
```

（6）getByAuthor 方法

该方法表示获取由指定作者的媒体文件组成的播放列表。

语法：

```
IWMPPlaylist getByAuthor (string bstrAuthor);
```

参数说明如下。

bstrAuthor：表示作曲家的名称。

返回值：表示一个播放列表。

例如，创建一个具有指定作曲家的媒体文件组成的播放列表。代码如下：

```
private void playAuthor_Click(object sender, System.EventArgs e)
{
    string author = getAuthor.Text;          //获取作曲家的名称
    //获取具有指定作曲家的媒体文件组成的播放列表
    WMPLib.IWMPPlaylist pl = player.mediaCollection.getByAuthor(author);
    player.currentPlaylist = pl;             //设置当前的播放列表
    player.Ctlcontrols.play();               //播放媒体文件
}
```

（7）getByGenre 方法

该方法表示获取由指定流派的媒体文件组成的播放列表。

语法：

```
IWMPPlaylist getByGenre (string bstrGenre);
```

参数列表如下。

bstrGenre：表示流派的名称。

返回值：表示一个播放列表。

例如，获取由指定流派的媒体文件组成的播放列表。代码如下：

```
private void playGenre_Click(object sender, System.EventArgs e)
{
    string genre = getGenre.Text;      //获取流派的名称
    //获取由指定流派下的媒体文件组成的播放列表
    WMPLib.IWMPPlaylist pl = player.mediaCollection.getByGenre(genre);
    player.currentPlaylist = pl;       //设置当前的播放列表
    player.Ctlcontrols.play();         //播放媒体文件
}
```

（8）getByName 方法

该方法表示获取具有指定名称的播放列表。

语法：

```
IWMPPlaylist getByName (string bstrName);
```

参数说明如下。

bstrName：表示一个播放列表名称。

返回值：表示一个具有指定名称的播放列表对象。

例如，从媒体库中获取 3 个媒体文件并添加到当前播放列表的末尾。代码如下：

```
//从互联网上获取一个媒体文件
WMPLib.IWMPPlaylist one = player.mediaCollection.getByName("http://www.proseware.com/Media/Laure");
WMPLib.IWMPPlaylist two = player.mediaCollection.getByName("Jeanne");        //从局域网上获取一个媒体文件
WMPLib.IWMPPlaylist three = player.mediaCollection.getByName("house");       //从本机中获取一个媒体文件
player.currentPlaylist.appendItem(one.get_Item(0));                          //向当前播放列表中添加 one 中的媒体文件
player.currentPlaylist.appendItem(two.get_Item(0));                          //向当前播放列表中添加 two 中的媒体文件
player.currentPlaylist.appendItem(three.get_Item(0));                        //向当前播放列表中添加 three 中的媒体文件
```

（9）getMediaAtom 方法

该方法表示获取指定属性的索引。

语法：

```
int getMediaAtom (string bstrItemName);
```

参数列表如下。

bstrItemName：表示有效的属性名称。

返回值：表示有效的属性名称在列表中对应的索引。

（10）remove 方法

该方法表示从媒体集合中移除指定的媒体文件。

语法：

```
void remove (IWMPMedia pItem,bool varfDeleteFile);
```

参数说明如下。

pItem：表示即将移除的媒体文件。

varfDeleteFile：表示是否从媒体库中移除指定的文件。

例如，在删除一个文件前给出信息提示，然后从媒体库中永久地删除指定媒体文件。代码如下：

```
//实例化一个 IWMPMedia 对象
WMPLib.IWMPMedia3 media = (WMPLib.IWMPMedia3)player.mediaCollection.getAll().get_Item(0);
string mediaName = media.name;                          //获取媒体文件的名称
string message = ("确认删除 " + mediaName + "?");       //实例化消息框中显示的文本信息
string caption = "删除媒体文件";                        //实例化消息框的标题
//实例化一个消息框中显示的按钮
System.Windows.Forms.MessageBoxButtons buttons = System.Windows.Forms.MessageBoxButtons.OKCancel;
//实例化一个对话框对象
System.Windows.Forms.DialogResult result = System.Windows.Forms.MessageBox.Show(message, caption, buttons);
if (result == System.Windows.Forms.DialogResult.OK)    //当单击确定按钮时
{
    player.mediaCollection.remove(media, true);         //从媒体库中永久的移除指定的媒体文件
    System.Windows.Forms.MessageBox.Show("已经移除" + mediaName);//弹出信息提示
}
```

（11）setDeleted 方法

该方法表示将指定的媒体文件移动到已删除文件夹中。

语法：

```
void setDeleted (IWMPMedia pItem,bool varfIsDeleted);
```

参数说明如下。

pItem：表示将移动的媒体文件。

varfIsDeleted：表示是否将媒体文件移动到已删除文件夹中。

例如，将指定的媒体文件移动到已删除文件夹中。代码如下：

```
if (!player.mediaCollection.isDeleted(media))                    //当该媒体文件未被删除时
{
    player.mediaCollection.setDeleted(media, true);              //删除指定的媒体文件
    System.Windows.Forms.MessageBox.Show("该文件已经移动到已删除文件夹中");//弹出删除成功的消息提示
}
else
{
    System.Windows.Forms.MessageBox.Show("该文件已经被删除");     //弹出该文件被删除的信息提示
}
```

9．playlistCollection 属性

该属性表示获取一个 IWMPPlaylistCollection 接口。

语法：

```
public virtual IWMPPlaylistCollection playlistCollection {get;}
```

例如，获取播放列表中我最喜爱的 4 星或 5 星的媒体文件，将它们设置为当前的播放文件。代码如下：

```
//获取指定名称下的文件个数
int checkit = player.playlistCollection.getByName("Favorites -- 4 and 5 star rated").count;
if (checkit > 0)//当指定名称下存在文件时
{
  //获取一个播放列表
  player.currentPlaylist = player.playlistCollection.getByName("Favorites -- 4 and 5 star rated").Item(0);
}
```

从 IWMPPlaylistCollection 接口可以得到一个 IWMPPlaylistCollection 对象。IWMPPlaylist Collection 对象为操作 IWMPPlaylist 对象和 IWMPPlaylistArray 对象提供方法。IWMPPlaylist Collection 对象的常用方法如表 29.15 所示。

表 29.15　　IWMPPlaylistCollection 对象的常用方法及说明

方　法	说　明
GetAll	提供一个访问媒体库中所有播放列表的对象 IWMPPlaylistArray
getByName	提供一个访问具有指定名称的播放列表对象 IWMPPlaylistArray
importPlaylist	向媒体库中添加一个静态的播放列表
IsDeleted	判断指定的播放列表是否处于已删除文件夹中
newPlaylist	在媒体库中创建一个新的播放列表
Remove	从媒体库中移除播放列表

下面对 IWMPPlaylistCollection 对象的方法进行详细的说明。

（1）getAll 方法

该方法表示提供一个访问媒体库中所有播放列表的对象 IWMPPlaylistArray。

语法：

```
IWMPPlaylistArray getAll ();
```

参数列表如下。

返回值：一个 IWMPPlaylistArray 对象。

（2）getByName 方法

该方法表示提供一个访问具有指定名称的播放列表对象 IWMPPlaylistArray。

语法：

```
IWMPPlaylistArray getByName (string bstrName);
```

参数列表如下。

bstrName：表示播放列表的名称。

返回值：表示一个 IWMPPlaylistArray 对象。

（3）importPlaylist 方法

该方法表示向媒体库中添加一个静态的播放列表。

语法：

```
IWMPPlaylist importPlaylist (IWMPPlaylist pItem);
```

参数列表如下。

pItem：表示即将添加的 IWMPPlaylist 对象。

返回值：表示已经添加的 IWMPPlaylist 对象。

（4）isDeleted 方法

该方法表示判断指定的播放列表是否处于已删除文件夹中。

语法：

```
bool isDeleted (IWMPPlaylist pItem);
```

参数说明如下。

pItem：表示需要判断的播放列表。

返回值：表示指定的播放列表是否已经被删除。

（5）newPlaylist 方法

该方法表示在媒体库中创建一个新的播放列表。

语法：

```
IWMPPlaylist newPlaylist (string bstrName);
```

参数列表如下。

bstrName：表示新播放列表的名称。

返回值：表示新添加的播放列表。

例如，向播放列表集合中添加一个空的播放列表“ThreeList”。代码如下：

```
WMPLib.IWMPPlaylist newList = player.playlistCollection.newPlaylist("ThreeList");
```

（6）remove 方法

该方法表示从媒体库中移除播放列表。

语法：

```
void remove (IWMPPlaylist pItem);
```

参数列表如下。

pItem：表示即将移除的播放列表对象。

在上面知识的讲解过程中用到了 IWMPPlaylistArray 对象，IWMPPlaylistArray 对象提供一个属性和一个方法用来访问 IWMPPlaylist 对象。

下面对 IWMPPlaylistArray 对象的属性 count 进行讲解。

该属性表示获取播放列表集合中播放列表的个数。

语法：

```
int count {get;}
```

属性值：表示播放列表的个数。

下面对 IWMPPlaylistArray 对象的方法 Item 进行讲解。

该方法表示获取指定索引下的播放列表。

语法：

```
IWMPPlaylist Item (int lIndex);
```

参数列表如下。

lindex：表示集合中播放列表对应的索引。

返回值：表示一个播放列表。

10．settings 属性

该属性表示获取一个 IWMPSettings 接口。

语法：

```
public virtual IWMPSettings settings {get;}
```

例如，设置媒体文件的循环模式为单曲循环。代码如下：

```
player.settings.setMode("loop",true);
```

从 IWMPSettings 接口中可以得到一个 IWMPSettings 对象，IWMPSettings 对象用来提供一系列方法和属性对 Windows Media Player 进行设置。IWMPSettings 对象的常见属性如表 29.16 所示。

表 29.16　　IWMPSettings 对象的常用属性及说明

属　性	说　明
autoStart	获取或设置一个值用来表示是否自动播放当前媒体文件
balance	获取或设置当前立体声的平衡
baseURL	获取或设置嵌入数字媒体目录用于和 URL 脚本命令相连的路径协议 URL
defaultFrame	获取或设置从脚本命令事件接收用于显示 URL 的结构名称
enableErrorDialogs	获取或设置一个值用来表示是否随机显示错误消息对话框
invokeURLs	获取或设置一个值用来表示是否将 URL 事件发送到网页浏览器
mute	获取或设置一个值用来表示音频是否处于无声状态
playCount	获取或设置媒体文件播放的次数
rate	获取或设置媒体文件的播放速度
voluhme	获取或设置播放声音

下面对 IWMPSettings 对象的属性进行详细讲解。

（1）autoStart 属性

该方法表示获取或设置一个值用来表示是否自动播放当前媒体文件。

语法：

```
bool autoStart {get; set;}
```

属性值：如果自动播放当前媒体文件，则该值为 true；否则，该值为 false。

（2）balance 属性

该属性表示获取或设置当前立体声的平衡。

语法：

```
int balance {get; set;}
```

属性值：表示一个平衡值，可以在-100～100，默认值为 0。

（3）baseURL 属性

该属性表示获取或设置嵌入数字媒体目录用于和 URL 脚本命令相连的路径协议 URL。

语法：

```
string baseURL {get; set;}
```

属性值：表示一个 URL。

（4）defaultFrame 属性

该属性表示获取或设置从脚本命令事件接收用于显示 URL 的结构名称。

语法：

```
string defaultFrame {get; set;}
```

属性值：表示目标结构元素的名称属性值。

（5）enableErrorDialogs 属性

该属性表示获取或设置一个值，该值用来表示是否随机显示错误消息对话框。

语法：

```
bool enableErrorDialogs {get; set;}
```

属性值：表示是否随机显示错误消息对话框，默认值为 true。

（6）invokeURLs 属性

该属性表示获取或设置一个值，该值用来表示是否将 URL 事件发送到网页浏览器。

语法：

```
bool invokeURLs {get; set;}
```

属性值：表示是否将 URL 事件发送到网页浏览器，默认值为 true。

（7）mute 属性

该属性表示获取或设置一个值用来表示音频是否处于无声状态。

语法：

```
bool mute {get; set;}
```

属性值：表示音频是否处于无声状态，默认值为 false。

例如，将音频是否处于无声状态和一个 CheckBox 控件的状态绑定。代码如下：

```
private void Mute_CheckStateChanged(object sender, System.EventArgs e)
{
    //实例化一个 CheckBox 对象
    System.Windows.Forms.CheckBox Mute = (System.Windows.Forms.CheckBox)sender;
    Mute.Text = Mute.Checked ? "有声状态" : Mute.Text = "无声状态";    //设置 CheckBox 对象的文本
    //将 Windows Media Player 控件的音频是否处于无声状态与 CheckBox 控件的选定状态绑定
    player.settings.mute = Mute.Checked;
}
```

（8）playCount 属性

该属性表示获取或设置媒体文件播放的次数。

语法：

```
int playCount {get; set;}
```

属性值：表示播放次数，最小值和默认值均为 1。

（9）rate 属性

该属性表示获取或设置媒体文件的播放速度。

语法：

```
double rate {get; set;}
```

属性值：表示媒体文件的播放速度，最小值为 1.0。

例如，通过一个 NumericUpDown 控件来控制媒体文件的播放速度。代码如下：

```
private void playbackRate_Click(object sender, System.EventArgs e)
{
    //获取当前 NumericUpDown 控件的值
    double newRate = (double)((System.Windows.Forms.NumericUpDown)sender).Value;
    if (player.settings.get_isAvailable("Rate"))    //当 Rate 属性处于可用状态时
    {
        player.settings.rate = newRate;             //设置媒体文件的播放速度
    }
}
```

（10）voluhme 属性

该属性表示获取或设置播放声音。

语法：

```
int volume {get; set;}
```

属性值：表示声音的大小，可选值 0～100。

IWMPSettings 对象的方法如表 29.17 所示。

表 29.17　　IWMPSettings 对象的常用方法及说明

方　法	说　明
get_isAvailable	判断指定的操作是否可以执行
getMode	获取当前文件的播放模式
setMode	设置当前文件的播放模式

下面对 IWMPSettings 对象的方法进行详细讲解。

（1）get_isAvailable 方法

该方法表示判断指定的操作是否可以执行。

语法：

```
bool get_isAvailable (string bstrItem);
```

参数说明如下。

bstrItem：常见的取值如表 29.18 所示。

返回值：如果当前操作可执行，则该值为 true；否则，该值为 false。

表 29.18　　get_isAvailable 方法参数的可取值及对应的说明

可　取　值	说　　明
AutoStart	判断 AutoStart 属性是否可用
Balance	判断 Balance 属性是否可用
BaseURL	判断 BaseURL 属性是否可用
DefaultFrame	判断 DefaultFrame 属性是否可用
EnableErrorDialogs	判断 EnableErrorDialogs 属性是否可用
GetMode	判断 GetMode 方法是否可用
InvokeURLs	判断 InvokeURLs 属性是否可用
Mute	判断 Mute 属性是否可用
PlayCount	判断 PlayCount 属性是否可用
Rate	判断 Rate 属性是否可用
SetMode	判断 SetMode 方法是否可用
Volume	判断 Volume 属性是否可用

例如，判断 IWMPSettings 对象的所有属性和方法是否可用，然后将结果和对应的属性或者方法显示在多行文本框中。代码如下：

```
string[] propertyList = new string[12]{
    "AutoStart", "Balance", "BaseURL", "DefaultFrame", "EnableErrorDialogs",
    "GetMode", "InvokeURLs", "Mute", "PlayCount", "Rate", "SetMode", "Volume"
};                                          //实例化一个存储属性和方法名称的字符串数组
string[] results = new string[12];          //实例化一个用来存储判断结果的字符串数组
for (int i = 0; i < propertyList.Length; i++)  //循环遍历集合中的每一个元素
{
    bool isAvailable = player.settings.get_isAvailable(propertyList[i]); //获取当前元素是否可用

    results[i] = (propertyList[i] + " = " + isAvailable.ToString());      //实例化判断结果数组中的每一个元素
}
playerSettings.Lines = results;                                           //在多行文本框中显示内容
```

（2）getMode 方法

该方法表示获取当前文件的播放模式。

语法：

```
bool getMode (string bstrMode);
```

参数说明如下。

bstrMode：常见的参数值如表 29.19 所示。

返回值：如果指定的模式被激活，则返回值为 true；否则，返回值为 false。

表 29.19　　getMode 方法参数的可取值及对应的说明

可　取　值	说　　明
autoRewind	播放至列表末尾时返回播放列表中的第一个媒体文件
Loop	循环播放指定的媒体文件
showFrame	当不播放时显示最近的帧，这和文件的音轨无关
Shuffle	随机播放媒体文件

（3）setMode 方法

该方法表示设置当前文件的播放模式。

语法：

```
void setMode (string bstrMode,bool varfMode);
```

参数说明如下。

bstrMode：表示即将设置的播放模式，如表 29.20 所示。

varfMode：表示是否激活指定的模式。

11．URL 属性

该属性表示获取或设置播放的文件名。

语法：

```
public virtual string URL {get; set;}
```

例如，播放文本框中指定目录下的媒体文件。代码如下：

```
player.URL = inputURL.Text;
player.Ctlcontrols.play();
```

另外，将 Windows Media Player 控件添加到窗体上，在该控件上单击鼠标右键，弹出“Windows Media Player 控件属性”对话框，为 Windows Media Player 控件提供中文属性对话框，如图 29.3 所示。

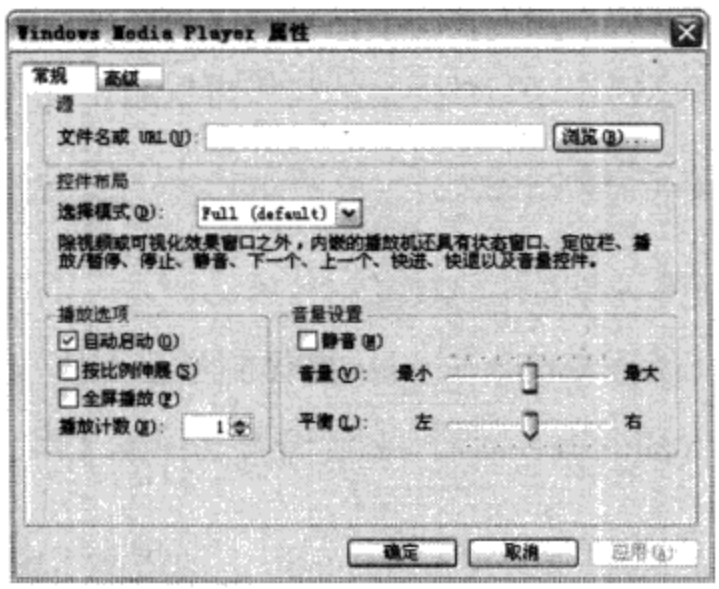


图 29.3　Windows Media Player 控件属性设置

Windows Media Player 控件方法如表 29-20 所示。

表 29.20　　Windows Media Player 控件主要方法及说明

方　法	说　明
Close	释放 Windows Media Player 控件所占用的资源
LaunchURL	在用户默认浏览器中加载 URL
NewMedia	为一个新的媒体文件返回一个 IWMPMedia 接口
NewPlaylist	为一个新的播放列表返回一个 IWMPPlaylist 接口
OpenPlayer	根据指定的路径通过 Windows Media Player 打开并播放音频或视频文件

下面对 Windows Media Player 控件比较重要的方法进行讲解。

（1）close 方法

该方法表示释放 Windows Media Player 控件所占用的资源。

语法：

```
public virtual void close ()
```

例如，用一个 Button 按钮实现 Windows Media Player 控件的停止播放功能。代码如下：

```
private void closeIt_Click(object sender, System.EventArgs e)
{
    player.close();
}
```

（2）newMedia 方法

该方法表示为一个新的媒体文件返回一个 IWMPMedia 接口。

语法：

```
public virtual IWMPMedia newMedia (string bstrURL)
```

参数说明如下。

bstrURL：表示一个媒体文件的 URL。

返回值：表示一个 IWMPMedia 对象。

（3）newPlaylist 方法

该方法表示为一个新的播放列表返回一个 IWMPPlaylist 接口。

语法：

```
public virtual IWMPPlaylist newPlaylist (string bstrName,string bstrURL)
```

参数说明如下。

bstrName：表示播放列表的名称。

bstrURL：表示播放列表的 URL。

返回值：表示一个 IWMPPlaylist 对象。

（4）OpenPlayer 方法

该方法表示根据指定的路径通过 Windows Media Player 打开并播放音频或视频文件。

语法：

```
public virtual void openPlayer (string bstrURL)
```

参数说明如下。

bstrURL：表示即将播放的媒体文件的 URL。

示例 29-02　Windows Media Player 控件制作多媒体播放器

本示例利用 Windows Media Player 控件来实现播放视频文件。

程序开发步骤如下。

（1）创建一个项目，命名为 MultimediaApplication，设置默认窗体的 Text 属性为“多媒体播放器”。

（2）在窗体上添加 Windows Media Player 控件和 3 个 Button 控件。

（3）程序代码如下。

代码 29-02　光盘位置：光盘\mingrisoft\sl\29\02

```
public static int count = 0;                                    //实例化一个计数器
OpenFileDialog openFile = new OpenFileDialog();                 //实例化一个打开文件对话框对象
IWMPPlaylistCollection PlayListCollection;                      //定义一个 IWMPPLaylistCollection 对象
WMPLib.IWMPPlaylist PlayList;                                   //定义一个 IWMPPlaylist 对象
private void Form1_Load(object sender, EventArgs e)
{
    this.axWindowsMediaPlayer1.settings.autoStart = false;      //设置 Windows Media Player 控件禁止自动播放
    this.listBox1.SelectionMode = SelectionMode.One;            //设置 listBox1 控件为单选模式
    //设置 Windows Media Player 控件的播放模式为目录一次
    this.axWindowsMediaPlayer1.settings.setMode("autoRewind", true);
    PlayListCollection = this.axWindowsMediaPlayer1.playlistCollection;   //实例化一个播放列表集合对象
    PlayList = PlayListCollection.newPlaylist("我最喜爱的音乐");          //实例化一个播放列表对象
}
private void AddInto_Click(object sender, EventArgs e)
{
    openFile.Filter = "WMA 格式(*.wma)|*.wma|MPEG 格式(*.mpeg)|*.mpeg|AVI 格式(*.avi)|*.avi|WMV 格式
(*.wmv)|*.wmv|所有文件(*.*)|*.*";                                    //设置打开文件对话框的过滤参数
    openFile.ShowDialog();                                          //显示通用对话框
    //实例化一个 IWMPMedia 对象
    WMPLib.IWMPMedia NewMedia = this.axWindowsMediaPlayer1.newMedia(this.openFile.FileName);
    this.PlayList.appendItem(NewMedia); //向播放列表中添加内容
    IWMPMedia media = PlayList.get_Item(PlayList.count - 1);        //获取播放列表中的媒体文件
```

