Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Beginning JavaScript Fourth Edition

JavaScript 入门经典（第4版）

(美) Paul Wilton
Jeremy McPeak 著
张 敏 高宇辉
王东亚 译

清华大学出版社

循序渐进精细讲解所有JavaScript知识点
指导您构建出神入化的完美Web应用程序

JavaScript允许用户交互浏览精彩纷呈的个性化内容，可以极大地提升网页和Web应用程序的吸引力。精品入门教程《JavaScript入门经典(第4版)》全面透彻地讲解JavaScript编程基础知识，引导您立即开始在网页中嵌入JavaScript脚本。本书对上一版本做了全面更新，使JavaScript代码适用于最新版本的IE、Firefox和Safari浏览器。

主要内容

- 循序渐进介绍JavaScript基础知识，包括JavaScript的含义、原理和功能
- 介绍用于创建JavaScript Web应用程序的各种工具
- 分析如何从字符串中选择单个字符、如何将字符码转换为字符串以及如何复制字符串的部分内容
- 讲述如何对数组执行连接、复制、排序和倒序操作
- 说明如何使用jQuery、Prototype和MooTools等JavaScript框架快捷高效地编写JavaScript程序
- 深入讨论Ajax
- 分析常见错误以及调试和处理方法

作者简介

Paul Wilton曾加盟一家Internet开发公司，期间历时3年构建了一个出色的Internet解决方案。现自创一家软件公司，该公司主要使用JavaScript开发在线假日酒店预订系统，目前的运作十分成功。

Jeremy McPeak是一位自学成才的程序员，从1998年开始从事网站开发工作。他曾与他人合著*Professional Ajax, 2nd Edition* (Wiley 2007)一书，并在线发表多篇有关XSLT、ASP.NET Web窗体和C#的文章。他目前为一家能源公司开发Web应用程序。

Wrox Beginning guides are crafted to make learning programming languages and technologies easier than you think, providing a structured, tutorial format that will guide you through all the techniques involved.

源代码下载

http://www.wrox.com

http://www.tupwk.com.cn/downpage

Wrox
An Imprint of
WILEY

上架建议：Web开发/JavaScript
读者信箱：wkservice@vip.163.com
投稿信箱：bookservice@263.net

wrox.com

Programmer Forums

Join our Programmer to Programmer forums to ask and answer programming questions about this book, join discussions on the hottest topics in the industry, and connect with fellow programmers from around the world.

Code Downloads

Take advantage of free code samples from this book, as well as code samples from hundreds of other books, all ready to use.

Read More

Find articles, ebooks, sample chapters and tables of contents for hundreds of books, and more reference resources on programming topics that matter to you.

ISBN 978-7-302-24561-2
9 787302 245612 >
定价：88.00元

JavaScript 入门经典

(第 4 版)

(美) Paul Wilton
Jeremy McPeak 著
张 敏 高宇辉
王东亚 译

清华大学出版社
北 京

Paul Wilton，Jeremy McPeak
Beginning JavaScript, Fourth Edition
EISBN：978-0-470-52593-7
Copyright © 2010 by Wiley Publishing, Inc., Indianapolis Indiana
All Rights Reserved. This translation published under license.

本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2010-1185

本书封面贴有 Wiley 公司防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

JavaScript 入门经典(第 4 版) / (美) 威尔顿(Wilton,P.)，(美) 麦克匹克(McPeak,J.) 著；
张敏，高宇辉，王东亚 译. —北京：清华大学出版社，2011.2
书名原文：Beginning JavaScript, Fourth Edition

ISBN 978-7-302-24561-2

Ⅰ. J…　Ⅱ. ①威…　②麦…　③张…　④高…　⑤王…　Ⅲ. Java 语言—程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2010)第 257970 号

责任编辑：王　军　韩宏志
装帧设计：孔祥丰
责任校对：胡雁翎
责任印制：何　芊
出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：三河市金元印装有限公司
经　销：全国新华书店
开　本：185×260　**印　张**：41.75　**字　数**：1016 千字
版　次：2011 年 2 月第 1 版　　**印　次**：2011 年 2 月第1 次印刷
印　数：1～4000
定　价：88.00 元

产品编号：035600-01

前　言

JavaScript 是一门脚本语言，它通过提供动态的、个性化交互式内容，来增强静态 Web 应用程序的功能。JavaScript 使访问站点的用户能够享受到更美妙的体验，增强了网站对用户的吸引力。现在，美观的下拉菜单、滚动文字和动态内容已经广泛应用于各种 Web 站点，这一切都是通过 JavaScript 来实现的。各种主流浏览器都支持 JavaScript，JavaScript 已经成为从事 Web 开发的首选脚本语言。另外，JavaScript 语言也可用于 Web 之外的其他场合，例如自动完成管理任务。

本书旨在介绍使用 JavaScript 进行开发的基础知识，即 JavaScript 的含义，JavaScript 代码是如何运行的，以及使用 JavaScript 能够实现哪些功能等。本书将首先介绍 JavaScript 的基本语法，然后介绍如何创建功能强大的 Web 应用程序。读者不必为没有编程经验而担心，本书将循序渐进地介绍所有相关知识。JavaScript 是通向程序设计世界的大门，学习和理解了本书的基础知识，就可以进一步学习更新、更高级的编程技术。

本书读者对象

为了最好地汲取本书中的知识，读者应该对 HTML 有所了解，并知道如何创建静态的 Web 页面。除此之外，读者不必拥有任何编程经验。

本书同样适合于具有编程经验、且希望进行 Web 程序设计的读者。这些读者可能比较了解计算机知识，但未必掌握 Web 技术。

另外，一些读者具备设计背景，但对计算机知识和 Web 技术不大了解。那么，本书可以作为一个进入编程和 Web 应用程序开发世界的快捷通道。

对于所有的读者，本书都物有所值。

本书内容

本书首先介绍 JavaScript 的含义，以及 JavaScript 的基础语法。然后详细介绍程序设计的基本概念，包括数据、数据类型以及选择语句和循环语句等结构化程序设计的概念。

熟悉了这些基础知识之后，本书将介绍 JavaScript 的一个重要概念——对象，讨论如何利用 JavaScript 的内置对象，如日期和字符串等，来管理复杂的数据，简化程序的设计。本书还将介绍如何使用 JavaScript 操作浏览器提供的对象，如表单、窗口和其他控件。运用这些知识，就可以创建专业水准的 Web 应用程序，并与用户交互。

即使是经验丰富的程序员，随着代码长度的增加，错误也在所难免，JavaScript 代码亦是如此。本书介绍了常见的语法错误和逻辑错误，还介绍了如何找到这些错误，以及如何使用针对 Firefox、Internet Explorer、Safari/Chrome 和 Opera 的 JavaScript 调试器来帮助找出错误。本书还介绍了如何处理漏掉的错误，并确保这些错误不会对最终用户的体验造成不良影响。

随后，本书将介绍更高级的主题，例如使用 cookie，通过动态 HTML 和 XML 使 Web 页面变得生动鲜活。最后还介绍了一种令人激动的较新技术 Ajax。Ajax 允许 HTML 页面中的 JavaScript 代码直接与服务器通信，最有用的就是通过 Ajax 查询服务器上的数据库中的信息。Google 工具栏就是一个 Ajax 技术应用案例。如果在浏览器中安装了 Google 工具栏，只需在 Google 工具栏中输入搜索关键字，就可以获得相关信息，这些信息正是通过查询 Google 搜索数据库获取的。

最后探讨了一些省时的 JavaScript 框架，例如 jQuery、Prototype 和 MooTools，了解它们的工作原理，以及它们如何帮助创建复杂而强大的 JavaScript 应用程序。

本书介绍的所有新概念都用实例加以说明。通过这些实例可以对所学的 JavaScript 原理进行实践，以巩固所学的知识。

本书末尾有 4 个附录，附录 A 是本书各章的习题答案，其他附录包含内容丰富且极富价值的参考资料。附录 B 是 JavaScript 语言的核心参考，附录 C 是完整的 W3C DOM 核心参考——还包括 HTML DOM 和 DOM Level 2 事件模型的信息，附录 D 是 Latin-1 字符集的十进制和十六进制字符码。

如何使用本书

JavaScript 代码是基于文本的技术，因此要创建 JavaScript 程序，只需一个文本编辑器即可，例如 Windows 中的记事本或其他文本编辑器。

另外，为了测试本书中的代码，还需要一个支持较新 JavaScript 版本的浏览器。最好使用 IE 8 及以上版本，或者 Firefox 3 及以上版本。本书在这两种浏览器中进行了详细的测试。本书的代码可以在当今的大多数浏览器中工作，但本书后面用于演示动态 HTML 和 DOM 脚本编程技术的部分代码专用于特定浏览器。本书的大部分代码都是跨浏览器兼容的，对于不能跨浏览器兼容的情况，本书会专门指出。

源代码

学习本书中的示例时，可以手工输入所有的代码，也可以使用本书附带的源代码文件。本书使用的所有源代码都可以从本书合作站点 www.wrox.com 上下载。登录到站点 www.wrox.com，使用 Search 框或书名列表就可以找到本书，接着单击本书细目页面上的 Download Code 链接，就可以获得所有的源代码。

提示：

许多图书的书名都很相似，所以通过 ISBN 查找本书是最简单的，本书的英文原版的 ISBN 是 978-0-470-52593-7。

下载了代码后，只需用自己喜欢的解压缩软件对它进行解压缩即可。另外，也可以进入 www.wrox.com/dynamic/books/download.aspx 上的 Wrox 代码下载主页，查看本书和其他 Wrox 图书的所有代码。

勘误表

尽管我们已经尽了各种努力来保证文章或代码中不出现错误，但错误总是难免的，如果你在本书中找到了错误，例如拼写错误或代码错误，请告诉我们，我们将非常感激。通过勘误表，可以让其他读者节省时间、避免阅读和学习受挫，当然，这还有助于提供更高质量的书籍。请给 wkservice@vip.163.com 发电子邮件，我们就会检查你的信息，如果是正确的，就把它发送到该书的勘误表页面上，或在后续版本中采用。

要在网站上找到本书的勘误表，可以登录 www.wrox.com，通过 Search 框或书名列表查找本书，然后在本书的细目页面上，单击 Book Errata 链接。

p2p.wrox.com

P2P 邮件列表是为作者和读者之间的讨论而建立的。读者可以在 p2p.wrox.com 上加入 P2P 论坛。该论坛是一个基于 Web 的系统，用于传送与 Wrox 图书相关的信息和相关技术，与其他读者和技术用户交流。该论坛提供了订阅功能，当论坛上有新贴子时，会给你发送你选择的主题。Wrox 作者、编辑和其他业界专家和读者都会在这个论坛上进行讨论。

在 http://p2p.wrox.com 上有许多不同的论坛，帮助读者阅读本书，在读者开发自己的应用程序时，也可以从这个论坛中获益。要加入这个论坛，需执行下面的步骤：

(1) 进入 p2p.wrox.com，单击 Register 链接。

(2) 阅读其内容，单击 Agree 按钮。

(3) 提供加入论坛所需的信息及愿意提供的可选信息，单击 Submit 按钮。

(4) 然后就会收到一封电子邮件，其中的信息描述了如何验证账户，完成加入过程。

提示：

不加入 P2P 也可以阅读论坛上的信息，但只有加入论坛后，才能发送自己的信息。

加入论坛后，就可以发送新信息，回应其他用户的贴子。可以随时在 Web 上阅读信息。如果希望某个论坛给自己发送新信息，可以在论坛列表中单击该论坛对应的 Subscribe to this Forum 图标。

对于如何使用 Wrox P2P 的更多信息，可阅读 P2P FAQ，了解论坛软件的工作原理，以及许多针对 P2P 和 Wrox 图书的常见问题的解答。要阅读 FAQ，可以单击任意 P2P 页面上的 FAQ 链接。

目　录

第 1 章　JavaScript 与 Web 概述 …… 1
1.1　JavaScript 简介 …… 1
1.1.1　JavaScript 的含义 …… 1
1.1.2　JavaScript 与 Web …… 2
1.1.3　为什么选择 JavaScript …… 3
1.1.4　JavaScript 的功能 …… 4
1.1.5　创建 JavaScript Web 应用程序所需的工具 …… 4
1.2　脚本的使用场合 …… 6
1.2.1　链接外部 JavaScript 文件 …… 6
1.2.2　使用外部文件的优点 …… 7
1.3　第一个简单的 JavaScript 程序 …… 8
1.4　浏览器以及兼容性问题 …… 13
1.5　小结 …… 14

第 2 章　数据类型与变量 …… 15
2.1　JavaScript 中的数据类型 …… 15
2.1.1　数值数据 …… 16
2.1.2　文本数据 …… 16
2.1.3　布尔数据 …… 17
2.2　变量——保存在内存中的数据 …… 17
2.2.1　创建变量并赋值 …… 18
2.2.2　用其他变量的值为变量赋值 …… 20
2.3　使用数据——计算数值及基本字符串操作 …… 22
2.3.1　数值计算 …… 22
2.3.2　运算符的优先级 …… 26
2.3.3　基本的字符串操作 …… 29
2.3.4　字符串与数值的混合操作 …… 31
2.4　数据类型转换 …… 32
2.5　数组 …… 35
2.6　小结 …… 43
2.7　习题 …… 44

第 3 章　决策、循环和函数 …… 45
3.1　决策语句——if 和 switch 语句 …… 45
3.1.1　比较运算符 …… 46
3.1.2　if 语句 …… 47
3.1.3　逻辑运算符 …… 51
3.1.4　在 if 语句中使用多个条件 …… 53
3.1.5　else 和 else if …… 56
3.1.6　字符串的比较 …… 58
3.1.7　switch 语句 …… 59
3.2　循环——for 语句和 while 语句 …… 63
3.2.1　for 循环 …… 63
3.2.2　for...in 循环 …… 66
3.2.3　while 循环 …… 66
3.2.4　do...while 循环 …… 68
3.2.5　break 和 continue 语句 …… 69
3.3　函数 …… 70
3.3.1　创建自定义函数 …… 70
3.3.2　变量的作用域和生存期 …… 73
3.4　小结 …… 74
3.5　习题 …… 75

第 4 章　常见错误、调试和错误处理 …… 77
4.1　一些常见错误 …… 77
4.1.1　未经定义的变量 …… 77
4.1.2　区分大小写 …… 79
4.1.3　不匹配的大括号 …… 79
4.1.4　不匹配的圆括号 …… 80
4.1.5　赋值而不是相等 …… 80

4.1.6 将方法和属性混为一谈……81
4.1.7 在连接字符串时未使用加号(+)……82
4.2 错误处理……82
4.2.1 避免错误……83
4.2.2 try...catch 语句……83
4.3 调试……92
4.3.1 在 Firefox 中用 Firebug 调试……93
4.3.2 在 Internet Explorer 中调试……104
4.3.3 在 Safari 中调试……109
4.3.4 使用 Opera 的开发工具 Dragonfly……114
4.4 小结……116
4.5 习题……117

第 5 章 JavaScript——基于对象的语言……119
5.1 基于对象的程序设计……119
5.1.1 对象的含义……119
5.1.2 JavaScript 中的对象……120
5.1.3 使用 JavaScript 对象……121
5.1.4 基本数据类型与对象……123
5.2 JavaScript 的内置对象类型……124
5.2.1 String 对象……124
5.2.2 Array 对象……134
5.2.3 Array 的新方法……140
5.2.4 Math 对象……144
5.2.5 Number 对象……150
5.2.6 toFixed()方法……151
5.2.7 Date 对象……151
5.3 创建对象的新类型(引用类型)……160
5.4 小结……169
5.5 习题……170

第 6 章 浏览器程序设计……171
6.1 浏览器对象简介……172
6.1.1 window 对象……173
6.1.2 history 对象……174
6.1.3 location 对象……175
6.1.4 navigator 对象……175
6.1.5 screen 对象……176
6.1.6 document 对象……176
6.1.7 使用 document 对象……176
6.1.8 images 集合……178
6.1.9 links 集合……180
6.2 用事件响应用户的操作……180
6.2.1 事件的含义……180
6.2.2 将代码连接到事件……181
6.2.3 确定用户的浏览器……186
6.3 小结……195
6.4 习题……197

第 7 章 HTML 表单：与用户交互……198
7.1 HTML 表单……198
7.2 Form 对象的其他属性和方法……201
7.2.1 表单中的 HTML 元素……201
7.2.2 共有的属性和方法……202
7.2.3 按钮元素……204
7.2.4 文本元素……207
7.2.5 textarea 元素……214
7.2.6 复选框和单选按钮……216
7.2.7 选择框……224
7.3 小结……237
7.4 习题……239

第 8 章 窗口和框架……240
8.1 框架和窗口对象……240
8.1.1 在框架之间编写代码……244
8.1.2 框架之间的代码访问……249
8.2 打开新窗口……259
8.2.1 打开新的浏览器窗口……259
8.2.2 窗口之间的脚本编程……266
8.2.3 移动窗口并重置其大小……270
8.3 安全性……271
8.4 小结……272
8.5 习题……273

第 9 章　字符串操作……274
9.1　新的字符串方法……274
9.1.1　split()方法……275
9.1.2　replace()方法……278
9.1.3　search()方法……278
9.1.4　match()方法……278
9.2　正则表达式……279
9.2.1　简单的正则表达式……279
9.2.2　正则表达式：特殊字符……286
9.2.3　包含所有的可能性……293
9.2.4　正则表达式的分组……293
9.2.5　重用字符组……295
9.3　String 对象——split()、replace()、search() 和 match()方法……296
9.3.1　split()方法……296
9.3.2　replace()方法……298
9.3.3　search()方法……301
9.3.4　match()方法……301
9.4　使用 RegExp 对象的构造函数……304
9.4.1　验证电话号码……307
9.4.2　验证邮政编码……308
9.4.3　验证电子邮件地址……309
9.4.4　验证域名……310
9.5　小结……312
9.6　习题……313

第 10 章　日期、时间和计时器……314
10.1　世界时……315
10.2　网页中的计时器……325
10.2.1　一次性计时器……325
10.2.2　创建定期触发的计时器……329
10.3　小结……330
10.4　习题……331

第 11 章　使用 Cookie 存储信息……332
11.1　烘焙第一个 Cookie……332
11.1.1　新鲜出炉的 Cookie……332
11.1.2　Cookie 字符串……338
11.2　创建 Cookie……340
11.3　获取 Cookie 的值……344
11.4　Cookie 的局限性……349
11.4.1　用户可能禁用 Cookie……349
11.4.2　数字和信息的限制……349
11.5　Cookie 的安全性和 IE 6+……350
11.6　小结……351
11.7　习题……351

第 12 章　动态 HTML 和 W3C 文档对象模型……352
12.1　Web 标准……353
12.1.1　HTML……354
12.1.2　XML……355
12.1.3　XHTML……356
12.1.4　ECMAScript……356
12.2　文档对象模型……357
12.2.1　DOM 标准……357
12.2.2　DOM 与 BOM 的区别……359
12.2.3　将 HTML 文档表示树形结构……359
12.2.4　DOM 核心对象……361
12.2.5　DOM 对象及其属性和方法……363
12.3　操作 DOM……378
12.3.1　访问元素……379
12.3.2　改变元素的外观……379
12.3.3　定位和移动内容……384
12.3.4　示例：动态广告……388
12.4　DOM 和事件……392
12.4.1　DOM 事件处理……392
12.4.2　IE 中的事件处理……398
12.5　编写跨浏览器的 DHTML……401
12.6　JavaScript 与 XML……407
12.6.1　使用 JavaScript 操作 XML……408
12.6.2　在 IE 中获取 XML 文件……408
12.6.3　在 Firefox 和 Opera 浏览器中获取 XML 文件……411

12.6.4 跨浏览器读取 XML 文件……412
12.6.5 示例：显示每日信息……413
12.7 小结……420
12.8 习题……421

第 13 章 在 JavaScript 中使用 ActiveX 和插件……423
13.1 在非 IE 浏览器中检查并嵌入插件……423
13.1.1 在页面中添加插件……424
13.1.2 检查并安装插件……425
13.2 在 IE 中检查和嵌入 ActiveX 控件……428
13.2.1 在页面中添加 ActiveX 控件……428
13.2.2 安装 ActiveX 控件……430
13.3 使用插件和 ActiveX 控件……432
13.3.1 确定插件/ActiveX 控件的可用性……433
13.3.2 完成……434
13.3.3 测试表单的禁用……436
13.3.4 潜在的问题……437
13.4 小结……439
13.5 习题……440

第 14 章 Ajax……441
14.1 Ajax 的含义……441
14.1.1 Ajax 的作用……442
14.1.2 浏览器支持……443
14.2 使用 XMLHttpRequest 对象……443
14.2.1 跨浏览器问题……444
14.2.2 使用 XMLHttpRequest 对象……447
14.2.3 异步请求……449
14.3 创建简单的 Ajax 模块……450
14.3.1 规划 HttpRequest 模块……450
14.3.2 HttpRequest 构造函数……450
14.3.3 创建方法……452
14.3.4 完整的代码……453
14.4 使用 Ajax 验证表单字段……455
14.4.1 请求信息……455
14.4.2 接收到的数据……455
14.4.3 准备工作……456
14.5 注意事项……464
14.5.1 安全性问题……464
14.5.2 可用性问题……465
14.6 小结……473
14.7 习题……473

第 15 章 JavaScript 框架……474
15.1 选择框架……474
15.2 入门……475
15.2.1 安装框架……475
15.2.2 测试框架……477
15.3 深入分析 jQuery……482
15.3.1 选择元素……482
15.3.2 修改样式……484
15.3.3 创建、追加和删除元素……486
15.3.4 jQuery 事件模型和事件的处理……489
15.3.5 用 jQuery 重写 DHTML 工具栏……490
15.3.6 把 jQuery 用于 Ajax……496
15.4 Prototype……503
15.4.1 检索元素……504
15.4.2 处理样式……505
15.4.3 创建、插入和删除元素……506
15.4.4 使用事件……507
15.4.5 用 Prototype 重写 DHTML 工具栏……508
15.4.6 使用 Ajax 支持……512
15.5 MooTools……520
15.5.1 查找元素……520
15.5.2 修改样式……521
15.5.3 创建、插入和删除元素……522
15.5.4 使用和处理事件……522
15.5.5 用 MooTools 重写 DHTML 工具栏……524

15.5.6　MooTools 中的 Ajax 支持 …… 529
15.6　小结 …… 535
15.7　习题 …… 535
附录 A　参考答案 …… 536
附录 B　JavaScript 核心参考 …… 606
附录 C　W3C DOM 参考 …… 627
附录 D　Latin-1 字符集 …… 647

第 1 章

JavaScript 与 Web 概述

本章介绍 JavaScript 的含义及功能，以及开发 JavaScript 程序所需的工具。了解这些基础知识之后，本书其余章节将逐步介绍如何使用 JavaScript 为 Web 站点创建功能强大的 Web 应用程序。

实践是最好的学习方式。本书将使用 JavaScript 创建大量实用的示例程序，而本章将创建第一段 JavaScript 代码。

1.1 JavaScript 简介

本节简要介绍 JavaScript 的含义、发展历程、工作原理以及功能。

1.1.1 JavaScript 的含义

在购买本书之前，读者也许已经知道 JavaScript 是一种计算机语言。但是，什么是计算机语言？简言之，计算机语言就是一系列告诉计算机做某件事的指令。其中，“某件事”可以是各种操作，包括显示文本、移动图片或者请求用户输入信息等。通常，指令(也称为代码)自上而下地处理。简单地说，计算机阅读编写好的代码，确定要执行的操作，然后执行该操作。处理代码的过程称为“运行代码”或“执行代码”。

在自然语言中，为了冲一杯速溶咖啡，可以写出如下指令或代码：

(1) 将咖啡放入杯中。

(2) 在水壶中加水。

(3) 用水壶烧水。

(4) 如果水已煮沸，则将水倒入咖啡杯，否则继续等水煮沸。

(5) 品尝咖啡。

在执行这些指令时，从第一行(指令 1)开始执行，接着执行第二行(指令 2)，然后是下一行，直到结束。这就是大多数计算机语言的执行方式，JavaScript 亦是如此。但是有时可能需要改变代码执行的顺序，甚至跳过某些语句，详见第 3 章。

JavaScript 是一种解释型语言，而不是编译型语言。什么是解释型语言与编译型语言呢？

计算机并不真正理解 JavaScript。计算机需要解释 JavaScript 代码，并将其转换成计算机能理解的机器码。因此，JavaScript 是一种解释型语言。计算机只能理解机器码，机器码实际上是一串二进制数字(即 0 和 1 组成的字符系列)。当浏览器遇到 JavaScript 时，就将 JavaScript 代码传递给一个称为“解释器”的程序，解释器将 JavaScript 代码解释为计算机能理解的机器码。这有点类似于请一个翻译将英语翻译为西班牙语。注意，JavaScript 的转换在代码运行时进行，每次运行都需要重复地进行转换。JavaScript 并非唯一的解释型语言，VBScript 也是一种解释型语言。

对于编译型语言，代码在程序运行之前转换成机器码，而且这个转换过程只执行一次。程序员使用“编译器”将写好的代码转换成机器码，这些机器码由程序的用户运行。编译型语言包括 VB 和 C++。这类似于一个翻译将一份西班牙语文档翻译成英文。除非改变了文档，否则就可以重复使用翻译好的文档，而不必重新翻译。

此处，应该消除一个普遍存在的误解：JavaScript 并非 Java 语言的脚本版。事实上，除了名字中都带有 Java 外，二者并没有任何相同点。而且，JavaScript 比 Java 更容易学习和使用。实际上，JavaScript 是所有语言中最简单的，且具有令人叹服的强大功能。

1.1.2 JavaScript 与 Web

本书大部分 JavaScript 代码都运行在由浏览器加载的 Web 页面中。创建网页只需要一个文本编辑器，例如 Windows 记事本。浏览页面需要一个浏览器，例如 Firefox 或者 Internet Explorer，这些浏览器都内置了 JavaScript 的解释器。

实际上，JavaScript 语言最初出现在 Netscape Navigator 2 浏览器中。当时它叫 LiveScript。然而，由于当时 Java 技术如日中天，Netscape 公司觉得改为 JavaScript 这个名字会更引人注目。随着 JavaScript 的发展，微软公司决定在 IE 浏览器中加入微软品牌的 JavaScript，即 JScript。随后，Netscape、微软和其他公司都发布了更新的 JavaScript 版本并在各自的最新浏览器中将其纳入。尽管这些不同公司、不同版本的 JavaScript 在很多方面是一致的，但是也存在一些差异，如果不小心，这些差异可能会导致一些问题。本书前面几章创建的代码适用于大多数浏览器，包括 Firefox、Internet Explorer 以及 Safari。后面章节介绍的特性仅适用于 Firefox 3 或以上版本以及 Internet Explorer 7 和 8。本章后面将简要说明不同浏览器和不同版本的 JavaScript 可能导致的问题，以及如何处理此类问题。

JavaScript 有时称为 ECMAScript。ECMA (European Computer Manufacturers Association) 是一个制订信息和通信系统标准的私人组织，它们控制的一个标准就用于 JavaScript，称为 ECMAScript。它们的标准控制着该语言的各个方面，帮助确保不同 JavaScript 版本的相互兼容性。但是，ECMA 为 JavaScript 语言制订标准时，没有规定如何在特定的主机上使用它。这里的“主机(host)”是指主机环境，在本书中就是 Web 浏览器。其他主机环境包括 PDF 文件、Web 服务器、Macromedia Flash 应用程序等许多其他地方。本书仅讨论该语言在 Web 浏览器上的使用。制订网页标准的组织是 World Wide Web Consortium(W3C)，它们不仅为 HTML、XHML 和 XML 制订了标准，还为 JavaScript 在 Web 浏览器上如何与网

页交互制订了标准。后面章节将详细介绍这些内容。在学习更高级的内容之前，先讨论 JavaScript 的基本内容。本书的附录列出了 JavaScript 语言及其与 Web 浏览器交互的有用指南。

可以将本书创建的大多数包含 JavaScript 代码的网页存储在硬盘上，并可以用浏览器直接从硬盘加载网页，就像加载普通的文件(如文本文件)一样。但是，浏览因特网上的网站时，浏览器却不是这样加载网页的。因特网是一个庞大的计算机互连网络。网站访问是因特网上特定计算机提供的一种专门服务，提供网站访问服务的计算机称为 Web 服务器。

Web 服务器的基本功能是在其硬盘上保存大量的网页。通常，另一台计算机上的浏览器请求一个保存在该服务器上的网页时，Web 服务器将从硬盘上载入该网页，并且通过一个称为超文本传输协议(Hypertext Transfer Protocol，HTTP)的专用通信协议，将网页回传给发出请求的计算机。运行浏览器并发出请求的计算机称为客户机。客户机/服务器的关系有点像顾客与售货员的关系，顾客进入商店并对售货员说："请给我拿某一件东西"。售货员就为顾客提供服务，找到顾客想要的东西并递给顾客。对于 Web 来说，运行浏览器的客户机就是顾客，而提供所请求网页的 Web 服务器就是售货员。

在浏览器中输入一个网址时，浏览器怎么知道向哪个 Web 服务器请求页面呢？商店有"某市中央大街 45 号"这样的地址，Web 服务器也有地址，但它的地址没有街道名，而是使用 IP(Internet Protocol，因特网协议)地址，IP 地址唯一标识了因特网上的 Web 服务器。IP 地址由句点("."）分隔的 4 组数字组成，例如：127.0.0.1。

如果用户曾在网上冲浪，可能会对上述内容感到不解。因为 Web 站点的名字是 www.somewebsite.com，而不是 IP 地址。实际上，名称 www.somewebsite.com 是实际 IP 地址的一个用户友好名字，便于人们记忆。域名服务器(domain name server)可以将用户友好的名字转换为实际的 IP 地址，域名服务器由因特网服务提供商(ISP)建立。

另外，本书把 Internet Explorer 浏览器简称为 IE。

1.1.3　为什么选择 JavaScript

JavaScript 不是唯一的脚本语言，还有其他脚本语言，如 VBScript 和 Perl。为什么偏要选择 JavaScript 呢？

选择 JavaScript 的主要原因在于 JavaScript 使用广泛，易于获得。两大主流的浏览器 IE 和 Firfox 都支持 JavaScript，大部分其他浏览器也支持 JavaScript。所以，访问网站的大多数人都已经安装了某个 JavaScript 版本。当然，也可以使用浏览器的选项来禁用它。

上面提到的其他脚本语言，例如 VBScript，可以实现与 JavaScript 相同的功能，但是只有 Windows 操作系统上的 IE 支持 VBScript。而 Perl 根本不在 Web 浏览器中使用。

JavaScript 不只是能在网页中使用，它还有很多用途。例如，JavaScript 还可用于自动完成 Windows 中的计算机管理任务；在 Adobe Acrobat 的 PDF 文件中使用 JavaScript 的简化版本来控制页面的显示，就像控制网页一样。对于哪种脚本语言更有用或者更强大这样的问题，并没有唯一正确的答案。VBScript 几乎能完成绝大部分 JavaScript 能完成的工作，反之亦然。

1.1.4 JavaScript 的功能

JavaScript 最常见的用途是与用户交互、获取用户输入的信息以及验证用户的操作。例如，在页面上放置一个下拉菜单，用户可以从中选择一个要进入的网页。下拉列表可能是普通的老式 HTML，但需要使用 JavaScript 处理用户的输入。使用 JavaScript 与用户交互的另一个场合是表单(form)，它可获取用户输入的信息。表单也可能是普通的 HTML，如何检查用户输入的有效性呢？例如，有一个表单要提取用户的信用卡信息，以准备进行在线购物，在发货之前，需要确保用户填入了这些信息。有时，还需要检查输入数据的类型是否正确，例如，年龄应输入数值，而不是文本。

JavaScript 还可以实现各种特效。例如，当鼠标经过页面中的一个图片时，将图片切换成另一幅图片，这在网页的菜单中比较常见。另外，在浏览器状态栏(通常位于浏览器窗口的底部)或者页面上显示滚动信息的特效，也是用 JavaScrip 实现的，详见本书后面的内容。当鼠标悬停在一个菜单项上时，将该菜单下的子菜单展开，以提供一个选项列表，这也是由 JavaScript 驱动的一种常见特效。

浏览器和 JavaScript 的复杂性意味着，现代 JavaScript 绝不仅仅用于几个特效。还可以创建出更高级的应用程序，例如 Google Maps、Google Calendar 甚至功能全面的字处理器 Google Docs。这些应用程序提供了真正的服务。只需发挥一下想象力，JavaScript 就能带来许多惊喜。当然，JavaScript 有强大的用户接口，实际的数据处理在后台由强大的服务器完成。JavaScript 很强大，但仍有局限性。

1.1.5 创建 JavaScript Web 应用程序所需的工具

学习 JavaScript 并不需要购买昂贵的软件，可以在任何 PC 或 Mac 上免费学习它。本节将介绍可用的工具，以及如何获得它们。

1. 开发工具

要为 Web 应用程序创建 JavaScript 代码，仅需要一个简单的文本编辑器，例如 Windows 记事本，或者某种提供了行号、查找和替换等功能的高级文本编辑器。也可以使用专用 HTML 编辑器，在 HTML 编辑器中可以编辑 HTML 源代码，并在其中加入 JavaScript 代码。有许多非常优秀的工具专门用于开发 Web 应用程序，例如 Adobe 公司的卓越产品 Dreamweaver。但是，本书将着重介绍 JavaScript 本身，而不是特定的开发工具。在学习 JavaScript 基础知识时，手工编写代码往往比依赖开发工具更有收获，这将有助于提高对基础知识的理解，之后再了解超越工具功能范围的更高级的逻辑。在深入掌握了 JavaScript 的知识后，就可以使用开发工具了，那样可以节省时间，从而将更多精力投入到高级和有价值的编码上。

掌握了更高级的知识后，会发现使用网页编辑器将更容易完成任务，因为它包含的特性有检查代码的有效性，给重要的 JavaScript 保留字加上色彩，在把页面加载到浏览器上之前更便于浏览页面。一个免费的 Web 开发软件是 Microsoft 的 Visual Web Developer 2008 Express Edition，可以从 http://www.microsoft.com/express/vwd/上下载它。

还有许多其他非常不错的免费网页编辑器。用 Google 搜索“网页编辑软件”，会得到一个长长的软件列表。其中最著名的付费软件是 Adobe Dreamweaver。

随着所编写的 Web 应用程序复杂程度的提高，应使用有助于找出并改正错误的有效工具。程序员把代码中的错误称为 bug。尽管程序出了问题，但我们会把它们称为“未预料到的额外特性”。非常有效的改正错误的开发工具称为调试器，它可以在代码执行时监控代码中的操作。第 4 章将深入探讨错误和调试器开发工具。

2. Web 浏览器

除了编辑网页的软件之外，还需要浏览器来查看网页。如果预计用户将使用某些类型的浏览器来访问网站，那么最好在这些类型的浏览器下进行 JavaScript 开发。如本章后面所述，尽管浏览器大都基于标准，但它们浏览网页和处理 JavaScript 代码的方式并不相同。本书的所有实例都在 Firefox 3+和 IE 7 、IE 8 中进行了测试。如果某段代码与这些浏览器不兼容，本书就会对这些不兼容的代码的影响作出说明。

如果使用的是 Windows 系统，则肯定安装了 IE。如果未安装，请访问如下网址，以获取最新版本的 IE：www.microsoft.com/windows/internet-explorer/default.aspx。

可以到 www.mozilla.com/firefox/all.html 下载 Firefox。

大多数浏览器都默认启用了 JavaScript 支持，但也可能禁用了浏览器中的这个功能。所以在开始学习下一节的第一批 JavaScript 示例前，应该先检查一下浏览器是否启用了 JavaScript。

对于 Firefox，请选择浏览器上的 Tools | Options。在弹出窗口中，单击 Content 选项卡。在这个选项卡中，请确认选中 Enable JavaScript 复选框，如图 1-1 所示。

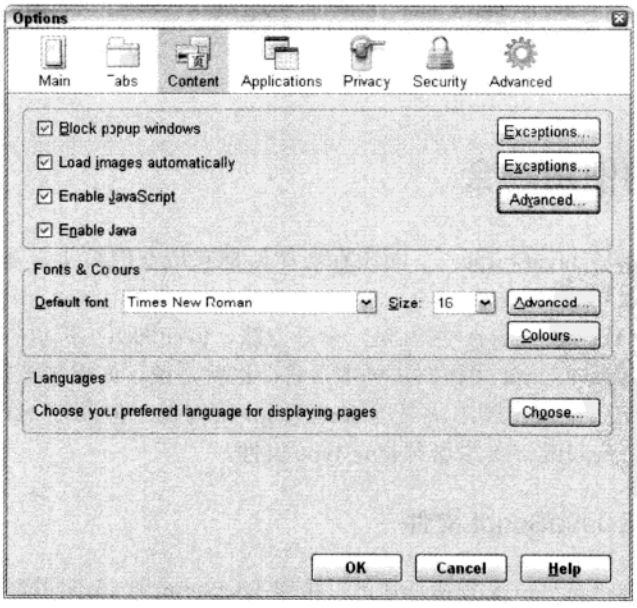

图　1-1

关闭 IE 的脚本功能稍微有点麻烦。从 IE 中选择 Tools | Internet Options。单击 Security 选项卡，检查 Internet 和 Local intranet 两项是否有自定义安全设置。如果有，就单击 Custom Level 按钮，找到 Scripting 一节，将 Active Scripting 设置为 Enable。

最后说明如何在浏览器中打开代码示例。对于本书中的代码，只需从硬盘上打开代码文件即可。打开代码文件有很多方法，比如在 IE 6 中，可以选择 File | Open，单击 Browse 按钮，找到代码文件。类似地，在 Firefox 中，也是选择 File | Open File，然后找到要打开的文件，再单击 Choose File 按钮即可。

而 IE 7 和 IE 8 有一个新的菜单结构，菜单中没有 Open File 选项。可以直接在浏览器的地址栏中输入盘符和冒号，例如，在地址栏中输入“C:”，以找到要打开的文件。在 Microsoft Windows 中，可以按下 Ctrl+O，以打开 Open File 菜单。另外，也可以切换回低版本 IE 的经典菜单。为此，可以单击 Tools | Toolbars，然后选择 Menu Bar 选项(如图 1-2 所示)。

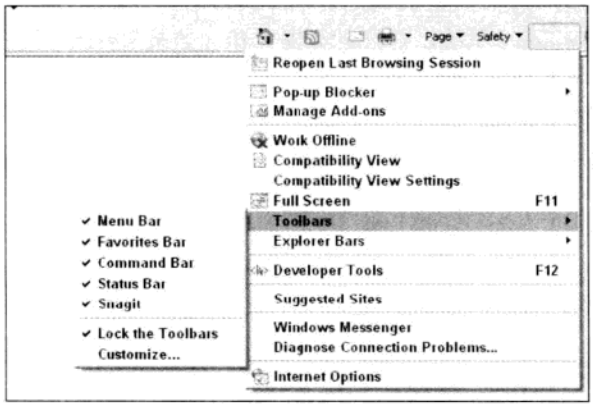

图 1-2

1.2 脚本的使用场合

前面讨论的都是 JavaScript，下面该看看如何把它放在网页上。本节将学习如何把 JavaScript 代码添加到网页上。

给 JavaScript 代码添加 type 属性是一个好方法，但在网页上可以省略该属性。IE 和 Firefox 等浏览器把 JavaScript 用作默认脚本语言，也就是说，如果浏览器遇到未设置 type 属性的<script>标记，就假定脚本块是用 JavaScript 编写的。但制订 HTML 标准的 W3C (World Wide Web Consortium)要求必须指定 type 属性。

1.2.1 链接外部 JavaScript 文件

使用<script>标记是另一个方法，它可以指定 JavaScript 代码不在网页上，而在一个单独的文件中。外部文件的扩展名应是.js。尽管这不是强制的，但使用该扩展名便于确定每

个文件包含的内容。

为了链接到外部 JavaScript 文件上，需要创建前面描述的<script>标记，并使用其 src 属性指定外部文件的位置。例如，假定创建了一个文件 MyCommonFunctions.js，该文件与网页位于同一个目录下，则为了把该文件链接到网页上，应使用如下<script>标记：

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

浏览器会读取这行代码，并把文件的内容作为网页的一部分包含进来。链接外部文件时，不能在<script>标记中放置任何代码，例如，下面的代码是无效的：

```
<script type="text/javascript" src="MyCommonFunctions.js">
var myVariable;
if ( myVariable == 1 )
{
        // do something
}
</script>
```

如果网页是 XHTML 文档，就可以省略</script>结束标记，而写成：

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

一般使用<script>标记加载本地文件(本地文件与网页位于同一台计算机上)。也可以指定文件的网址，从 Web 服务器上加载外部文件。例如，如果 MyCommonFunctions.js 文件加载到域名为 www.mysite.com 的 Web 服务器上，则<script>标记应如下所示：

```
<script type="text/javascript" src="http://www.mysite.com/MyCommonFunctions.js">
</script>
```

如果外部文件由其他人控制，则链接外部文件时要小心。因为这需要给控制外部文件的人授予控制和修改自己网页的权限，必须确保这些人是值得信任的。

1.2.2　使用外部文件的优点

外部文件的最大优势是促进了代码的重用。假定要编写一些较复杂的 JavaScript 代码，来构建许多页面都需要的一个通用函数。如果内联包含这些代码(布置在网页中，而不使用外部文件)，就需要把这些代码剪切并粘贴到使用它的每个网页上。如果从不需要修改代码，这就非常好。但现实是很可能需要在某个时刻修改或改进代码。如果把代码剪切并粘贴到 30 个不同网页上，就需要在 30 个不同的地方更新它们，这很麻烦！而使用一个外部文件，再把它添加到所有需要它的网页上，就只需更新一次代码，所有 30 个页面都会立即更新，这就简单多了！

使用外部文件的另一个优点是浏览器会缓存它们，就像在页面之间处理共享的图像一样。如果文件很大，这可以节省下载时间，减少占用的带宽。

1.3 第一个简单的 JavaScript 程序

关于 JavaScript 的话题已经谈了不少，下面分析如何将 JavaScript 添加到网页中。本节将编写第一个 JavaScript 代码段。

在网页中添加 JavaScript 代码，与在网页中添加其他任何 HTML 内容一样，也使用标记来标识脚本代码的开始和结束。该标记就是<script>，它告诉浏览器，在<script>标记和</script>结束标记之间的文本块并不是要显示的 HTML，而是需要处理的脚本代码。由<script>标记和</script>标记包围的代码块称为脚本块。

浏览器遇到<script>标记时，并不将脚本块中的内容显示给用户，而是使用浏览器内置的 JavaScript 解释器来运行代码中的指令。当然，代码中的指令有可能改变页面的显示方式或者改变显示的内容，但是脚本代码永远不会显示给用户。

可将<script>标记放在 HTML 页面的头部(<head>与</head>标记之间)，或放在页面体中(<body>和</body>标记之间)。尽管可将<script>标记放在其他地方，比如<html>标记之前或者</html>标记之后，但是 Web 标准禁止这样做，这是一种非常糟糕的做法。

<script>标记包含很多属性，其中最重要的属性是 type。前面曾提到，JavaScript 并不是唯一可用的脚本语言，不同的脚本语言需要按不同的方式处理。必须告诉浏览器，当前页面使用的是何种脚本语言，以便浏览器正确地加以处理。<script>开始标记的形式如下：

```
<script type="text/javascript">
```

在<script>标记中添加 type 属性是一个好方法，但在网页上可以省略该属性。IE 和 Firefox 等浏览器把 JavaScript 用作默认脚本语言，也就是说，如果浏览器遇到未设置 type 属性的<script>标记，就假定脚本块是用 JavaScript 编写的。但制订 HTML 标准的 W3C (World Wide Web Consortium)要求必须指定 type 属性。

下面看看第一个包含 JavaScript 代码的页面。

试一试　将页面背景色改成红色

下面的简单例子使用 JavaScript 改变了浏览器的背景色。在文本编辑器(如 Windows 记事本)中，输入下列代码：

```
<html>
<body bgcolor="WHITE">
<p>Paragraph 1</p>
<script type="text/javascript">
   document.bgColor="RED";
</script>
</body>
</html>
```

在硬盘上合适的目录中，将页面保存为 ch1_examp1.htm，然后在浏览器中加载该文件。浏览器会显示一个红色页面，且页面的左上角有一行文字 Paragraph 1。但是，上述代码不是把<body>标记的 bgcolor 属性设置为白色了吗？这是怎么回事呢？

页面包含在<html>和</html>标记之间，其中包含一个<body>元素。在定义<body>开始标记时，使用 bgcolor 属性将页面的背景色设置为白色。

```
<BODY bgcolor="WHITE">
```

接下来，使用<script>开始标记告诉浏览器，下面的代码是 JavaScript 代码：

```
<script type="text/javascript">
```

从这里一直到</script>结束标记的任何内容，浏览器都作为 JavaScript 代码来处理。在这个脚本块中，使用 JavaScript 将文档的背景色设置为红色。

```
document.bgColor="RED";
```

对网页编写脚本时，把页面称为文档。文档有很多属性，比如背景色属性 bgColor。要引用文档的属性，只需在 document 后加一个句点和属性名。如果现在对如何使用 document 对象不是很了解，请不要担心，本书后面将深入介绍它。

上面这行代码就是一个 JavaScript 语句例子。标记<script>和</script>之间的每行代码都称为语句。当然，有的语句需要跨越多行。

上面这行代码以分号(;)结束。在 JavaScript 中，分号用于表示语句的结束。实际上，JavaScript 对是否需要分号的要求很宽松，新起一行时，JavaScript 通常可以确定是否应开始一个新代码行。但是，最好在每行代码的结尾都加上分号，单个 JavaScript 语句写在一行上，而不要放在两行或多行上。另外，有时必须包含一个分号，详见本书后面的内容。

最后，使用结束标记</script>告诉浏览器，停止将文本解释为 JavaScript，而是解释为 HTML：

```
</script>
```

前面解释了代码的运行机制，但没有说明代码的执行顺序。浏览器载入一个网页时，它将遍历整个网页，自上而下地逐一呈现各个标记。这个过程称为解析(parsing)。浏览器自上而下解析页面，首先将遇到<body>标记，并将文档的背景色设置为白色。然后，浏览器继续解析页面，遇到 JavaScript 代码时，就将文档背景色设为红色。