```
        if (this.listBox1.Items.Count == 0)                    //当 listBox1 控件中不存在内容时
        {
            this.listBox1.Items.Add(media.name);               //向 listBox1 控件中添加从播放列表中获取的媒体文件
        }
        else                                                   //当 listBox1 控件中存在内容时
        {
            for (int j = 0; j < this.listBox1.Items.Count; j++) //循环遍历 listBox1 控件中的每一项内容
            {
                //当即将添加的媒体文件与当前 listBox1 控件中的内容有相同的情况时
                if (media.name == this.listBox1.Items[j].ToString())
                {
                    count++;                                   //累加 count 的值
                }
            }
            if (count == 0)                                    //当 count 的值为 0 时
            {
                this.listBox1.Items.Add(media.name);           //向 listBox1 控件中添加当前的媒体文件
                this.listBox1.SelectedIndex = this.listBox1.Items.Count - 1;    //设置 listBox1 控件的当前选定项索引
            }
            else                                               //当 count 的值不为 0 时
            {
                MessageBox.Show("列表中已存在将添加的媒体文件！", "信息提示", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk);                                      //弹出列表中已经存在即将添加的媒体文件
            }
        }
        //设置 Windows Media Player 控件的当前播放列表为已经定义的
        this.axWindowsMediaPlayer1.currentPlaylist = this.PlayList;
    }
    private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (this.listBox1.SelectedItem != null)                //当 listBox1 控件中的选定内容不为 null 时
        {
            for (int i = 0; i < this.PlayList.count; i++)      //循环遍历 PlayList 播放列表中的每一项内容
            {
                //当选定项与播放列表中的某一项同名时
                if (this.listBox1.SelectedItem.ToString().Trim() == this.PlayList.get_Item(i).name)
                {
                    //设置 Windows Media Player 控件的当前播放列表
                    this.axWindowsMediaPlayer1.currentPlaylist = this.PlayList;
                    //设置 Windows Media Player 控件的 URL 地址
                    this.axWindowsMediaPlayer1.URL = this.PlayList.get_Item(i).sourceURL;
                }
            }
        }
    }
    private void RemoveOut_Click(object sender, EventArgs e)
    {
        for (int i = 0; i < this.PlayList.count; i++)          //循环遍历播放列表中的每一项内容
        {
            //当选定项与播放列表中的某一项同名时
            if (this.listBox1.SelectedItem.ToString().Trim() == this.PlayList.get_Item(i).name)
            {
                this.PlayList.removeItem(this.PlayList.get_Item(i));    //移除播放列表中的指定项
                this.listBox1.Items.Remove(this.listBox1.SelectedItem); //从 listBox1 控件中移除指定的内容
            }
        }
    }
```

运行本示例，效果如图 29.4 所示。

图 29.4　多媒体播放器

29.3.2　DirectShow 设计

1．DirectShow 技术简介

DirectShow 是 DirectX 开发包中关于流媒体处理的一个开发包，这个开发包可以进行音频和视频的捕捉，使用它可以开发 DVD 应用程序和数字 TV 应用程序。DirectShow 支持的媒体文件主要有 WMA、MOV、MPG、AVI、MP3、WAV、WMV 等，无论是读取媒体文件还是将捕捉的数据写入到媒体文件，DirectShow 开发包都能提供很好的技术支持。

2．DirectShow 技术结构

DirectShow 技术利用标准组件来处理流媒体数据，这些组件称为过滤器。过滤器带有输入和输出针角，或二者兼而有。在 DirectShow 技术中处于最核心位置的就是作为“过滤器”的可插入标准组件，它是执行特定任务的 COM 对象。过滤器又分为源过滤器（source filter）、变换过滤器（Rransform filter）和表现过滤器（Renderer filter）等。过滤器通过向文件读写、修改数据和显示数据到输出设备上来操作流媒体。为了完成整个任务，必须要将所有过滤器 Filter 连接起来，这 3 种过滤器组成了过滤器图表结构，如图 29.5 所示。

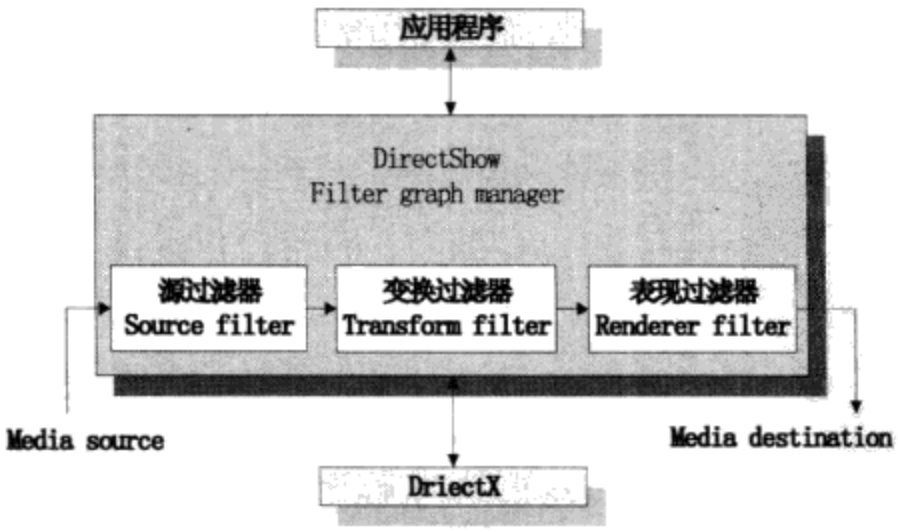


图 29.5　过滤器图表结构图

从图 29.5 中可以看出，过滤器图表是各种过滤器的集合，它是通过过滤器的输入、输出针脚顺序连接而成的，这些过滤器的针脚通过协商来决定它们将支持何种形式的多媒体。由于 DirectShow 支持可重构的过滤器图表结构，所以使用相同的软件组件可以播放多种类型的媒体。开发人员可以通过定义自己的过滤器来扩展 DirectShow 对媒体的支持功能。

在过滤器图表结构中，源过滤器用来从数据源获取数据并将数据传送到过滤器图表中。数

据源可以是摄像机、因特网、磁盘文件等。转换过滤器用来获取、处理和传送媒体数据，它包括分离视频和音频的分解变换过滤器（Splitter transform filter）、解压视频数据的视频转换过滤器（Video transform filter）、解压音频数据的音频转换过滤器（Audio transform filter）。表现过滤器用来在硬件上（如显卡和声卡）或者是任何可以接受媒体数据的地方（如磁盘文件）表现媒体数据，包括用来显示图像的视频表现过滤器（Video renderer filter）和将音频数据送到声卡上去的音频表现过滤器（Audio renderer filter）。

在过滤器图表中，为了完成特定的任务，必须将所有需要的过滤器连接起来，因此前级过滤器的输出必定成为下级过滤器的输入。一个过滤器至少有一个输入针（Input pin），并将特定的输出送到输出针（Output pin），如图 29.6 所示，为一个过滤器连接图。

图 29.6　过滤器连接图

应用程序不需要对过滤器图表中的各个过滤器进行单独的处理，因为在更高的层次上，DirectShow 提供一个称为过滤图表管理器的部件（FGM）管理着这些过滤器的连接和流媒体数据在过滤器之间的流动。FGM 提供了一套 COM 接口，应用程序可以通过它来访问过滤器图表，控制流媒体或者接收过滤器事件。如果需要，它可以自动的插入一个合适的解码器，并将转换过滤器的输出针脚连接到表现过滤器。应用程序可以通过与过滤图表管理器的通信来控制过滤器图表的活动。程序开发人员只需要调用 API 函数来实现对流媒体的控制，如 run 方法启动流媒体在过滤器图表（Filter graph）中的流动，pause 方法暂停流媒体的播放，stop 方法停止播放流媒体等。

3．DirectShow 技术开发准备

DirectShow 组件在“C:\WINDOWS\system32”目录下的 Quartz.dll 动态库中，要使 C#代码引用 COM 对象和接口，必须将 COM 类型库转换为.NET 框架元数据，从而有效地创建一个可以从任何托管语言调用的托管包装。在转换过程中需要使用 FrameWork SDK 自带的 TlbImp 命令工具，该命令工具在“C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\TlbImp.exe”目录下。该命令的使用方法为：

```
TlbImp C:\WINDOWS\system32\quartz.dll out：C:\WINDOWS\system32\quartzDriectShow.dll
```

在 DOS 命令下执行转换成功，如图 29.7 所示。

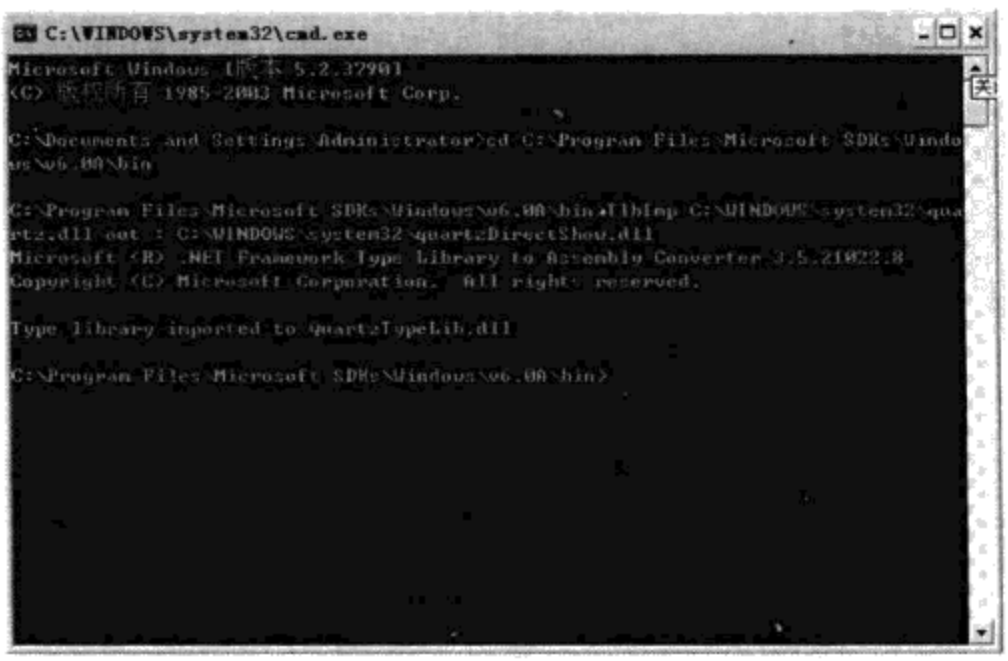


图 29.7　装换 Quartz.dll

转换完成后需在应用程序中引用 quartzDriectShow，引用 quartzDriectShow.dll 组件步骤如下。

（1）在 Visual Studio 2008 开发环境中，选择菜单“项目”→“添加引用”命令，弹出“添加引用”对话框。

（2）选择“浏览”选项卡，通过浏览找到引用 quartzDriectShow.dll 所在的位置，并引用到程序中，如图 29.8 所示。

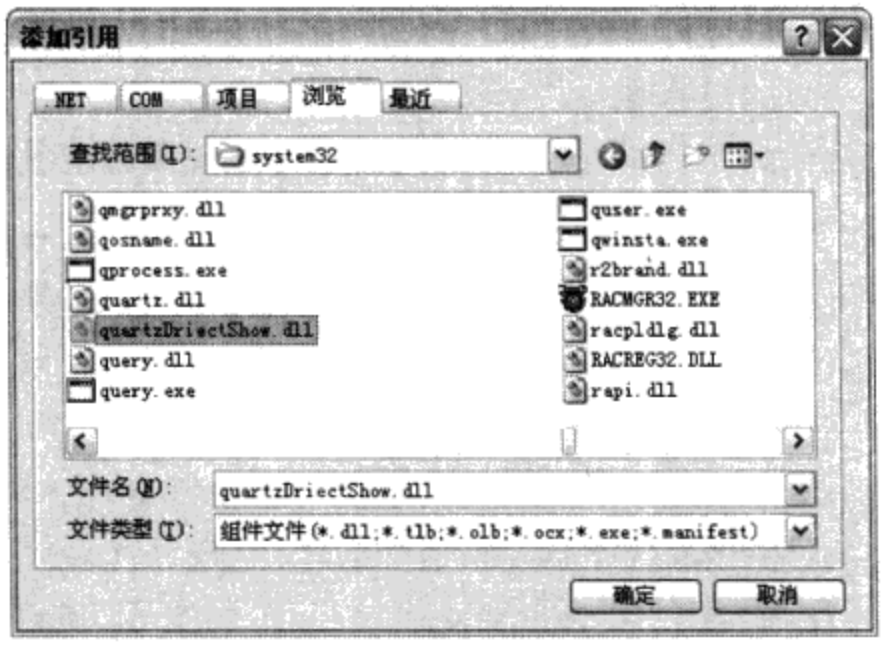


图 29.8　引用 quartzDriectShow 组件

（3）引入 using quartzDriectShow 后，在程序中就可以开发相关多媒体程序了。

开发人员还可以通过编写自己的过滤器扩展 DirectShow 对多媒体的支持。下面是 Direct Show 组件的接口。

- IFilterGraph：过滤通道接口。
- IFilterGraph2：增强的 IfilterGraph。
- IGraphBuilder：最为重要的 COM 接口，用于手动或者自动构造过滤通道 Filter Graph Manager。
- IMediaControl：用来控制流媒体（如流的启动和停止暂停等）播放控制接口。
- IMediaEvent：播放事件接口，该接口在 FilterGraph 发生一些事件时用来创建事件的标志信息并传送给应用程序。
- IMediaEventEx：扩展播放事件窗口。
- IMediaPosition：播放的位置和速度控制接口（控制播放放置只能为设置时间控制方式）。
- IMediaSeeking：另一个播放的位置和播放速度控制接口，有较强的位置选择功能设置播放格式。常用的控制播放方式为 TIME_FORMAT_MEDIA_TIME，单位 100ms；TIME_FORMAT_ FRAME，按帧播放。
- IBasicAudio：声音控制接口。
- IBasic Video：图像控制接口（波特率、宽度、长度等信息）。
- IVideoWindow：显示窗口控制接口（有关播放窗口的一切控制，包括 caption 显示、窗口位置控制等）。
- ISampleGrabber：捕获图像接口（可用于抓图控制）。
- IVideoFrameStep：控制单帧播放的接口。

注意：使用 DirectShow 接口编程有 3 个步骤，初始化接口→利用接口中的控制函数使用控制操作→释放接口。

4．使用测试工具 GraphEdit

测试工具（GraphEdit）所在安装路径为“C:\ProgramFiles\MicrosoftSDKs\Windows\v6.0A\bin \\graphedt.exe”。通过该工具，可以很直观地看到 Filter Graph 的运行及处理流程，方便进行程序调试。

首先运行 GraphEdit，执行菜单“File”→“Render Media File”命令，然后选择一个媒体文件；当 Filter Graph 构建成功后，单击工具栏的“运行”按钮，播放已选择的媒体文件，如图 29.9 所示。

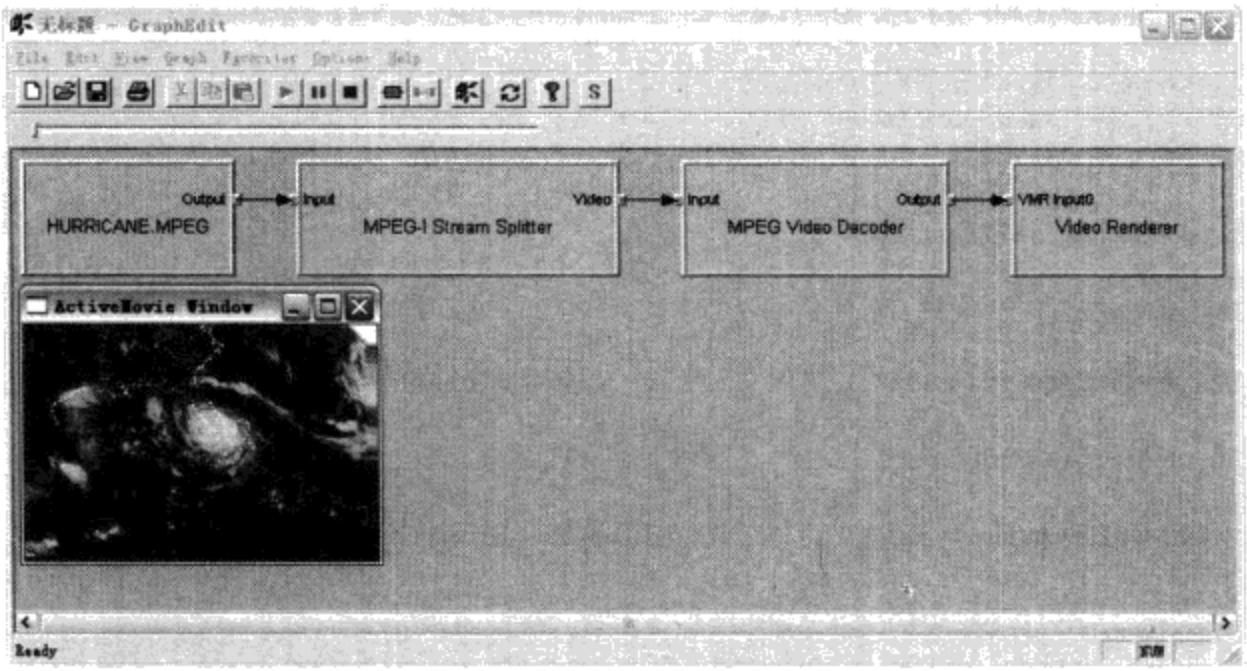


图 29.9　GraphEdit 测试工具的使用

示例 29-03　DirectShow 技术设计多媒体播放器的应用

本示例利用 DirectShow 技术来实现多媒体播放器。

程序开发步骤如下。

（1）创建一个项目，命名为 DirectShowApplication，设置默认窗体的 Text 属性为“多媒体播放器”。

（2）在窗体上添加 6 个 Button 控件，主要用来操作播放的媒体文件，如图 29.10 所示。

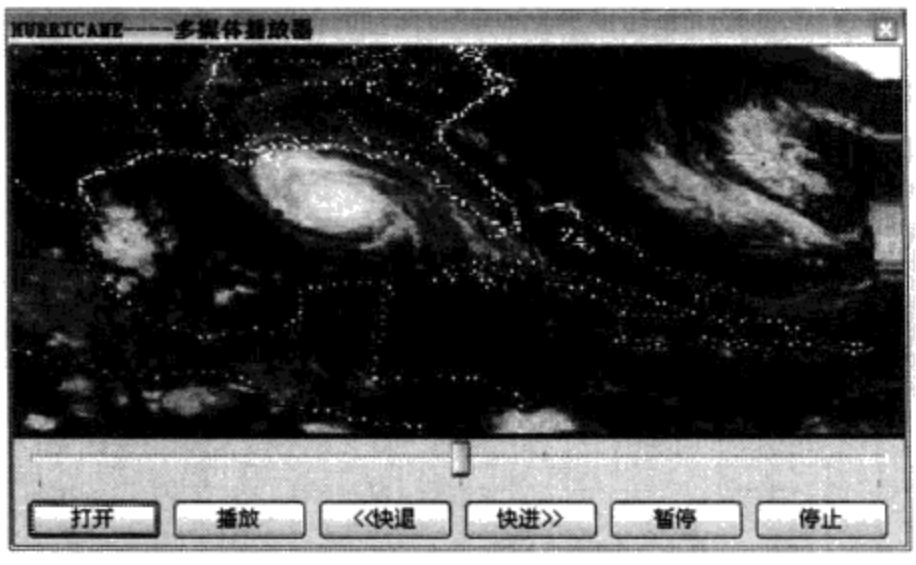


图 29.10　DirectShow 技术设计多媒体播放器的应用

（3）在窗体上添加一个 TrackBar 控件，用来显示媒体文件播放的进度。

（4）在窗体上添加一个 Timer 控件，用来计算媒体当前的进度。

（5）程序代码如下：

代码 29-03　光盘位置：光盘\mingrisoft\sl\29\03

```
const int WM_APP = 0x8000;                          //设置常量
const int WM_GRAPHNOTIFY = WM_APP + 1;
const int EC_COMPLETE = 0x01;
const int WS_CHILD = 0x40000000;
const int WS_CLIPCHILDREN = 0x2000000;
FilgraphManager m_objFilterGraph = null;            //实例化程序中用到的各个对象
IBasicAudio m_objBasicAudio = null;
IVideoWindow m_objVidwoWindow = null;
IMediaEvent m_objMediaEvent = null;
IMediaEventEx m_obnMediaEventEx = null;
IMediaPosition m_objMediaPosition = null;
IMediaControl m_objMediaControl = null;
public enum MediaStatus                             //实例化一个枚举对象
{
    None, Stopped, Paused, Running
};
private void Clear()
{
    if (m_objMediaControl != null)                  //当 m_objMediaControl 的值不为 null 时
    {
        m_objMediaControl.Stop();                   //停止视频的播放
    }
    if (m_obnMediaEventEx != null)                  //当 m_objMediaEventEx 中的值不为 null 时
    {
        m_obnMediaEventEx.SetNotifyWindow(0, 0, 0); //设置消息窗口
    }
    if (m_objVidwoWindow != null)                   //当 m_objVidwoWindow 的值不为 null 时
    {
        m_objVidwoWindow.Visible = 0;               //设置对象 m_objVidwoWindow 的 Visible 属性值为 0
        m_objVidwoWindow.Owner = 0;                 //设置对象 m_objVidwoWindow 的 Owner 属性值为 0
    }
    if (m_objBasicAudio != null)                    //当 m_objBasicAudio 的值不为 null 时
    {
        m_objBasicAudio = null;                     //设置 m_objBasicAudio 的值为 null
    }
    if (m_objFilterGraph != null)                   //当 m_objFilterGraph 的值不为 null 时
    {
        m_objFilterGraph = null;                    //设置 m_objFilterGraph 的值为 null
    }
    if (m_objMediaControl != null)                  //当 m_objMediaControl 的值不为 null 时
    {
        m_objMediaControl = null;                   //设置 m_objMediaControl 的值为 null
    }
    if (m_objMediaEvent != null)                    //当 m_objMediaEvent 的值不为 null 时
    {
        m_objMediaEvent = null;                     //设置 m_objMediaEvent 的值为 null
    }
    if (m_objMediaPosition != null)                 //当 m_objMediaPosition 的值不为 null 时
    {
        m_objMediaPosition = null;                  //设置 m_objMediaPosition 的值为 null
    }
    if (m_objVidwoWindow != null)                   //当 m_objVidwoWindow 的值不为 null 时
    {
        m_objVidwoWindow = null;                    //设置 m_objVidwoWindow 的值为 null
    }
    if (m_obnMediaEventEx != null)                  //当 m_obnMediaEventEx 的值不为 null 时
    {
        m_obnMediaEventEx = null;                   //设置 m_obnMediaEventEx 的值为 null
    }
    this.Text = "多媒体播放器";                      //设置当前窗体的标题栏文本
    timer1.Enabled = false;                         //禁用计时器 timer1
```