试一试　**执行顺序**

现在扩展上面的示例，演示页面的解析过程。请在文本编辑器中输入如下代码：

```
<html>
<body bgcolor="WHITE">
<p>Paragraph 1</p>
<script type="text/javascript">
   // Script block 1
   alert("First Script Block");
</script>
<p>Paragraph 2</p>
<script type="text/javascript">
   // Script block 2
   document.bgColor = "RED";
   alert("Second Script Block");
</script>
<p>Paragraph 3</p>
</body>
</html>
```

将文件保存为 ch1_examp2.htm，然后用浏览器加载它。当载入页面时，首先看到第一段的内容，即 Paragraph 1，然后是第一个脚本块显示的消息框。这时，浏览器挂起(halt)页面解析过程，等待用户单击 OK 按钮。如图 1-3 所示，页面的背景是白色，这是在<body>标记中设置的，且仅显示第一段。

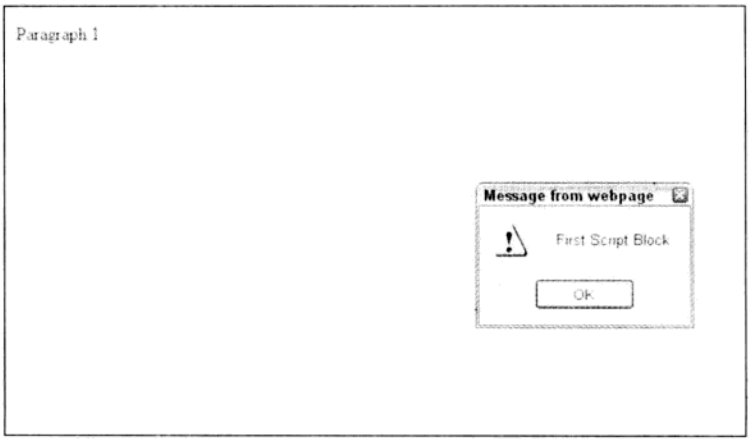

图 1-3

单击 OK 按钮之后，将继续进行解析。浏览器显示第二段，并到达第二个脚本块，它将背景色设置为红色，然后显示另一个消息框，如图 1-4 所示。

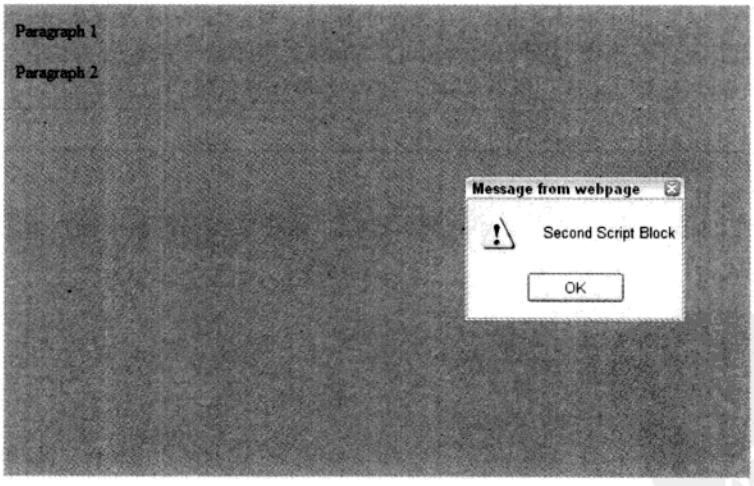

图 1-4

单击 OK，继续解析，显示第三段 Paragraph 3。网页解析完成，如图 1-5 所示。

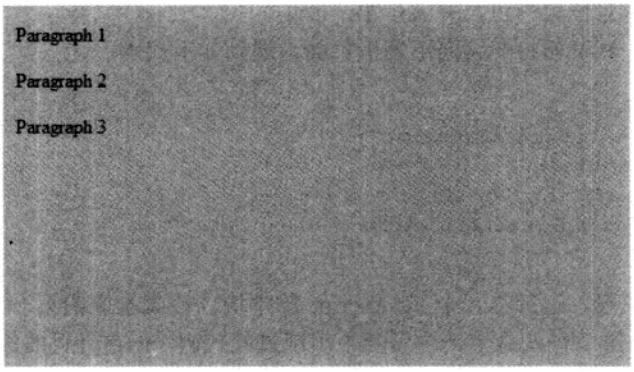

图　1-5

这个网页的第一部分与前一个示例相同。在<body>标记的定义中，将页面背景设为白色。然后在页面上显示 Paragraph 1。

```
<html>
<body bgcolor="WHITE">
<p>Paragraph 1</p>
```

第一个新块包含在第一个脚本块中：

```
<script type="text/javascript">
   // Script block 1
   alert("First Script Block");
</script>
```

这个脚本块包含两行新代码，第一行代码是：

```
// Script block 1
```

这是一行注释，可酌情添加。浏览器把双斜线(//)之后的任何内容视为注释，对此类内容不作任何处理。注释对程序员是非常有用的，注释可以说明代码的作用，用户再次阅读代码时，注释将有助于回顾代码的功能。

第二行代码中的 alert()函数也是一个新语句。在学习 alert()函数之前，先回顾一下函数的概念。

第 3 章将详细介绍函数。现在，只需要将函数视为可以完成某些任务的 JavaScript 代码块。根据了解的数学知识，读者可能已经知道函数的含义：函数提取一些信息，并进行处理，然后返回一个结果。函数简化了程序员的工作，因为程序员只需关注需要使用哪个函数，而无需关注函数是如何实现的。

确切地讲，alert()函数可以显示一个消息框，以便向用户发送通知或警告信息。在 alert()函数的括号中指定消息框中的消息，它称为函数的参数(parameter)。

alert()函数显示的消息框是模态的。这是一个应该掌握的重要概念。其含义是：除非用户单击了 OK 按钮来关闭消息框，否则消息框不会消失。实际上，页面解析过程在使用 alert()函数的代码处停止，直到关闭消息框之后才重启动。这对本示例非常有用，它可以展示解

析的结果：页面背景色为白色，并显示第一段。

单击 OK 后，浏览器将继续向下解析页面，遇到如下代码：

```
<p>Paragraph 2</p>
<script type="text/javascript">
   // Script block 2
   document.bgColor = "RED";
   alert("Second Script Block");
</script>
```

现在显示第二段，运行第二个 JavaScript 脚本块。该脚本块的第一行是另一个注释，浏览器会忽略它。第二行就是上一个示例中出现过的代码——它将页面的背景色改成红色。第三行是 alert()函数，它显示第二个消息框。页面解析将被挂起，直到单击 OK 按钮关闭消息框为止。

关闭消息框后，浏览器继续解析接下来的代码，显示第三段，然后网页就结束了。

```
<p>Paragraph 3</p>
</body>
</html>
```

这个示例的另一个要点是，通过 HTML 和 JavaScript 设置页面属性的区别，比如设置背景色。通过 HTML 设置属性是一种静态的方法，属性值只设置一次，且不会改变。通过 JavaScript 进行设置可以动态改变属性的值。此处，“动态”指某些元素的属性值或外观可在代码中改变。

上面的代码仅用于演示。实际上，如果要将页面的背景色设置为红色，可以将<body>标记的 bgcolor 属性设置为 RED，并不需要使用 JavaScript。当需要为页面添加某种智能或逻辑时才使用 JavaScript。例如，如果用户的屏幕分辨率过低，可以使用 JavaScript 调整页面上的内容。另外，使用 JavaScript 修改属性还可以实现一些特效，比如使页面从白色渐变为最终色彩。

试一试　在网页中显示结果

最后一个示例将介绍如何使用 JavaScript 把信息直接写到网页上。输出计算结果或使用 JavaScript 创建的文本时，这很有效，如下一章所述。现在仅使用 JavaScript 把“Hello World!”写入空白页面。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<body >

<p id="ResultsP"></p>

<script type="text/javascript">
   // Script block 1
   document.getElementById('ResultsP').innerHTML = 'Hello World!';
</script>
```

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

在硬盘的适当位置把页面保存为 ch1_examp3.htm，并加载到 Web 浏览器中，页面上就会显示“Hello World!”。使用 HTML 完成这一任务更容易，但这个技术在后面的章节中比较有用。

页面的第一部分同于前面的示例，只是添加了下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

这一行代码告诉浏览器，我们正在使用 XHTML，这是本书采用的标准。它并没有使代码发生任何变化，没有这行代码，页面照样也能工作。

考虑下面的代码：

```
<p id="ResultsP"></p>
```

注意使用 id 属性给<p>标记指定了一个 id，这个 id 在网页中必须是唯一的，因为 JavaScript 使用它在下面的代码行中标识指定的 HTML 元素：

```
document.getElementById('ResultsP').innerHTML = 'Hello World!';
```

如果觉得这很复杂，不必担心，后面将详细介绍其工作原理(详见第 6 章和第 12 章)。这段代码表示“提取 id 为 ResultsP 的文档元素，把该元素中的 HTML 设置为“Hello World!”。

在这个示例中，重要的是访问段落的代码要放在该段落的后面，否则，代码就会尝试访问一个不存在的段落，从而抛出一个错误。

1.4　浏览器以及兼容性问题

在前面的示例中可以看出，在 JavaScript 中，可以使用 document 的 bgColor 属性来改变网页的背景色。微软的 IE 浏览器和 Netscape 的浏览器都支持 document.bgColor 属性，所以上面的代码在这两种浏览器中都能正常运行。也可以说上面的示例是跨浏览器兼容的。但有时，某个浏览器支持的属性和语言特性，可能另一个浏览器并不支持。甚至有时同一种浏览器的不同版本所支持的属性和语言功能也不尽相同。

注意：

IE 和 Firefox 浏览器通常使用小数形式表示版本号。例如，Firefox 1.5。本书使用如下术语规范表示浏览器的版本号：1.x 表示主版本号为 1 的所有版本；1.0+表示 1.0 或 1.0 以上的版本。

开发 JavaScript 网页时，一个令人头痛的问题是不同 Web 浏览器之间存在差异，它们支持的 HTML 级别不同，JavaScript 解释器的功能也不同。例如，一个浏览器仅需要几行

代码就能移动图片，而另一个浏览器需要编写数十行代码，甚至根本无法实现这个功能。某个 JavaScript 版本有一个将文本转换为大写格式的方法，而另一版本的 JavaScript 则不存在这个方法。在 IE 和 Firefox 浏览器的每个新版本中，都会加入一些令人激动的新特性，以增强对 HTML 和 JavaScript 的支持。浏览器设计者越来越注重遵循 W3C 组织制订的 Web 标准。而且，只需使用一点小技巧，就可以使 JavaScript 代码既用于 IE 浏览器，又用于 Firefox 浏览器。

如果某网站的访问者(用户群)大多使用某种浏览器，就应支持这种浏览器。本书的代码适用于 IE 7 及以上版本或者 Firefox 2 及以上版本。

如果希望使网站达到专业水准，则需要对老版本的浏览器做适当的处理。可以确保代码向后兼容——即仅使用老版本浏览器支持的特性。但是，如果觉得不应仅局限于老版本的浏览器，就需要使页面正常降级。换句话说，尽管页面在老版本的浏览器中不能运行，但应使用户意识不到页面的失败，或者告知用户网站的某些功能与他的浏览器不兼容。否则代码会产生大量的错误消息，在页面上显示一些奇怪的东西，让人一头雾水！

那么，如何使网页正常降级呢？在完全加载或部分加载页面时，可以使用 JavaScript 判断载入页面的是哪种浏览器。利用这个信息，可以决定运行哪些脚本，或者将用户重定向到一个为特定浏览器编写的页面上。后续章节将说明如何确定浏览器支持的功能，并作出适当的处理，使页面可以在尽可能多的浏览器上运行。

1.5 小结

现在，读者对 JavaScript 及其功能有了初步的概念。本章介绍了以下内容：

- 本章介绍了浏览器解释网页的过程。浏览器逐一解析页面上的元素，将 HTML 元素呈现在页面上，解释和执行 JavaScript 代码。
- 与许多程序设计语言不同，JavaScript 只需要一个文本编辑器(例如 Windows 的记事本)就可以开始创建代码。积累了一定的经验后，功能强大的扩展工具就比较物有所值了。
- JavaScript 代码嵌入到网页的 HTML 中，用<script>标记来标识。与 HTML 一样，脚本也是自上而下逐句解释并执行的。

第 2 章

数据类型与变量

计算机的一项重要功能是处理和显示信息。处理信息指的是计算机按某种方式修改、解释或者过滤信息。例如，在一个在线银行网站上，用户要求提供其银行账户在上个月的支付明细。这时，计算机就获取该信息，过滤掉与上个月的支出无关的信息，最后将相关内容显示在网页上。某些情况下，只需要处理信息，而不必显示，而有时只获取信息，而不必处理。例如，在银行环境下，定期付款可以采用电子化处理和转账方式，并不需要人工干预，也不需要显示出来。

在计算机中，信息称为数据。数据有各种类型，比如数值、文本、日期和时间等。本章将详细讨论 JavaScript 如何处理数值和文本等数据。理解数据处理方式，是理解任何程序设计语言的基础。

首先介绍 JavaScript 中的各种数据类型。然后介绍如何将数据保存在计算机的内存中，以便在代码中反复使用。最后讨论 JavaScript 如何使用和处理数据。

2.1 JavaScript 中的数据类型

数据有各种形式，或者说类型。JavaScript 可以处理一些直接来自于现实世界的数据类型，例如数值和文本。其他数据类型有点抽象，但为编程提供了便利，例如对象数据类型，详见第 4 章。

很多编程语言都是强类型语言。在这些语言中，只要使用某段数据，就需要明确指定要处理什么类型的数据，并且数据的使用必须严格遵守其类型的规则。例如，不能将数值和单词加在一起。

而 JavaScript 是一种弱类型语言，对如何使用不同类型的数据并没有严格的要求。处理数据时，常常不需要指定其类型，JavaScript 自己会自动确定它们的类型。另外，同时使用不同类型的数据时，JavaScript 会在后台推断用户尝试执行的操作。

既然 JavaScript 对数据没有严格的要求，为什么还要讨论数据类型？为什么不直接使用数据，而不考虑数据的类型呢？

首先，尽管 JavaScript 擅长推断当前使用的数据类型，但有时它会推断错误，至少没

有执行用户预期的操作。在这些情况下，就需要明确告诉 JavaScript 数据的类型以及用法。因此，首先需要了解数据类型。

其次，了解数据类型可以让代码高效地使用数据。即使没有显式地声明数据的类型，数据操作及其结果也依赖于数据的类型。例如，两个数相乘是有意义的，但是两个文本字符串相乘是没有意义的。另外，两个数相加与两个字符串相加是完全不同的，两个数相加得到其和，而两个字符串相加将得到这两个字符串连成长字符串。

下面简要介绍几种常用的数据类型：数值、文本和布尔类型。然后分析如何使用这些类型。

2.1.1 数值数据

数值数据有两种形式：

- 整数：比如 145。整数可正可负，在 JavaScript 中，整数的取值范围很大：-2^{53}~2^{53}。
- 小数：比如 1.234，也称为浮点数。与整数一样，小数也可正可负，其取值范围也很大。

简言之，除非编写专用于科学计算的应用程序，否则不用关心 JavaScript 中可用数值的范围问题。另外，尽管在存储时，可以区别对待整数和浮点数，但实际上 JavaScript 将它们都视为浮点数。JavaScript 隐藏了其间转换的细节，通常我们不用考虑这个问题。只是将浮点数转换为整数时，需要使用四舍五入方法。本章后面将介绍该方法。

2.1.2 文本数据

包含一个或多个字符的文本称为字符串。把文本放在引号(")中，JavaScript 就会把它处理为文本，而不是代码。比如"Hello World"和"A"都是 JavaScript 可识别的字符串。可以使用单引号，因此，'Hello World'和'A'也是 JavaScript 可识别的字符串。但是，字符串的开头和结束必须使用相同的引号，因此，"A'和'Hello World"是无效的 JavaScript 字符串。

如果字符串中间有一个单引号，比如 Peter O'Toole，就可以用双引号将其括起来，"Peter O'Toole"是 JavaScript 可识别的字符串。但'Peter O'Toole'将产生一个错误。因为 JavaScript 把中间的单引号视为字符串结束符号，认为字符串是 Peter O，但无法识别 Toole'。

还有一个方法可以使 JavaScript 把字符串中间的单引号视为字符串的一部分，而不是字符串的结束符号，即使用反斜线(\)，它在 JavaScript 中有特殊的含义，称为转义字符。反斜线告诉浏览器，其后的下一个字符是文本的一部分，而不是字符串的结束符号。因此，'Peter O\'Toole'也是正确的。

如果要在双引号包含的字符串内部使用双引号，情况如何呢？可以将字符串用单引号括起来，所以'Hello "Paul" "是正确的，而"Hello "Paul" "是错误的。当然"Hello \"Paul\" "也是正确的。

JavaScript 有许多其他的特殊字符，它们不能直接输入，但可以用转义字符和其他字符合在一起，构成一个转义序列。它们与 HTML 的情况类似，例如 HTML 把一行中连续的多个空格忽略，因此用 来表示一个空格。同样在 JavaScript 中，有时不能直接使用

字符，而必须使用转义序列。表 2-1 列出了一些有用的转义序列。

表　2-1

转义字符序列	字 符 含 义
\b	退格字符
\f	换页符
\n	换行符
\r	回车符
\t	制表符(Tab)
\'	单引号
\"	双引号
\\	反斜线
\xNN	NN 是一个十六进制数，表示 Latin-1 字符集中的一个字符

上表中最后一行不太直观，它使用字符在 Latin-1 字符集中的编号来表示该字符，而没有使用字符本身表示。例如，如果想在字符串中包含版权符号(©)，就可以使用版权符号(©)在 Latin-1 字符集中的编号来表示："\xA9 Paul Wilton"。

同样，也可以使用 Unicode 转义序列来表示字符，其形式是\uNNNN，其中 NNNN 代表某字符的 Unicode 编号。例如，使用这种方法表示版权符号(©)，就是转义序列\u00A9。

2.1.3　布尔数据

在日常生活中，我们经常使用“是”与“否”、“正”与“负”、“真”与“假”。“真”与“假”对于数字计算机来说也是基本概念，计算机不理解“或许”的概念，只能理解“真”或“假”。实际上，“是”与“否”的功能非常有用，在 JavaScript 中有独立的数据类型，即布尔数据类型。布尔类型只有两个值：true 表示“是”，false 表示“否”。

JavaScript 中布尔数据的用途与现实生活类似：都是根据某个问题的答案做出选择。例如，如果问“这本书是关于 JavaScript 的吗？”答案可能是“是”或者“正确”。也可能说：“如果这本书不是关于 JavaScript 的，就把它放在一边。”这就是布尔逻辑语句(以发明人 George Boole 的名字命名)，它根据某个问题的答案是 true 或 false 来决定是否做某件事。在 JavaScript 中，可以使用这种布尔逻辑，让程序做出决策。第 3 章将详细介绍布尔逻辑。

2.2　变量——保存在内存中的数据

数据可以永久或临时存储下来。

重要的数据应永久保存，比如用户的银行账号信息。例如，Bloggs 女士从账户中取出 10 美元(或者 10 英镑、或 10 欧元)，银行就需要从她的账户中扣除 10 美元，并将新的余额永久存储下来。这样的重要信息一般存储在数据库中。

但是，有时并不需要永久存储数据，而只需临时保存一下。例如，Bloggs 女士从 BigBadBank 银行贷了一笔钱，她想看看还有多少未偿贷款。她进入银行在线贷款页面，单击一个链接，查询自己欠银行多少钱。这个数据永久存储在某个地方。假如 Bloggs 女士采用递增还款法来加快偿还速度，她在网页的文本框中输入递增后的还款额，页面就会显示她的贷款再过多长时间就能还清。这涉及几个复杂的运算，为简便起见，可以使代码分多个阶段计算结果，存储每个阶段的结果，然后计算出最终结果。但是，在计算完成并显示出结果后，就不再需要永久存储各个阶段的结果了。所以，各个阶段的中间结果并不需要使用数据库来存储，而需要使用变量来存储。为什么称之为变量？因为它可以保存会变化的临时数据。

使用变量的另一个好处是：变量保存在计算机的内存中，而持久存储的数据保存在磁盘或磁带上。这意味着变量中数据的存取会快许多。

变量非常适于保存临时数据。变量的生存期有限。用户关闭页面或者移动到新页面时，就会释放变量，除非采取其他措施将变量保存在某个地方。

每个变量都有一个名字，以便在代码的其他地方引用它。变量名必须遵循一些规则。

与许多 JavaScript 代码一样，变量名区分大小写。例如，变量 myVariable 与变量 myvariable 是不同的。即使是精通 JavaScript 的专家，也很容易在这一点上出错。

另外，某些单词或字符不能用作变量名。这些单词称为保留字。保留字是 JavaScript 留给自己用的，例如，var 或者 with。某些字符也不能用在变量名中，例如，&字符和百分号(%)字符。变量名中可以使用数字，但是变量名不能用数字开头。因此，myVariable101 是合法的，而 101myVariable 不合法。下面列出了一些例子。

下面的变量名是不合法的：

- with
- 99variables
- my%Variable
- theGood&theBad

下面的变量名是合法的：

- myVariable99
- myPercent_Variable
- the_Good_and_the_Bad

应该使用某种命名约定给变量命名。例如，变量名应描述变量存储的数据类型。为变量命名的方法有很多种——它们无所谓对错，只是最好保持变量命名规则的一致性。一种常见方法是匈牙利表示法，每个变量名的开头是一个表示数据类型的三字母标识符。例如，整型变量名以 int 开头；浮点数变量名以 flt 开头；字符串变量名以 str 开头等。只要变量名有意义并保持命名约定的一致性，使用哪种命名约定并不重要。

2.2.1 创建变量并赋值

在使用变量之前，应该用关键字 var 声明变量。这将告诉计算机保留相应的内存用于存储该变量的数据。例如，要声明一个新变量 myFirstVariable，可以使用如下代码：

```
var myFirstVariable;
```

注意，行尾处的分号并不是变量名的一部分，而表示 JavaScript 语句的结束。

声明了变量之后，就可以使用它来保存任何类型的数据。如前所述，许多强类型语言不仅要求声明变量，还要求指定变量存储的数据类型，例如数值或文本。然而，JavaScript 是一种弱类型语言，不需要指定变量所能保存数据的类型。

使用等号(=)将数据放到变量中的过程称为变量赋值。例如，如果要将数值 101 赋给变量 myFirstVariable，可以使用如下代码：

```
myFirstVariable = 101;
```

等号(=)用在为变量赋值时，通常称为赋值运算符。

试一试　声明变量

下面的例子声明了一个变量，然后将一些数据保存在变量中，最后访问该变量的数据。变量可以保存任何类型的数据，所保存的数据类型也可以改变。例如，可以先在变量中保存文本，再保存数值，这在 JavaScript 中是毫无问题的。在文本编辑器中输入下列代码，并保存为 ch2_examp1.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
</head>
<body>

<script type="text/javascript">

var myFirstVariable;

myFirstVariable = "Hello";
alert(myFirstVariable);

myFirstVariable = 54321;
alert(myFirstVariable);

</script>

</body>
</html>
```

一旦用 Web 浏览器加载这个文件，就会弹出一个包含 Hello 的警告框，如图 2-1 所示。这是代码中变量 myFirstVariable 在此时保存的值。

单击 OK 按钮，会弹出另一个包含 54321 的警告框，如图 2-2 所示。这是代码赋给变量 myFirstVariable 的新值。

图 2-1

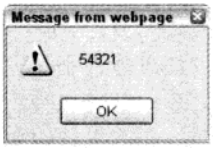

图 2-2

在上面的脚本块中，首先声明了变量：

```
var myFirstVariable;
```

此时，这个变量的值是 undefined。这是因为变量声明仅表明变量的存在，并没有保存任何实际的数据。这听起来有点古怪，但是 undefined 是 JavaScript 中一个基本值，还可以用这个值来进行比较。例如，可以用 undefined 值来检查变量是否包含实际的值，如果变量没有包含任何值，它的值就是 undefined。下面这行代码将一个字符串 Hello 赋给变量 myFirstVariable。

```
myFirstVariable = "Hello";
```

此时，变量 myFirstVariable 有了一个字面值(literal)。字面值就是实际的数据，而不是通过计算得到的值或者来自其他变量的值。只要使用了数值字面值或者字符串字面值，就可以用一个包含数值或字符串数据的变量来替代它。例如，下面的代码行在第 1 章介绍的 alert()函数中使用变量 myFirstVariable。

```
alert(myFirstVariable);
```

这行代码将弹出第一个警告框。接着在变量中存储一个新值，这次是一个数值。

```
myFirstVariable = 54321;
```

此时 myFirstVariable 原来的值就永远丢失了。用于存储值的内存空间会由 JavaScript 自动释放，这个过程称为“垃圾回收”。当 JavaScript 检测到某个变量的内容不再可用时，例如给变量分配了一个新值，它将执行垃圾回收过程，使被占用的内存恢复为可用状态。如果没有自动垃圾回收机制，计算机的内存将被不断消耗并最终用尽，最后系统将陷入停顿。但是，垃圾回收并不总是那么高效，有可能直到加载新页面时才会执行。

为了说明变量 myFirstVariable 保存了新值，再次使用 alert()函数显示该变量的新值。

```
alert(myFirstVariable);
```

2.2.2 用其他变量的值为变量赋值

前面用数值或字符串为变量赋值，还可以用另一个变量保存的数据为变量赋值？这非常简单，方法与用字面值为变量赋值一样。例如，声明两个变量 myVariable 和 myOtherVariable，并将 myOtherVariable 赋值为 22，代码如下：

```
var myVariable;
var myOtherVariable;
```

```
myOtherVariable = 22;
```

然后使用下面的代码，将变量 myOtherVariable 的值 22 赋给变量 myVariable。

```
myVariable = myOtherVariable;
```

试一试　用其他变量的值为变量赋值

下面的例子用其他变量的值为当前变量赋值：

(1) 在文本编辑器中输入下列代码，并保存为 ch2_examp2.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script language="JavaScript" type="text/javascript">

var string1 = "Hello";
var string2 = "Goodbye";

alert(string1);
alert(string2);

string2 = string1;

alert(string1);
alert(string2);

string1 = "Now for something different";

alert(string1);
alert(string2);

</script>

</body>
</html>
```

(2) 在浏览器中加载该页面，会弹出 6 个警告框。

(3) 单击每个警告框中的 OK 按钮，会弹出下一个警告框。前两个警告框分别显示 string1 和 string2 的值——Hello 和 Goodbye。然后将 string1 的值赋给 string2，接下来的两个警告框将显示 string1 和 string2 的值，现在它们都是 Hello。

(4) 最后修改 string1 的值。注意，string2 的值不受影响。最后的两个警告框分别显示 string1 的新值(Now for something different)，以及 string2 未改变的值(Hello)。

首先在脚本块中声明两个变量，string1 和 string2。注意，在声明的同时对变量进行了赋值。这是一种简便的写法，称为变量的初始化，可减少不少输入量。

```
var string1 = "Hello";
var string2 = "Goodbye";
```

注意，这种简化写法可用于所有数据类型，而不仅仅是字符串。下面两行代码使用 alert()

函数分别显示了两个变量的当前值。

```
alert(string1);
alert(string2);
```

接下来，将 string1 的值赋给 string2。为了验证赋值操作，再次使用 alert()函数来显示两个变量的值。

```
string2 = string1;

alert(string1);
alert(string2);
```

随后，将一个新值赋给 string1：

```
string1 = "Now for something different";
```

此时，变量 string2 保留当前值，上一步将变量 string1 的值赋给了 string2，所以 string2 拥有自己的一份数据副本。后面的章节将提到，并不是每个变量都拥有自己的数据副本。但一般来说，基本数据类型(比如文本和数值)在赋值时总是复制副本。而较复杂的数据类型在赋值时会被共享，而不是复制副本，比如第 4 章介绍的对象。例如，将一个变量的值"Hello"赋给另外 5 个变量，则"Hello"将具有原始数据和 5 个独立的数据副本。但是如果变量包含的是对象，而不是字符串，在执行同样的操作时，数据就只有一个副本，这 6 个变量都共享同一个数据。使用其中的一个变量修改对象的数据，则其他 5 个变量的值也会随之改变。

最后使用 alert()函数显示两个变量的当前值。

```
alert(string1);
alert(string2);
```

2.3 使用数据——计算数值及基本字符串操作

下面将介绍本章的主题：数据及其使用方式。前面介绍了如何声明变量，它们如何存储信息，但我们还没有用变量来做什么有用的事情——为什么要使用变量？

变量可以临时保存信息，这些信息可用于数学计算、构造文本消息或者处理用户的输入。变量有点类似于普通袖珍计算器上的“内存存储”按钮，可用它来作合计。例如，先把要支出的费用加起来，再保存在临时内存中。当把收入总和累加起来后，就可以减去刚才保存的支出，得到余额。变量的作用与其类似，可以获取必要的用户输入信息，保存在变量中，然后使用变量来运算。

本节将介绍如何将值保存在变量中，进行数值运算和文本操作。

2.3.1 数值计算

JavaScript 支持各种基本的数学运算，如加、减、乘、除。这些基本数学函数用符号表

示：加(+)、减(－)、乘(*)、除(/)。这些符号对给定数值进行某种运算，因此称为运算符。换句话说，它们执行某种运算或操作，返回一个结果。在可以使用数值或变量的几乎所有地方，都可以使用这些计算的结果。

假如要计算购物清单的总额，可以按如下方式计算：

购物总额 = 10 + 5 + 5

或者，可以直接计算总和：

购物总额 = 20

那么，在 JavaScript 中该怎么做呢？实际上很简单，只需要用变量来保存最后的总额：

```
var TotalCostOfShopping;
TotalCostOfShopping = 10 + 5 + 5;
alert(TotalCostOfShopping);
```

首先声明一个变量 TotalCostOfShopping，用于保存总额。

第二行的代码为 10+5+5。这是一个表达式。这个表达式的值赋给 TotalCostOfShopping 变量时，JavaScript 将自动计算表达式的值(为 20)，并保存在变量中。注意，等号运算符(=)告诉 JavaScript 将计算结果保存在变量 TotalCostOfShopping 中。这通常称为用计算的值为变量赋值，因此，等号运算符(=)称为赋值运算符。

最后，在警告框中显示变量的值。

减法和乘法运算符的用法与加法运算符类似，除法运算略有不同。

试一试　数值计算

下面是一个使用除法运算符的例子。

(1) 在文本编辑器中输入以下代码并保存为 ch2_examp3.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script language="JavaScript" type="text/javascript">
var firstNumber = 15;
var secondNumber = 10;
var answer;
answer = 15 / 10;
alert(answer);

alert(15 / 10);

answer = firstNumber / secondNumber;
alert(answer);

</script>

</body>
```

```
</html>
```

(2) 在 Web 浏览器中加载该文件，将会接连弹出三个警告框，每个警告框都显示值为 1.5。这是三个计算的结果。

(3) 在脚本块中首先声明了三个变量，并为前两个变量赋值。后面要使用这两个值。

```
var firstNumber = 15;
var secondNumber = 10;
var answer;
```

(4) 接下来将表达式 15/10 的计算结果赋给变量 answer，并用警告框显示该变量的值。

```
answer = 15 / 10;
alert(answer);
```

这个示例演示了执行计算的一种方式，但实际上很少这样做。

为了说明表达式可以用在任何使用数值或变量的地方，下面将表达式 15/10 的计算结果放在 alert()函数中，直接显示计算结果。

```
alert(15 / 10);
```

最后，用两个变量 firstNumber(值为 15)、secondNumber(值为 10)来做同样的计算。本例将表达式 firstNumber/secondNumber 的计算结果存储在变量 answer 中。为了对代码进行验证，仍然使用 alert()函数来显示变量 answer 的值。

```
answer = firstNumber / secondNumber;
alert(answer);
```

大多数计算都用第三种方式进行，即使用变量(或数值与变量)进行计算，然后将结果保存在另一个变量中。原因是，如果使用字面值(实际值，比如 15/10)进行计算，就可以先计算出表达式的值，而不是让 JavaScript 计算它。例如，不是计算 15/10 的值，而可以直接在代码中写 1.5。毕竟，JavaScript 要做的计算越多，它的速度就越慢，即使一个简单的计算也会稍微加重 JavaScript 的负担。

这样做的另一个好处是代码便于理解。例如，代码中的 69.231 肯定比 1.5 * 45 - 56 / 67 + 2.567 更简单。还可将变量命名为 PricePerKG，使不熟悉代码的人更容易理解代码。

递增和递减运算符

使用数学运算符的运算非常常见，所以它们有自己的运算符。本节介绍递增和递减运算符，它们分别用两个加号(++)和两个减号(--)表示。它们分别把变量的值加 1 和减 1。也可以使用普通的加(+)和减(-)运算符实现这个功能，例如：

```
myVariable = myVariable + 1;
myVariable = myVariable - 1;
```

注意：

可以使用变量参与某个表达式的运算，然后将运算结果赋给这个变量。

使用递增和递减运算符，可将上面两行代码简化为：

```
myVariable++;
myVariable--;
```

结果是一样的——myVariable 的值加 1 或减 1——但代码变短了。熟悉了这种语法后，代码会非常简洁，且容易理解。

现在用户可能觉得这两个运算符比较刺眼，但第 3 章介绍如何多次执行一段代码时，会发现这两个运算符非常有用，且应用很广泛。实际上，++运算符的应用非常广泛，所以有一门程序设计语言用它来命名——C++，即 C++是给 C 加 1(当然，这只是程序员的一点小幽默)。

可以把++和--放在变量之后，也可以把它们放在变量之前。例如：

```
++myVariable;
--myVariable;
```

当++和--仅作用于某个变量时，把它们放在变量之前还是之后通常是没有区别的。但可能在表达式中与其他运算符一起使用++或--：

```
myVar = myNumber++ - 20;
```

这行代码从变量 myNumber 中减去 20，然后把 myNumber 加 1，最后把得到的结果赋给变量 myVar。如果把++运算符放在变量 myNumber 的前面，则代码如下：

```
myVar = ++myNumber - 20;
```

这行代码先把变量 myNumber 加 1，然后从中减去 20，二者有微妙的差别，但在某些情况下，这个差别却非常重要。例如下面的代码：

```
myNumber = 1;
myVar = (myNumber++ * 10 + 1);
```

变量 myVar 的值是多少？由于++作为后缀放在 myNumber 变量的后面，所以递增操作在计算表达式之后进行。因此，这个等式的含义是：将 myNumber 乘以 10，加上 1，再把 myNumber 加 1。

```
myVar = 1 * 10 + 1 = 11
```

变量 myNumber 加 1 后值为 12，但这个运算在将运算结果 11 赋给变量 myVar 之后执行。再看看下面的代码：

```
myNumber = 1;
myVar = (++myNumber * 10 + 1);
```

这次 myNumber 先加 1，再乘以 10，最后加上 1。

```
myVar = 2 * 10 + 1 = 21
```

人们很容易忽视如此细微的差异，从而造成代码中的错误。所以，应该避免使用这样

的语法。

接下来介绍另一个运算符+=。这个运算符用于将某个变量的值加上一个数。例如：

```
myVar += 6;
```

等价于：

```
myVar = myVar + 6;
```

减法和乘法也可以使用类似的运算符，例如：

```
myVar -= 6;
myVar *= 6;
```

等效于：

```
myVar = myVar - 6;
myVar = myVar * 6;
```

2.3.2 运算符的优先级

执行特定功能的符号称为运算符，比如加号(+)把两个数加在一起，减号(-)从一个数中减去另一个数。并非所有运算符的优先级都是相同的，某些运算符具有较高的优先级，会得到优先处理。下面的简单例子说明了这个问题：

```
var myVariable;

myVariable = 1 + 1 * 2;

alert(myVariable);
```

如果输入上面的代码，警告框中显示的变量 myVariable 的值是多少？我们希望先执行 1+1=2，然后执行 2*2=4，结果是 4。实际上，警告框中显示的运算结果却是 3。怎么回事？难道 JavaScript 加错了？

根据数学知识，就可以知道得到 3 的原因。JavaScript 先执行 1*2=2，再执行加法 1+2=3，所以最终结果是 3。

为什么？因为乘法的优先级高于加法。等于运算符(称为赋值运算符)的优先级最低，它总是最后执行。

加法和减法运算符的优先级相同，先执行哪一个？JavaScript 遇到优先级相同的运算符时，将根据运算符出现的顺序，按从左到右的次序执行运算。这也适用于乘法和除法，因为它们也有相同的优先级。

试一试　华氏度转换为摄氏度

下面是一个稍微复杂的例子——将华氏温度转换为摄氏温度。在文本编辑器中，输入下面的代码，并保存为 ch2_examp4.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">
// Equation is °C = 5/9 (°F - 32).
var degFahren = prompt("Enter the degrees in Fahrenheit",50);
var degCent;

degCent = 5/9 * (degFahren - 32);

alert(degCent);

</script>

</body>
</html>
```

在浏览器中加载该页面，将会看到一个如图 2-3 所示的提示框，它要求输入要转换的华氏温度。提示框中已填充了默认值 50。

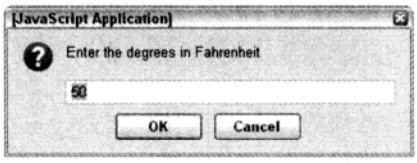

图　2-3

如果使用默认值，单击 OK 按钮，弹出的警告框就显示 10，这是华氏 50 度转换成摄氏度的结果。

重载页面，修改提示框中的值，看看会得到什么结果。例如，把值改为 32，这次警告框显示 0。

这仍然是一个较简单的例子。我们并没有检查输入的数据，所以允许把 abc 输入为华氏温度值。后面的 2.5 节“数据类型转换”将介绍如何找出作为数值数据输入的非法字符。

试一试　IE8 的安全问题

把页面加载到 Internet Explorer 8 (IE8)中时，可能会看到如图 2-4 所示的安全警告问题，而不显示提示窗口。

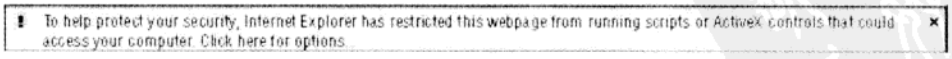

图　2-4

此时，需要修改 IE8 的安全设置，把 file://*..host 添加为一个信任站点，步骤如下：

(1) 打开 IE8，从 Tools 菜单栏中选择 Internet Options 菜单，如图 2-5 所示。

(2) 单击 Security 选项卡，再单击绿色的 Trusted Sites 按钮，如图 2-6 所示。

(3) 单击 Sites 按钮，在 Add This Website to the Zone 文本框中输入 file://*..host，如图 2-7 所示。

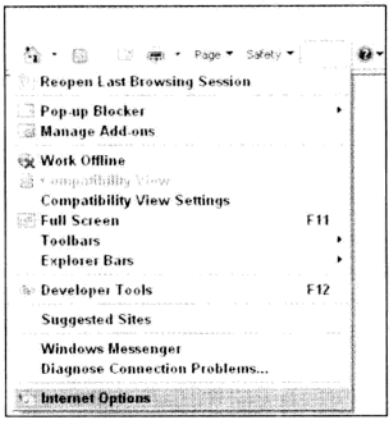

图 2-5

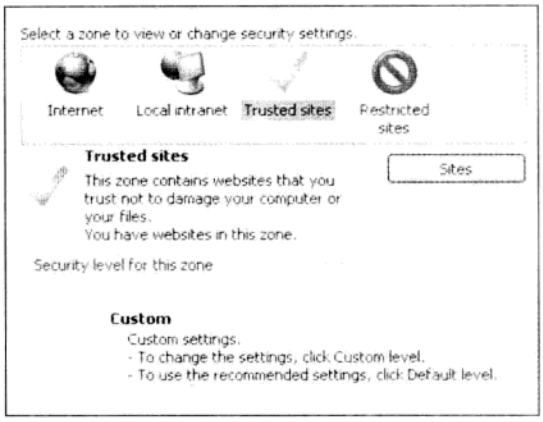

图 2-6

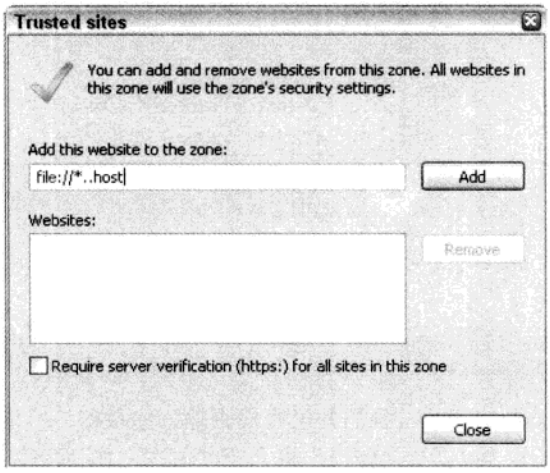

图 2-7

(4) 取消对 Require Server Verification 复选框的选择，单击 Add 按钮，再单击 Close 按钮。

(5) 单击 Internet Options 对话框中的 OK 按钮，返回网页，按下 F5 键刷新页面。示例现在可以正常工作了。

在上例中，脚本块的第一行是一个注释，因为它以双斜线(//)开头。注释列出了将华氏温度转换为摄氏温度的计算公式。代码就是根据这个公式编写的。

```
// Equation is °C = 5/9 (°F - 32).
```

接着将这个公式转换为 JavaScript 代码。首先声明两个变量：degFahren 和 degCent。

```
var degFahren = prompt("Enter the degrees in Fahrenheit",50);
var degCent;
```

在声明变量 degFahren 时，没有为变量 degFahren 赋予字面值，而是使用 prompt()函数

提示用户输入一个值。prompt()函数的用法与 alert()函数很类似，但除了显示信息外，它还包含一个供用户输入数值的文本框。这个值将保存在变量 degFahren 中。prompt()函数的返回值是一个文本字符串，把它用作数值时，JavaScript 将它隐式转换成数值。如 2.5 节“数据类型转换”所述。

prompt()函数需要两个信息：

- 要显示的文本，通常用于提醒用户输入信息。
- 第一次显示提示对话框时，在输入框中包含的默认值。

这两个信息必须按指定的顺序给出，且用逗号分隔。如果在弹出提示对话框时，不需要设置输入框的默认值，则把第二个信息设置为空字符串""。

在上面的代码中，提示信息是"Enter the degrees in Fahrenheit"，输入框中的默认值是 50。

接着，脚本块用 JavaScript 代码表示转换公式，并将计算结果保存在变量 degCent 中。JavaScript 的转换公式与注释中的公式很类似，只是用变量 degFahren 代表华氏温度(ºF)，用变量 degCent 代表摄氏温度(ºC)。

```
degCent = 5/9 * (degFahren - 32);
```

等号右边的表达式提供了一些重要信息。首先，与数学运算一样，JavaScript 等式也是从左到右地计算，至少基本的算术运算如+、－等是这样。另外，JavaScript 中的运算与数学运算一样也有优先级。

从左边开始，JavaScript 先计算出 5/9 = 0.5556(近似值)。接着计算乘法，但是等式的最后一项 degFahren－32 放在括号中。这就提高了最后一项的优先级，所以 JavaScript 先计算 degFahren－32，再进行乘法运算。例如，假如 degFahren 的值为 50，则 JavaScript 先计算 (degFahren－32) = (50－32) = 18，然后计算乘法 0.5556 * 18，结果约等于 10。

如果未使用括号，代码就会变成：

```
degCent = 5/9 * degFahren - 32;
```

JavaScript 仍然先计算 5/9 = 0.5556(近似值)，但接下来计算乘法，因为乘法运算的优先级比减法高。如果 degFahren 的值为 50，则先计算 5/9 * 50 = 27.7778，再减去 32，得到错误的结果：－4.2221。

在脚本代码块的最后，使用 alert()函数显示转换结果。

```
alert(degCent);
```

前面简要介绍了 JavaScript 的基本运算。第 4 章将介绍 Math 对象，来实现更复杂的运算。

2.3.3　基本的字符串操作

前一节介绍了文本或字符串类型和数值数据。数值数据有相应的运算符，字符串也有类似的运算符。本节介绍一些基本的字符串操作。第 4 章将深入介绍字符串，第 8 章将介绍处理字符串的高级技术。

在 JavaScript 中，经常需要将两个字符串连接成一个字符串，这称为连接(Concatenation)。例如，将两个字符串"Hello "和"Paul"连接成一个字符串"Hello Paul"。怎么实现连接呢？非常简单，使用+运算符。对于数值，+运算符把两个数加在一起，而对于字符串，+运算符将把两个字符串连在一起。

```
var concatString = "Hello " + "Paul";
```

变量 concatString 中保存的字符串是"Hello Paul"。注意，字符串"Hello "的最后一个字符是一个空格，如果忘记输入空格，连接后的字符串将是"HelloPaul"。

试一试　连接字符串

下面的代码用+运算符来连接字符串。

(1) 输入下列代码，并保存为 ch2_examp5.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var greetingString = "Hello";
var myName = prompt("Please enter your name", "");
var concatString;

document.write(greetingString + " " + myName + "<br>");

concatString = greetingString + " " + myName;

document.write(concatString);

</script>

</body>
</html>
```

(2) 在 Web 浏览器中加载该文件，将会看到一个提示对话框，要求输入名字。

(3) 输入名字后，单击 OK 按钮，网页上将显示两条问候语和名字。

脚本块先声明了三个变量。把第一个变量 greetingString 设置为字符串值，第二个变量 myName 设置为用户在提示框中输入的内容，第三个变量 concatString 则未作初始化，它在后面的代码中用来保存连接字符串的结果。

```
var greetingString = "Hello";
var myName = prompt("Please enter your name", "");
var concatString;
```

前一章介绍了如何用 document 代表网页，它有许多不同的属性，如 bgColor。还可以用 document 在网页中直接写入文本或 HTML，方法是使用单词 document 后跟一个句点和 write()。document.write()的使用方式与 alert()函数一样，也是把要在网页中显示的文本放在

write 后面的括号中。现在不必过于关注 document.write()，因为第 4 章将详细介绍它。现在只需在代码中使用 document.write()将表达式的结果写在页面上：

```
document.write(greetingString + " " + myName + "<br>");
```