```
        trackBar1.Value = 0;                                    //设置控件 trackBar1 控件的当前值为 0
    }
    private void Open_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFile = new OpenFileDialog();   //实例化一个 OpenFileDialog 对象
        //设置对象 openFile 对象的过滤参数
        openFile.Filter = "选择播放文件|*.mpeg;*.avi;*.mov;*.wma;*.wav;*.mp3|All Files|*.*";
        if (DialogResult.OK == openFile.ShowDialog())       //当选定所需的媒体文件单击“打开”时
        {
            this.Clear();                                   //清理正在播放的文件
            m_objFilterGraph = new FilgraphManager();       //实例化对象 m_objFilterGraph
            try
            {
                m_objFilterGraph.RenderFile(openFile.FileName);                    //打开指定文件
                m_objBasicAudio = m_objFilterGraph as IBasicAudio;                 //图像控制接口
                m_objVidwoWindow = m_objFilterGraph as IVideoWindow;               //声音控制接口
                m_objVidwoWindow.Owner = (int)this.Handle;                 //设置对象 m_objVidwoWindow 的所有者
                m_objVidwoWindow.WindowStyle = WS_CHILD | WS_CLIPCHILDREN; //设置窗口类型
                m_objVidwoWindow.SetWindowPosition(this.ClientRectangle.Left, this.ClientRectangle.Top,
this.ClientRectangle.Width, this.ClientRectangle.Height - 60);             //设置播放容器
            }
            catch (Exception ex)                                           //捕获异常
            {
                MessageBox.Show(ex.Message,"信息提示");                    //弹出错误的信息提示
                m_objVidwoWindow = null;                                   //当出错时，将视频对象清空
            }
            m_objMediaEvent = m_objFilterGraph as IMediaEvent;             //播放事件接口
            m_obnMediaEventEx = m_objFilterGraph as IMediaEventEx;         //扩展播放事件接口
            m_obnMediaEventEx.SetNotifyWindow((int)this.Handle, WM_GRAPHNOTIFY, 0);  //设置通知消息窗口
            m_objMediaPosition = m_objFilterGraph as IMediaPosition;       //播放位置和速度控制
            m_objMediaControl = m_objFilterGraph as IMediaControl;         //用来控制流媒体播放控制接口
            //重新设置标题栏文本内容
            this.Text = System.IO.Path.GetFileNameWithoutExtension( openFile.FileName) + "----" + this.Text;
            m_objMediaControl.Run();                                       //播放文件
            try
            {
                trackBar1.LargeChange = 1;                                 //设置 trackBar1 控件的变化值
                trackBar1.Maximum = (int)m_objMediaPosition.Duration;      //设置 trackBar1 控件的最大值
                timer1.Enabled = true;                                     //启用计时器 timer1
            }
            catch { }                                                      //捕获异常
        }
    }
    private void Play_Click(object sender, EventArgs e)
    {
        if (m_objMediaControl != null)   //当 m_objMediaControl 的值不为 null 时
        {
            m_objMediaControl.Run();  //播放指定的媒体文件
        }
    }
    private void Pause_Click(object sender, EventArgs e)
    {
        m_objMediaControl.Pause();                                         //暂停媒体文件的播放
    }
    private void Stop_Click(object sender, EventArgs e)
    {
        m_objMediaControl.Stop();                                          //停止媒体文件的播放
        m_objMediaPosition.CurrentPosition = 0;                            //设置对象的 CurrentPosition 为起始位置
        this.Clear();                                                      //调用方法 Clear
    }
    private void MoveBack_Click(object sender, EventArgs e)
    {
        if (m_objMediaPosition.CurrentPosition >= 1)                       //当媒体文件已经开始播放时
```

```
        {
            //设置媒体文件的当前播放位置
            m_objMediaPosition.CurrentPosition = m_objMediaPosition.CurrentPosition - 1;
        }
        else  //当媒体文件未开始播放时
        {
            m_objMediaPosition.CurrentPosition = 0; //设置媒体文件的当前播放位置
        }
    }
    private void MoveForward_Click(object sender, EventArgs e)
    {
        m_objMediaPosition.CurrentPosition = m_objMediaPosition.CurrentPosition + 1;      //增加媒体文件的播放进度
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        trackBar1.Value = (int)m_objMediaPosition.CurrentPosition;                        //显示文件播放的进度
    }
    private void trackBar1_Scroll(object sender, EventArgs e)
    {
        m_objMediaPosition.CurrentPosition = trackBar1.Value;//根据滚动进度条来播放文件
    }
```

运行本示例，效果如图 29.10 所示。

29.4 动 画 设 计

29.4.1 ImageAnimator 类功能

ImageAnimator 类是 GDI+提供的实现动画效果类，动画处理包含基于时间的帧图像。

29.4.2 ImageAnimator 类方法

ImageAnimator 类常用方法及说明如表 29.21 所示。

表 29.21　　ImageAnimator 类常用方法及说明

方　法	说　明
Animate	将多帧图像显示为动画
CanAnimate	返回一个布尔值，该值指示指定图像是否包含基于时间的帧
Equals	已重载。确定两个 Object 实例是否相等（从 Object 继承）
GetHashCode	用作特定类型的哈希函数。GetHashCode 适合在哈希算法和数据结构（如哈希表）中使用（从 Object 继承）
GetType	获取当前实例的 Type（从 Object 继承）
ReferenceEquals	确定指定的 Object 实例是否是相同的实例（从 Object 继承）
StopAnimate	终止正在运行的动画
ToString	返回表示当前 Object 的 String（从 Object 继承）
UpdateFrames	该方法为可重载方法，用于更新时间帧

下面对 ImageAnimator 类比较重要的方法进行详细介绍。

（1）Animate 方法

该方法表示将多帧图像显示为动画。

语法：

```
public static void Animate (Image image,EventHandler onFrameChangedHandler)
```

参数说明如下。

image：要动画处理的 Image 对象。

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

该方法返回一个布尔值，该值指示指定图像是否包含基于时间的帧。

语法：

```
public static bool CanAnimate (Image image)
```

参数说明如下。

image：表示要测试的 Image 对象。

返回值：如果指定图像包含基于时间的帧，则此方法返回 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：

```
public static void StopAnimate (Image image,EventHandler onFrameChangedHandler)
```

参数说明如下。

image：要停止动画处理的 Image 对象。

onFrameChangedHandler：一个 EventHandler 对象，它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：

```
public static void UpdateFrames ()
```

返回值：此方法无返回值。

（5）UpdateFrames 方法(Image)

此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。此方法只适用于包含基于时间的帧的图像。

语法：

```
public static void UpdateFrames (Image image)
```

参数说明如下。

image：要为其更新帧的 Image 对象。

注意：用 AutoPlay 属性播放的 AVI 文件将不断重复，直到将其值设置为 false 为止。

29.4.3 ImageAnimator 类的应用

实例位置：光盘\mingrisoft\dxyy\29\01

运行程序，单击“播放动画”按钮，播放 Gif 动画；单击“停止播放”按钮，停止播放 Gif 动画。

代码 29-01　光盘位置：光盘\mingrisoft\dxyy\29\01

```
Bitmap bitmap = new Bitmap("1.gif");        //实例化一个 GDI+位图对象
bool current = false;                       //定义一个全局变量标识
public void AnimateImage()
{
    if (!current)                           //当 current 的值为 false 时
    {
        ImageAnimator.Animate(bitmap, new EventHandler(this.OnFrameChanged));//以指定的形式动画显示图片
        current = true;                     //设置 current 的值为 true
    }
}
private void OnFrameChanged(object o, EventArgs e)
{
    this.Invalidate();                      //使整个画面无效并导致重绘事件
}
protected override void OnPaint(PaintEventArgs e)
{
```