向页面写入的表达式由变量 greetingString 的值、一个空格(" ")、变量 myName 的值和 HTML 标记
(产生一个换行符)连接而成。例如，如果在提示框中输入 Paul，这个表达式的值将是：

```
Hello Paul<br>
```

下一行代码是一个类似的表达式。这次仅连接了变量 greetingString 的值、一个空格(" ")和变量 myName 的值，并把这个表达式的结果保存在变量 concatString 中。最后使用 document.write()将变量 concatString 的内容输出在页面上。

```
concatString = greetingString + " " + myName;
document.write(concatString);
```

2.3.4　字符串与数值的混合操作

如何在表达式中混合使用数值和字符串。在前面的温度转换例子中，只显示了数字，而没有说明其含义。可以在数字的前面加上描述文本，来说明其含义，例如："The value converted to degrees centigrade is 10。"

混合使用数值和文本实际上很简单，只需使用+运算符把它们连接在一起。同时操作数值和字符串时，JavaScript 知道我们不是要进行数值计算，而是想把数值当作字符串，与文本连接在一起。例如，要把文本 My age is 和数值 101 连接在一起，只需使用如下代码：

```
alert("My age is " + 101);
```

这将弹出一个警告框，显示 My age is 101。

试一试　将温度转换为便于阅读的形式

下面尝试在温度转换的例子中，将字符串和数值连接在一起，输出一些描述信息和转换结果。修改幅度很小，在文本编辑器中加载 ch2_examp4.htm，修改最后一行代码，并保存为 ch2_examp6.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var degFahren = prompt("Enter the degrees in Fahrenheit", 50);
var degCent;

degCent = 5/9 * (degFahren - 32);
```

```
alert(degFahren + "\xB0 Fahrenheit is " + degCent + "\xB0 centigrade");

</script>

</body>
</html>
```

在浏览器中载入这个页面。在提示框中单击 OK 以提交值 50，结果将如图 2-8 所示。

图 2-8

除了下面的代码行之外，本例与 ch2_examp4.htm 完全相同：

```
alert(degFahren + "\xB0 Fahrenheit is " + degCent + "\xB0 centigrade");
```

因此，我们将仅关注这行代码。alert()函数包含一个表达式，下面仔细分析它。

首先是变量 degFahren，它包含数值数据。将它与字符串"\xB0 Fahrenheit is "连接在一起。如果将数值与字符串加在一起，JavaScript 知道应把二者连接起来，而不是计算二者的和，所以自动把包含在变量 degFahren 中的数值转换成字符串。接下来把这个字符串与包含数值数据的变量 degCent 连接起来。JavaScript 也会把这个变量 degCent 的值转换成字符串。最后加上字符串"\xB0 centigrade"。

另外，转义序列(\xB0)用来在字符串中插入温度字符。前面介绍过，\xNN 可用来插入不能直接输入的特殊字符。这里，NN 是一个十六进制数，表示 Latin-1 字符集中的字符。当 JavaScript 遇到\xB0 时，它将显示 B0 编码所代表的字符。

在使用特殊字符时，注意它们未必跨平台兼容。在 Windows 平台中，\xNN 代表某个字符，但在 Mac 或 Unix 系统中，它可能代表另一个字符。

第 4 章将介绍更多的字符串操作技术，例如查找字符串，以及在字符串中插入其他字符串。第 8 章还将介绍有关字符串操作的高级技术。

2.4 数据类型转换

如前所述，把数值和字符串加在一起时，JavaScript 将把数值转换成字符串，然后连接它们。JavaScript 通常能酌情正确地转换数据类型。但有时需要显式地转换数据类型。例如，有时需要将字符串转换成数值。特别是当使用表单(form)收集用户数据时。用户输入的任何数据都被视为字符串，即使这些数据包含数值(例如年龄)也被视为字符串。

数据类型的转换是很重要的。考虑下面的情况：使用表单来获取用户输入的两个数字，并计算它们的和。从表单获得的两个数都是字符串，例如"22"和"15"。如果使用"22"+"15"计算这两个数的和，将得到"2215"。因为 JavaScript 认为我们要把两个字符串连起来，而不

是求两个数之和。有时顺序不同，结果也会不同。例如：

```
1 + 2 + 'abc'
```

得到的字符串是“3abc”，而

```
'abc' + 1 + 2
```

得到的字符串是“abc12”。

本节介绍两个可以将字符串转换为数值的函数：parseInt()和 parseFloat()。

先讨论 parseInt()函数，它将字符串转换成整数。这个函数的名称有点让人疑惑，为什么不是 convertToInt()，而是 parseInt()呢？该函数名来自于这个函数的工作方式。它解析字符串的每个字符，检查该字符是否是一个有效的数字。如果是，parseInt()函数将使用这个数字来生成字符串对应的数值。否则，命令就停止转换，并返回之前转换的数值。

例如，对于函数 parseInt("123")，JavaScript 将把字符"123"转换为数值 123。而对于函数 parseInt("123abc")，JavaScript 也将返回数值 123。当 JavaScript 解析到字符 a 时，将认为字符串的数字部分已经结束，并把 123 作为字符串"123abc"的整数部分。

parseFloat()函数的工作机制与 parseInt()类似，只是它返回浮点数，并把小数点视为数值的一部分进行解析。

试一试　将字符串转换为数值

下面是一个使用 parseInt()和 parseFloat()函数的例子。输入如下代码，并保存为 ch2_examp7.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var myString = "56.02 degrees centigrade";
var myInt;
var myFloat;

document.write("\"" + myString + "\" is " + parseInt(myString) +
   " as an integer" + "<BR>");

myInt = parseInt(myString);
document.write("\"" + myString + "\" when converted to an integer equals " +
   myInt + "<BR>");

myFloat = parseFloat(myString);
document.write("\"" + myString +
   "\" when converted to a floating point number equals " + myFloat);

</script>

</body>
```

```
</html>
```

在浏览器中载入这个文件，网页上会输出三行信息，如图 2-9 所示。

"56.02 degrees centigrade" is 56 as an integer
"56.02 degrees centigrade" when converted to an integer equals 56
"56.02 degrees centigrade" when converted to a floating point number equals 56.02

图 2-9

脚本块先声明了一些变量，并将变量 myString 声明和初始化为要转换的字符串。也可以直接使用字符串而非将其存储在变量中，但实际上变量比字面值更常用。本例还声明了变量 myInt 和 myFloat，用于保存转换后的数值。

```
var myString = "56.02 degrees centigrade";
var myInt;
var myFloat;
```

接下来，使用字符串连接，在页面上把 myString 转换后的整数值显示在一个易于理解的句子中。注意，使用转义序列\"来显示要转换的字符串两边的双引号。

```
document.write("\"" + myString + "\" is " + parseInt(myString)+
   " as an integer" + "<BR>");
```

可以看出，在使用数字或包含数值的变量的地方，都可以使用 parseInt()和 parseFloat()函数。实际上，在这行代码中，JavaScript 解释器作了两个类型转换。首先将变量 myString 转换为整数，因为我们要求对该变量使用 parseInt()函数。接着自动将该整数转换回字符串，以便与其他字符串连接成一个句子。还要注意，在处理整数时，变量 myString 中只有 56 这部分是有效的数字，6 之后的字符是无效的，会被忽略。

然后使用 parseInt()函数对变量 myString 做相同的转换，但这次把结果保存在变量 myInt 中。下面的代码在显示给用户的一些文本中使用该结果。

```
myInt = parseInt(myString);
document.write("\"" + myString + "\" when converted to an integer
   equals " + myInt + "<BR>");
```

同样，尽管变量 myInt 保存了一个数值，但 JavaScript 解释器知道用+操作字符串和数值时，就表示把变量 myInt 的值转换成字符串，再与其他字符串连接起来，并显示给用户。

最后用 parseFloat()函数把 myString 中的字符串转换为浮点数，并将转换结果保存在变量 myFloat 中。这次小数点是数值的有效部分，因此字符 2 之后的部分将被忽略。然后用 document.write()将结果放在一个友好的字符串中，并输出到网页上。

```
myFloat = parseFloat(myString);
document.write("\"" + myString +
   "\" when converted to a floating point number equals " + myFloat);
```

处理不能转换的字符串

某些字符串不能转换成数值，例如不包含任何数字的字符串。如果转换这些字符串，将得到什么结果？假定修改前面的例子，使变量 myString 包含不能转换的数据，例如，将如下代码：

```
var myString = "56.02 degrees centigrade";
```

改为

```
var myString = "I'm a name not a number";
```

在浏览器中重载这个页面，将看到如图 2-10 所示的结果。

```
"I'm a name not a number" is NaN as an integer
"I'm a name not a number" when converted to an integer equals NaN
"I'm a name not a number" when converted to a floating point number equals NaN
```

图　2-10

可以看到，转换结果并不是数值，而是 NaN。NaN 就是“Not a Number”，表示该值不是数值。

如果使用 parseInt()函数或 parseFloat()函数处理空字符串，或者不以数字开头的字符串，就返回 NaN 值，表示“Not a Number”。

NaN 实际上是 JavaScript 中的一个特殊值。它有自己的函数 isNaN()，来检查某个值是否是 NaN。例如：

```
myVar1 = isNaN("Hello");
```

把值 true 存储在变量 myVar1 中，因为"Hello"不是数值。然而：

```
myVar2 = isNaN("34");
```

把 false 值存储在变量 myVar2 中，因为 isNaN()函数可以把字符串"34"成功地转换为数值。

第 3 章将介绍如何使用 isNaN()函数来检查将字符串转换为数值时是否成功。第 7 章将使用 isNaN()函数来处理用户的输入。

2.5　数组

现在学习一个新概念——数组。数组与普通的变量类似，也可以保存任何类型的数据。但是，数组与变量有一个重要的区别，如下所述。

如前所述，普通的变量一次只能保存一个数据。例如，可以将变量 myVariable 设置为 25：

```
myVariable = 25;
```

接着可将它设置为另一个值，比如 35：

```
myVariable = 35;
```

但是，把变量 myVariable 设置为 35 时，原来保存的值 25 就丢失了，变量 myVariable 现在只保存 35，如表 2-2 所示：

表 2-2

变 量 名	值
myVariable	35

普通变量与数组的区别是，数组一次可以保存多个数据项。例如，可以用 myArray 数组同时保存 25 和 35。数组中的每个数据项称为元素。

如何区分数组中的两个数据项？数组的每个数据项都有一个索引值。把索引值放在数组名后面的方括号中，就可以访问对应的元素。例如，myArray 数组包含了 25 和 35，如表 2-3 所示：

表 2-3

元 素 名	值
myArray[0]	25
myArray[1]	35

注意，数组的索引从 0 开始，而不是从 1 开始。人们习惯于将数据的第一项计为 1，第二项计为 2，依此类推。但是计算机从 0 开始计数，并把第一项计为 0，第二项计为 1。这虽然有点令人疑惑，但我们很快就能习惯。

数组是非常有用的，可以根据需要在数组中保存很多数据项(JavaScript 规定，数组最多可以保存 2^{32} 个元素)。另外，不必预先指定数组中可以保存的元素数，当然也可以指定。

那么，怎么创建数组呢？这与声明普通的变量略有不同。要创建新数组，需要声明一个变量名，并使用 new 关键字和 Array()函数来说明这是一个数组。例如，myArray 数组的定义如下：

```
var myArray = new Array();
```

注意，与 JavaScript 中的其他内容一样，代码也是区分大小写的。因此，如果输入 array()而不是 Array()，代码将不能正常工作。第 5 章将进一步介绍数组的定义。

与普通变量一样，也可以首先声明变量，再将该变量定义成数组，例如：

```
var myArray;
myArray = new Array();
```

上面提到过，可以预先指定数组保存的元素数，尽管这是不必要的。为此，可以把要指定的元素数放在 Array 后面的圆括号中。例如，要创建一个包含 6 个元素的数组，可使用如下代码：

```
var myArray = new Array(6);
```

前面介绍了如何声明新数组，但如何在数组中存储数据呢？只需在定义数组时，把用逗号分隔的数据放在 Array()后面的圆括号中。例如：

```
var myArray = new Array("Paul",345,"John",112,"Bob",99);
```

其中，第一个数据项"Paul"保存在索引为 0 的元素中，第二个数据项 345 保存在索引为 1 的元素中，依此类推。换句话说，元素 myArray[0]将包含值"Paul"，元素 myArray[1]将包含值 345，依此类推。

注意，这个方法不能用于声明仅包含 1 个数值的数组，例如 Array(345)。否则，JavaScript 会认为所声明的数组包含 345 个元素。

还有一种声明数组数据的方法，可将上面的代码改写为：

```
var myArray = new Array();
myArray[0] = "Paul";
myArray[1] = 345;
myArray[2] = "John";
myArray[3] = 112;
myArray[4] = "Bob";
myArray[5] = 99;
```

把数组中的每个元素视为一个变量，并对其赋值。下面的“试一试”示例就采用这种方法声明数组元素的值。

显然，定义数据项的第一种方法比较简单。但是，有时需要在声明之后修改某个元素存储的数据，此时只能使用第二种方法定义数组元素的值。

在上面的示例中，在同一数组中可以保存不同类型的数据。在这一点上，JavaScript 是非常灵活的。

在开始下一个示例前，注意假如定义了一个包含 3 个元素的数组 myArray，例如：

```
var myArray = new Array(3);
```

然后，为索引为 130 的元素定义一个值：

```
myArray[130] = "Paul";
```

JavaScript 将认为我们需要一个至少可以包含 131 个元素的数组。

试一试　数组

下面的例子将创建一个保存一些名字的数组。本例使用上一节介绍的第二种方法，把数据存储在数组中，然后显示给用户。输入以下代码，并保存为 ch2_examp8.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var myArray = new Array();
```

```
    myArray[0] = "Bob";
    myArray[1] = "Pete";
    myArray[2] = "Paul";

    document.write("myArray[0] = " + myArray[0] + "<BR>");
    document.write("myArray[2] = " + myArray[2] + "<BR>");
    document.write("myArray[1] = " + myArray[1] + "<BR>");

    myArray[1] = "Mike";
    document.write("myArray[1] changed to " +
myArray[1]);

    </script>

    </body>
    </html>
```

myArray[0] = Bob
myArray[2] = Paul
myArray[1] = Pete
myArray[1] changed to Mike

图 2-11

在 Web 浏览器中载入这个文件，将看到如图 2-11 所示的网页。

脚本块首先声明一个变量，并告诉 JavaScript 解释器该变量是一个新数组。

```
var myArray = new Array();
```

定义了数组之后，就可以在数组中保存数据了。每次使用新索引来保存数据时，JavaScript 都会自动创建一个新的存储空间。注意，第一个元素是 myArray[0]。

在数组中依次添加数据，并观察变化。在添加任何数据之前，数组是空的。然后使用下面的代码增加一个数组元素：

```
myArray[0] = "Bob";
```

现在数组如表 2-4 所示：

表 2-4

索　引	存储的值
0	Bob

然后，为数组增加索引 1 的元素：

```
myArray[1] = "Pete";
```

现在数组如表 2-5 所示。

表 2-5

索　引	存储的值
0	Bob
1	Pete

最后，为数组增加索引 2 的元素：

```
myArray[2] = "Paul";
```

现在数组如表 2-6 所示：

表　2-6

索　　引	存 储 的 值
0	Bob
1	Pete
2	Paul

接下来，使用一系列 document.write()函数，将数组各元素的值输出到网页上。访问数组元素时，也可以不按索引顺序。

```
document.write("myArray[0] = " + myArray[0] + "<BR>");
document.write("myArray[2] = " + myArray[2] + "<BR>");
document.write("myArray[1] = " + myArray[1] + "<BR>");
```

可以将数组中的每个元素都视为标准变量，因而可以用数组元素执行计算、给其他变量或数组赋值。但是，如果访问尚未定义的数组元素，则该元素的值是 undefined。

最后，把数组中第二个元素的值改为"Mike"。也可以改为一个数值，因为同普通变量一样，数组中的任何一个元素都可以保存任意类型的数据。

```
myArray[1] = "Mike";
```

现在数组如表 2-7 所示：

表　2-7

索　　引	存 储 的 值
0	Bob
1	Mike
2	Paul

为了说明前面的修改是有效的，可使用 document.write()显示第二个元素的值：

```
document.write("myArray[1] changed to " + myArray[1]);
```

多维数组

假如要在数组中保存某公司的员工信息，例如姓名、年龄和地址等。创建该数组的一种方法是按顺序保存这些信息。例如，把第一个人的姓名作为数组的第一个元素，其年龄作为第二个元素，其地址作为第三个元素，把第二个人的姓名作为第四个元素，依此类推。这个数组如表 2-8 所示。

表 2-8

索　引	存储的值
0	Name1
1	Age1
2	Address1
3	Name2
4	Age2
5	Address2
6	Name3
7	Age3
8	Address3

这是可行的，还有一个更简洁的方法：使用多维数组。前面使用的都是一维数组。在一维数组中，元素仅由一个索引值来确定。所以在前面的例子中，Name1 的索引是 0，Age1 的索引是 1，依此类推。

而多维数组的每个元素都有 2 个或多个索引。例如，使用二维数组来保存人员信息，如表 2-9 所示：

表 2-9

索　引	0	1	2
0	Name1	Name2	Name3
1	Age1	Age2	Age3
2	Address1	Address2	Address3

下面的“试一试”示例将介绍如何创建多维数组。

试一试　二维数组

下面的代码说明了在 JavaScript 代码中如何创建多维数组，如何访问该数组的元素。输入以下代码，并保存为 ch2_examp9.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var personnel = new Array();

personnel[0] = new Array();
personnel[0][0] = "Name0";
personnel[0][1] = "Age0";
```

```
personnel[0][2] = "Address0";

personnel[1] = new Array();
personnel[1][0] = "Name1";
personnel[1][1] = "Age1";
personnel[1][2] = "Address1";

personnel[2] = new Array();
personnel[2][0] = "Name2";
personnel[2][1] = "Age2";
personnel[2][2] = "Address2";

document.write("Name : " + personnel[1][0] + "<BR>v);
document.write("Age : " + personnel[1][1] + "<BR>");
document.write("Address : " + personnel[1][2]);

</script>

</body>
</html>
```

在浏览器中载入这个页面，页面上会有三行输出，分别表示存储在数组元素 personnel[1]中的人的姓名、年龄和地址，如图 2-12 所示。

Name : Name1
Age : Age1
Address : Address1

图　2-12

脚本块首先声明一个变量 personnel，并向 JavaScript 说明该变量是一个数组。

```
var personnel = new Array();
```

接下来告诉 JavaScript，把 personnel 数组中索引为 0 的元素 personnel[0]声明为一个新的数组。

```
personnel[0] = new Array();
```

这是怎么回事呢？实际上，JavaScript 只支持一维数组，不支持多维数组。但是，JavaScript 允许在一个数组的内部创建另一个数组，从而模拟出多维数组。因此，上面的代码在 personnel 数组中索引为 0 的元素内部创建一个新数组。

下面三行代码将值放在新建的 personnel[0]数组中。这在 JavaScript 中很简单：只需在数组名 personnel[0]的后面加上放在方括号中的其他索引。第一个索引(0)属于数组 personnel，第二个索引属于数组 personnel[0]。

```
personnel[0][0] = "Name0";
personnel[0][1] = "Age0";
personnel[0][2] = "Address0";
```

执行以上代码之后，数组如表 2-10 所示。

表 2-10

索　引	0
0	Name0
1	Age0
2	Address0

表头中的数字 0 表示数组 personnel，其后左列的数字 0、1、2 就是 personnel 数组内部的新数组 personnel[0]的索引。

对于第二个人的信息，可以重复该过程，但这次使用数组 personnel 中索引为 1 的元素：

```
personnel[1] = new Array();
personnel[1][0] = "Name1";
personnel[1][1] = "Age1";
personnel[1][2] = "Address1";
```

现在，数组如表 2-11 所示：

表 2-11

索　引	0	1
0	Name0	Name1
1	Age0	Age1
2	Address0	Address1

下面几行代码创建了第三个人的信息，这次使用数组 personnel 中索引为 2 的元素创建一个新数组。

```
personnel[2] = new Array();
personnel[2][0] = "Name2";
personnel[2][1] = "Age2";
personnel[2][2] = "Address2";
```

数组如表 2-12 所示：

表 2-12

索　引	0	1	2
0	Name0	Name1	Name2
1	Age0	Age1	Age2
2	Address0	Address1	Address2

多维数组就创建好了。在脚本块的结尾，访问了第二个人的信息(Name1，Age1，Address1)，并使用 document.write()将其显示在页面上。如代码所示，二维数组的访问与存储它们相同。在可以使用普通变量或一维数组的地方，都可以使用多维数组。

```
document.write("Name : " + personnel[1][0] + "<BR>");
```

```
document.write("Age : " + personnel[1][1] + "<BR>");
document.write("Address : " + personnel[1][2]);
```

更改 document.write()命令，以显示第一个人的信息，代码如下所示：

```
document.write("Name : " + personnel[0][0] + "<BR>");
document.write("Age : " + personnel[0][1] + "<BR>");
document.write("Address : " + personnel[0][2]);
```

也可以创建三维数组、四维数组甚至一百维的数组，但是维数越多，数组就越复杂。实际上，超过二维的数组很少使用。下面演示了如何声明和访问五维数组：

```
var myArray = new Array();
myArray[0] = new Array();
myArray[0][0] = new Array();
myArray[0][0][0] = new Array();
myArray[0][0][0][0] = new Array();

myArray[0][0][0][0][0] = "This is getting out of hand"

document.write(myArray[0][0][0][0][0]);
```

数组就介绍到这里。第 4 章将再次回到数组的主题，并介绍数组的高级特性。

2.6　小结

本章介绍了有关 JavaScript 数据类型和变量的基础知识，以及如何在操作中使用它们。主要内容如下：

- JavaScript 支持多种数据类型，如数值、文本和布尔类型。
- 文本用字符串表示，放在引号中，且引号必须是匹配的。转义字符可以在字符串中包含不能直接输入的字符。
- 变量是 JavaScript 在内存中保存数据(例如数值或字符串)的方式，以便在代码中反复使用它们。
- 变量名不得包含非法字符，如百分号(%)和&，也不能是 JavaScript 的保留字，如 with。
- 在给变量赋值之前，必须先向 JavaScript 解释器声明变量的存在。
- JavaScript 有 4 个基本的数学运算符，即加(+)、减(－)、乘(*)、除(/)。为变量赋予计算结果时，使用等号(=)运算符，它又称为赋值运算符。
- 运算符有不同的优先级，所以乘法和除法将先于加法和减法计算。
- +运算符可以把多个字符串连成一个长字符串。用+运算符连接数值和字符串时，JavaScript 将自动把数值转换成字符串。
- 尽管很多时候 JavaScript 都可以自动转换数据类型，但有时需要强制转换类型。parseInt()和 parseFloat()函数可以将字符串转换为数值。对于不能转换的字符串，将返回 NaN。

- 数组是一种特殊的变量类型，可以保存多个数据。通过唯一索引可以插入和访问这些数据。

2.7 习题

在附录 A 中可以找到本章习题的答案。

习题 1：

编写一个 JavaScript 程序，将摄氏温度转换为华氏温度，并将结果保存在一个描述性语句中，输出到页面上。将摄氏温度转换为华氏温度的 JavaScript 等式如下所示：

```
degFahren = 9 / 5 * degCent + 32
```

习题 2：

下面的代码使用 prompt()函数，从用户的输入中获得两个数值，并将这两个值相加后输出到页面中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<script language="JavaScript" type="text/javascript">

var firstNumber = prompt("Enter the first number","");
var secondNumber = prompt("Enter the second number","");
var theTotal = firstNumber + secondNumber;
document.write(firstNumber + " added to " + secondNumber + " equals " +
   theTotal);

</script>
</body>
</html>
```

但是，如果执行上面的代码，会发现代码不能正常工作。为什么？请修改代码，使之能正常工作。

第 3 章

决策、循环和函数

前面学习了如何使用 JavaScript 获取用户的输入，并使用该输入进行计算和执行任务，然后将结果输出到网页上。然而，袖珍计算器就能完成这些工作，使用计算机又有什么不同呢？或者说，是什么使计算机具有一定的智能呢？答案是：计算机可以根据所收集的信息做出选择。

如何利用决策语句来帮助创建 Web 站点？第 2 章编写了一些代码，将华氏温度转换为摄氏温度。我们使用 prompt()函数来获得用户输入的华氏温度值。当用户输入有效的数值，如 50 时，代码将正常运行。但是，当用户输入一个无效的华氏温度值，如字符串 aaa 时，代码将无法正常工作。如果在代码中加入一些语句，就可以检查用户的输入是否有效。如果有效，则执行温度转换的计算；如果无效，则通知用户输入是无效的，并提示用户输入有效的数值。

在 JavaScript 中，检查用户输入的有效性是决策语句最常用的场合。但是，决策语句还有其他用途。

本章将介绍 JavaScript 中的决策语句，使代码有一定的逻辑判断能力。

3.1 决策语句——if 和 switch 语句

所有的程序设计语言都有决策语句，使程序能够根据是否满足某个条件，来执行某个操作。选择语句使编程语言有一定的智能。

例如，如果 JavaScript 代码仅与 4.0 或以上版本的浏览器兼容，条件就是用户使用 4.0 或以上版本的浏览器。如果该条件不满足，就把用户重定向到与低版本浏览器兼容的页面上。

条件就是变量与数据之间的比较，例如：

- A 是否大于 B？
- X 是否等于 Y？
- M 是否不等于 N？

例如，假如变量 browserVersion 保存了用户浏览器的版本号，上面的条件就是：

变量 browserVersion 是否大于等于 4?

所有的问题都只有“是”与“否”这两个答案。即条件是一个布尔值，只能等于 true 和 false。那么如何在代码中使用这个布尔值来做出决定。例如，测试浏览器的版本，看看该条件是否是 true，只有结果是 true，才执行特定的代码段。

下面是另一个例子。第 1 章用自然语言作为指令，演示了冲泡速溶咖啡的代码流程。其中一个指令是：

如果水已煮沸，则将水倒入咖啡杯中，否则继续等水煮沸。

这就是一个决策语句的示例。该语句中的条件是“水是否已煮沸？”，它只有 true 和 false 两个答案。如果答案是 true，则将水倒入杯中。如果答案不是 true，则继续等水煮沸。

在 JavaScript 中可以使用 if 语句和 switch 语句，根据某个条件的真假来改变程序的执行流。稍后介绍它们，但在此之前，先介绍定义条件的要素——比较运算符。

3.1.1 比较运算符

如第 2 章所述，数学函数(如加法和减法)用特定的符号表示，如加号(+)、除号(/)，称为运算符。另外，使用等号(=)可以将一个值或计算结果赋给变量，等号又称为赋值运算符。

决策语句也有自己的运算符，以测试条件，称为比较运算符。比较运算符与第 2 章中的数学运算符类似，也有左操作数(Left-Hand Side，LHS)和右操作数(Right-Hand Side，RHS)，比较就在这两者之间进行。例如，小于运算符(<)就是一个比较运算符。表达式 23 < 45 表示“23 是否小于 45？”。这个表达式的结果为 true，如图 3-1 所示。

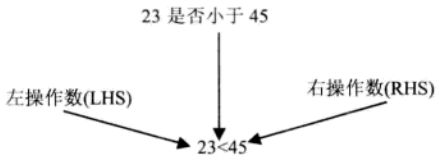

图 3-1

其他常用的比较运算符如表 3-1 所示：

表 3-1

运 算 符	作 用
==	左操作数是否等于右操作数
<	左操作数是否小于右操作数
>	左操作数是否大于右操作数
<=	左操作数是否小于等于右操作数
>=	左操作数是否大于等于右操作数
!=	左操作数是否不等于右操作数

下一节介绍 if 语句时，将用到这些比较运算符。

1. 优先级

第 2 章提到，运算符有优先级。这也适用于比较运算符。==和!=的优先级最低，其他比较运算符<、>、<=、>=的优先级相同。

所有比较运算符的优先级都比算术运算符(+、-、*、/)低。这意味着，对于表达式 3 * 5 > 2 * 5，先进行乘法运算，再对乘积进行比较。但在这些情况下，把算术运算放在括号中比较安全、清晰，例如(3 * 5) > (2 * 5)。通常最好使用圆括号，来保证优先级的清晰，否则可能得到意想不到的结果。

2. 赋值运算符与比较运算符

有一点要特别注意，赋值运算符(=)非常容易与比较运算符(==)混淆。=运算符将值赋给变量；而==运算符比较两个变量的值。即使很清楚这一点，也非常容易将比较运算符写成赋值运算符。

3. 将比较结果赋给变量

可将比较的结果存储在变量中，如下面的示例所示：

```
var age = prompt("Enter age:", "");
var isOverSixty = parseInt(age) > 60;
  document.write("Older than 60: " + isOverSixty);
```

这里用 prompt()函数获取用户的年龄。无论用户输入何值，prompt()函数都返回一个字符串，接着使用第 2 章讲过的 parseInt()函数将它转换为一个数值。然后，使用大于运算符(>)判断它是否大于 60。比较的结果(true 或 false)将保存在变量 isOverSixty 中。

如果用户输入 35，最后一行的 document.write()将在页面上输出：

```
Older than 60: false
```

如果用户输入 61，则显示：

```
Older than 60: true
```

3.1.2　if 语句

几乎每个程序都会用到 if 语句。if 语句与它在日常语言中的作用很类似。例如，我们说“如果室内温度超过华氏 80º，就打开空调”。在 JavaScript 中，可以写成如下代码：

```
if (roomTemperature > 80)
{
   roomTemperature = roomTemperature - 10;
}
```

代码的含义如图 3-2 所示。

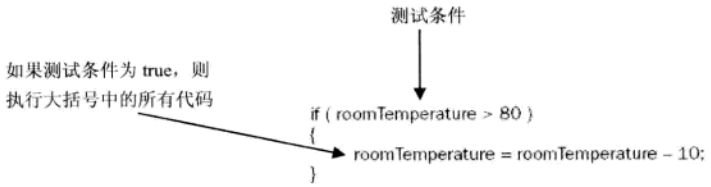

图 3-2

注意，测试条件置于 if 关键字后的括号中，且这行代码的结尾没有分号。条件为 true 时执行的代码放在其后的大括号中，每行语句都以分号结尾。

大括号在 JavaScript 中有特殊作用，它标记了一个代码块。JavaScript 将代码行放在一个代码块中，表示将其视为代码的同一部分。如果 if 语句的条件为 true，JavaScript 将执行紧随 if 语句之后的下一条语句或代码块。在前面的例子中，代码块中仅有一条语句，所以可将代码写成：

```
if (roomTemperature > 80)
   roomTemperature = roomTemperature - 10;
```

但是如果需要执行多行代码，则应该使用大括号将这些语句标记为一个代码块。例如，修改上面的代码，使之包含三条语句，则必须使用大括号：

```
if (roomTemperature > 80)
{
   roomTemperature = roomTemperature - 10;
   alert("It's getting hot in here");
   alert("Air conditioning switched on");
}
```

一个特别易犯的错误是，标记要执行的代码块时忘记使用大括号。当条件为 true 时，仅执行 if 语句之后的第一条语句。而无论测试条件的结果如何，其他代码行总是会执行。为了避免这样的错误，即使只有一条语句，也最好使用大括号。如果养成这样的编程习惯，就不太可能忘记使用大括号。

试一试 **if 语句**

回到第 2 章中温度转换的例子，添加一些决策功能。

(1) 输入以下代码，并保存为 ch3_examp1.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var degFahren = Number(prompt("Enter the degrees Fahrenheit",32));
var degCent;

degCent = 5/9 * (degFahren - 32);
```

```
document.write(degFahren + "\xB0 Fahrenheit is " + degCent +
   "\xB0 centigrade<br />");

if (degCent < 0)
{
   document.write("That' s below the freezing point of water");
}

if (degCent == 100)
   document.write("That' s the boiling point of water");

</script>

</body>
</html>
```

(2) 在浏览器中载入该页面，并在提示框中输入 32，作为要转换的华氏温度值。对于华氏 32°，两个 if 语句的条件都不是 true，所以页面上仅有一行输出，如图 3-3 所示。

32° Fahrenheit is 0° centigrade

图　3-3

(3) 现在重载该页面，然后输入 31 作为要转换的华氏温度。这次，页面上有两行输出，如图 3-4 所示。

31° Fahrenheit is–0.5555555555555556° centigrade That's below the freezing point of water

图　3-4

(4) 最后，再次重载该页面，在提示框中输入 212。页面上也有两行输出，如图 3-5 所示。

212° Fahrenheit is 100° centigrade That's the boiling point of water

图　3-5

该示例中脚本块的前半部分取自第 2 章的示例 ch2_examp4.htm 文件。我们声明了两个变量 degFahren 和 degCent。变量 degFahren 的初始值是利用 prompt()函数从用户输入中获得的。注意 prompt()函数返回一个字符串值，因此使用 Number()函数将它显式转换为数值。变量 degCent 用于保存 5/9 * (degFahren - 32)的计算结果，这个表达式将华氏温度转换为摄氏温度。

```
var degFahren = Number(prompt("Enter the degrees Fahrenheit",32));
var degCent;

degCent = 5/9 * (degFahren - 32);
```

接下来，将计算结果输出在页面上。

```
document.write(degFahren + "\xB0 Fahrenheit is " + degCent +
   "\xB0 centigrade<BR>");
```

下面的两个 if 语句是新增的代码，这是新增的第一个 if 语句。

```
if (degCent < 0)
{
   document.write("That's below the freezing point of water");
}
```

这个 if 语句的条件为："变量 degCent 的值是否小于 0？"。如果答案为 true，则执行大括号中的代码，使用 document.write()向页面输出一个句子；如果答案是 false，则程序跳到右大括号之后的下一行代码。注意，if 语句的大括号内的代码是缩进的，虽然这并不是必需的，但代码缩进是一个值得提倡的编程习惯，可以使代码更易读。

测试这个示例时，先输入 32，因此变量 degFahren 初始化为 32。这样，计算 degCent = 5/9 * (degFahren - 32)后，变量 degCent 的值为 0。所以，条件"变量 degCent 的值是否小于 0？"的结果为 false，因为 degCent 为 0，但不是小于 0。因此跳过大括号中的代码，而不执行。下一个要执行的代码行是第二个 if 语句的条件，稍后讨论。

在提示框中输入 31 时，degFahren 就设置为 31，因此变量 degCent 是 - 0.55555555556。现在，if 语句的条件是" - 0.55555555556 小于 0 吗？"，这次的答案是 true，所以执行大括号中的代码，即 document.write()语句。

最后，输入 212 时，if 语句会有什么改变？经过计算，变量 degCent 设置为 100，因此 if 语句的条件是"100 是否小于 0？"，结果为 false，跳过大括号中的代码。

在第二个 if 语句中，条件是"变量 degCent 的值是否等于 100？"

```
if (degCent == 100)
   document.write("That's the boiling point of water");
```

这个 if 语句没有使用大括号。如果条件为 true，则只执行 if 语句后的第一行代码。如果要在条件为 true 时执行多行代码，则应该加上大括号。

变量 degFahren 为 32 时，变量 degCent 就是 0。因此，这个 if 语句是"0 是否等于 100?"，显然结果为 false，不执行 if 语句后的代码。同样，变量 degFahren 设置为 31 时，degCent 的值为 - 0.55555555556，" - 0.55555555556 是否等于 100?"也是 false，也不执行 if 语句后的代码。

最后，在将 degFahren 设置为 212 时，degCent 是 100。此时，if 语句是"100 是否等于 100?"，结果为 true，所以执行 document.write()语句。

如前所述，包括 JavaScript 专家在内的程序设计人员容易犯下的一个错误是把比较运算符等于(==)写成赋值运算符(=)。例如下面的代码：

```
if (degCent = 100)
   document.write("That's the boiling point of water");
```

条件永远是 true，所以总是执行 if 语句后的代码。更糟糕的是，变量 degCent 设置为 100。为什么？因为一个等号(=)仅给变量赋值；只有两个等号(==)才执行比较操作。赋值

操作总是 true 的原因是，赋值表达式的结果是右操作数的值，即 100，它会隐式转换为布尔类型，除了 0 和 NaN 之外，其他数值都会转换为 true。

3.1.3　逻辑运算符

前面介绍了如何在 if 语句中使用条件，但是如何使用如下条件："degFahren 是否大于 0，但小于 100？"它要测试两个条件，即不仅判断 degFahren 是否大于 0，还要判断 degFahren 是否小于 100。

JavaScript 允许使用这样的多个条件。为此，需要学习另外 3 个运算符：逻辑运算符 AND、OR 和 NOT。它们的符号如表 3-2 所示：

表　3-2

逻 辑 运 算	运　算　符
AND(逻辑与)	&&
OR(逻辑或)	||
NOT(逻辑非)	!

注意，AND 和 OR 运算符是两个连续的符号：&& 和 ||。如果只输入一个符号& 或 |)，其结果会很奇怪，因为单个的 & 或 | 符号是二进制运算的按位运算符。逻辑操作必须使用两个符号：&& 或 || 。

在学完这三个逻辑运算符后，我们将通过一些实例来说明如何在 if 语句中使用它们。如果现在不大清楚逻辑运算符，不用着急，下面将逐一介绍它们，从 AND 运算符开始。

1. AND

前面介绍过运算符的左操作数和右操作数。它们同样适用于 AND 运算符。只不过，AND 的左操作数和右操作数是布尔值，通常是条件表达式的结果。

AND 运算符非常类似于英语中 and 的概念。例如，在"如果感到很冷，并且(and)带了外衣，就穿上外衣"这个语句中，and 的左操作数是"是否感到很冷"，其结果是 true 或 false。右操作数是"是否带了外衣"，其结果也是 true 或 false。如果左操作数是 true(感到很冷)，并且右操作数也是 true(带了外衣)，则"穿上外衣"。

这非常类似于 JavaScript 中的 AND 运算符。AND 运算符只得到一个结果，就好像把两个数加在一起得到一个结果一样。只不过 AND 运算符处理两个布尔值(左操作数和右操作数)，得到另一个布尔值。如果左操作数和右操作数都是 true，结果就是 true；否则结果就是 false。

表 3-3 是 AND 的真值表，包含了左操作数和右操作数所有可能的情况及其结果。

表 3-3

左操作数	右操作数	结果
true	true	true
false	true	false
true	false	false
false	false	false

严格地说，上表是 AND 的真值表。但注意 JavaScript 并不做无用的工作。当左操作数为 false 时，无论右操作数的结果如何，都对最终结果没有影响——最终结果是 false。为了避免浪费时间，当左操作数为 false 时，JavaScript 将不计算右操作数，而直接返回结果 false。

2. OR

OR 运算符也与英语中 or 的概念非常类似。例如，在“如果下雨了，或者(or)下雪了，我就带上雨伞”语句中，只要“下雨了”或者“下雪了”两个条件之一为真，我就带上雨伞。

另外，与 AND 运算符类似，OR 运算符也比较两个布尔值(左操作数和右操作数)，返回另一个布尔值。如果左操作数为 true，或者右操作数为 true，OR 就返回 true；否则返回 false。表 3-4 是 OR 运算符的真值表。

表 3-4

左操作数	右操作数	结果
True	true	true
False	true	true
True	false	true
False	false	false

与 AND 运算符类似，JavaScript 也会避免执行对最终结果没有影响的操作。如果左操作数为 true，那么右操作数是 true 还是 false 对最终结果没有任何影响——最终结果都是 true。因此，当左操作数为 true 时，JavaScript 将不计算右操作数，而直接返回 true。实际上最终结果是一样的，唯一的区别在于 JavaScript 得出该结果的方式。但是，这表示不应把需要执行的计算放在 OR 运算符的右边。

3. NOT

在“如果我感觉不热，我就喝汤”语句中，要测试的条件是“我是否感到热”。结果是 true 或 false。但是，在这个例子中，当结果为 false 时，我将执行操作——喝汤。

然而，JavaScript 仅在条件为 true 时执行代码，所以如果要在条件为 false 时执行代码，就需要把 false 转变为 true(把 true 转变为 false)。这样，JavaScript 就可以在条件为 false 时执行代码。

为此，需要使用 NOT 运算符。这个运算符反转逻辑，它提取一个布尔值，并把它变成另一个布尔值，即把 true 变成 false，把 false 变成 true。有时这称为取反。

要使用 NOT 运算符，只需把要反转的条件放在括号中，再在括号前加上!符号，如下所示：

```
if (!(degCent < 100))
{
   // Some code
}
```

仅当条件 degCent < 100 为 false 时，才执行大括号中的代码。

表 3-5 是 NOT 运算符的真值表：

表　3-5

右 操 作 数	逻辑非的结果
True	false
false	true

3.1.4　在 if 语句中使用多个条件

前一节提到如何使用条件“degFahren 是否大于 0，但又小于 100？”，一个办法是使用嵌套的两个 if 语句。嵌套指的是在一个外层 if 语句中，使用一个内层的 if 语句。当且仅当外层的 if 语句的条件为 true 时，才检验内层的 if 语句的条件。

使用嵌套的 if 语句来解决上面的问题，可使用如下代码：

```
if (degCent < 100)
{
   if (degCent > 0)
   {
      document.write("degCent is between 0 and 100");
   }
}
```

这段代码是正确的，但有点长，不太直观。JavaScript 提供了一个更好的办法——在 if 语句的条件部分使用多个条件。多个条件使用前面介绍的逻辑运算符连接起来。所以上面的代码可改写为：

```
if (degCent > 0 && degCent < 100)
{
   document.write("degCent is between 0 and 100");
}
```

上面的 if 语句首先确定 degCent 是否大于 0，如果是，就接着确定 degCent 是否小于 100。当两个条件都为 true 时，才执行代码 document.write()。

试一试 **多重条件**

下面的示例演示了使用 AND、OR 和 NOT 运算符的多条件 if 语句。输入下列代码，并保存为 ch3_examp2.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var myAge = Number(prompt("Enter your age",30));

if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10<br />");
}

if ( !(myAge >= 0 && myAge <= 10) )
{
   document.write("myAge is NOT between 0 and 10<br />" );
}

if ( myAge >= 80 || myAge <= 10 )
{
   document.write("myAge is 80 or above OR 10 or below<br />");
}

if ( (myAge >= 30 && myAge <= 39) || (myAge >= 80 && myAge <= 89) )
{
   document.write("myAge is between 30 and 39 or myAge is between 80 and 89");
}

</script>

</body>
</html>
```

在浏览器中载入该文件，会弹出一个提示框。输入 30 并按回车键，网页将显示几行文本，如图 3-6 所示。

myAge is NOT between 0 and 10
myAge is between 30 and 39 or myAge is between 80 and 89

图 3-6

这个脚本块首先定义了变量 myAge，然后把用户在提示框中输入的值转换为数值后赋给 myAge。

```
var myAge = Number(prompt("Enter your age",30));
```

接下来是 4 个 if 语句，每个 if 语句都使用了多个条件。下面按顺序详细讲解。

计算多个条件的最简单办法，是将它分解成小段，再把各小段的结果合并起来。在这

个例子中，输入的值 30 保存在变量 myAge 中。将这个值代入各条件，看看代码是如何工作的。

第一个 if 语句是：

```
if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10<br/>");
}
```

第一个 if 语句确定“myAge 是否在 0 和 10 之间？”先来看看这个条件的左操作数，代入 myAge 的值，则左操作数是“30 是否大于等于 0?”结果为 true。右操作数是“30 是否小于等于 10?”结果为 false。左操作数和右操作数用 AND 运算符&&来连接。根据前面的 AND 真值表，true && false 的结果是 false，所以这个条件的结果为 false，不执行大括号内的代码。

接下来，第二个 if 语句是：

```
if ( !(myAge >= 0 && myAge <= 10) )
{
   document.write("myAge is NOT between 0 and 10<br/>");
}
```

第二个 if 语句确定“myAge 是否不在 0 到 10 之间？”这与第一个 if 语句的条件类似，但有一个较小的差异：该条件放在括号中，并在前面加上 NOT 运算符(!)。

计算括号中的条件，与前面一样，结果仍为 false。但是，NOT 运算符会反转结果，将 false 变成 true，所以最终结果为 true，执行大括号中的代码，document.write()向页面输出信息。

第三个 if 语句的情况又如何呢？

```
if ( myAge >= 80 || myAge <= 10 )
{
   document.write("myAge is either 80 and above OR 10 or below<br/>");
}
```

第三个 if 语句确定“myAge 是否大于等于 80，或者小于等于 10？”先看看左操作数，即“30 是否大于等于 80？”，结果为 false。右操作数是“30 是否小于等于 10？”，结果也是 false。左操作数和右操作数使用 OR 运算符||来连接。根据本节前面的 OR 真值表，false || false 的结果是 false，所以这个 if 语句的条件为 false，不执行大括号内的代码。

最后一个 if 语句稍复杂些：

```
if ( (myAge >= 30 && myAge <= 39) || (myAge >= 80 && myAge <= 89) )
{
   document.write("myAge is between 30 and 39 " +
                  "or myAge is between 80 and 89<br/>");
}
```

它确定“myAge 是在 30 到 39 之间，还是在 80 到 89 之间？”将这个条件分解为其各组成部分，包括左操作数、右操作数和 OR 运算符。但是，左操作数和右操作数本身也有

各自的左操作数和右操作数，并用 AND 运算符(&&)连接起来。注意，与算术运算一样，用圆括号来表示首先计算条件的哪个部分。

条件的左操作数是(myAge >= 30 && myAge <= 39)。把条件放在圆括号内，就可以确保把它视为一个条件，无论圆括号包含几个条件，都只产生一个结果，即 true 或 false。分解圆括号内的条件，“30 是否大于等于 30？”的结果为 true，“30 是否小于等于 39？”的结果为 true。根据 AND 的真值表，true && true 的结果为 true。

条件的右操作数是(myAge >= 80 && myAge <= 89)。再次分解它。其左操作数是“30 是否大于等于 80？”，结果为 false，右操作数是“30 是否小于等于 89？”，结果为 true。false && true 的结果为 false。

现在，if 语句的条件可以看作(true || false)。根据 OR 的真值表，true || false 的结果为 true。因此执行 if 语句之后大括号中的代码，在页面上输出一行信息。

注意，JavaScript 不会计算不影响最终结果的条件。前面的条件就属于这种情况：计算出左操作数的结果 true 后，右操作数的结果为 true 还是 false 并不重要，因为只要 OR 操作中的一个条件是 true，其结果就是 true。因此，JavaScript 实际上并不计算条件的右操作数。前面计算右操作数仅仅为了演示。

如上所述，理解或创建多个条件的最简单方法就是将它们分解为最小的逻辑块。根据经验，我们几乎会不假思索地运用这种方法。除非要计算特别复杂的条件。

尽管使用多个条件通常比使用多个 if 语句好，但是有时这会使代码难以阅读，不易理解和调试。一个 if 语句可能包含 10 个、20 个甚至 100 多个条件，但是即使一个 if 语句只包含 10 个条件，也难以理解。如果多个条件过于复杂，就应将它们分成较小的逻辑块。

例如，假如要使 myAge 在区间[30,39]、[80,89]或[100,115]时，执行一些代码，则语句如下：