```
        AnimateImage();                          //动画显示图片
        ImageAnimator.UpdateFrames();            //使该帧在当前正被动画处理的所有图像中前移
        e.Graphics.DrawImage(this.bitmap, new Point(0, 0));                    //在指定位置绘制图像
    }
    private void Stop_Click(object sender, EventArgs e)
    {
        ImageAnimator.StopAnimate(bitmap, new EventHandler(this.OnFrameChanged));    //停止动画的播放
    }
    private void Play_Click(object sender, EventArgs e)
    {
        ImageAnimator.Animate(bitmap, new EventHandler(this.OnFrameChanged));        //播放指定动画
    }
```

运行本示例，效果如图 29.11 所示。

图 29.11 ImageAnimator 类播放 GIF 动画

第7篇

报表与打印技术

C#

第 30 章

水晶报表

- 水晶报表简介
- .NET 平台下的 Crystal 报表
- Crystal 报表数据源和数据库的操作

Crystal Reports 水晶报表是内置于 Visual Studio.NET 开发环境中的一种报表设计工具，它能帮助程序员在.NET 平台上创建专业级的互动式报表。本章将对 Crystal Reports 水晶报表进行详细介绍。

30.1 水晶报表简介

Crystal Reports 水晶报表用于创建交互式报表的软件包，它为开发人员提供了丰富的工具。

使用 Crystal Reports 水晶报表可以创建简单的报表，也可以创建复杂的、专业的报表，它可以由任何数据源生成所需要的报表。内置报表专家在生成报表和完成一般报表的过程中，会一步一步地指导开发人员进行操作，它可以通过公式、交叉表、子报表和设置条件格式等表现数据的实际意义，揭示可能被隐藏的重要关系。

Crystal Reports 水晶报表可以用各种形式发布，如 Word、Excel、电子邮件和 Web 等。高级的 Web 水晶报表还允许工作组中的其他成员在各自的 Web 浏览器中查看或共享报表。

将 Crystal Reports 水晶报表整合到数据库应用程序中，不仅可以节省开发时间，还可以更大程度地满足用户需求。Crystal Reports 水晶报表支持大多数流行的开发语言。

30.2 .NET 平台下的 Crystal 报表

30.2.1 Crystal Reports.Net 简介

Crystal Reports 自 1993 年开始就已经成为 Visual Studio 的一部分，并且现在已经成为了 Visual Studio 2008 中的标准报表创建工具。每套 Visual Studio 2008 都附带了该工具，并且直接集成到程序开发环境中。

30.2.2 创建 Crystal 报表

创建一个新的报表，首先在 Visual Studio 2008 环境中建立一个项目，创建 Crystal 报表的具体步骤如下。

（1）打开 Visual Studio 2008 开发环境，选择“文件”→“新建”→“项目”，弹出如图 30.1 所示的“新建项目”对话框。在该对话框的“项目类型”中选中“Visual C#”，在“模板”中选中“Windows 应用程序”，在“名称”对话框中输入项目名称，并选择项目的保存位置。

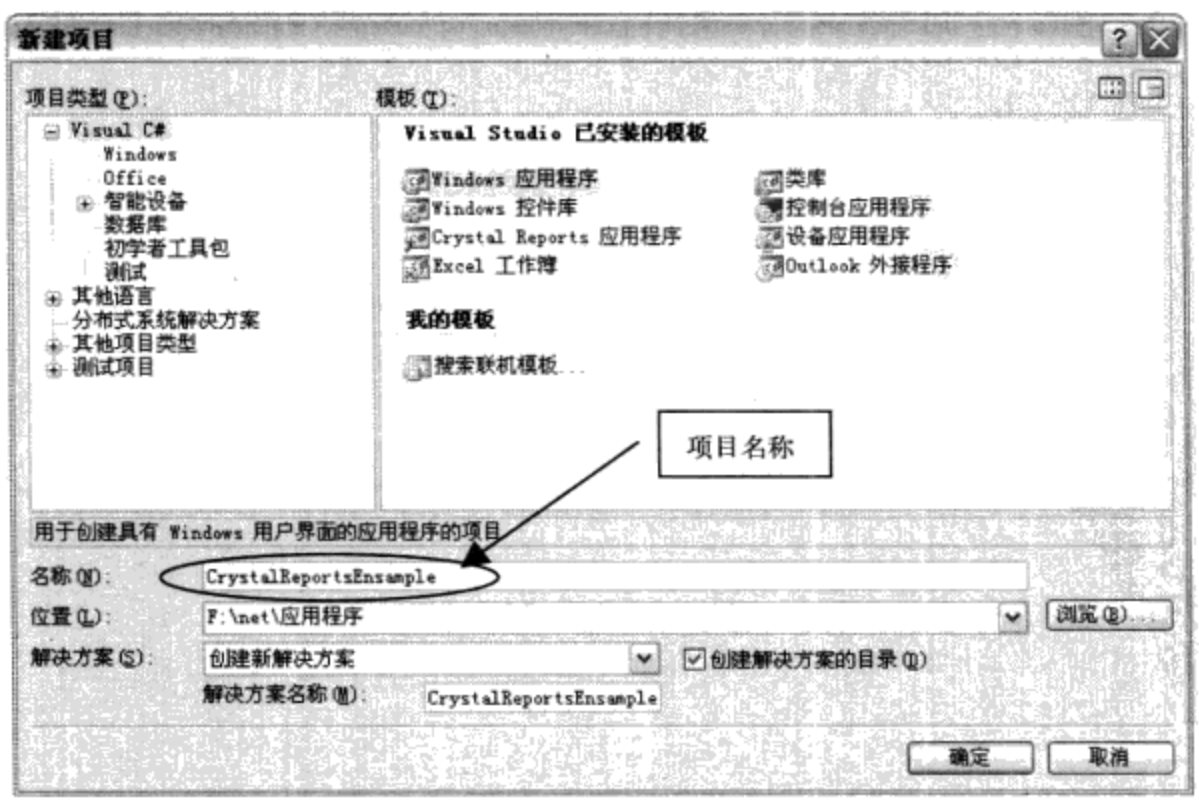


图 30.1 “新建项目”对话框

（2）单击“确定”按钮，此时在 Visual Studio 2008 开发环境的右边将弹出一个“解决方案资源管理器”窗口，如图 30.2 所示，若该窗口未出现，可以通过选择“视图”→“解决方案管理器”选项来显示该窗口。

（3）选中当前项目名称，单击鼠标右键，在弹出的快捷菜单中选择“添加”→“新建项”选项，弹出“添加新项”对话框，如图 30.3 所示。选择“Crystal 报表”，并在“名称”文本框中输入报表名称。

（4）单击“添加”按钮，弹出“Crystal Reports 库”对话框，如图 30.4 所示。

图 30.2 “解决方案资源管理器”窗口

（5）选择“使用报表向导”选项，单击“确定”按钮，用户即可根据向导完成水晶报表的创建工作。另外，“作为空白报表”选项用来打开 Crystal Reports 设计器，以便手动设计报表；“来自现有的报表”选项用来根据已经存在的报表创建一张相同的报表。

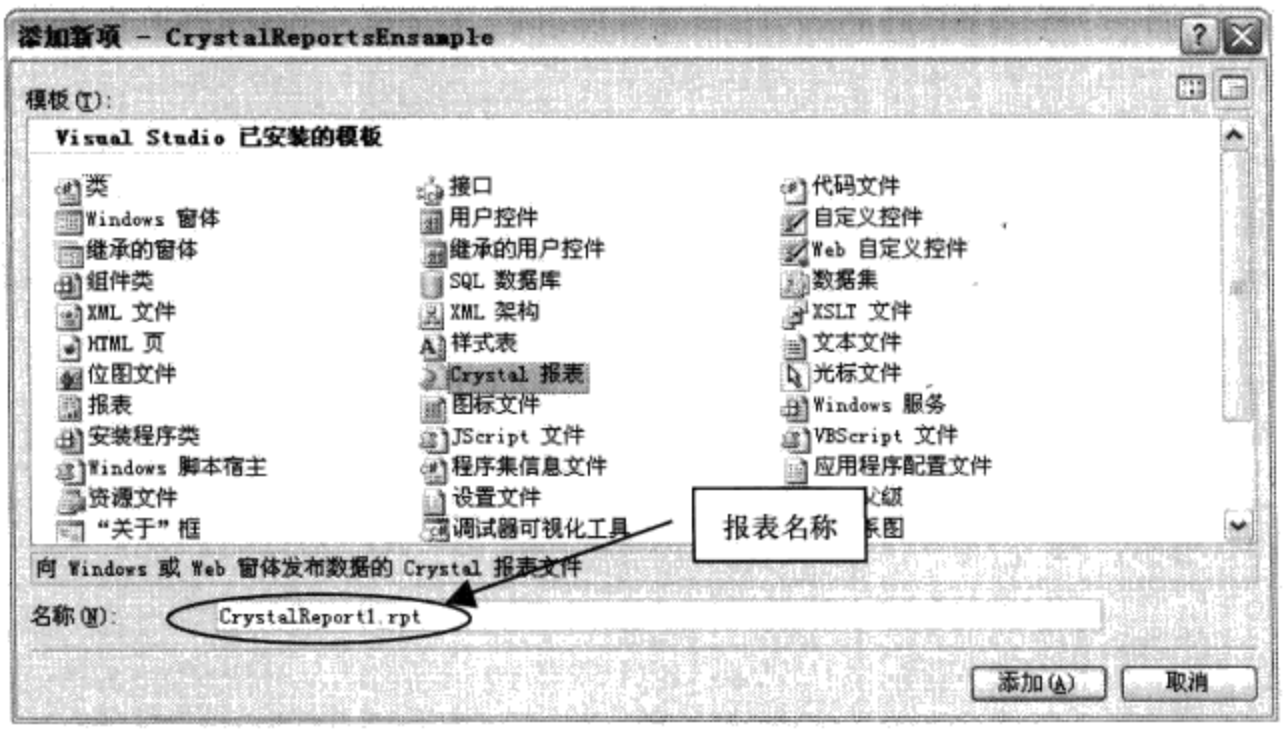


图 30.3 “添加新项”对话框

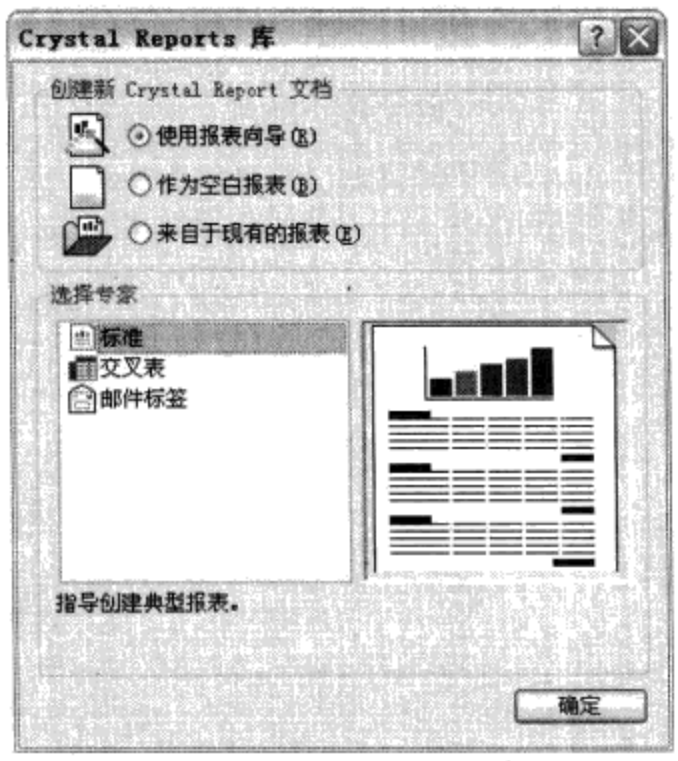


图 30.4 “Crystal Reports 库”对话框

30.2.3 Crystal 报表设计器的环境介绍

在 Visual Studio 2008 开发环境中打开一个报表文件（.rpt），这里打开 Visual Studio 2008 开发环境自带的一个报表文件“World Sales Report.rpt”，该文件位于 Visual Studio 2008 安装目录中的“Crystal Reports\Samples\zh-chs\Reports\General Business”文件夹中。

Crystal 报表设计器环境如图 30.5 所示，右边是“字段资源管理器”窗口，该窗口中显示的树型图中包含了可以添加到报表中的数据库字段、特殊字段、公式字段、参数字段、组名字段、运行总计字段、SQL 表达式字段、特殊字段和未绑定字段，“字段资源管理器”窗口会在已经被引用的字段旁边显示一个选中标记。如果 Visual Studio 2008 开发环境中没有显示“字段资源管理器”窗口，可以通过选择菜单栏中的“视图”→“其他窗口”→“文档大纲”选项将其显示出来。Visual Studio 2008 开发环境的中间是“报表设计区”，在该区域中，用户可以根据需要设计各种样式的报表。

图 30.5 Crystal 报表设计器环境

30.2.4 Crystal 报表区域介绍

对于一张新的 Crystal 报表，报表设计区被分成 5 部分，分别为报表页眉、页眉、详细资料、报表页脚和页脚，如图 30.6 所示。用户在设计过程中，可以选择创建其他区域，也可以隐藏已有区域。下面对 Crystal 报表的区域进行详细介绍。

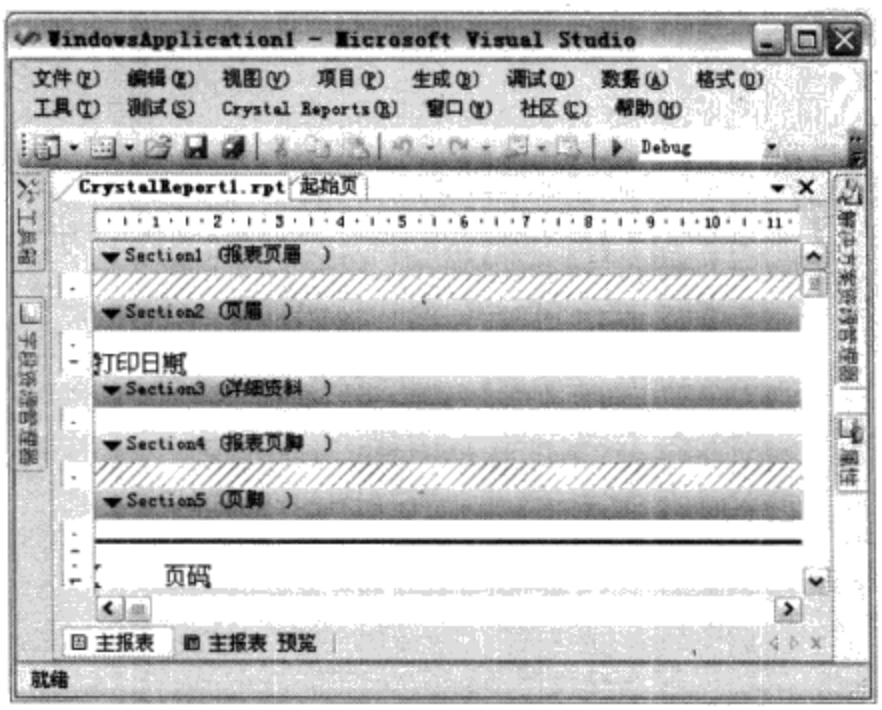


图 30.6 Crystal 报表区域

1．报表页眉

- 该区域中的信息和对象只在 Crystal 报表的开头显示一次。
- 该区域通常包含 Crystal 报表的标题和其他只在 Crystal 报表开始位置出现的信息。
- 放在该区域中的图表和交叉表包含整个 Crystal 报表的数据。
- 放在该区域中的公式只在 Crystal 报表开始进行一次求值。

2．页眉

- 该区域中的信息和对象显示在每个新页的开始位置。
- 该区域通常包含只出现在每页顶部的信息，例如，它可以包含文本字段，也可以用来包含字段标题等。
- 图表或交叉表不能放在该区域。
- 放在该区域中的公式在每个新页的开始进行一次求值。

3．详细资料

- 该区域中的信息和对象随每条新记录显示。
- 该区域包含 Crystal 报表正文数据，例如，批量报表数据通常出现在该区域。
- 图表或交叉表不能放置在该区域中。
- 放在该区域中的公式对每条记录进行一次求值。

4．报表页脚

- 该区域中的信息和对象只在 Crystal 报表的结束位置显示一次。
- 该区域可用来输出只在 Crystal 报表的末尾出现一次的信息（如统计）。
- 放在该区域中的图表和交叉表包含整个 Crystal 报表的数据。
- 放在该区域中的公式只在 Crystal 报表的结束位置进行一次求值。

5．页脚

- 该区域中的信息和对象显示在每页的底部。
- 该区域通常包含页码和任何其他希望出现在每页底部的信息。
- 图表和交叉表不能放置在该节中。
- 放在该节中的公式在每个新页的结束位置进行一次求值。

另外，如果将组、摘要或小计添加到 Crystal 报表，则程序会自动创建两个区域：组页眉和组页脚。其中，“组页眉”区域出现在“详细资料”区域的正上方，而“组页脚”区域出现在“详细资料”区域的正下方。跟原始报表区域一样，每个新添加的区域也可以包含一个或多个子区域，默认情况下，它们都只包含一个区域。

6．组页眉

- 放在该区域中的对象显示在每个新组的开始位置。
- 该区域通常保存组名字段，也可以用来显示包括特定数据的图表或交叉表。
- 放在该区域中的图表和交叉表仅包含本组数据。
- 放在该区域中的公式在每组的开始位置对本组进行一次求值。

7．组页脚

- 放在该区域中的对象显示在每个新组的结束位置。
- 该区域通常保存汇总数据，也可以用来显示图表或交叉表。
- 放在该区域中的图表和交叉表仅包含本组数据。
- 放在该区域中的公式在每组的结束位置对本组进行一次求值。

30.3 Crystal 报表数据源和数据库的操作

30.3.1 Visual Studio 2008 中 Crystal Reports 数据源列举

Crystal 水晶报表通过数据库驱动程序与数据库进行连接，用户可以根据下列数据源中的数据进行报表设计。

- 使用 ODBC 驱动程序的任何数据库（RDO）。
- 使用 OLEDB 提供程序的任何数据库（ADO）。
- Microsoft Access 数据库或 Excel 工作簿（DAO）。
- ADO.NET 记录集。

30.3.2 Visual Studio 2008 中 Crystal Reports 数据库操作

1．数据源

本节中以 ADO.NET 数据集为例介绍如何在 Crystal 报表中操作数据源。在 Crystal 报表中操作数据源的步骤如下。

（1）在“解决方案资源管理器”窗口中，选中当前项目，单击鼠标右键，选择“添加”→“新建项”选项，弹出如图 30.7 所示的“添加新项”对话框。在该对话框中，选择“数据集”选项，并在“名称”文本框中输入数据集名称。

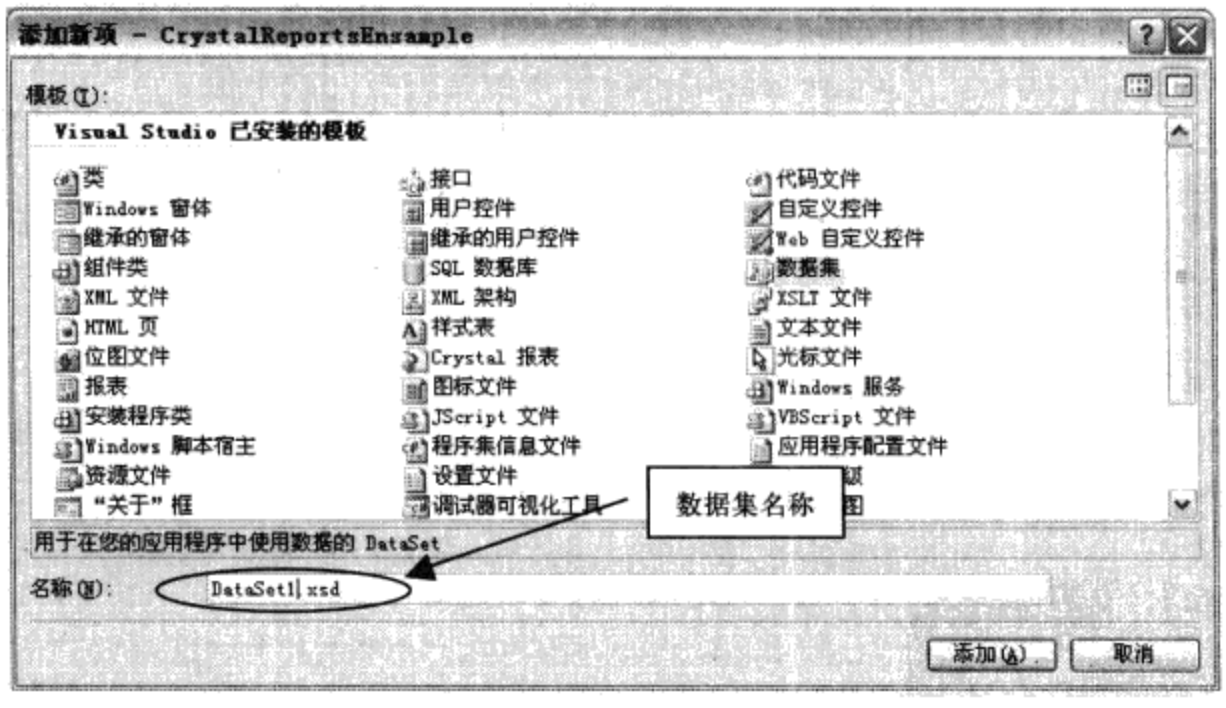


图 30.7 “添加新项”对话框

（2）单击“添加”按钮，在当前项目中添加一个 DataSet.xsd 文件，双击打开该文件，在“工具箱”窗口中，把“TableAdatper”拖入窗体中，弹出如图 30.8 所示的“TableAdapter 配置向导”对话框。

（3）单击“新建连接”按钮，弹出“添加连接”对话框，如图 30.9 所示。

（4）单击“更改”按钮，弹出如图 30.10 所示“更改数据源”对话框，在该对话框中，选择“Microsoft SQL Server 数据库文件”。

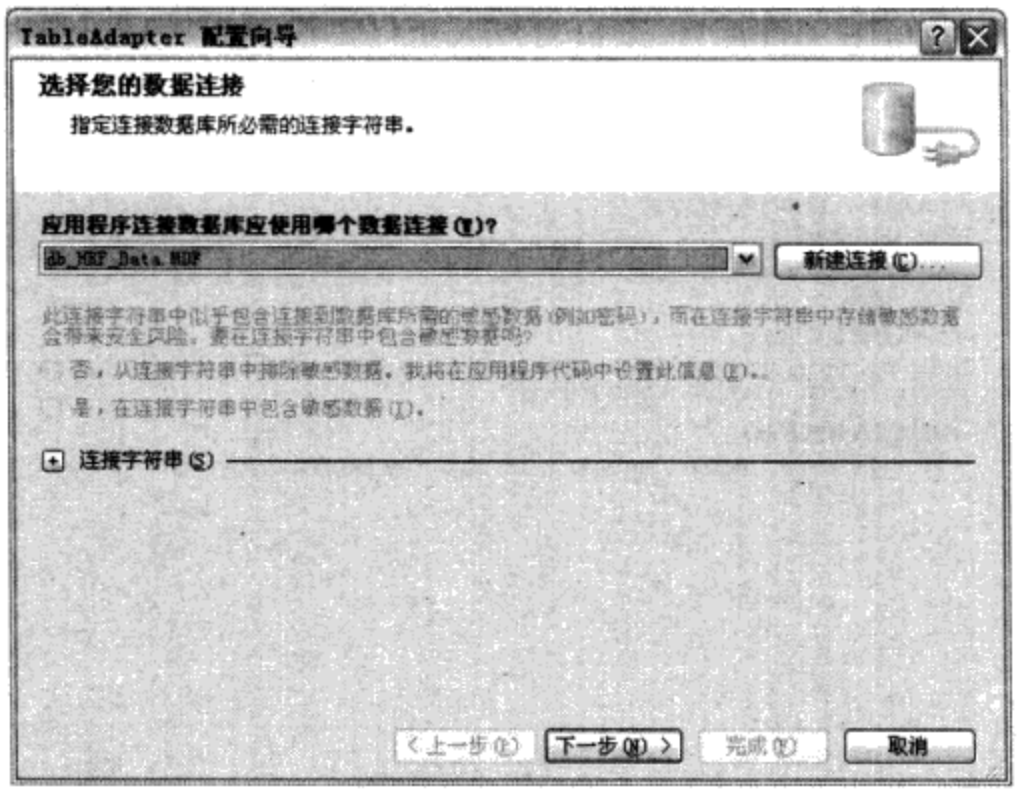


图 30.8 “TableAdapter 配置向导”对话框

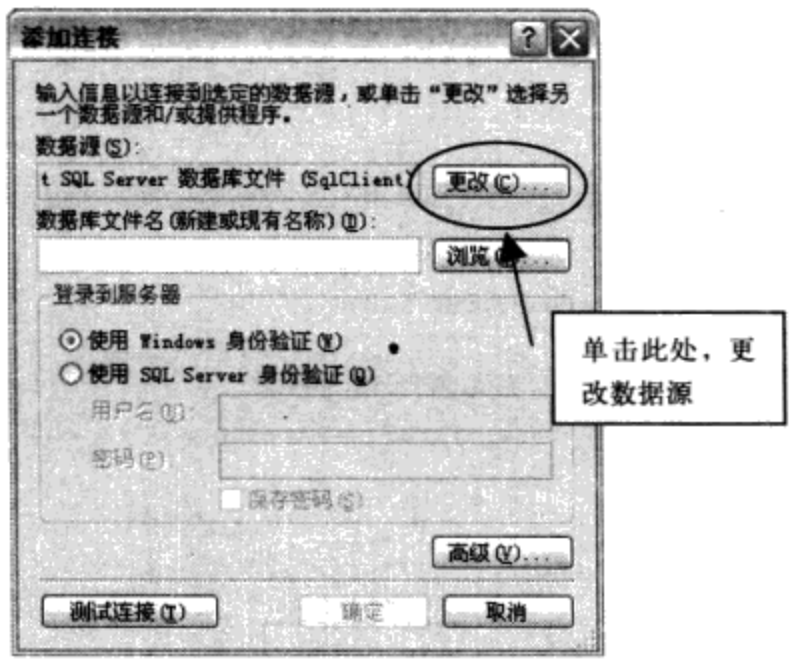


图 30.9 “添加连接”对话框

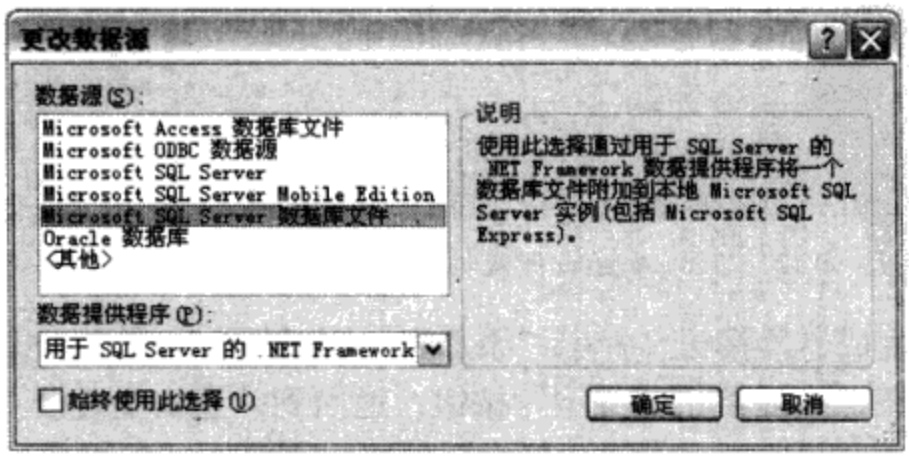


图 30.10 “更改数据源”对话框

(5) 单击“确定”按钮，返回“添加连接”对话框。单击“浏览”按钮，选择数据库文件，单击“确定”按钮，返回“TableAdapter 配置向导”对话框。连续单击“下一步”按钮，出现如图 30.11 所示的“TableAdapter 配置向导——选择命令类型”对话框，在该对话框中，选中“使用 SQL 语句”单选框。

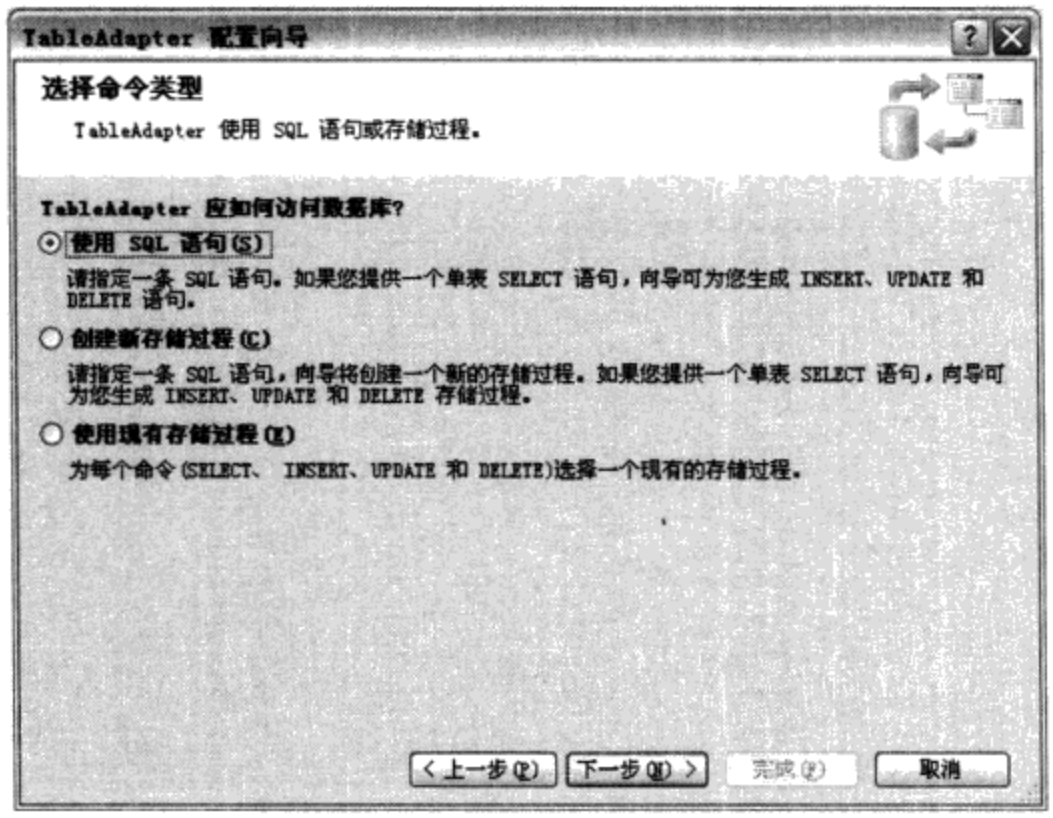


图 30.11 “TableAdapter 配置向导——选择命令类型”对话框

(6) 单击“下一步”按钮，出现如图 30.12 所示的“TableAdapter 配置向导——输入 SQL 语句”对话框。

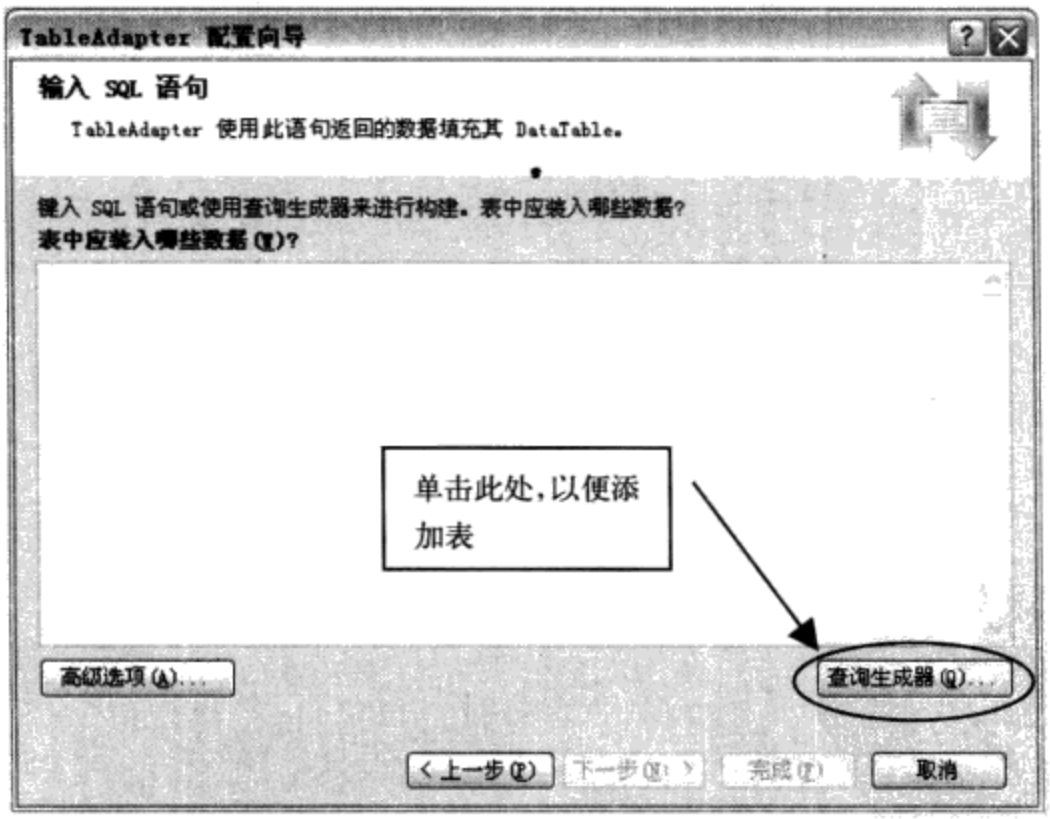


图 30.12 “TableAdapter 配置向导——输入 SQL 语句”对话框

(7) 单击“查询生成器”按钮，弹出“添加表”对话框，如图 30.13 所示。选中要添加的表，单击“添加”按钮，然后单击“关闭”按钮，返回到“查询生成器”对话框，如图 30.14 所示。根据需要选中要生成的列，单击“确定”按钮，再单击“下一步”按钮，直到单击“完成”按钮。

(8) 打开已创建的报表文件，在“字段资源管理器”窗口中，右键单击“数据库字段”，选择“数据库专家”选项，弹出“数据库专家”对话框，如图 30.15 所示。在“可用数据源”中的“ADO.NET 数据集”树目录下选中要添加的表，单击“>”按钮，然后单击“确定”按钮，数据源设置操作即可完成。

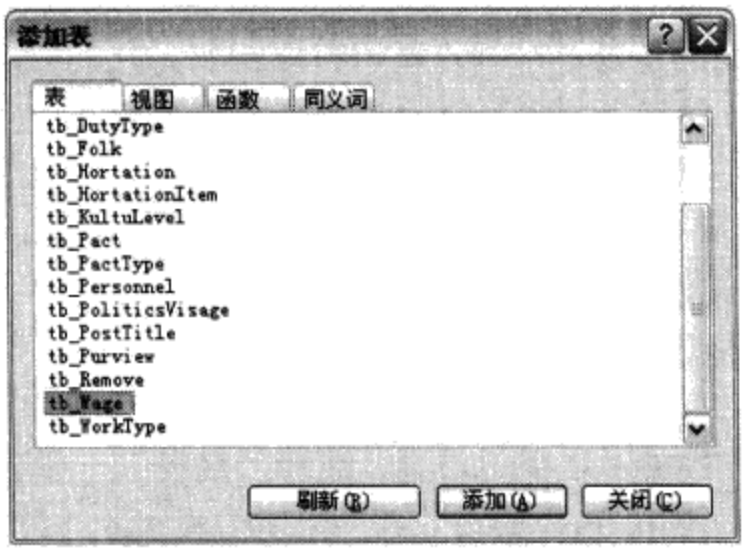


图 30.13 “添加表”对话框

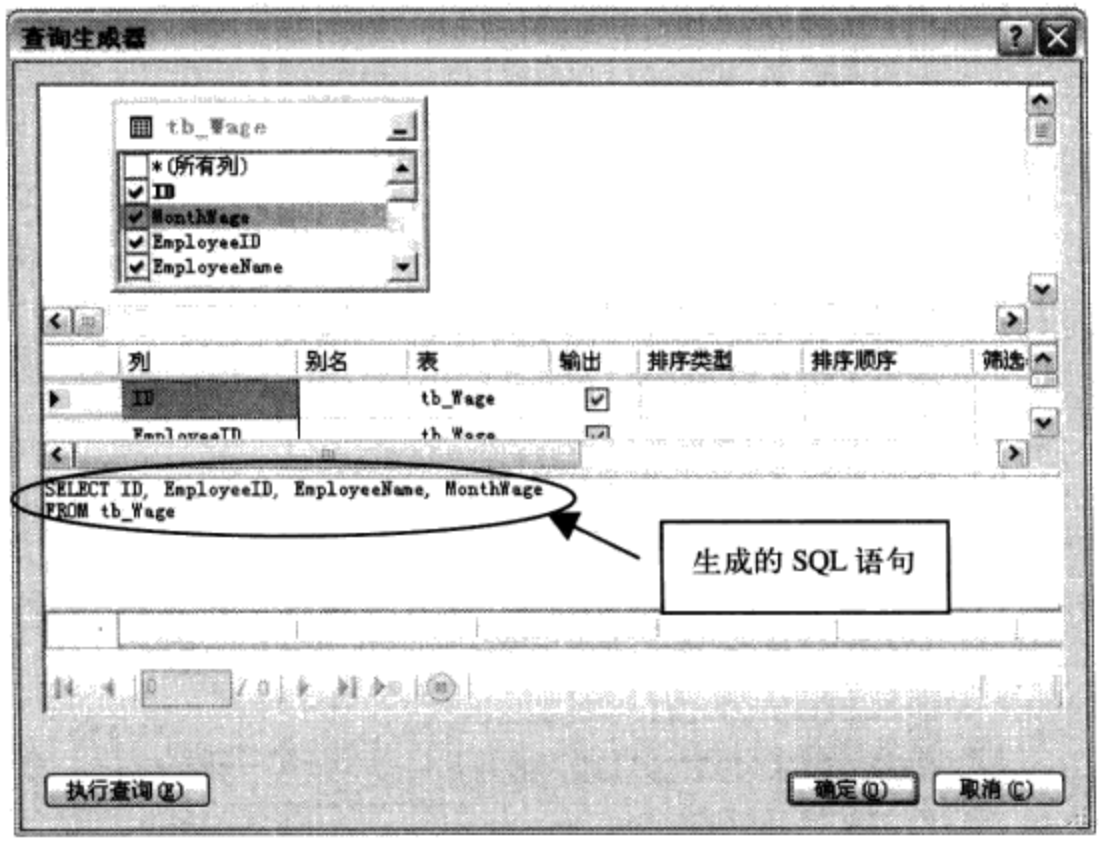


图 30.14 “查询生成器”对话框

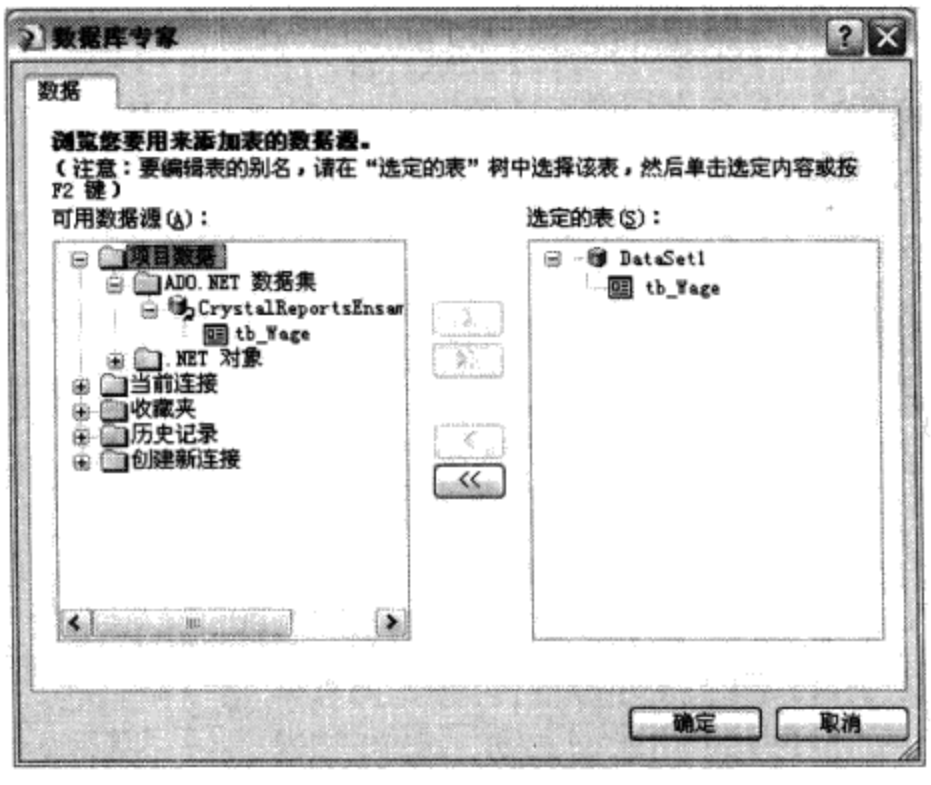


图 30.15 “数据库专家”对话框

2．数据的筛选

默认情况下，数据源中的每条记录都将显示在报表中，但在实际应用中，可能只需要显示其中一些符合条件的记录，这时就需要程序员对数据源中的记录进行筛选了。一般情况下，可以用如下两种方式来筛选记录。

- 利用“选择专家”筛选记录。
- 开发人员自定义公式筛选记录。

下面分别对这两种筛选记录方式进行介绍。

（1）利用“选择专家”筛选记录。

在报表设计环境中的空白处单击鼠标右键，选择“报表”→“选择专家”选项，弹出“选择字段”对话框。在该对话框中，选择要设置条件限制的表字段并单击“确定”按钮，弹出“选择专家”对话框，如图 30.16 所示。

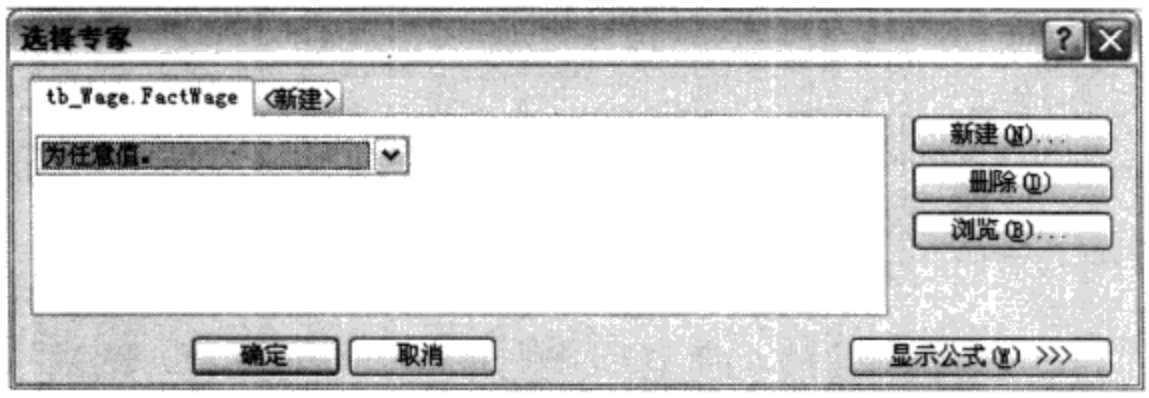


图 30.16 “选择专家”对话框

该对话框的设置简单灵活，对于一些字符型的字段，其过滤条件包括“起始为”字段，如图 30.17 所示。例如，在“起始为”后面的文本框中输入“张”，报表便会显示出“tb_Wagw.EmployeeName”字段中以“张”开头的所有记录。

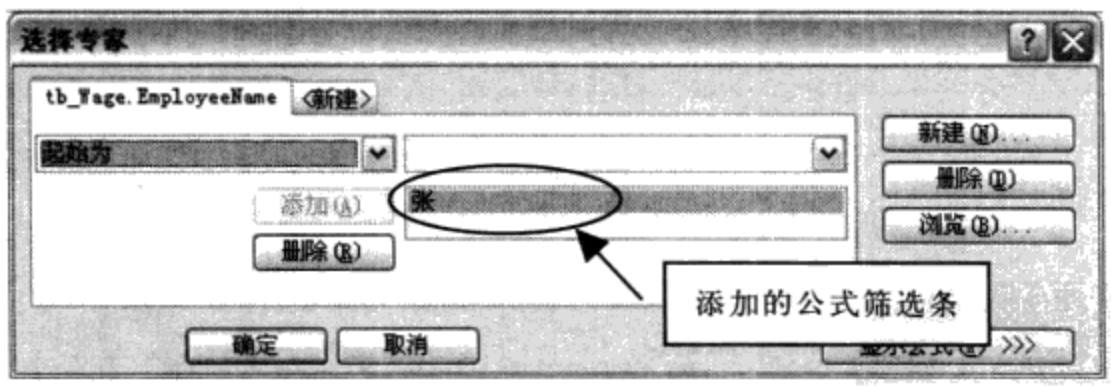


图 30.17 “选择专家”对话框中添加过滤条件

在数据筛选和查询时，对于字符型数据一般是不区分大小写的，但有时可能要求区分大小写，这时可以把报表的 CaseInsensitiveSQLData 属性设置为“False”，如图 30.18 所示。

另外，开发人员还可以对报表中的打印日期进行设置。在报表设计环境中的空白处单击鼠标右键，选择“报表”→“设置打印日期和时间”选项，弹出“设置打印日期和时间”对话框，如图 30.19 所示。在该对话框中，可以设置当前打印的日期和时间，也可以设置打印指定日期和时间的内容。

（2）开发人员自定义公式筛选记录。

在报表设计环境中的空白处单击鼠标右键，选择“报表”→“选定公式”→“记录”选项，弹出“公式工作室——记录选定公式编辑器”对话框，如图 30.20 所示。

图 30.18　设置报表 CaseInsensitiveSQLData 属性

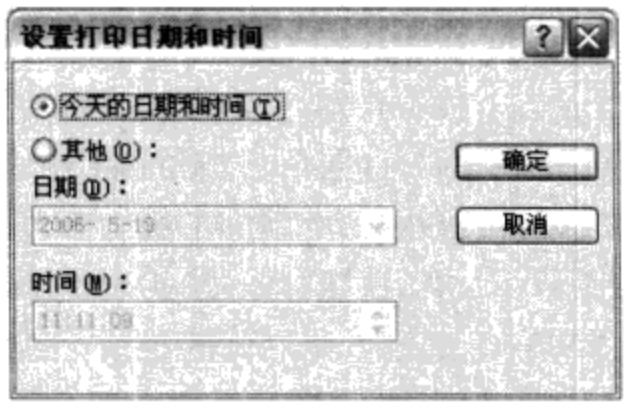


图 30.19　“设置打印日期和时间”对话框

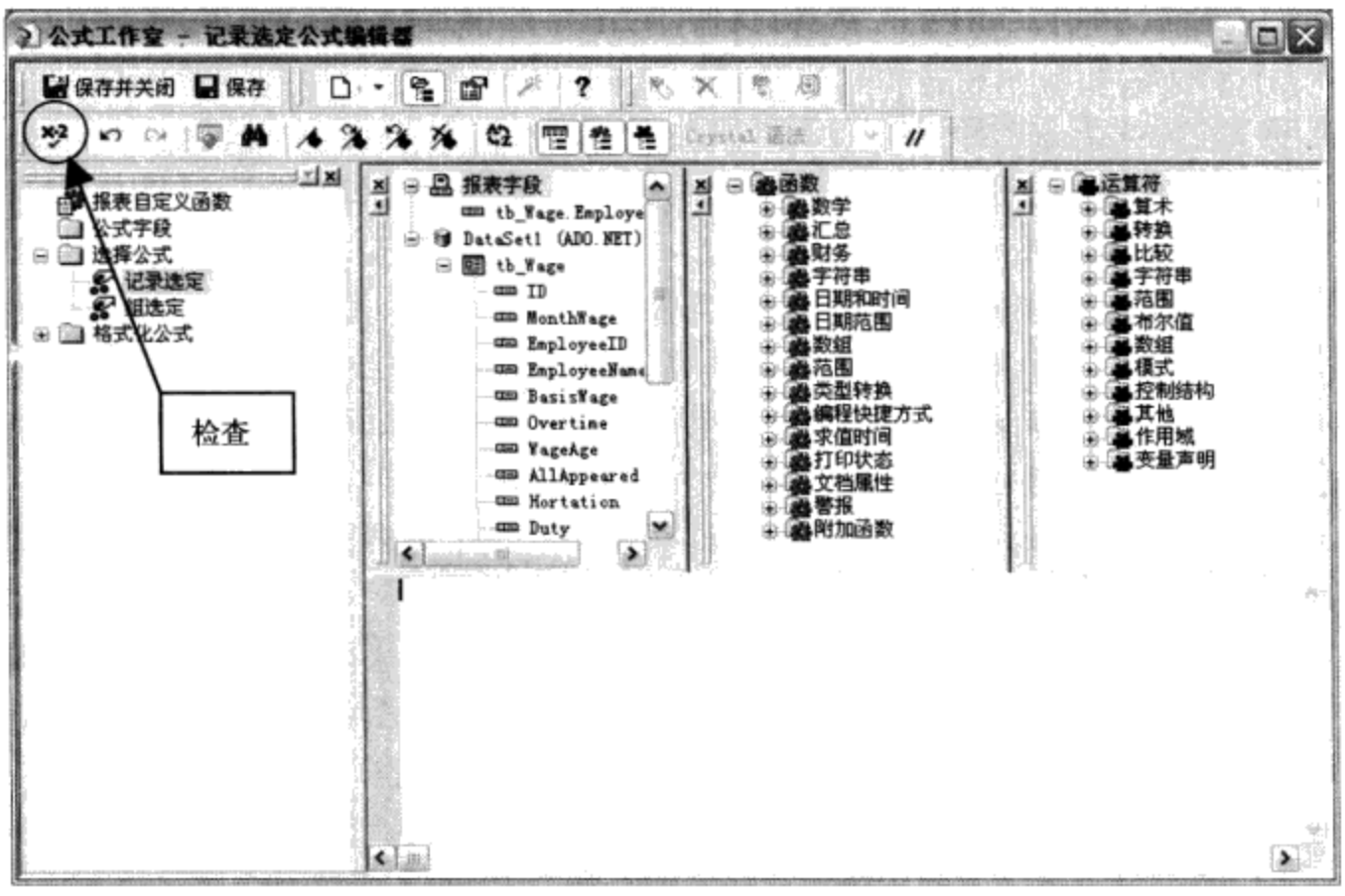


图 30.20　“公式工作室——记录选定公式编辑器”对话框

在此对话框中输入具有限制条件的公式，并单击“检查”按钮，检查公式是否存在错误，在确认无误后单击“保存并关闭”按钮，完成记录的筛选操作。

注意：“公式工作室——记录选定公式编辑器”对话框中输入的公式必须是布尔型的返回结果，即“真”或“假”。

下面列举了一些公式模板，请读者参考。

● 使用数字选择记录

{文件.字段}>10：选择{文件.字段}值大于 10 的记录。

{文件.字段}<10：选择{文件.字段}值小于 10 的记录。

{文件.字段}>10 and {文件.字段}<100：选择{文件.字段}值大于 10，并且小于 100 的记录(不包括 10 和 100)。

{文件.字段}>=10 and {文件.字段}<=100：选择{文件.字段}值大于等于 10，并且小于等于 100 的记录。

● 用字符串选择记录

{文件.字段} startswich "a"：选择{文件.字段}值以字符“a”开始的记录。

not（{文件.字段}）：选择{文件.字段}值不以字符“a”开始的记录。

"530" in {文件.字段}[3 to 5]：选择{文件.字段}值中的第 3 位到第 5 位数字为“530”的记录。

"cp" in {文件.字段}：选择{文件.字段}值包含字符串“cp”的记录。

● 使用日期选择记录

Year（{文件.日期}）<2006：选择{文件.日期}字段中的年份小于 2006 年的记录。

Year（{文件.日期}）>2000 and Year（{文件.日期}）<2006：选择{文件.日期}字段中的年份在 2000 年和 2006 年之间的记录（不包括 2000 年和 2006 年）。

Year（{文件.日期}）>=2000 and Year（{文件.日期}）<=2006：选择{文件.日期}字段中的年份在 2000 年和 2006 年之间的记录（包括 2000 年和 2006 年）。

Month（{文件.日期}）in 1 to 3：选择{文件.日期}字段中月份为一年中前 3 个月的记录（包括一月、二月和三月）。

Month（{文件.日期}）in[1,3]：选择{文件.日期}字段中月份为一年中的第一个月和第 3 个月的记录（不包括二月）。

● 使用预置数据范围选择记录

{文件.日期} in LastFullMonth：选择{文件.日期}字段中的日期在上个月整月范围内的记录（如果本月是 3 月，则选择二月的所有记录）。

not{文件.日期} in LastFullMonth：选择{文件.日期}字段中日期在上个月整月范围之外的记录。（如果本月是 3 月，那么选择除了二月以外日期的所有记录）。

{文件.日期}<CurrentDate：选择{文件.日期}字段中日期在今日之前的所有记录。

● 使用日期/数字/字符组合选择记录

"a" in {文件.字段}[1] and Month（{文件.日期}）in [1,3]：选择{文件.日期}字段值以“a”开头并且月份是一月和 3 月的记录。

"TS" in {文件.编号}[3 to 4] and {文件.数量}>100：选择{文件.编号}字段中的第 3 和第 4 个字符分别为“T”和“S”，并且{文件.数量}字段值大于 100 的记录。

● 为用户创建参数进行筛选

对于报表中数据的筛选，也可以通过传递参数的方法来实现。参数用来提示报表用户输入信息，可以将参数看作报表生成之前需要用户回答的问题，用户输入的信息决定了报表输出的内容。

注意：

① 如果要通过参数提示用户输入内容，并根据用户输入的内容在报表中显示结果，必须将该参数包含在报表中的某个位置或者直接将该参数作为报表中的字段。

② 如果要在记录或组选定公式中使用参数，必须创建参数字段，然后像插入其他字段一样将其输入到公式中。

③ 参数字段必须与所要比较字段的数据类型相同。参数字段的数据类型如表 30.1 所示。

表 30.1　　参数字段的数据类型及说明

数据类型	说　明
布尔值	是/否或真/假
货币	要求一个美元金额
日期	日期格式
日期和时间	日期和时间格式
数字	一个数值
字符串	文本格式
时间	时间格式

为用户创建参数选择列表的步骤如下。

(1) 在“字段资源管理器”窗口中，选中“参数字段”，单击鼠标右键，选择“新建”命令，弹出“创建参数字段”对话框，如图 30.21 所示。

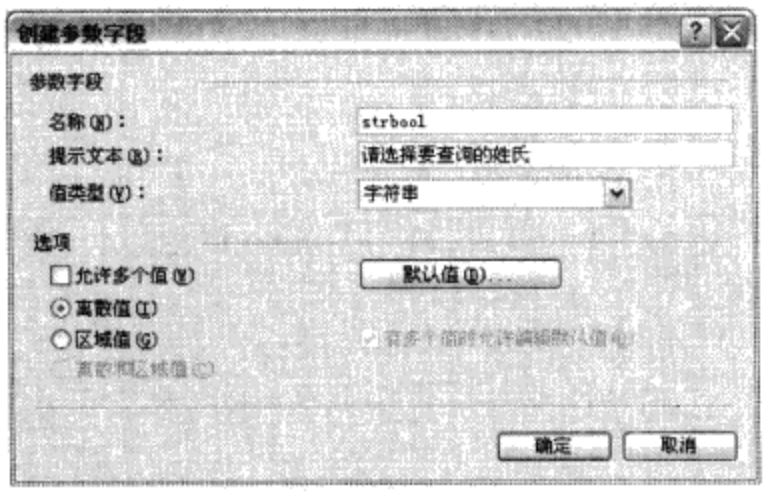


图 30.21 “创建参数字段”对话框

(2) 在“名称”文本框中输入参数的名称，在“提示文本”文本框中输入想要提示的文本，此文本将在刷新报表时出现在“输入参数值”对话框中，在“值类型”下拉框中选择一个值类型。“选项”区中有一个“允许多个值”选项，如果选择该项，则使用者可以输入一个以上的参数值（如查询条件为“张”和“王”的用户，“张”和“王”就是两个参数值）或者从默认值清单中选择一个以上的值。

(3)“选项”区中有一个“离散值”单选按钮，“离散值”即只有一个参数值，如每个参数值只代表一个产品编号或者客户等；如果“选项”区中的“区域值”单选按钮被选中，则要求用户要输入一个值的范围，例如，日期从 2007/1/1 到 2007/4/1 等。

(4) 在“选项”区中单击“默认值”按钮，弹出“设置默认值”对话框，如图 30.22 所示。在“选择或输入值”文本框中输入要添加选项的名称，然后单击“>”按钮。

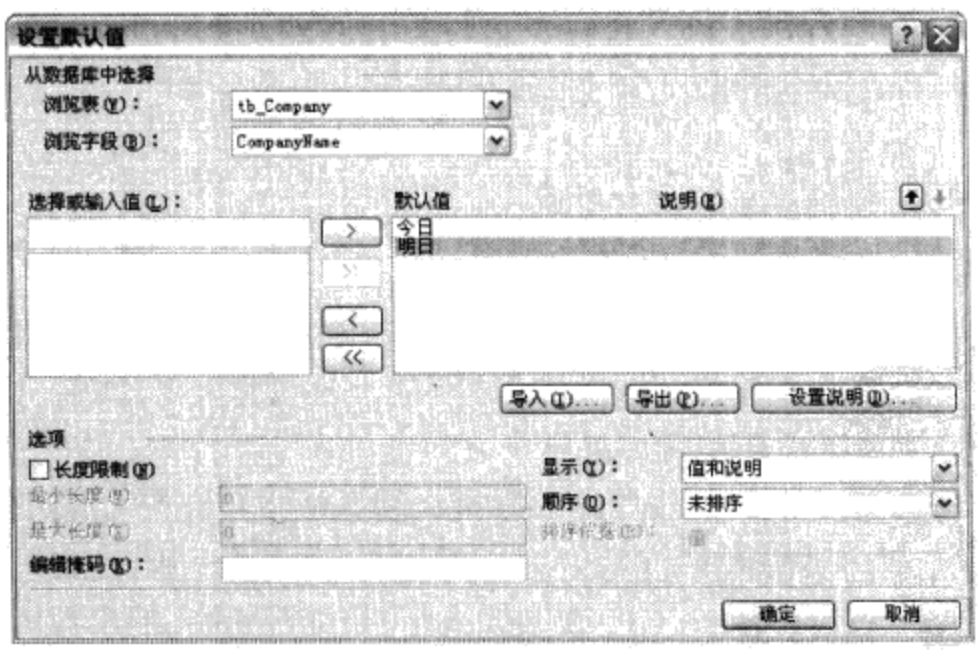


图 30.22 “设置默认值”对话框

如果要修改默认值，在“默认值”文本框中选中要修改的值，单击“<”按钮，在“选择或输入值”文本框内会出现要修改的值，此时可以对其进行修改，然后单击“>”按钮，再将其添加到“默认值”文本框中。

另外，在“创建参数字段”对话框中有一个“有多个值时允许编辑默认值”复选框，其作用是使当前用户既可以输入一个新的参数，也可以选择设定好的默认值。

如果创建了一个“数字、货币或日期”类型的参数，在“选项”区域中会出现一个“范围限制字段”复选框，如图 30.23 所示，用户可以在此设置“最小值”和“最大值”。

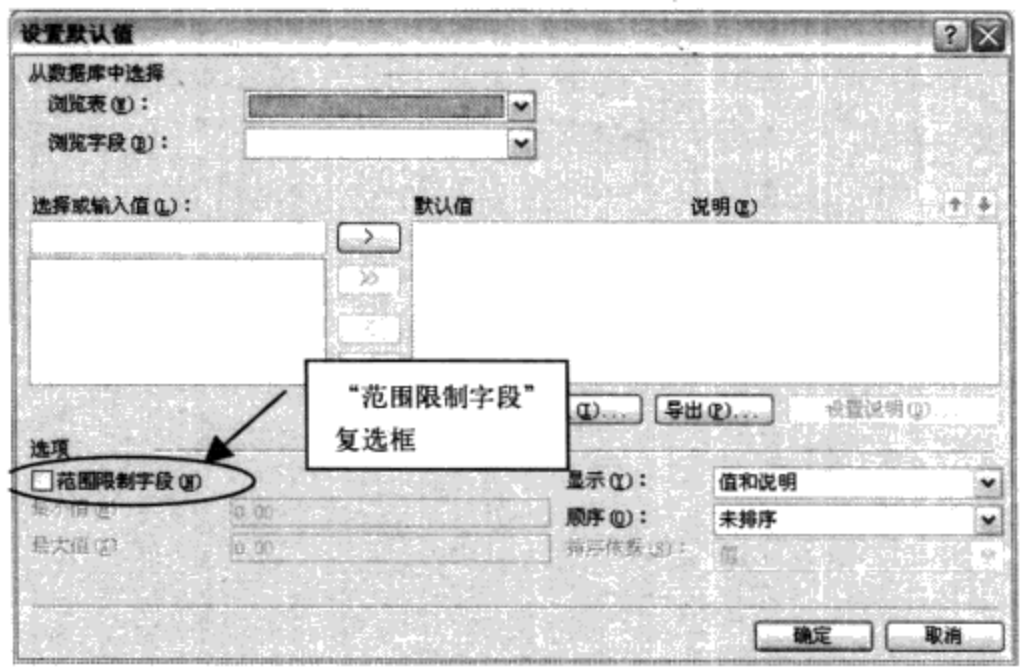


图 30.23 "设置默认值"对话框中的"范围限制字段"复选框

对于要为其设置默认值的字符串参数字段，可以在"编辑掩码"文本框中输入编辑掩码，而不是指定范围。编辑掩码可以是任何一组屏蔽字符，它主要用于限制输入的参数值。

编辑掩码可以是下列任意屏蔽字符，或这些字符的任意组合。

- "A"：允许字母、数字和字符，并要求在参数值中输入字符。
- "a"：允许字母、数字和字符，不要求在参数值中输入字符。
- "0"：允许数字 0～9，并要求在参数值中输入字符。
- "9"：允许数字和空格，不要求在参数值中输入字符。
- "#"：允许数字、空格和加/减号，不要求在参数值中输入字符。
- "L"：允许字母 A～Z，并要求在参数值中输入字符。
- "?"：允许字母，不要求在参数值中输入字符。
- "&"：允许任何字符和空格，并要求在参数值中输入字符。
- "C"：允许任何字符或空格，不要求在参数值中输入字符。
- ".,:;-/"：分隔字符。
- "<"：使随后的字符转换成小写字母。
- ">"：使随后的字符转换成大写字母。
- "\"：使随后的字符显示为字面值。例如，编辑掩码"\A"将显示参数值"A"。
- "Password"：允许将编辑掩码设置成"密码"，以创建条件公式，指定报表的某些窗口只有在输入特定的用户密码后才可见。

一些编辑掩码字符要求在它们的位置输入字符，而另一些允许在需要时保留空格。例如，如果编辑掩码是 000099，由于编辑掩码字符"9"不要求输入字符，因此可输入 4 位数字、5 位数字或 6 位数字的参数值。但是，由于"0"要求输入字符，因此输入的参数值不能少于 4 位数字。

30.3.3 水晶报表的应用

实例位置：光盘\mingrisoft\dxyy\30\01

本节将通过一个实例来介绍如何在 Windows 应用程序中使用水晶报表。实例运行结果如图 30.24 所示。

程序开发步骤如下。

（1）新建一个 Windows 应用程序，命名为 CrystalReportViewerApplication，其主窗体默认为 Form1.cs。

图 30.24　水晶报表的应用

（2）Form1.cs 窗体中，添加一个 ToolTip 控件和一个 CrystalReportViewer 控件，分别用来作为工具栏和显示用户信息报表。

（3）在当前项目中，新建一张 Crystal 报表，命名为 CrystalReport1.rpt，它主要用来显示用户信息。

（4）程序主要代码如下。

运行本示例前，需要对用到的变量进行定义和声明，代码如下：