```
if ( (myAge >= 30 && myAge <= 39) || (myAge >= 80 && myAge <= 89) ||
     (myAge >= 100 && myAge <= 115) )
{
  document.write("myAge is between 30 and 39 " +
                 "or myAge is between 80 " +
                  "and 89 or myAge is between 100 and 115");
}
```

这没有什么问题，但是代码较长，难以阅读。对此，可以为区间[100,115]执行的代码创建另一个 if 语句。

3.1.5 else 和 else if

假设某条件为 true 时执行一些代码，该条件为 false 时执行另外一些代码。这可以使用两个 if 语句来实现，如下所示：

```
if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10");
}
```

```
if ( !(myAge >= 0 && myAge <= 10) )
{
   document.write("myAge is NOT between 0 and 10");
}
```

第一个 if 语句测试 myAge 是否在 0 和 10 之间。第二个 if 语句测试变量是否不在 0 和 10 之间。JavaScript 提供了更简便的实现方法：else 子句。else 子句的用法与英语类似。例如，“如果下雨了，我就带上雨伞；否则，我就带上太阳帽”。在 JavaScript 中，可以表示为：如果条件为 true，执行一段代码，否则执行另一段代码。使用这个技术可将上面的代码改写为：

```
if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10");
}
else
{
   document.write("myAge is NOT between 0 and 10");
}
```

改写后的代码更简洁，也更容易阅读。这也避免 JavaScript 测试一个结果已知的条件。也可以在 else 子句中包含另一个 if 语句，例如：

```
if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10");
}
else if ((myAge >= 30 && myAge <= 39) || (myAge >= 80 && myAge <= 89) )
{
   document.write("myAge is between 30 and 39 " +
                  "or myAge is between 80 and 89");
}
else
{
   document.write("myAge is NOT between 0 and 10, " +
                  "nor is it between 30 and 39, nor is it between 80 and 89");
}
```

第一个 if 语句检查 myAge 是否在 0 到 10 之间，如果结果为 true 则执行一些代码。如果结果为 false，则 else if 语句将检查 myAge 是否在 30 到 39 之间，或者在 80 到 89 之间，只要其中任一条件成立，则执行另一些代码。如果上述条件都不成立，则执行最后一个 else 子句的代码。

在使用 if 语句和 else if 语句时，应该特别小心，要使用大括号确保 if 语句和 else if 语句在希望的位置开始和结束，且 else 子句与对应的 if 语句正确匹配。下面用一个例子来加以说明，相应代码如下所示：

```
if (myAge >= 0 && myAge <= 10)
{
```

```
document.write("myAge is between 0 and 10");
if (myAge == 5)
{
document.write("You're 5 years old");
}
else
{
document.write("myAge is NOT between 0 and 10");
}
```

注意没有缩进代码。尽管这对 JavaScript 不重要，但将使人们难以阅读，而且看不出最后一个 else 语句之前少了右大括号。

正确设置代码格式，再补上遗漏的大括号后，代码如下所示：

```
if (myAge >= 0 && myAge <= 10)
{
   document.write("myAge is between 0 and 10<br>");
   if (myAge == 5)
   {
      document.write("You're 5 years old");
   }
}
else
{
   document.write("myAge is NOT between 0 and 10");
}
```

可以看到，代码现在正常运行，也更容易看出哪段代码属于哪个 if 块。

3.1.6 字符串的比较

前面学习如何把比较运算符用于数值，这些运算符也可以用于字符串。所有可应用于数值的比较操作也可以应用于字符串，唯一不同的是字符串按字母顺序进行比较，所以有几个注意事项。

下面的代码比较包含字符串"Paul"的变量 myName 与字符串字面值"Paul"：

```
var myName ="Paul";
if (myName == "Paul")
{
   alert("myName is Paul");
}
```

JavaScript 如何进行比较？JavaScript 按顺序比较左右两边字符串在同一位置上的字母，看看它们是否相等。只要发现有区别就停止比较，并返回 false。如果依次检查了所有字符，发现每个字符都相同，JavaScript 将返回 true。例如，上面代码中 if 语句的判断条件将返回 true，将执行 if 语句中的代码并弹出一个消息对话框。

但是，JavaScript 中的字符串比较是区分大小写的。因此，"P"与"p"不同。在前面的例子中，如果将变量 myName 改为"paul"，则条件是 false，不执行 if 语句中的代码。

```
var myName ="paul";
if (myName == "Paul")
{
   alert("myName is Paul");
}
```

>、>=、<和<=运算符可应用于字符串和数字，而字符串是按字母顺序进行比较的。因此"A" < "B"为 true，因为在字母表中，A 在 B 的前面。另外，JavaScript 中字符串的比较是区分大小写的。因此，虽然"A" < "B"为 true，但"a" < "B"为 false。因为大写字母总是排在小写字母的前面。实际上，每个字符在 ASCII 和 Unicode 字符集中都有一个编码，大写字母的编码总是小于小写字母的编码。在编写代码时，应该注意这一点。

避免混淆大小写的最简单办法是在比较前，将两个字符串都转换为大写或小写。为此，可以使用 toUpperCase()函数或 toLowerCase()函数，详见第 4 章。

3.1.7　switch 语句

如前所述，使用 if 语句和 else if 语句可以检查多个条件。如果第一个条件无效，则检查第二个，如果无效，则继续检查下一个，依此类推。当比较某个变量与许多可能的值时，有一个更高效的替代方案，即 switch 语句。switch 语句的结构如图 3-7 所示。

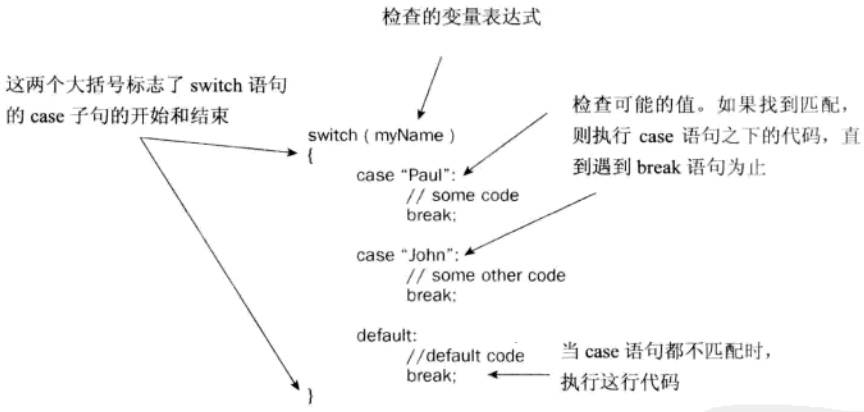

图　3-7

switch 语句就是“将代码切换到条件匹配的分支处”。switch 语句有 4 个重要的元素：

- 测试表达式
- case 语句
- break 语句
- default 语句

测试表达式放在 switch 关键字后面的圆括号中。上面的示例测试变量 myName。实际上，可以在括号中使用任何有效的表达式。

接下来是 case 语句，case 语句检查条件。为了指定 case 语句属于 switch 语句，必须

将它们放在测试表达式后面的大括号中。每个 case 语句都指定了一个值，例如"Paul"。接着 case 语句执行类似 if (myName == "Paul")的检查。如果变量 myName 包含"Paul"，则跳到 case "Paul" 语句下的代码，一直执行到 switch 语句的结束。上面的示例只包含了两个 case 语句，但 switch 语句可以包含任意多个 case 语句。

大多数情况下，只执行某个 case 语句下的代码块，而不是该 case 语句之后、包含其他 case 语句的所有代码。为此，只需在要执行的代码结尾添加一个 break 语句，告诉 JavaScript 在此处停止执行，并退出 switch 语句。

最后是 default 语句。顾名思义，它表示其他 case 语句都不匹配时要执行的代码。default 语句是可选的，如果没有要执行的默认代码，就可以省略 default 语句，即所有 case 语句都不匹配时，不执行任何代码。最好包含一个 default 语句，除非绝对肯定包含了所有可能的选项。

试一试 使用 switch 语句

下面是使用 switch 语句的示例，它演示了一个简单的猜谜游戏。输入以下代码，并保存为 ch3_examp3.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var secretNumber = prompt("Pick a number between 1 and 5:", "");
secretNumber = parseInt(secretNumber);

switch (secretNumber)
{
case 1:
   document.write("Too low!");
   break;

case 2:
   document.write("Too low!");
   break;

case 3:
   document.write("You guessed the secret number!");
   break;

case 4:
   document.write("Too high!");
   break;

case 5:
   document.write("Too high!");
   break;

default:
```

```
    document.write("You did not enter a number between 1 and 5.");
    break;
}

document.write("<BR>Execution continues here");

</script>

</body>
</html>
```

在浏览器中载入该文件，并在提示框中输入 1，则结果如图 3-8 所示。

如果输入 3，页面上会输出一个友好的提示信息，说明猜对了，如图 3-9 所示。

Too low!
Execution continues here

图　3-8

You guessed the secret number!
Execution continues here

图　3-9

首先声明变量 secretNumber，并设置为用户在提示框中输入的值。然后使用 parseInt()函数将 prompt()返回的字符串转换为一个整数。

```
var secretNumber = prompt("Pick a number between 1 and 5:", "");
secretNumber = parseInt(secretNumber);
```

接下来分析 switch 语句的开头部分：

```
switch (secretNumber)
{
```

括号中的表达式就是变量 secretNumber，其值就是用于匹配 case 语句的数字。

用大括号声明了包含 case 语句的代码块。每个 case 语句检查一个 1~5 之间的数字，因为这是指定用户输入的数字。第一个 Case 语句仅在页面上输出一个数字太小的提示。

```
case 1:
   document.write("Too low!");
   break;
```

第二个 case 语句用于匹配数字 2，其输出与数字 1 相同。第三个 case 语句告诉用户他猜对了。

```
case 3:
   document.write("You guessed the secret number!");
   break;
```

最后，第四个和第五个 case 语句输出一个提示，表明数字太大。

```
case 4:
   document.write("Too high!");
   break;
```

这个例子需要添加一个 default 语句，因为用户完全可能不管输入的要求，输入一个不在 1～5 之间的数字，甚至输入一个字母。这里添加一个消息，告诉用户出了问题。

```
default:
   document.write("You did not enter a number between 1 and 5.");
   break;
```

default 语句也用来查找错误。如果 default 语句在不该运行时运行了，则可以很快断定某些 case 语句是错误的。

最后，添加右大括号来表示 switch 语句的结束。接着输出一行信息，表示从哪里继续执行。

```
}
document.write("<br/>Execution continues here");
```

注意，每个 case 语句都以 break 语句结尾，以确保代码跳到 switch 语句之后的代码行。如果忘记包含 break 语句，则代码将从匹配的 case 语句开始，执行其后的每个 case 语句。

不同的 case 执行相同的代码

这个示例中的 switch 语句有一个问题：如果用户输入 1 或 2，执行的代码是相同的；如果输入 4 或 5，执行的代码也是相同的。因此必须重复编码。最好有一种更简单的方法，让 JavaScript 为不同的 case 执行相同的代码。这非常简单，只需将上面的代码改成：

```
switch (secretNumber)
{
case 1:
case 2:
   document.write("Too low!");
   break;

case 3:
   document.write("You guessed the secret number!");
   break;

case 4:
case 5:
   document.write("Too high!");
   break;

default:
   document.write("You did not enter a number between 1 and 5.");
   break;
}
```

在浏览器中载入该文件，试着输入不同的数字，结果与前面的代码相同。

这里使用了 switch 语句的一个特性：如果某个 case 语句下的代码未以 break 结尾，则代码会依次执行其后的 case 语句，直到遇到 break 语句，或者 switch 语句结束。

如果值为 1 的 case 语句匹配，JavaScript 将继续执行到 case 2 语句中的 break 语句。这样，就可以高效地为两个 case 执行相同的代码。值为 4 和 5 的 case 语句也采用了相同的办法。

3.2　循环——for 语句和 while 语句

循环指的是当条件为 true 时，反复执行某个代码块。JavaScript 使用两个语句来达到该目的：for 语句和 while 语句。在介绍它们之前，先看看为什么需要重复代码块。

假如我们有一系列数据，表示一年中每个月的平均温度，我们想把它绘制在图上。绘制每个点的代码几乎是完全相同的。所以，不必将代码重复编写 12 次(每次绘制一个点)，而是使用该系列中接下来的数据项，重复执行同一段代码 12 次。此时使用 for 语句比较方便，因为代码的执行次数是已知的。

另一种情况是，需要在某个条件为 true 时(例如用户是否单击了 Start Again 按钮)，重复相同的代码。这时，while 语句最有用。

3.2.1　for 循环

for 语句可以将某段代码重复执行指定的次数，其语法如图 3-10 所示。

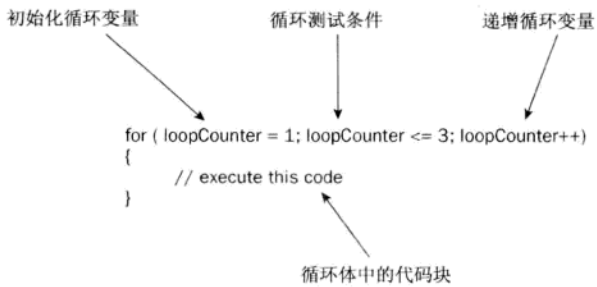

图　3-10

先看看 for 语句的组成，如图 3-10 所示，与 if 和 switch 语句一样，for 语句的逻辑也放在括号中。但其逻辑分为 3 部分，各部分之间用分号分隔。例如，在图 3-10 中，for 语句的逻辑是：

```
(var loopCounter = 1; loopCounter <= 3; loopCounter++)
```

For 语句的第一部分是其初始化部分。为了跟踪代码循环的次数，需要一个变量来计数。该变量在初始化部分进行初始化。上面的示例声明了 loopCounter，并设置为 1。在循环的执行过程中，这个部分只执行一次，而其他部分可能执行多次。如果之前已经声明了该变量，就不需要声明它了：

```
var loopCounter;
for (loopCounter = 1; loopCounter <= 3; loopCounter++)
```

分号的后面是 for 语句的测试条件部分。只要测试条件为 true，就一直执行 for 语句中的代码。每次执行这些代码后，就测试这个条件。如图 3-10 所示，只要 loopCounter 小于等于 3，就执行代码。循环执行的次数常常称为迭代次数。

最后是 for 循环的递增部分，它用于递增循环测试条件中的变量。在上面的示例中，loopCounter 使用第 2 章介绍的运算符++递增。For 语句的这个部分在每次循环时都执行一遍。尽管它称为递增部分，但也可以递减变量。例如，从数组中的最后一个元素开始向后访问到第一个元素。

只要测试条件为 true，就重复执行 for 语句后面的代码块。这个代码块包含在大括号中。如果条件一直不是 true，即使是第一次测试循环条件，也会跳过 for 循环中的代码，不执行它们。

综上所述，for 循环的工作原理是：

(1) 执行 for 语句的初始化部分。

(2) 检查测试条件，如果为 true，则继续执行，如果为 false，则退出 for 语句。

(3) 执行 for 语句后面的代码块。

(4) 执行 for 语句的递增部分。

(5) 重复第 2 步到第 4 步，直到测试条件为 false 为止。

试一试　使用循环语句将一系列华氏温度转换为摄氏温度

下面修改温度转换程序，把一系列保存在数组中的华氏温度值转换为摄氏温度。本例使用 for 语句来遍历数组中的元素。输入下列代码，并保存为 ch3_examp4.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var degFahren = new Array(212, 32, -459.15);
var degCent = new Array();
var loopCounter;

for (loopCounter = 0; loopCounter <= 2; loopCounter++)
{
    degCent[loopCounter] = 5/9 * (degFahren[loopCounter] - 32);
}

for (loopCounter = 2; loopCounter >= 0; loopCounter--)
{
    document.write("Value " + loopCounter + " was " + degFahren[loopCounter] +
                   " degrees Fahrenheit");
    document.write(" which is " + degCent[loopCounter] +
                   " degrees centigrade<BR>");
}

</script>
```

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

在浏览器中载入该文件，页面上将有 3 行输出，显示了将华氏温度数组中的元素转换为摄氏温度的结果，如图 3-11 所示。

```
Value 2 was–459.15 degrees Fahrenheit which is–272.8611111111111 degrees centigrade
Value 1 was 32 degrees Fahrenheit which is 0degrees centigrade
Value 0 was 212 degrees Fahrenheit which is 100 degrees centigrade
```

图　3-11

上面的代码首先声明了要使用的变量 degFahren，并将其初始化为包含 3 个值 212、32、-459.15 的数组。接下来，把变量 degCent 声明为一个空数组。最后，声明了 loopCounter，用于跟踪循环过程中访问的数组下标。

```
var degFahren = new Array(212, 32, -459.15);
var degCent = new Array();
var loopCounter;
```

接下来是第一个 for 循环：

```
for (loopCounter = 0; loopCounter <= 2; loopCounter++)
{
    degCent[loopCounter] = 5/9 * (degFahren[loopCounter] - 32);
}
```

第一行首先把 loopCounter 初始化为 0。然后检查 for 循环的测试条件 loopCounter <= 2。如果该条件为 true，则第一次执行循环。执行完大括号中的代码后，执行 for 循环的递增部分 loopCounter++，并再次计算测试条件，如果该条件仍然为 true，则再次执行循环代码。这个过程将反复进行，直到 for 循环的测试条件变成 false，此时循环结束，接着执行右大括号之后的第一条语句。

大括号中的代码就是前面示例中的等式，但这次将结果放在 degCent 数组中，其下标是 loopCounter 的值。

在第二个 for 语句中，把 degCent 数组包含的结果输出在屏幕上。

```
for (loopCounter = 2; loopCounter >= 0; loopCounter--)
{
    document.write("Value " + loopCounter + " was " + degFahren[loopCounter]+
                   " degrees Fahrenheit");
    document.write(" which is " + degCent[loopCounter] +
                   " degrees centigrade<BR>");
}
```

这次从 2 到 0 递减计数。变量 loopCounter 初始化为 2，循环条件一直是 true，直到 loopCounter 小于 0 为止。这次通过 loopCounter--，使 loopCounter 递减，而不是递增。另外，loopCounter 有两个作用：记录循环次数，以及提供数组中的下标位置。

注意：

上述循环例子的 loopCounter 都使用了整数。也可以使用分数，但不常用。

3.2.2 for...in 循环

这个循环主要用于数组，如下一章所述，它也用于对象。for...in 循环可以遍历数组中的元素，而无须知道数组中元素的个数。其含义是：对于数组中的每个元素，执行一些代码。for...in 循环语句在每次迭代时，会自动确定每个元素的下标，自动移动到下一个元素上。

for...in 语句用于数组的语法如下：

```
for (index in arrayName)
{
    //some code
}
```

其中，index 是在循环前声明的变量，会用数组中的下一个下标值自动填充它。Array Name 是包含要遍历的数组的变量名。

下面举例说明。定义一个数组，用 3 个值初始化它：

```
var myArray = new Array("Paul","Paula","Pauline");
```

使用传统的 for 循环来访问每个元素，则代码如下：

```
var loopCounter;
for (loopCounter = 0; loopCounter < 3; loopCounter++)
{
    document.write(myArray[loopCounter]);
}
```

如果使用 for...in 循环，则代码如下所示：

```
var elementIndex;
for (elementIndex in myArray)
{
    document.write(myArray[elementIndex]);
}
```

显然，第二段代码更加简洁明了。两种方法的作用一样，都是循环 3 次。然而，假如增加数组的长度，例如添加元素 myArray[3] = "Philip"，则第一种方法仍只遍历数组的前 3 个元素，而第二种方法会遍历所有 4 个元素。

3.2.3 while 循环

for 循环用于迭代特定的次数，而 while 循环可测试一个条件，在条件为 true 时继续迭代。for 语句在循环次数已知时比较方便，例如遍历元素个数已知的数组。while 循环在不知道循环次数时比较有效，例如在遍历温度值数组时，如果数组元素包含的温度值小于 100 时继续循环，就需要使用 while 语句。

while 语句的结构如图 3-12 所示。

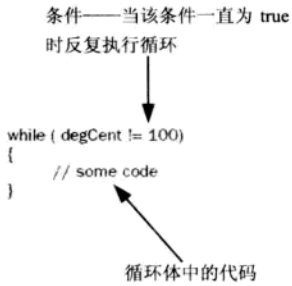

图　3-12

可以看出，while 循环的组成部分比 for 循环少。如果 while 循环的条件为 true，则执行一次大括号中的代码块；然后再次计算条件，如果条件仍然为 true，则再次执行代码块，并再次计算条件，一直继续下去，直到条件变为 false 为止。

注意，如果条件一开始就是 false，while 循环体将永远不会执行。例如：

```
degCent = 100;

while (degCent != 100)
{
      // some code
}
```

如果 degCent 不等于 100，则运行循环。但是，由于 degCent 是 100，条件是 false，所以代码块永远不执行。

通常我们希望循环先执行一次，是否再次执行则取决于循环体中的代码对循环条件中的变量执行的处理，例如：

```
degCent = new Array();
degFahren = new Array(34, 123, 212);
var loopCounter = 0;
while (loopCounter < 3)
{
   degCent[loopCounter] = 5/9 * (degFahren[loopCounter] - 32);
   loopCounter++;
}
```

只要 loopCounter 小于 3，就执行循环。循环中的代码(loopCounter++;)会递增 loop Counter，最终使 loopCounter < 3 为 false，结束循环，并从 while 语句右大括号之后的第一行代码继续执行。

要注意无限循环——永远不会结束的循环。假定在上面的代码中忘记加入 loopCounter++; 语句，loopCounter 将一直是 0，因此条件(loopCounter < 3)总是 true，则循环将一直继续，直到用户因感到厌烦而关闭浏览器为止。这是一个很容易出现的错误，JavaScript 也不会发出警告。

不但忘记上述代码行会导致无限循环，代码中的其他错误也可能导致无限循环，例如：

```
var testVariable = 0;
while (testVariable <= 10)
{
   alert("Test Variable is " + testVariable);
   testVariable++;
   if (testVariable = 10)
   {
      alert("The last loop");
   }
}
```

能看出故意设置的导致无限循环的错误吗？该错误是在 if 语句的条件中，把比较运算符==错误地写成了赋值运算符=，所以尽管有 testVariable++;语句，但在每次循环时，testVariable 都设置为 10。这意味着在每次循环的开始，测试条件总是 true，即 10<=10 成立。添加遗漏的=，改为 if (testVariable==10))，代码就会正常运行。

3.2.4 do...while 循环

在 while 语句中，循环体仅在条件为 true 时执行。如果条件为 false，代码就不执行，而是跳到 while 循环之后的第一条语句。但有时希望不管条件是否为 true，while 循环体中的代码都至少执行一次。甚至有时 while 循环中的代码要在测试 while 语句的条件之前执行。此时应使用 do...while 循环。

下面的例子通过提示框获得用户的年龄。不但要显示提示框，还要检查用户输入是否是一个数值。