```
SqlConnection sqlcon;                                   //声明一个数据连接对象
SqlDataAdapter myda;                                    //声明一个更新数据库对象
DataSet myds;                                           //声明一个数据集对象
public string M_str_sql = "server=(local);database=db_30;uid=sa;pwd=;";   //实例化一个连接数据库语句
```

Form1 窗体在运行时，将数据表中的所有记录显示在 CrystalReport1.rpt 报表中，并通过 crystalReportViewer1 控件显示出来。Form1 窗体的 Load 事件代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    sqlcon = new SqlConnection(M_str_sql);                          //实例化一个数据库连接
    myda = new SqlDataAdapter("select * from tb_01", sqlcon);       //执行 SQL 查询语句
    myds = new DataSet();                                           //实例化一个数据集
    myda.Fill(myds, "tb_01");                                       //向数据集中填充数据
    CrystalReport1 CReport = new CrystalReport1();                  //实例化一个报表对象
    CReport.SetDataSource(myds.Tables["tb_01"]);                    //设置报表的数据源
    crystalReportViewer1.ReportSource = CReport;                    //将报表设置为绑定到 CrystalReportViewer 控件
    this.reportViewer1.RefreshReport();                             //更新报表中的数据记录
}
```

单击“查找”按钮，程序使用 ReportDocument 类对象的 Load 方法加载指定报表，并通过设置 DataDefinition 类 RecordSelectionFormula 属性，将指定 SQL 查询语句的执行结果绑定到水晶报表上。“查找”按钮的 Click 事件代码如下：

```
private void toolStripButton1_Click(object sender, EventArgs e)
{
    //实例化一个 SQL 查询语句字符串 P_str_sql
    string P_str_sql = " {tb_01.Sex} like '" + toolStripComboBox1.Text.Trim() + "'";
    ReportDocument reportDocument = new ReportDocument();        //实例化一个报表文件对象
    string P_str_creportPath = Application.StartupPath.Substring(0, Application.StartupPath.Substring(0, Application. Startup
Path.LastIndexOf("\\")).LastIndexOf("\\"));                      //截取水晶报表文件所在的路径
    P_str_creportPath += "\\CrystalReport1.rpt";                 //保存水晶报表文件所在的路径
    reportDocument.Load(P_str_creportPath);                      //向水晶报表文件中加载内容
    reportDocument.DataDefinition.RecordSelectionFormula = P_str_sql;   //设置记录选定的公式内容
    crystalReportViewer1.ReportSource = reportDocument;          //将报表设置绑定到 CrystalReportViewer 控件
}
```

第 31 章

Windows 打印对象

- PrintDocument 类
- PageSetupDialog 类
- PrintDialog 类
- PrintPreviewControl 类
- PrintPreviewDialog 类
- Windows 打印对象的综合应用

31.1 PrintDocument 类

在 Windows 窗体中 PrintDocument 类用于设置一些属性，这些属性说明在基于 Windows 的应用程序中要打印什么内容以及打印文档的能力，常将它与 PrintDialog 类一起使用来控制文档打印的各个方面。

PrintDocument 类是最重要的 Windows 打印对象，它负责建立与其他打印对象的联系。要想建立输出到打印机的输出程序，就要首先创建 PrintDocument 对象。代码如下：

```
PrintDocument pd = new PrintDocument();
```

31.1.1 PrintDocument 类的属性

PrintDocument 类的常用属性如表 31.1 所示。

表 31.1　　PrintDocument 对象常用属性及说明

属　性	说　明
DefaultPageSettings	获取或设置页设置，这些页设置用作要打印的所有页的默认设置
DocumentName	获取或设置打印文档时要显示的文档名（例如，在打印状态对话框或打印机队列中显示）
OriginAtMargins	获取或设置一个值，该值指示与页关联的图形对象的位置是位于用户指定边距内，还是位于该页可打印区域的左上角
PrintController	获取或设置指导打印进程的打印控制器
PrinterSettings	获取或设置对文档进行打印的打印机

下面对 PrintDocument 类比较重要的属性进行详细说明。

（1）DefaultPageSettings 属性。

该属性表示获取或设置页设置，这些页设置用于要打印的所有页的默认设置。

语法：

```
public PageSettings DefaultPageSettings { get; set; }
```

属性值：表示为文档指定默认页设置的 PageSettings。

例如，将页面的文档方向设置为横向，并打印该文档。在代码中做了一些假定：将名为 filePath 的变量设置为要打印的路径，已定义的 pd_PrintPage 方法用来处理 PrintPage 事件，将名为 printer 的变量设置为打印机的名称。代码如下：

```
public void Printing()
{
    try
    {
        streamToPrint = new StreamReader (filePath);                    //实例化一个 StreamReader 对象
        try
        {
            printFont = new Font("Arial", 10);                          //实例化一个字体格式对象
            PrintDocument Document = new PrintDocument();               //实例化一个 PrintDocument 对象
            //向 PrintDocument 的 PrintPage 事件中添加处理事件
            Document.PrintPage += new PrintPageEventHandler(pd_PrintPage);
            Document.PrinterSettings.PrinterName = printer;             //设置打印文档的打印机
            Document.DefaultPageSettings.Landscape = true;              //设置文档打印方向为横向打印
            Document.Print();                                           //打印指定的文档
        }
        finally
        {
            streamToPrint.Close() ;                                     //关闭文件读取流
        }
    }
    catch(Exception ex)
    {
```

```
        MessageBox.Show(ex.Message);                    //显示异常信息
    }
}
```

（2）DocumentName 属性。

该属性表示获取或设置打印文档时要显示的文档名（例如，在打印状态对话框或打印机队列中显示）。

语法：

```
public string DocumentName { get; set; }
```

属性值：表示打印文档时要显示的文档名。默认情况下为“文档”。

例如，打印文档并彩印文档的第一页，同时将 DocumentName 属性设置为用户友好名称。在使用代码前，需要进行一些预定义：已定义名为 printDoc 的 PrintDocument 变量，同时处理 PrintPage 和 QueryPageSettings 事件。代码如下：

```
private void MyButtonPrint_OnClick(object sender, System.EventArgs e)
{
    printDoc.PrinterSettings.PrinterName = "\\mynetworkprinter";    //设置打印机的名称
    if (printDoc.PrinterSettings.IsValid)                           //当有效的打印机被指定后
    {
        if (printDoc.PrinterSettings.SupportsColor)                 //当打印机支持彩印时
        {
            printDoc.DefaultPageSettings.Color = false;             //设置当前页不进行彩印
        }
        printDoc.DocumentName = "试印页";                           //设置打印文档的名称
        currentPageNumber = 1;                                      //设置当前页所处的页数
        printDoc.Print();                                           //对设置好的文档进行打印作业
    }
    else                                                            //当没有指定有效的打印机时
    {
        MessageBox.Show("打印机不可用","信息提示");                 //弹出信息提示
    }
}
private void MyPrintQueryPageSettingsEvent(object sender, QueryPageSettingsEventArgs e)
{
    if (printDoc.PrinterSettings.SupportsColor)                     //当打印机支持彩印时
    {
        if (currentPageNumber == 1 )                                //当变量 currentPageNumber 的值为 1 时
        {
            e.PageSettings.Color = true;                            //设置彩印当前页
        }
    }
}
```

（3）PrintController 属性。

该属性表示获取或设置指导打印进程的打印控制器。

语法：

```
public PrintController PrintController { get; set; }
```

属性值：表示指导打印进程的 PrintController。默认情况下为 PrintControllerWithStatusDialog 类的一个新实例。

例如，先创建一个 PrintDocument 类的实例，将它命名为 myPrintDocument。创建一个 PrintController 类的新实例，将该实例分配给 myPrintDocument 的 PrintController 属性，然后打印该文档。代码如下：

```
public void myPrint()
{
    if (useMyPrintController == true)                   //当变量 useMyPrintController 为 true 时
    {
        //设置 PrintDocument 对象的 PrintController 属性
        myPrintDocument.PrintController =new myControllerImplementation();
        if (wantsStatusDialog == true)                  //当变量 wantsStatusDialog 为 true 时
```

```
            {
                myPrintDocument.PrintController =new
PrintControllerWithStatusDialog(myPrintDocument.PrintController);   //设置 PrintDocument 对象的 PrintController 属性
            }
        }
        myPrintDocument.Print();                                    //调用打印作业
    }
```

（4）PrinterSettings 属性。

该属性表示获取或设置一个值，该值指示对文档进行打印的打印机。

语法：

```
public PrinterSettings PrinterSettings { get; set; }
```

属性值：PrinterSettings，它指定在何处以及如何打印文档。默认值为 PrinterSettings。

31.1.2 PrintDocument 类的方法

下面对 PrintDocument 类比较重要的 Print 方法进行说明，该方法是最重要也是最常用的方法，它将开始打印进程，并触发 PrintPage 事件。

语法：

```
public void Print ()
```

示例 31-01 打印指定路径下的文本文件

下面通过控制台的形式将指定路径下的文件进行打印输出。代码如下：

代码 31-01 光盘位置：光盘\mingrisoft\sl\31\01

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Printing;
using System.Windows.Forms;
public class PrintingExample
 {
     private Font printFont;                                    //实例化一个字体样式对象
     private StreamReader streamToPrint;                        //实例化一个流读取对象
     static string filePath;                                    //实例化一个用于保存文件路径的字符串
     public PrintingExample()
     {
        Printing();                                             //调用打印方法
     }
        private void pd_PrintPage(object sender, PrintPageEventArgs ev)
        {
            float linesPerPage = 0;                             //实例化一个单精度型变量用来保存每页的行数
            float yPos = 0;                                     //实例化一个单精度型变量用来保存打印文本的纵坐标
            int count = 0;                                      //实例化一个整型变量用来保存打印的行数
            float leftMargin = ev.MarginBounds.Left;            //保存边距以内的页面部分的 x 坐标
            float topMargin = ev.MarginBounds.Top;              //保存边距以内的页面部分的 y 坐标
            String line = null;                                 //实例化一个字符串实例
            linesPerPage = ev.MarginBounds.Height / printFont.GetHeight(ev.Graphics); //保存每页的行数
            while (count < linesPerPage && ((line = streamToPrint.ReadLine()) != null)) //当页面中存在内容时
            {
                yPos = topMargin + (count * printFont.GetHeight(ev.Graphics));          //递增变量 yPos 的值
                //绘制字符串
                ev.Graphics.DrawString(line, printFont, Brushes.Black, leftMargin, yPos, new StringFormat());
                count++;                                        //递增变量 count 的值
            }
            if (line != null)                                   //当 line 的值不为 null 时
                ev.HasMorePages = true;                         //设置打印附加页
            else                                                //当 line 的值为 null 时
                ev.HasMorePages = false;                        //设置不打印附加页
        }
    public void Printing()
```

```
    {
        try
        {
            streamToPrint = new StreamReader (filePath);      //实例化一个字节流读取对象
            try
            {
                printFont = new Font("Arial", 10);            //实例化一个字体类对象
                PrintDocument pd = new PrintDocument();       //实例化一个 PrintDocument 对象
                //向 PrintDocument 对象的 PrintPage 事件中添加处理事件
                pd.PrintPage += new PrintPageEventHandler(pd_PrintPage);
                pd.Print();                                   //处理打印作业
            }
            finally
            {
                streamToPrint.Close() ;                       //关闭字节流读取对象
            }
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message);                      //显示异常信息
        }
    }
    public static void Main(string[] args)
    {
        string sampleName = Environment.GetCommandLineArgs()[0];      //获取命令行参数
        if(args.Length != 1)                                          //当 args 中不存在内容时
        {
            Console.WriteLine("Usage: " + sampleName +" <file path>");  //弹出提示信息
            return;                                                   //返回
        }
        filePath = args[0];              //实例化一个文件路径字符串
        new PrintingExample();           //打印指定路径下的文件
    }
}
```

说明：本实例的运行是通过命令行的方式进行的，首先定位到当前可执行文件的路径，然后指定打印文件。例如，本实例打印根目录下的指定文件“ABC.doc”，文件的位置可以是任意的。如果所要打印的文件没有在根目录下，那么写的时候应该写它的绝对路径。本实例的命令行代码为“E:\练习区\ConsolePractiseApplication\ConsolePractiseApplication\bin\Debug>ConsolePractiseA pplication.exe ABC.doc”。

指定打印输出的方法是处理 PrintPage 事件并使用 PrintPageEventArgs 参数中包含的 Graphics 对象。使用 PrinterSettings.PrinterName 属性可指定用哪一台打印机来打印文档。Print 方法在打印文档时不使用打印对话框。若要为用户提供选择打印设置的对话框，则使用 PrintDialog 类。

31.1.3 PrintDocument 类的事件

下面介绍 PrintDocument 类的常用事件，如表 31.2 所示。

表 31.2　　PrintDocument 对象常用事件及说明

事　件	说　明
BeginPrint	在调用 Print 方法时并且在打印文档的第一页之前发生
Dispose	添加事件处理程序以侦听类上的 Disposed 事件
EndPrint	打印完文档的最后一页时发生
PrintPage	当需要为当前页打印输出时发生
QueryPageSettings	在 PrintPage 事件的前面发生

下面对 PrintDocument 类比较重要的事件进行详细介绍。

（1）BeginPrint 事件。

该事件表示在调用 Print 方法时并且在打印文档的第一页之前发生。

语法：

```
public event PrintEventHandler BeginPrint
```

（2）EndPrint 事件。

该事件表示在打印完文档的最后一页时发生。

语法：

```
public event PrintEventHandler EndPrint
```

（3）PrintPage 事件。

该事件表示当需要为当前页打印输出时发生。

语法：

```
public event PrintPageEventHandler PrintPage
```

这些事件中最常用的是 PrintPage 事件，该事件应当与 PrintPageEventHandle 参数委托的签名匹配。例如：

```
Public delegate void PrintPageHandler(object sender,PrintPageEventArgs e);
```

参数说明如下。

sender：表示事件源。

e：表示包含事件数据的 PrintPageEventArgs 对象。

PrintPageEventArgs 参数为打印提供必要的系统数据，相关参数及说明如表 31.3 所示。

表 31.3　　PrintPageEventArgs 参数列表及说明

参　数	说　明
Cancel	布尔值，设置为 True 则取消打印
Graphics	用于写到打印机的 Graphics 对象
HasMorePages	布尔值，指示是否还有待打印的页面。默认为 False
MarginBounds	表示整个页面的全部区域
PageBounds	表示整个页面的矩形区域
PageSettings	待打印页面的设置

Graphics 参数是 PrintPageEventArgs 参数的成员，包含一个 Graphics 对象用来绘制将要打印的文字和图像。

语法：

```
public Graphics Graphics { get; }
```

例如，打印指定路径下的图像文件。本实例假定已经在 Form 上创建了名为 printButton 的 Button 和名为 pd 的 PrintDocument。确保此实例中 Button 的 Click 事件与 printButton_Click 方法关联，PrintDocument 的 PrintPage 事件与 pd_PrintPage 关联。实例中的 printButton_Click 方法调用引发 PrintPage 事件的 Print 方法，并打印在 pd_PrintPage 方法中指定的.bmp 文件。需要注意的是：当运行本实例时，请更改要打印的位图的路径。代码如下：

```
private void printButton_Click(object sender, EventArgs e)
{
    try
    {
        pd.Print();          //调用 PrintDocument 对象的打印方法，同时触发 PrintPage 事件
    }
    catch (Exception ex)                                          //捕获异常
    {
        MessageBox.Show("在打印时出现一个错误", ex.ToString());      //显示错误信息
    }
}
private void pd_PrintPage(object sender, PrintPageEventArgs ev)
```

```
{
    //绘制图片
    ev.Graphics.DrawImage(Image.FromFile("C:\\My Folder\\MyFile.bmp"), ev.Graphics.VisibleClipBounds);
    ev.HasMorePages = false;                                    //设置当前打印页为最后一页
}
```

31.2 PageSetupDialog 类

在 Windows 窗体中，PageSetupDialog 类是一个预先配置的页面对话框，用于在基于 Windows 的应用程序中设置页面信息，便于打印。通过对该类的属性进行设置。可以使用户更改与页面相关的打印设置，包括纸张大小、纸张方向、页边距和选择打印机等。

PageSetupDialog 类具有与单页（PrintDocument 类）或任何文档（PageSettings 类）相关的属性。此外，PageSetupDialog 类可用于确定特定的打印机设置，这些设置存储在 PrinterSettings 类中。可以用 ShowDialog 方法将对话框显示出来，从而对给定 Document 的 PageSettings 和 PrinterSettings 信息进行修改。用户可启用该对话框的各个部分以控制打印和边距，控制纸张方向、大小和来源，还可以显示“帮助”和网络按钮。

31.2.1 PageSetupDialog 类的属性

PageSetupDialog 类的属性及说明如表 31.4 所示。

表 31.4　　PageSetupDialog 对象常用属性及说明

属　性	说　明
AllowMargins	获取或设置一个值，该值指示是否启用对话框的边距部分
AllowOrientation	获取或设置一个值，该值指示是否启用对话框的方向部分（横向或纵向）
AllowPaper	获取或设置一个值，该值指示是否启用对话框的纸张部分（纸张大小和纸张来源）
AllowPrinter	获取或设置一个值，该值指示是否启用“打印机”按钮
Container	获取 Icontainer 接口，它包含 Component 类
Document	获取或设置一个值，指示从中获取页面设置的 PrintDocument 类对象
EnableMetric	获取或设置一个值，该值指示以毫米为单位显示边距设置时，是否自动将边距设置转换为以百分之一英寸为单位（或相反）
MinMargins	获取或设置一个值，该值指示允许用户选择的最小边距（以百分之一英寸为单位）
PageSettings	获取或设置一个值，该值指示要修改的页设置
PrinterSettings	获取或设置用户单击对话框中“打印机”按钮时修改的打印机设置
ShowHelp	获取或设置一个值，该值指示“帮助”按钮是否可见
ShowNetwork	获取或设置一个值，该值指示“网络”按钮是否可见
Site	获取或设置 Component 类的 ISite 接口
Tag	获取或设置一个对象，该对象包含控件的数据

下面对 PageSetupDialog 类比较重要的属性进行详细说明。

（1）AllowOrientation 属性。

该属性表示获取或设置一个值，该值指示是否启用对话框的方向部分（横向或纵向）。

语法：

```
public bool AllowOrientation { get; set; }
```

属性值：如果启用了对话框的方向部分，则为 true；否则为 false。默认为 true。

例如，设置当前打印文档为横向打印。代码如下：

```
this.pageSetupDialog1.AllowOrientation=true;
```

（2）Document 属性。

该属性用于获取或设置一个值，该值指示在设置 PageSetupDialog 时所对应的 PrintDocument 对象。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：从中获得页面设置的 PrintDocument。默认值为空引用。

例如，设置打印预览的文档对象，假定已经存在一个 PrintDocument 对象 printDocument1。代码如下：

```
this.pageSetupDialog1.Document=this.printDocument1;
```

31.2.2 PageSetupDialog 类的方法

PageSetupDialog 类的方法及说明如表 31.5 所示。

表 31.5　　PageSetupDialog 对象常用方法及说明

方　法	说　明
Reset	将所有选项重新设置为其默认值
ShowDialog	已重载。运行通用对话框
ToString	返回包含类的名称的字符串（如果有）。不应重写此方法

下面对 PageSetupDialog 类比较重要的 ShowDialog 方法进行说明。

该方法用于运行通用对话框。显示设置页详细设计的对话框。

语法：

```
public DialogResult ShowDialog ()
```

例如，显示当前的打印设置窗体。代码如下：

```
this.pageSetupDialog1.ShowDialog();
```

31.3 PrintDialog 类

在 Windows 窗体的众多类中，PrintDialog 类是一个预先配置的对话框，可用于在 Windows 应用程序中选择打印机、选择要打印的页以及确定其他与打印相关的设置。该控件用作选择打印机和打印相关设置，可全部打印、打印指定的页范围或打印选定内容。

从应用程序的代码内调用 ShowDialog 方法，可以显示打印对话框。在显示该类后，用户将与之进行交互，设置打印作业的属性。这些项保存在与该打印作业关联的 PrinterSettings 类和 PageSettings 类中，可以调用它们所设置的属性以确定打印作业的细节。

31.3.1 PrintDialog 类的属性

PrintDialog 的属性及说明如表 31.6 所示。

表 31.6　　PrintDialog 对象常用属性及说明

属　性	说　明
AllowCurrentPage	获取或设置一个值，该值指示是否显示“当前页”选项按钮
AllowPrintToFile	获取或设置一个值，该值指示是否启用“打印到文件”复选框
AllowSelection	获取或设置一个值，该值指示是否启用“选择”选项按钮
AllowSomePages	获取或设置一个值，该值指示是否启用“页”选项按钮
Document	获取或设置一个值，指示用于获取 PrinterSettings 类的 PrintDocument 对象
PrinterSettings	获取或设置对话框修改的打印机设置
PrintToFile	获取或设置一个值，该值指示是否选中“打印到文件”复选框
ShowHelp	获取或设置一个值，该值指示是否显示“帮助”按钮
ShowNetwork	获取或设置一个值，该值指示是否显示“网络”按钮
Site	获取或设置 Component 类的 ISite 接口
Tag	获取或设置一个对象，该对象包含控件的数据

下面对 PrintDialog 类比较重要的 Document 属性进行说明。

该属性用于获取或设置一个值，指示用于获取 PrinterSettings 类的 PrintDocument 对象。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：用于获取 PrinterSettings 的 PrintDocument，默认值为空引用。

例如，下面的实例演示如何使用 PrintDialog 类的对象来设置 AllowSomePages 属性、ShowHelp 属性和 Document 属性。要运行此实例，请将下面的代码粘贴到一个窗体中，该窗体包含一个名为 Button1 的按钮。此实例要求该按钮的 Click 事件和 docToPrint 的 PrintPage 事件已连接到此实例中定义的事件处理方法。代码如下：

```
//实例化一个 PrintDocument 对象
private System.Drawing.Printing.PrintDocument docToPrint =new System.Drawing.Printing.PrintDocument();
//实例化一个 PrintDialog 对象
private System.Drawing.Printing PrintDialog1=new System.Drawing.Printing.PrintDialog();
private void Button1_Click(System.Object sender,System.EventArgs e)
{
    PrintDialog1.AllowSomePages = true;                //启用页选项按钮
    PrintDialog1.ShowHelp = true;                      //启用帮助按钮
    PrintDialog1.Document = docToPrint;                //设置打印文档
    DialogResult result = PrintDialog1.ShowDialog();   //显示打印对话框
    if (result == DialogResult.OK)                     //当单击“打印”按钮时
    {
        docToPrint.Print();                            //调用 PrintDocument 对象的打印方法
    }
}
private void document_PrintPage(object sender,System.Drawing.Printing.PrintPageEventArgs e)
{
    string text = "明日科技有限公司";                   //设置打印的内容
    //设置打印内容的字体
    System.Drawing.Font printFont = new System.Drawing.Font("Arial", 35, System.Drawing.FontStyle.Regular);
    e.Graphics.DrawString(text, printFont,System.Drawing.Brushes.Black, 10, 10);        //绘制打印的内容
}
```

31.3.2 PrintDialog 类的方法

PrintDialog 的方法及说明如表 31.7 所示。

表 31.7　　PrintDialog 对象常用方法及说明

方　法	说　明
Reset	将所有选项、最后选定的打印机和页面重新设置为其默认值
ShowDialog	已重载。运行通用对话框
ToString	返回包含类的名称的字符串（如果有）。不应重写此方法

PrintDialog 类中最常用的方法是 ShowDialog 方法，ShowDialog 方法用来显示与打印机相关设置的对话框。该方法常与 Document 属性一起使用。

语法：

```
public DIALOGRESULT ShowDialog ()
```

属性值：如果用户在对话框中单击“确定”，则返回值为 DialogResult.OK，否则返回值为 Dialog Result.Cancel。

例如，显示打印窗体。代码如下：

```
this.printDialog.ShowDialog();
```

31.4 PrintPreviewControl 类

PrintPreviewControl 类表示 Windows 窗体应用程序打印预览的原始预览部分，没有任何对

话框或按钮。大多数 PrintPreviewControl 对象都可在 PrintPreviewDialog 对象上找到，但这不是必需的。

31.4.1 PrintPreviewControl 类的属性

PrintPreviewControl 类的常用属性如表 31.8 所示。

表 31.8　　PrintPreviewControl 对象常用属性及说明

属　性	说　明
AccessibleDescription	获取或设置辅助功能客户端应用程序使用的控件说明
AccessibleName	获取或设置辅助功能客户端应用程序所使用的控件名称
AccessibleRole	获取或设置控件的辅助性角色
AllowDrop	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
Anchor	获取或设置控件绑定到的容器的边缘并确定控件如何随其父级一起调整大小
AutoScrollOffset	获取或设置一个值，该值指示在 ScrollControlIntoView 方法中将控件滚动到何处
AutoZoom	获取或设置一个值，该值指示在调整控件的大小或更改显示的页数时是否自动调整 Zoom 属性
BindingContext	获取或设置控件的 BindingContext 属性
Bottom	获取控件下边缘与其容器的工作区上边缘之间的距离（以像素为单位）
Bounds	获取或设置控件（包括其非工作区元素）相对于其父控件的大小和位置（以像素为单位）
Capture	获取或设置一个值，该值指示控件是否已捕获鼠标
CausesValidation	获取或设置一个值，该值指示控件是否会引起在任何需要在接收焦点时执行验证的控件上执行验证
CheckForIllegalCrossThreadCalls	获取或设置一个值，该值指示是否捕获对错误线程的调用，这些调用访问控件的 Handle 属性
Columns	获取或设置屏幕横向显示的页数
ContainsFocus	获取一个值，该值指示控件或它的一个子控件当前是否有输入焦点
ContextMenu	获取或设置与控件关联的快捷菜单
ContextMenuStrip	获取或设置与此控件关联的 ContextMenuStrip 属性
Controls	获取包含在控件内的控件的集合
Created	获取一个值，该值指示控件是否已经创建
Cursor	获取或设置当鼠标指针位于控件上时显示的光标
DataBindings	为该控件获取数据绑定
DefaultBackColor	获取控件的默认背景色
DefaultFont	获取控件的默认字体
DefaultForeColor	获取控件的默认前景色
DisplayRectangle	获取表示控件的显示区域的矩形
Dock	获取或设置哪些控件边框停靠到其父控件并确定控件如何随其父级一起调整大小
Document	获取或设置一个值，该值指示要预览的文档
Enabled	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
Focused	获取一个值，该值指示控件是否有输入焦点
Font	获取或设置控件显示的文字的字体
ForeColor	获取或设置控件的前景色
Handle	获取控件绑定到的窗口句柄
Height	获取或设置控件的高度
InvokeRequired	获取一个值，该值指示调用方在对控件进行方法调用时是否必须调用 Invoke 方法，因为调用方位于创建控件所在的线程以外的线程中

续表

属　性	说　明
Location	获取或设置该控件的左上角相对于其容器的左上角的坐标
Name	获取或设置控件的名称
Parent	获取或设置控件的父容器
Rows	获取或设置屏幕纵向显示的页数
StartPage	获取或设置页面左上角的页码
UseAntiAlias	获取或设置一个值，该值指示打印是否使用操作系统的防锯齿功能
Visible	获取或设置一个值，该值指示是否显示该控件
Width	获取或设置控件的宽度
Zoom	获取或设置一个值，该值指示页面的显示大小

下面对 PrintPreviewControl 类比较重要的属性进行详细介绍。

(1) Document 属性。

该属性用于获取或设置一个值，该值指示要预览的文档。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：PrintDocument，它表示要预览的文档。

(2) UseAntiAlias 属性。

该属性用于获取或设置一个值，该值指示打印是否使用操作系统的防锯齿功能。

语法：

```
public bool UseAntiAlias { get; set; }
```

属性值：如果使用防锯齿功能，则为 true，否则为 false。

注意：UseAntiAlias 设为 true 可以消除锯齿，提高打印质量；但会损耗性能，使显示速度变慢。

(3) Zoom 属性。

该属性用于获取或设置一个值，该值指示页面的显示大小。

语法：

```
public double Zoom { get; set; }
```

属性值：指示页面显示大小的值。若属性值为 1.0 则指示实际大小。

ZOOM 属性可以设置预览页面为实际大小的比例，例如：

```
printPreviewControl1.Zoom = 0.75;
```

例如，下面的实例演示 PrintPreviewControl 的 Document 属性、UseAntiAlias 属性和 Zoom 属性。若要运行此实例，请将下面的代码放到一个窗体中，并调用位于该窗体的构造函数或 Load 事件处理方法中的 InitializePrintPreviewControl 方法。代码如下：

```
internal PrintPreviewControl PrintPreviewControl1;                    //定义一个 PrintPreviewControl 对象
//实例化一个 PrintDocument 对象
private System.Drawing.Printing.PrintDocument docToPrint =new System.Drawing.Printing.PrintDocument();
private void InitializePrintPreviewControl()
{
    this.PrintPreviewControl1 = new PrintPreviewControl();     //实例化 PrintPreviewControl1 对象
    this.PrintPreviewControl1.Location = new Point(88, 80);    //设置 PrintPreviewControl1 对象的位置
    this.PrintPreviewControl1.Name = "PrintPreviewControl1"; //设置 PrintPreviewControl1 对象的名称
    this.PrintPreviewControl1.Dock = DockStyle.Fill;           //设置 PrintPreviewControl1 对象的填充格式
    this.PrintPreviewControl1.Document = docToPrint;           //设置进行预览的文档
    this.PrintPreviewControl1.Zoom = 0.25;                     //设置页面的大小
    this.PrintPreviewControl1.Document.DocumentName = "c:\\someFile"; //设置打印文档时要显示的文档名
    this.PrintPreviewControl1.UseAntiAlias = true;             //设置打印文档时使用系统的防锯齿功能
    this.Controls.Add(this.PrintPreviewControl1);              //向当前窗体中添加 PrintPreviewControl 对象
```

```
        //向 PrintDocument 对象的 PrintPage 事件中追加处理事件
        this.docToPrint.PrintPage +=new System.Drawing.Printing.PrintPageEventHandler(docToPrint_PrintPage);
    }
    private void docToPrint_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
    {
        string text = "明日编程词典";                                          //设置打印的内容
        System.Drawing.Font printFont = new Font("Arial", 35, FontStyle.Regular);   //设置打印内容的字体格式

        e.Graphics.DrawString(text, printFont, Brushes.Black, 10, 10);          //绘制将打印的内容
    }
```

31.4.2 PrintPreviewControl 类的方法

PrintPreviewControl 类的常用方法如表 31.9 所示。

表 31.9　　PrintPreviewControl 对象常用方法及说明

方　法	说　明
CreateGraphics	为控件创建 Graphics 类
FindForm	检索控件所在的窗体
Focus	为控件设置输入焦点
FromHandle	返回当前与指定句柄关联的控件
Hide	对用户隐藏控件
GetType	获取当前实例的类型
Refresh	强制控件使其工作区无效并立即重绘自己和任何子控件
Select	已重载。激活控件
Show	向用户显示控件
ToString	返回包含类的名称的字符串。不应重写此方法

下面对 PrintPreviewControl 类比较重要的 Show 方法进行介绍，该方法表示向用户显示控件。

语法：

```
public void Show ()
```

说明：显示控件等效于将 Visible 属性设置为 true。在调用 Show 方法之后，只要不调用 Hide 方法，Visible 属性就返回 true 值。

例如，显示打印预览窗体。代码如下：

```
this.printPreviewControl1.Show();
```

31.5　PrintPreviewDialog 类

PrintPreviewDialog 类表示包含用于从 Windows 窗体应用程序中进行打印的 PrintPreview Control 的对话框窗体。与 PrintPreviewControl 类不同的是，PrintPreviewDialog 类对一些属性进行了默认的设置。

31.5.1 PrintPreviewDialog 类的属性

PrintPreviewDialog 类的常用属性如表 31.10 所示。

表 31.10　　PrintPreviewDialog 对象常用属性及说明

属　性	说　明
AutoSize	获取或设置一个值，该值指示 PrintPreviewDialog 类是否自动调整大小以完整显示其内容
Capture	获取或设置一个值，该值指示控件是否已捕获鼠标
ContextMenu	获取或设置控件的快捷菜单
ContextMenuStrip	获取或设置该控件的快捷菜单
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

续表

属　性	说　明
Dock	获取或设置控件在其父控件中的停靠方式
Document	获取或设置要预览的文档
Enabled	获取或设置一个值，该值指示是否已启用该控件
Font	获取或设置用于控件的字体
ForeColor	获取或设置控件的前景色
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
MaximumSize	获取或设置窗体可调整到的最大大小
UseAntiAlias	获取或设置一个值，该值指示打印是否使用操作系统的防锯齿功能

下面对 PrintPreviewDialog 类比较重要的属性进行详细介绍。

（1）Document 属性。

该属性用于获取或设置一个值，该值指示要预览的文档。

语法：

```
public PrintDocument Document { get; set; }
```

属性值：PrintDocument，它表示要预览的文档。

（2）UseAntiAlias 属性。

该属性表示获取或设置一个值，该值指示打印是否使用操作系统的防锯齿功能。

语法：

```
public bool UseAntiAlias { get; set; }
```

属性值：如果使用防锯齿功能，则为 true；否则为 false。

注意：UseAntiAlias 设为 true 可以消除锯齿，提高打印质量；但会损耗性能，使显示速度变慢。

例如，下面的实例演示 PrintPreviewDialog 如何设置 Document 属性和 UseAntiAlias 属性。该实例假定窗体包含一个名为 TreeView1 的 TreeView，其中包含 TreeNode 对象。每个 TreeNode 对象的 Tag 属性必须设置为一个能被运行该实例的计算机访问的完全限定文档名。将每个 TreeNode.Text 属性设置为一个标识由 TreeNode.Tag 属性指定的文件的字符串。例如，可以将 TreeNode1.Tag 设置为“c:\myDocuments\recipe.doc”，将 TreeNode1.Text 设置为“recipe.doc”。该实例还假定窗体包含一个名为 PrintPreviewDialog1 的 PrintPreviewDialog 和一个名为 Button1 的按钮。若要运行此实例，请调用位于该窗体的构造函数或 Load 事件处理程序中的 InitializePrintPreviewDialog 方法。代码如下：

```
internal PrintPreviewDialog PrintPreviewDialog1;                    //定义一个 PrintPreviewDialog 对象
//实例化一个 PrintDocument 对象
private System.Drawing.Printing.PrintDocument document =new System.Drawing.Printing.PrintDocument();
private void InitializePrintPreviewDialog()
{
    this.PrintPreviewDialog1 = new PrintPreviewDialog();            //实例化对象 PrintPreviewDialog1 对象
    //设置 PrintPreviewDialog1 对象的工作区大小
    this.PrintPreviewDialog1.ClientSize =new System.Drawing.Size(400, 300);
    this.PrintPreviewDialog1.Location =new System.Drawing.Point(29, 29); //设置 PrintPreviewDialog1 对象的位置
    this.PrintPreviewDialog1.Name = "PrintPreviewDialog1";           //设置 PrintPreviewDialog1 控件的名称
    //向 PrintDocument 对象的 PrintPage 事件中添加处理事件
    this.document.PrintPage += new System.Drawing.Printing.PrintPageEventHandler(document_PrintPage);
    this.PrintPreviewDialog1.MinimumSize =new System.Drawing.Size(375, 250);//获取窗体可调整的最小大小
    this.PrintPreviewDialog1.UseAntiAlias = true;                   //设置打印时使用系统的防锯齿功能
}
private void Button1_Click(object sender, System.EventArgs e)
{
    if (TreeView1.SelectedNode != null)                             //当 TreeView1 控件中不存在选中节点时
```

```
        {
            document.DocumentName = TreeView1.SelectedNode.Tag.ToString();      //设置打印文档的文档名
        }
        PrintPreviewDialog1.Document = document;                                //设置打印预览的文档
        PrintPreviewDialog1.ShowDialog();                                       //显示打印预览窗体
    }
    private void document_PrintPage(object sender,System.Drawing.Printing.PrintPageEventArgs e)
    {
        string text = "C-Sharp 编程词典";                                       //设置要打印的内容
        //设置打印内容的字体
        System.Drawing.Font printFont = new System.Drawing.Font("Arial", 35, System.Drawing.FontStyle.Regular);
        e.Graphics.DrawString(text, printFont, System.Drawing.Brushes.Black, 0, 0);   //绘制打印的内容
    }
```

31.5.2 PrintPreviewDialog 类的方法

PrintPreviewDialog 类的常用方法如表 31.11 所示。

表 31.11　　PrintPreviewDialog 对象常用方法及说明

方　法	说　明
Activate	激活窗体并给予它焦点
FindForm	检索控件所在的窗体
GetNextControl	按照子控件的 Tab 键顺序向前或向后检索下一个控件
Hide	获取一个值，该值指示是否可以选中控件
Show	显示窗体
ShowDialog	将窗体显示为模式对话框
ToString	从 Form 继承

下面对 PrintPreviewDialog 类比较重要的方法 ShowDialog 进行详细介绍，该方法表示运行通用对话框。

语法：

```
public DIALOGRESULT ShowDialog ()
```

返回值：如果用户在对话框中单击“确定”，则为 DialogResult.OK；否则为 DialogResult.Cancel。

31.6 Windows 打印对象的综合应用

下面通过一个综合实例来说明这 5 个对象的具体使用方法。

实例位置：光盘\mingrisoft\dxyy\31\01

程序开发步骤如下。

(1) 在 Visual Studio 2008 中新建一个项目，并命名为 TransferPrint。

(2) 运行本实例，在当前的窗体中显示数据库中的内容，代码如下：

```
//实例化一个数据库连接字符串
public static string M_str_sqlcon = @"Data Source=127.0.0.1;Initial Catalog=db_mrsql;User ID=sa";
//定义一个 SqlConnection 类型的公共变量 My_con，用于判断数据库是否连接成功
public static SqlConnection My_con;
public string SQLstr = "select * from tb_YearD09"; //实例化一个数据库查询字符串
public string SQLTable = "tb_YearD09";             //实例化一个字符串 SQLTable 用来保存表 tb_YearD09
public DataSet DSet = new DataSet();               //实例化一个数据集对象
public DataSet getDataSet(string SQLstr, string tableName)
{
    My_con = new SqlConnection(M_str_sqlcon); //用 SqlConnection 对象与指定的数据库相连接
    My_con.Open();                            //打开数据库连接
    //创建一个 SqlDataAdapter 对象，并获取指定数据表的信息
```

```
        SqlDataAdapter SQLda = new SqlDataAdapter(SQLstr, My_con);
        DataSet My_DataSet = new DataSet();          //创建 DataSet 对象
        //通过 SqlDataAdapter 对象的 Fill 方法，将数据表信息添加到 DataSet 对象中
        SQLda.Fill(My_DataSet, tableName);
        My_con.Close();                              //关闭数据库的连接
        return My_DataSet;                           //返回 DataSet 对象的信息
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        DSet = getDataSet(SQLstr, SQLTable);         //从数据集中获取数据内容
        dataGridView1.DataSource = DSet.Tables[0];   //设置 DataGridView 控件的数据源
    }
    private void button1_Click(object sender, EventArgs e)
    {
        MyDLL.TakeOver(DSet);                        //向打印模块中的 DLL 文件传递参数
    }
    private void button2_Click(object sender, EventArgs e)
    {
        this.Close();                                //关闭当前窗体
    }
```

当单击“打印”按钮时用到类 MyDLL，类 MyDLL 的对应代码如下：

```
    private static DataSet DataCom = new DataSet();
    public static void TakeOver(DataSet Dset)
    {
        DataCom = Dset;                              //获取 DataSet 对象
        Frm_PrintSet FrmPring = new Frm_PrintSet(); //实例化打印窗体
        FrmPring.Show();                             //显示打印窗体
    }
    public static DataSet SendOut()
    {
        return DataCom;
    }
```

在类 MyDLL 中用到窗体对象 Frm_PrintSetFrmPring，该窗体对应的代码如下：

```
    public static DataSet MyDS_Grid;                 //声明一个数据集对象
    public int header = 30;                          //实例化一个整型变量 header 表示标题栏的默认高度
    public int row = 23;                             //实例化一个整型变量 row 表示单元格的默认高度
    public Color linecolor = Color.Black;            //实例化一个 ARGB 颜色对象用来表示边框的颜色
    public int linewidth = 1;                        //实例化一个整型变量 linewidth 用来表示边线的宽度
    public bool PLintUP = true;                      //实例化一个 bool 型变量 PLintUp
    public bool PLintLeft = true;                    //实例化一个 bool 型变量 PLintLeft
    public bool PLintUnit = true;                    //实例化一个 bool 型变量 PLintUnit
    public bool Aspect = true;                       //实例化一个 bool 型变量 Aspect 用来表示打印方向
    public bool boundary = false;                    //实例化一个 bool 型变量 boundary 用来表示是否打印分割线
    public void MSetUp(PrintClass dgp)
    {
        string n = "false";                          //实例化一个字符串变量 n
        string[] margin = new string[4];             //实例化一个字符串数组 margin
        if (checkBox_margin.Checked == true)         //当多选框 checkBox_margin 处于选中状态时
            n = "true";                              //设置字符串变量 n 的值为 true
        else                                         //当多选框 checkBox_margin 处于未选中状态时
            n = "false";                             //设置字符串变量的值为 false
        margin[0] = n;                               //实例化字符串数组 margin 中第一个元素的值
        margin[1] = textBox_topmargin.Text;          //实例化字符串数组 margin 中第二个元素的值
        margin[2] = textBox_leftmargin.Text;         //实例化字符串数组 margin 中第三个元素的值
        margin[3] = textBox_buttommargin.Text;       //实例化字符串数组 margin 中第四个元素的值
        dgp.PrintSetUp(margin);                      //调用页边距设置方法
        dgp.headerheight = this.header;              //设置列标题的默认高度
        if (checkBox_Header.Checked == true)         //当复选框 checkBox_Header 处于选中状态时
        {
            //设置前景色
            dgp.brushHeaderFont = new SolidBrush(dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor);
            dgp.headerfont = dataGridView1.ColumnHeadersDefaultCellStyle.Font;   //设置字体样式
```

```
            //设置背景色
            dgp.brushHeaderBack = new SolidBrush(dataGridView1.ColumnHeadersDefaultCellStyle.BackColor);
            dgp.headerheight = dataGridView1.ColumnHeadersHeight;   //设置列标题的高度
        }
        if (checkBox_Cell.Checked == true)                          //当复选框 checkBox_Cell 处于被选中状态时
        {
            dgp.Cellfont = dataGridView1.RowsDefaultCellStyle.Font;   //设置字体样式
            dgp.brushCellBack = new SolidBrush(dataGridView1.RowsDefaultCellStyle.BackColor);      //设置背景色
            dgp.brushCellFont = new SolidBrush(dataGridView1.RowsDefaultCellStyle.ForeColor);      //设置前景色
            dgp.rowgap = row;                                       //设置行的高度
        }
        if (checkBox_Table.Checked == true)                         //当复选框 checkBox 处于被选中状态时
        {
            dgp.AlignmentSgin = this.comboBox_Alignment.SelectedIndex;   //设置对齐方式
        }
        dgp.iseverypageprinttitle = checkBox_Title.Checked;         //是否每一页都打印标题
        dgp.needprintpageindex = checkBox_Page.Checked;             //是否每一页都打印页脚
        dgp.PageAspect = Aspect;                                    //设置横向打印
        //设置表格的边线
        dgp.LineUP = PLintUP;                                       //是否打印上边线
        dgp.LineLeft = PLintLeft;                                   //是否打印左边线
        dgp.LineUnit = PLintUnit;                                   //是否打印单元格边线
        dgp.LineColor = linecolor;                                  //设置线的颜色
        dgp.Boundary = checkBox_Boundary.Checked;                   //是否打印分割线
        dgp.HAlignment = comboBox_HAlignment.SelectedIndex;         //列标题的对齐方式
        dgp.UAlignment = comboBox_UAlignment.SelectedIndex;         //单元格的对齐方式
    }
    public void Barbarism_DataGrid(int n)
    {
        if (n == 0)                                                 //当 n 的值为 0 时
        {
            Font Hfont = new Font("宋体", 10);                      //实例化一个字体类对象 Hfont
            dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = Color.White;      //设置列标题的背景颜色
            dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor = Color.Black;//设置列标题的字体颜色
            dataGridView1.ColumnHeadersDefaultCellStyle.Font = Hfont;          //设置列标题的字体样式
            dataGridView1.ColumnHeadersHeight = 30;                            //设置列标题的高度
            textBox_Size.Text = dataGridView1.ColumnHeadersHeight.ToString();  //设置列标题的文本内容
        }
        if (n == 1)                                                            //当 n 的值为 1 时
        {
            Font Hfont = new Font("宋体", 9);                                  //实例化一个字体类对象 Hfont
            dataGridView1.RowsDefaultCellStyle.BackColor = Color.White;        //设置单元格的背景颜色
            dataGridView1.RowsDefaultCellStyle.ForeColor = Color.Black;        //设置单元格的字体颜色
            dataGridView1.RowsDefaultCellStyle.Font = Hfont;                   //设置单元格的字体样式
            textBox_CellSize.Text = dataGridView1.Rows[0].Height.ToString();   //设置单元格的高度
        }
    }
    public void InputCount(KeyPressEventArgs e, object sender, int n)
    {
        int tsele = 0;                                                     //实例化一个整型变量 tsele
        //当按下的按键不是键盘上的指定按键时
        if (!(e.KeyChar <= '9' && e.KeyChar >= '0') && e.KeyChar != '\r' && e.KeyChar != '\b')
        {
            e.Handled = true;                                              //处理 KeyPress 事件
        }
        else                                                               //当按下的按键是键盘上的指定按键时
        {
            if (((TextBox)sender).SelectedText != "")                      //当文本框中的选定内容不为空时
            {
                tsele = ((TextBox)sender).SelectionLength;                 //保存文本框中选定文本的长度
            }
            if (n >0)                                                      //当 n 的值大于 0 时
            {
```

```
                if (e.KeyChar <= '9' && e.KeyChar >= '0') //当按键为指定的按键时
                    if (((((TextBox)sender).Text).Length - tsele + 1) > n)//当输入文本的长度满足一定条件时
                        e.Handled = true;                                //处理 KeyPress 事件
            }
        }
    }
    public void SetCheckBox(CheckBox CheckB, GroupBox GroupB)
    {
        if (CheckB.Checked == true)                                      //如果 CheckBox 控件为可用状态
            GroupB.Enabled = false;                                      //设置 GroupB 为不可用状态
        else                                                             //当 CheckB 为不可用状态时
            GroupB.Enabled = true;                                       //设置 GroupB 为可用状态
    }
    public void TextBoxValue(object sender,int d,int n,int m)
    {
        if (((TextBox)sender).Text == "")                                //当文本框的内容为空时
        {
            ((TextBox)sender).Text = d.ToString();                       //设定文本框中的内容
        }
        //当文本框中的内容大于 m 或者小于 n 时
        if (int.Parse(((TextBox)sender).Text) < n || int.Parse(((TextBox)sender).Text) > m)
        {
            if (int.Parse(((TextBox)sender).Text) < n)                   //当文本框中的内容小于 n 时
                ((TextBox)sender).Text = n.ToString();                   //设置文本框中的内容为 n
            if (int.Parse(((TextBox)sender).Text) > m)                   //当文本框中的内容大于 m 时
                ((TextBox)sender).Text = m.ToString();                   //设置文本框中的内容为 m
        }
    }
    private void button_Preview_Click(object sender, EventArgs e)
    {
        PrintClass dgp = new PrintClass(this.dataGridView1, this.textBox_Title.Text, 16,
comboBox_PageSize.SelectedIndex, checkBox_Aspect.Checked);               //对打印信息进行设置
        MSetUp(dgp);                                                     //记录窗体中打印信息的相关设置
        string[] header = new string[dataGridView1.ColumnCount];         //创建一个与数据列相等的字符串数组
        for (int p = 0; p < dataGridView1.ColumnCount; p++)              //循环遍历 DataGridView 控件中的每一列
        {
            header[p] = dataGridView1.Columns[p].HeaderCell.Value.ToString();//为字符串数组 header 赋值
        }
        dgp.print();                                                     //显示打印预览窗体
    }
    private void Form1_Load(object sender, EventArgs e)
    {
        MyDS_Grid = MyDLL.SendOut();                                     //发送传递的数据集对象
        dataGridView1.DataSource = MyDS_Grid.Tables[0]; //设置 dataGridView1 控件的数据源
        dataGridView1.RowHeadersVisible = false;                         //设置不显示包含行标题的列
        comboBox_PageSize.SelectedIndex = 0;                             //设定 comboBox_PageSize 的当前选定项的索引为 0
        comboBox_HAlignment.SelectedIndex = 0;                           //设置 comboBox_HAlignment 的当前选定项的索引为 0
        comboBox_UAlignment.SelectedIndex = 0;                           //设置 comboBox_UAlignment 的当前选定项的索引为 0
        Barbarism_DataGrid(0);                                           //调用 Barbarism_DataGrid 方法
        Barbarism_DataGrid(1);
    }
    private void checkBox_margin_MouseDown(object sender, MouseEventArgs e)
    {
        SetCheckBox(((CheckBox)sender), groupBox3);                      //调用 setCheckBox 方法
    }
    private void button_BackColor_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)                //当单击颜色对话框中的“确定”按钮时
            dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = colorDialog1.Color;//设置列标题的背景颜色
    }
    private void button_Fint_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.OK)                 //当单击字体对话框中的“确定”按钮时
            dataGridView1.ColumnHeadersDefaultCellStyle.Font = fontDialog1.Font; //设置列标题的默认样式
```

```
        fontDialog1.Dispose();                                  //释放 fontDialog1 对象
    }
    private void checkBox_Title_MouseDown(object sender, MouseEventArgs e)
    {
        SetCheckBox(((CheckBox)sender), groupBox4);             //调用 SetCheckBox 方法
    }
    private void button_FontColor_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)       //当单击颜色对话框中的“确定”按钮时
            dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor = colorDialog1.Color;//设置列标题的字体颜色
        colorDialog1.Dispose();     //释放 colorDialog1 对象
    }
    private void button_Default_Click(object sender, EventArgs e)
    {
        Barbarism_DataGrid(0);     //调用 Barbarism_DataGrid 方法
    }
    private void button_CellFont_Click(object sender, EventArgs e)
    {
        if (fontDialog1.ShowDialog() == DialogResult.OK)                //当单击字体对话框中的“确定”按钮时
            dataGridView1.RowsDefaultCellStyle.Font = fontDialog1.Font;  //设置单元格中的默认字体
        fontDialog1.Dispose();                                          //释放对象 fontDialog1
    }
    private void button1_CellBackColor_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)               //当单击颜色对话框中的“确定”按钮时
            dataGridView1.RowsDefaultCellStyle.BackColor = colorDialog1.Color;  //设置单元格的字体背景颜色
        colorDialog1.Dispose();                                         //释放对象 colorDialog1
    }
    private void button_CellFontColor_Click(object sender, EventArgs e)
    {
        if (colorDialog1.ShowDialog() == DialogResult.OK)               //当单击颜色对话框中的“确定”按钮时
            dataGridView1.RowsDefaultCellStyle.ForeColor = colorDialog1.Color;  //设置单元格的字体颜色
        colorDialog1.Dispose();                                         //释放对象 colorDialog1
    }
    private void checkBox_Cell_MouseDown(object sender, MouseEventArgs e)
    {
        SetCheckBox(((CheckBox)sender), groupBox5);                     //调用方法 SetCheckBox
    }
    private void button_CellDefault_Click(object sender, EventArgs e)
    {
        Barbarism_DataGrid(1);                                          //调用方法 Barbarism_DataGrid
    }
    private void textBox_Size_Leave(object sender, EventArgs e)
    {
        TextBoxValue(sender, 30, 20, 99);                               //调用方法 TextBoxValue
        dataGridView1.ColumnHeadersHeight = int.Parse(textBox_Size.Text);//设置 dataGridView1 控件的标题行的高度
        header = int.Parse(textBox_Size.Text);                          //设置标题栏的高度
    }
    private void textBox_Size_KeyPress(object sender, KeyPressEventArgs e)
    {
        InputCount(e, sender, 2);                                       //调用方法 InputCount
        if (e.KeyChar == '\r')                                          //当按下的键对应的字符为换行时
            textBox_Size_Leave(sender, e);                              //调用方法 textBox_Size_Leave
    }
    public void DateGrid_CellHeight(DataGridView DG, int n)
    {
        for (int i = 0; i < DG.Rows.Count; i++)                         //循环遍历 DG 中的每一行
        {
            DG.Rows[i].Height = n;                                      //设置每一行的高度
        }
    }
    private void textBox_CellSize_Leave(object sender, EventArgs e)
    {
        TextBoxValue(sender, 23, 20, 99);                               //调用方法 TextBoxValue 方法
```

```
        DateGrid_CellHeight(dataGridView1, int.Parse(textBox_CellSize.Text));//调用方法 DateGrid_CellHeight 方法
        row = int.Parse(textBox_CellSize.Text);                          //设置行的默认高度
    }
    private void textBox_CellSize_KeyPress(object sender, KeyPressEventArgs e)
    {
        InputCount(e, sender, 2);                                        //调用方法 InputCount
        if (e.KeyChar == '\r')                                           //当按下的键对应的字符为换行符时
            textBox_CellSize_Leave(sender, e);                           //调用方法 textBox_CellSize_Leave
    }
    private void checkBox_Table_MouseDown(object sender, MouseEventArgs e)
    {
        SetCheckBox(((CheckBox)sender), groupBox6);                      //调用方法 SetCheckBox
    }
    private void textBox_topmargin_KeyPress(object sender, KeyPressEventArgs e)
    {
        InputCount(e, sender, 3);                                        //调用方法 InputCount
        if (e.KeyChar == '\r')                                           //当按下的键对应的字符为换行符时
            textBox_topmargin_Leave(sender, e);                          //调用方法 textBox_topmargin_Leave
    }
    private void textBox_topmargin_Leave(object sender, EventArgs e)
    {
        TextBoxValue(sender, 60, 10, 500);                               //调用方法 TextBoxValue
    }
    private void textBox_leftmargin_Leave(object sender, EventArgs e)
    {
        TextBoxValue(sender, 50, 10, 700);                               //调用方法 TextBoxValue
    }
    private void textBox_leftmargin_KeyPress(object sender, KeyPressEventArgs e)
    {
        InputCount(e, sender, 3);                                        //调用方法 InputCount
        if (e.KeyChar == '\r')                                           //当按下的键对应的字符为换行符时
            textBox_leftmargin_Leave(sender, e);                         //调用方法 textBox_leftmargin_Leave
    }
    private void textBox_buttommargin_KeyPress(object sender, KeyPressEventArgs e)
    {
        InputCount(e, sender, 3);                                        //调用方法 InputCount
        if (e.KeyChar == '\r')                                           //当按下的键对应的字符为换行符时
            textBox_buttommargin_Leave(sender, e);                       //调用方法 textBox_buttommargin_Leave
    }
    private void textBox_buttommargin_Leave(object sender, EventArgs e)
    {
        TextBoxValue(sender, 50, 10, 500);                               //调用方法 TextBoxValue
    }
    //在窗体中绘制一个预览表格
    private void Frm_PrintSet_Activated(object sender, EventArgs e)
    {
        Color Lcolor;                                                    //实例化一个颜色对象
        Graphics g = panel_Line.CreateGraphics();                        //实例化一个 GDI+绘图图面类对象
        int paneW = panel_Line.Width;                                    //设置表格的宽度
        int paneH = panel_Line.Height;                                   //设置表格的高度
        g.DrawRectangle(new Pen(Color.WhiteSmoke, paneW), 0, 0, paneW, paneH);  //绘制一个矩形
        if (PLintUnit == true)                                           //如果绘制单元格线
            Lcolor = linecolor;                                          //设置单元格的颜色
        else                                                             //当绘制单元格线时
            Lcolor = Color.WhiteSmoke;                                   //设置单元格线的颜为背景颜色
        int unitW = (int)((paneW - 20 * 2) / 3);                         //设置单元格的宽度
        int unitH = (int)((paneH - 20 * 2) / 3);                         //设置单元格的高度
        int bW = 0, bH = 0;
        //绘制一个 3 行 3 列的表格
        for (int i = 0; i < 2; i++)                                      //从 0 循环遍历至 1
        {
```

```
            g.DrawLine(new Pen(Lcolor, 1), 20 + unitW, 20, 20 + unitW, paneH - 20);        //绘制纵线
            if (boundary==true && i==0)                          //如果是分割线并且是第一条线
            {
                //设置分割线的坐标点
                bW = paneW - 20;                                 //设置分割点的横坐标
                bH = 20 + unitH;                                 //设置分割点的纵坐标
            }
            else                                                 //当不是分割线或者分割线不是第一条时
                g.DrawLine(new Pen(Lcolor, 1), 20, 20 + unitH, paneW - 20, 20 + unitH);   //绘制横线
            unitH += unitH;                                      //下一条横线的位置
            unitW += unitW;                                      //下一条纵线的位置
        }
        if (boundary == true)                                    //当是分割线时
            g.DrawLine(new Pen(linecolor, 1), 20, bH, bW, bH);                            //绘制分割线
        if (PLintUP == true)                                     //绘制最上面的线
            Lcolor = linecolor;                                  //设置线的颜色
        else                                                     //当不是最上面的线时
            Lcolor = Color.WhiteSmoke;                           //设置 Lcolor 的颜色
        g.DrawLine(new Pen(Lcolor, linewidth), 20, 20, paneW - 20, 20);                   //绘制上线
        g.DrawLine(new Pen(Lcolor, linewidth), 20, paneH - 20, paneW - 20, paneH - 20);   //绘制下线
        if (PLintLeft == true)                                   //绘制最左面的线
            Lcolor = linecolor;                                  //线的颜色
        else
            Lcolor = Color.WhiteSmoke;                                                    //与背景色相同的颜色
        g.DrawLine(new Pen(Lcolor, linewidth), 20, 20, 20, paneH - 20);                   //绘制左线
        g.DrawLine(new Pen(Lcolor, linewidth), paneW - 20, 20, paneW - 20, paneH - 20);   //绘制右线
    }
    private void Frm_PrintSet_Shown(object sender, EventArgs e)
    {
        Frm_PrintSet_Activated(sender, e);                       //调用方法 Frm_PrintSet_Activated
    }
    //改变窗体中预览表格的方向
    private void checkBox_Aspect_MouseDown(object sender, MouseEventArgs e)
    {
        int aspX = 0;     //宽度
        int aspY = 0;     //高度
        if (((CheckBox)sender).Checked == false)                 //如果不是纵向打印
        {
            aspX = 136;                                          //设置宽度值 aspX
            aspY = 98;                                           //设置高度值 aspY
            PrintClass.PageScape = true;                         //横向打印
        }
        else                                                     //当是纵向打印时
        {
            aspX = 100;                                          //设置宽度值 aspX
            aspY = 116;                                          //设置高度值 aspY
            PrintClass.PageScape = false;                        //纵向打印
        }
        panel_Line.Width = aspX;                                 //设置控件的宽度
        panel_Line.Height = aspY;                                //设置控件的高度
        aspX = (int)((groupBox7.Width - aspX) / 2);              //设置控件的 Top
        panel_Line.Location = new Point(aspX, 26);               //设置控件的位置
        Frm_PrintSet_Activated(sender, e);                       //调用 Activated 事件
    }
    private void checkBox_line_MouseDown(object sender, MouseEventArgs e)
    {
        SetCheckBox(((CheckBox)sender), groupBox7);              //调用方法 SetCheckBox
        Frm_PrintSet_Activated(sender, e);                       //调用方法 Frm_PrintSet_Activated
    }
    private void button_upline_Click(object sender, EventArgs e)
```

```
{
    PLintUP = !PLintUP;                          //设置 PLintUp 的值
    Frm_PrintSet_Activated(sender, e);           //调用方法 Frm_PrintSet_Activated
}
private void button_leftline_Click(object sender, EventArgs e)
{
    PLintLeft = !PLintLeft;                      //设置 PLintLeft 的值
    Frm_PrintSet_Activated(sender, e);           //调用方法 Frm_PrintSet_Activated
}
private void button_cellline_Click(object sender, EventArgs e)
{
    PLintUnit = !PLintUnit;                      //设置 PLintUnit 的值
    Frm_PrintSet_Activated(sender, e);           //调用方法 Frm_PrintSet_Activated
}
private void button_LineColor_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK) //当单击颜色对话框中的“确定”按钮时
        linecolor = colorDialog1.Color;          //设置边线的颜色
    colorDialog1.Dispose();                      //释放 colorDialog1 对象所占用的资源
    Frm_PrintSet_Activated(sender, e);           //调用方法 Frm_PrintSet_Activated
}
private void checkBox_Boundary_MouseDown(object sender, MouseEventArgs e)
{
    boundary = !(((CheckBox)sender).Checked);    //设置 boundary 的值
    Frm_PrintSet_Activated(sender, e);           //调用方法 Frm_PrintSet_Activated
}
```

说明：本示例用到类 PrintClass，详细实现代码见项目源程序。此外，运行本示例前，需要添加命名空间 using DLLFullPrint。

运行本实例，效果如图 31.1、图 31.2 和图 31.3 所示。

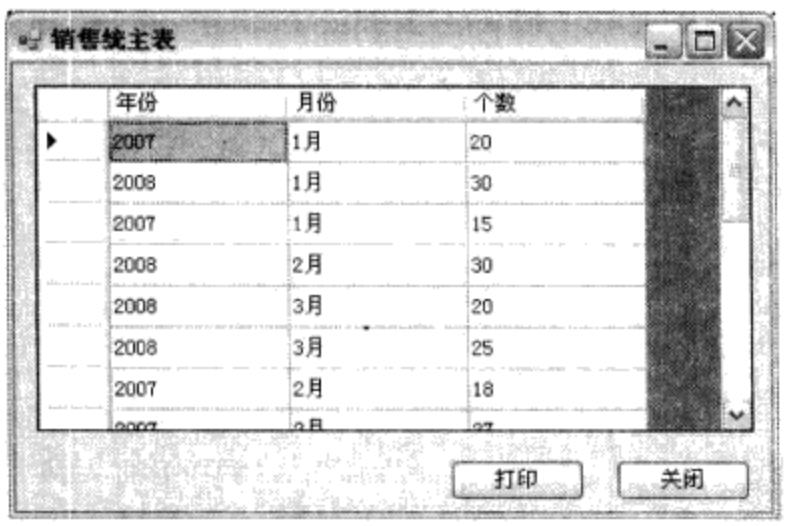


图 31.1　主窗体

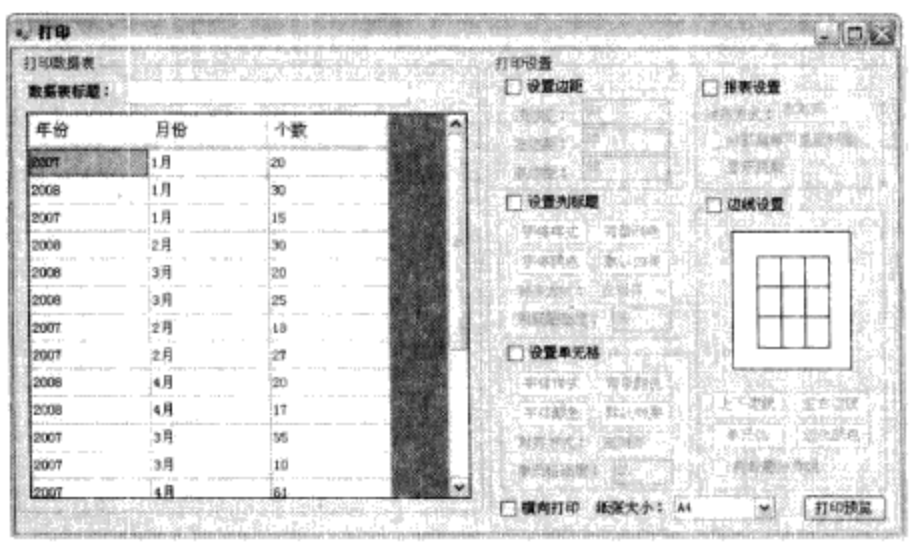


图 31.2　打印设置窗体

年份	月份	个数
2007	1月	20
2008	1月	30
2007	1月	15
2008	2月	30
2008	3月	20
2008	3月	25
2007	2月	18
2007	2月	27
2008	4月	20
2008	4月	17
2007	3月	55
2007	3月	10

图 31.3　打印预览窗体