```
var userAge;
do
{
   userAge = prompt("Please enter your age","")
}
while (isNaN(userAge) == true);
```

循环中的代码行为：

```
userAge = prompt("Please enter your age","")
```

无论 while 语句的条件为 true 还是 false，循环中的代码都会执行。这是因为在执行了一次循环之后才检查条件。如果条件为 true，则再次执行循环；如果条件为 false，则终止循环。

在 do...while 语句的条件中，使用了第 2 章介绍的 isNaN()函数，来检查变量 userAge 的值是否为 NaN(即非数值)。如果 userAge 是 NaN，则条件返回 true，否则返回 false。在上例中，isNaN()函数用于检查用户是否输入了正确数据。用户可能输入不真实的年龄，但至少输入了一个数值。

do...while 语句相当少见，必须使用 do...while 语句的情况很少，所以，除非确有必要，

否则最好避免使用它。

3.2.5　break 和 continue 语句

在前面的 switch 语句中遇到过 break 语句。在 switch 语句中，break 语句的作用是终止代码的执行，跳到 switch 语句右大括号之后的下一行代码。break 语句也可用在 for 循环和 while 循环中，提前退出循环。例如，在温度转换的示例中迭代数组时，遇到一个无效的值，此时希望停止代码的执行，通知用户该数据无效，并退出循环。这是使用 break 语句的一种场合。

下面修改转换一系列华氏温度值的示例(ch3_examp4.htm)，使之在遇到非数字的值时终止循环，并通知用户数据无效：

```
<script type="text/javascript">
var degFahren = new Array(212, "string data", -459.67);
var degCent = new Array();
var loopCounter;

for (loopCounter = 0; loopCounter <= 2; loopCounter++)
{
   if (isNaN(degFahren[loopCounter]))
      {
        alert("Data '" + degFahren[loopCounter] + "' at array index " +
              loopCounter + " is invalid");
        break;
      }
   degCent[loopCounter] = 5/9 * (degFahren[loopCounter] - 32);
}
```

上面的代码修改了数组 degFahren 的初始化，使之包含一个无效的数据。然后，在 for 循环中加入了一个 if 语句，来检查 degFahren 数组中的数据是否不是数值。为此使用了 isNaN()函数。如果括号中的值 degFahren[loopCounter]不是数字，它将返回 true，并通知用户，数组有无效的数据值，然后使用 break 语句来结束 for 语句的循环，代码从 for 语句之后的第一条语句继续执行。

这是 break 语句的作用，那么 continue 语句呢？continue 与 break 类似，也是停止循环的执行，但它不跳出循环，而是启动下一轮循环，重新计算 for 或 while 语句的条件，就像执行到循环代码的最后一行一样。

在 break 示例中，只要一个数据无效，就结束循环。在转换 degFahren 数组中的所有元素时，如果遇到无效的数组元素，最好通知用户，然后继续处理下一个元素；而不是像 break 语句那样退出循环。

```
if (isNaN(degFahren[loopCounter]))
   {
     alert("Data '" + degFahren[loopCounter] + "' at array index " +
           loopCounter + " is invalid");
     continue;
   }
```

把 break 语句改成了 continue 语句。当遇到无效数据时，将输出一条消息，但第三个值也已转换过来了。

3.3 函数

函数是执行特定任务的单元。以袖珍计算器为例，袖珍计算器可以执行很多基本计算，如加法和减法。它也有一些执行复杂操作的功能按钮。例如，某些计算器具有计算平方根的功能键，有的计算器甚至提供了统计功能，如计算平均值。大部分复杂操作均由基本数学操作(加、减、乘、除)完成，但步骤很多。但是用户只需按一下功能按钮，并提供数据，功能按钮就会完成其他工作。

JavaScript 中的函数类似于袖珍计算器上的功能按钮：它们封装了执行特定任务的代码块。前面学习了许多完成特定任务的 JavaScript 内置函数，如 parseInt()和 parseFloat()函数将字符串转换为数值；isNaN()函数检查某个值能否转换为数值。某些函数返回数据，如 parseInt()函数返回一个整数。而某些函数只执行某个操作，不返回数据。另外，某些函数需要传入数据，而有的不需要。例如，isNaN()函数需要传入某个数据，它检查这个数据是否为 NaN。需要传入函数的数据称为参数(parameter)。

本书将学习很多有用的 JavaScript 内置函数。更重要的是，可以创建自己的函数。设计、编写、调试执行某个任务的代码块后，就可以在需要时反复调用它。JavaScript 允许创建自定义函数，这是下一节的重点。

3.3.1 创建自定义函数

创建和使用自定义函数是非常简单的。图 3-13 就是一个自定义函数的例子。前面介绍过该函数的功能和代码的工作原理，这是将华氏温度转换为摄氏温度的代码。

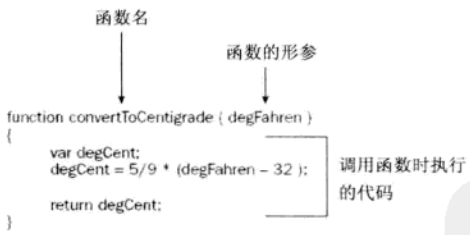

图 3-13

在 JavaScript 中定义的每个函数必须在该页面上有唯一的名称。函数名位于关键字 function 之后。为了让代码易于理解，应使用有意义的函数名，这样，以后在代码中用到这个函数时，就能知道该函数的功能。例如，某个函数把某人的生日和当前日期作为参数，返回该人的年龄，这个函数可以命名为 getAge()。但是，可以使用的函数名是受限制的，这与变量名一样。例如，函数名不能使用 JavaScript 的保留字，因此，函数名不能是 with()或者 while()。

函数的参数放在函数名后的圆括号中。参数是函数完成其工作所需要的一个数据项，

通常情况下，未传入必选参数将导致一个错误。函数可以有 0 个或多个参数，即使函数没有参数，函数名后的圆括号也不能省略。例如，函数定义应具有以下形式：

```
function myNoParamFunction()
```

之后编写调用函数时执行的代码。所有函数代码都位于一对大括号中。

通过 return 语句，函数还可以把值返回给调用它的代码。前面的例子返回计算出来的 degCent 变量值。如果不需要，就可以不返回任何值，但最好始终在函数结尾处包含 return 语句。由于 JavaScript 的语法很宽松，不使用 return 语句也不会造成任何错误。

JavaScript 执行到函数的 return 语句时，会像 for 循环中的 break 语句那样处理——退出函数，并返回 return 关键字之后指定的值。

可以将 JavaScript 代码中的常用函数组成一个函数库。在需要时，就可以复制它们并粘贴到页面中。

创建好自定义函数之后，如何使用它们？前面介绍的代码都是按顺序执行，而函数只有要求执行时才会执行，这称为调用函数。要调用函数，应在需要调用函数的地方写出函数名，并传入函数所需的参数，参数之间用逗号分隔。例如：

```
myTemp = convertToCentigrade(212);
```

这行代码调用了前面的函数 convertToCentigrade()，并将 212 作为参数传入。该函数的返回值(即 100)保存在变量 myTemp 中。

了解了如何创建自定义函数，下面看看如何传递参数。参数传递有点麻烦，所以下面先创建一个简单函数，该函数只接收一个参数(即用户名)，然后在页面上显示友好的欢迎信息。首先，给函数取一个简短的描述性函数名 writeUserWelcome()，然后定义需要传给函数的参数。该函数只需要一个参数，即用户名。定义参数与定义变量非常类似，其参数名需要遵守与变量名相同的命名规范，所以参数名不能包含空格、特殊字符以及 JavaScript 的保留字。将参数命名为 userName，放在函数名后的圆括号中(注意，这行代码的结尾处没有分号)：

```
function writeUserWelcome(userName)
```

在定义好函数名和参数之后，就可以创建函数体——调用函数时执行的代码。函数的这部分放在大括号中：

```
function writeUserWelcome(userName)
{
   document.write("Welcome to my website " + userName + "<br/>");
   document.write("Hope you enjoy it!");
}
```

这段代码非常简单，它用 document.write()方法向网页输出信息。userName 的用法与普通变量完全相同。实际上，最好把参数视为普通变量。调用该函数的 JavaScript 代码指定了该参数的值。

下面看看如何调用该函数：

```
writeUserWelcome("Paul");
```

非常简单，只需写出要调用的函数名，并在其后的括号中填入传递给每个参数的数据。这个例子只填入了一个数据。执行函数的代码时，函数体中使用的变量 userName 包含文本"Paul"。

假如要向函数传递两个参数，如何进行修改？首先要修改函数的定义，假定第二个参数包含用户的年龄，并命名为 userAge，清楚地表明参数数据的含义。下面是新代码：

```
function writeUserWelcome(userName, userAge)
{
   document.write("Welcome to my website" + userName + "<br/>");
   document.write("Hope you enjoy it<br/>");
   document.write("Your age is " + userAge);
}
```

在函数体中增加一行代码，以便使用所添加的第二个参数。使用如下代码调用这个函数：

```
writeUserWelcome("Paul",31);
```

第二个参数是一个数值，所以不需要放在引号中。这里，userName 参数是"Paul"，第二个参数 userAge 是 31。

试一试　创建一个将华氏温度转换为摄氏温度的函数

下面使用函数重写温度转换页面。可以从 ch3_examp4.htm 中复制并粘贴大部分代码。完成后保存为 ch3_examp5.htm。

```
<html>
<body>

<script language="JavaScript" type="text/javascript">

function convertToCentigrade(degFahren)
{
   var degCent;
   degCent = 5/9 * (degFahren - 32);

   return degCent;
}

var degFahren = new Array(212, 32, -459.15);
var degCent = new Array();
var loopCounter;

for (loopCounter = 0; loopCounter <= 2; loopCounter++)
{
   degCent[loopCounter] = convertToCentigrade(degFahren[loopCounter]);
}

for (loopCounter = 2; loopCounter >= 0; loopCounter-- )
{
```

```
    document.write("Value " + loopCounter + " was " + degFahren[loopCounter] +
                   " degrees Fahrenheit");
    document.write(" which is " + degCent[loopCounter] +
                   " degrees centigrade<br />");
}

</script>

</body>
</html>
```

在浏览器中载入该文件后，结果与 ch3_examp4.htm 示例完全相同。

在脚本代码块的开始声明了函数 convertToCentigrade()，前面介绍了这个函数。

```
function convertToCentigrade(degFahren)
{
    var degCent;
    degCent = 5/9 * (degFahren - 32);

    return degCent;
}
```

如果在页面中使用多个独立 script 块，则必须把要定义的函数放在调用它的任何脚本之前。如果有多个函数，可以把它们放在页面顶部一个专门的脚本块中——<head> 和 </head> 标记之间。这样就知道在哪里查找所有函数，并且确保函数在使用之前声明。

我们已经非常熟悉函数中的代码。首先声明一个变量 degCent，然后执行计算，把结果保存在 degCent 中。最后，把 degCent 返回给调用代码。该函数的参数是 degFahren，提供了计算需要的信息。

在函数定义的代码之后，是页面加载时执行的代码。首先定义需要的变量，然后使用两个循环计算并输出结果。大部分代码与前面相同，但第一个 for 循环除外：

```
for (loopCounter = 0; loopCounter <= 2; loopCounter++)
{
    degCent[loopCounter] = convertToCentigrade(degFahren[loopCounter]);
}
```

第一个 for 循环中的代码把 convertToCentigrade()函数的返回值放在 degCent 数组中。

这个示例还有一个微妙之处。在 convertToCentigrade()函数中声明了变量 degCent，在函数定义的后面又把它声明为一个数组。肯定可以这样做吗？

这引出了本章的下一个主题——变量的作用域。

3.3.2　变量的作用域和生存期

变量的作用域是什么意思？简言之，变量的作用域是变量的有效范围——哪部分代码可以访问该变量及其包含的数据。在网页上，任何在函数之外声明的变量，都可用于该页面的所有脚本，无论该脚本在函数内还是函数外。这称为页面级作用域或者全局作用域。在函数内部定义的变量只能在该函数内使用，函数外的任何代码都不能访问它。因此，可

以在页面上的每个函数中都声明一个变量 degCent，而在函数外部只能声明一个变量 degCent。但是，不能在一个函数中多次声明该变量，也不能在函数外部多次声明一个变量。虽然可以采用这种方式在页面上重用变量名，但这不是一个好的编程习惯，会造成代码难以理解。

函数参数与变量类似：函数的参数在函数外部是不可见的。虽然可以在函数中声明一个与其参数同名的变量，但会带来混乱，容易导致难于察觉的错误。这是一种很糟糕的编码方式，最好避免使用。否则，调试代码将很困难。

当函数中的代码执行完毕，将返回到函数的调用点，此时，函数中定义的变量会有什么变化？下次调用函数时，这些变量的值保持不变吗？

答案是不会！变量不仅有作用域，还有生存期。函数执行完毕后，该函数中的变量就被释放，其值丢失，除非把某个变量的值返回给调用代码。JavaScript 将时常执行垃圾回收(详见第 2 章)，它将扫描所有代码，检查是否有不再使用的变量。如果有，就释放这些变量，给其他变量腾出空间。

全局变量可以在任何地方使用，为什么不将所有变量都定义为全局变量？需要在全局范围内跟踪某个数据时，全局变量非常合适。但因为可以在代码的任何地方修改全局变量，如果由于某个错误对全局变量做了错误的修改，该错误就可能出现在代码的任何地方，从而增加代码的调试难度。因此，最好尽量少用全局变量，除非必须使用全局变量——例如需要在函数之间共享数据时。

3.4 小结

本章介绍了 JavaScript 语言的核心及其语法。所有 JavaScript 代码建立在这些基础之上，在掌握了基本的语法之后，就可以继续学习本书后面的有趣内容了。

本章的主要内容如下：

- **用 if 和 switch 语句作出决策**。正是作出**决策**的能力使代码具有“智能”。我们可以根据某个条件的 true 或 false，来决定是否执行某个操作。
- **比较运算符**。比较运算符可比较左操作数与右操作数，返回一个布尔值。主要的比较运算符如下：
  - == 左操作数是否等于右操作数？
  - != 左操作数是否不等于右操作数？
  - <= 左操作数是否小于等于右操作数？
  - >= 左操作数是否大于等于右操作数？
  - < 左操作数是否小于右操作数？
  - > 左操作数是否大于右操作数？
- **if 语句**。使用 if 语句可以在条件为 true 时，执行大括号中的代码。if 语句的测试条件放在括号中。如果这个条件为 true，就执行 if 语句后的代码。
- **else 语句**。如果希望代码在 if 语句为 false 时执行，就可以使用 if 语句之后的 else 语句。

- **逻辑运算符**。可以使用 3 个逻辑运算符 AND(&&)、OR(||)、NOT(!)合并多个条件。
  - 当两边的表达式都是 true 时，AND 运算符返回 true。
  - 当两边的表达式有一个或两个是 true 时，OR 运算符返回 true。
  - NOT 运算符反转表达式的逻辑。
- **switch 语句**。switch 语句将一系列可能的值与一个表达式的结果进行相比较，它类似于多个 if 语句。
- **循环语句**：for 语句、for...in 语句、while 语句和 do...while 语句。代码块常常需要执行多次，JavaScript 支持使用循环。
  - for 循环。用于将代码循环执行指定的次数。for 循环包括三部分：初始化、测试条件和递增部分。当测试条件为 true 时，循环会继续。每次循环都执行代码块，接着执行 for 循环的递增部分，然后重新计算测试条件，确定递增的结果是否改变了测试条件的结果。
  - for...in 循环。用于遍历数组，且无须知道数组中元素的个数。JavaScript 将自动实现遍历过程，不会遗漏任何元素。
  - while 循环。用于测试条件为 true 时循环执行代码的场合。while 语句包含一个测试条件和一段测试条件为 true 时执行的代码。如果条件永远不是 true，就始终不执行代码。
  - do...while 循环。类似于 while 循环，但 do...while 循环将先执行一次代码，然后，只要测试条件为 true，就继续执行代码。
  - break 和 continue 语句。有时需要提前结束循环，此时应使用 break 语句。一旦遇到 break 语句，就停止执行大括号中的代码块，从右大括号后面的第一条语句开始执行。continue 语句与 break 语句类似，但代码在 continue 语句处停止执行时，并不退出循环，而是继续下一次循环，就好像执行到该次迭代的末尾一样。
- **函数**是可重用的代码单元。JavaScript 提供了大量的内置函数供程序员使用，如将字符串转换为数值的函数。此外，JavaScript 还允许通过 function 关键字创建和使用自定义函数。函数可以有 0 个或多个参数，可以根据需要返回一个值。
- **变量的作用域和生存期**。在函数外部声明的变量是全局变量，在代码的任何地方都可以访问全局变量。在函数内定义的变量是该函数的私有变量，不能在函数外部访问。变量有生存期，其生存长度取决于声明变量的地方。全局变量的生存期就是页面的生存期，页面加载到浏览器中时，全局变量就始终存在。对于在函数内定义的局部变量，它的生存期就是函数的执行期间。函数执行完毕后，就会释放变量，其值将会丢失。如果在后面的代码中再次调用函数，其变量将是空的。

3.5　习题

在附录 A 中可以找到本章习题的参考答案。

习题 1：

下面是某个初学者编写的一段代码，它不能正常运行。请指出代码中的错误并改正。

```
var userAge = prompt("Please enter your age");

if (userAge = 0);
{
   alert("So you're a baby!");
}
else if ( userAge < 0 | userAge > 200)
   alert("I think you may be lying about your age");
else
{
   alert("That's a good age");
}
```

习题 2：

使用 document.write()，输出 12 的乘法表，输出应如下所示：

```
12 * 1 = 12
12 * 2 = 24
12 * 3 = 36
...
12 * 11 = 132
12 * 12 = 144
```

习题 3：

将习题 2 中的代码改为一个函数，其参数是乘法表以及起始值和终止值。例如，计算 4 的乘法表，就是从 4*4 开始，一直计算到 4*9。

习题 4：

修改习题 3 的代码，要求用户输入一个数值，然后显示这个数的乘法表。继续要求用户输入，并显示乘法表，直到用户输入 - 1。另进行检查，确保用户输入有效的值，如果用户输入无效，则提示用户重新输入。

第 4 章

常见错误、调试和错误处理

即使是 JavaScript 专家也会犯错，哪怕只是拼写错误。尤其是代码扩展到几百行之后，出错的几率就显著增加了。相应地，发现这些错误或缺陷的难度也增加了。本章将介绍有助于最小化这种情况下的问题的各种技术。

首先讨论 JavaScript 代码中最常见的 7 种错误。了解到这些错误后，就可以在编写代码时特别留意它们，减少此类错误的发生次数。

接下来讨论错误发生时该如何处理，以免用户看到编码错误。

最后讨论 Microsoft 的 Internet Explorer (IE8)、Firebug (用于 Firefox 的插件)、Safari 和 Chrome 的 Web Inspector 以及 Opera 的 Dragonfly 中的调试工具，学习如何使用这些工具单步调试代码，在代码运行时查看变量的内容，以找出难以发现的错误。还会简要介绍 Firefox 的调式工具。

4.1 一些常见错误

程序员经常犯的错误有 7 个，其中一些错误随着编程经验的积累是可以避免的，但其他错误仍会常犯。

注意：

把浏览器设置为显示错误是很有用的，具体的设置方法参见 2.3 节。如果还没有设置为显示错误，现在就应予以设置。

4.1.1 未经定义的变量

对于变量在赋值前是否应先定义，JavaScript 的要求非常宽松。例如，下面的代码隐式地创建一个新的全局变量 abc，并赋值 23：

```
abc = 23;
```

但严格地讲，应该使用 var 关键字显式地定义该变量：

```
var abc = 23;
```

是否使用 var 关键字来声明变量对变量的作用域有影响，因此最好使用 var 关键字。如果使用了一个未定义的变量，则将产生一个错误。例如，如果并未事先显式或隐式地定义变量 abc，则下面的代码在 IE8 中将产生如图 4-1 所示的错误。

```
alert(abc);
```

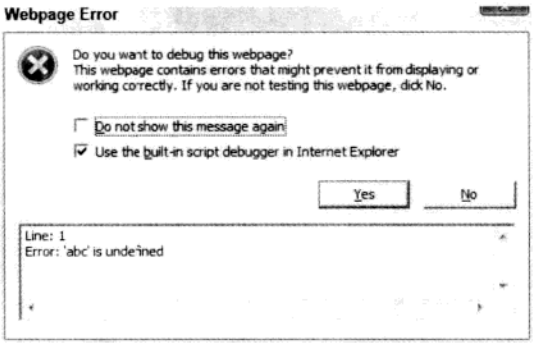

图 4-1

在 Firefox 中，需要在 JavaScript 控制台上查看错误信息，选择 Tools | Error Console 即可打开 JavaScript 控制台。

另外，函数定义也包含了其参数，如果未正确地声明参数，也将遇到同类错误。

例如下面的代码：

```
function foo(parametrOne)
{
   alert(parameterOne);
}
```

如果调用该函数，将产生如图 4-2 所示的错误消息。

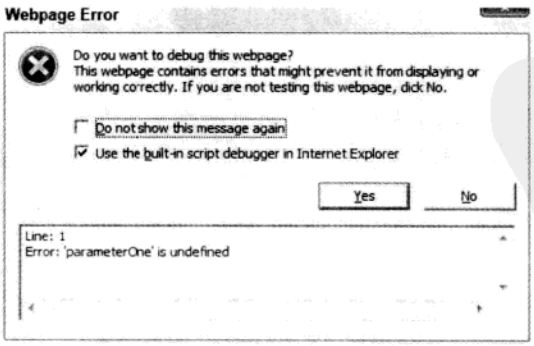

图 4-2

这个错误是函数定义中的一个简单的拼写错误。第一个参数的拼写错误：它应是 parameter

One，而不是 parametrOne，这类错误常常令人迷惑，尽管浏览器指出某行有错误，但该错误的源头在另一行。

4.1.2　区分大小写

大小写错误是最常见的错误之一，因为它有时可能难以发现。

例如，请找出下面代码中的三个大小写错误：

```
var myName = "Jeremy";
If (myName == "jeremy")
   alert(myName.toUppercase());
```

第一个错误是将关键字 if 写成了 If。但是，JavaScript 不会报告大小写错误，IE 将报告“Object expected”错误，而 Firefox 报告“If is not defined”错误。尽管这些错误消息提示了发生了什么错误，但往往并不直接。在这个例子中，IE 认为用户要调用一个 If 对象，而 Firefox 认为用户要引用一个未定义的函数 If。

找到第一个错误后，接着查找第二个错误，它不是一个 JavaScript 语法错误，而是一个逻辑错误。在 JavaScript 中，Jeremy 不等于 jeremy，因此，myName=="jeremy"返回 false，即使我们并不关心该单词是 Jeremy 还是 jeremy，也是如此。这种错误根本不会产生任何错误消息，但是代码不会按照预期的方式执行。

第三个错误是 myName 包含的 String 对象的 toUpperCase()方法存在拼写错误。上面的代码使用了 toUppercase()，其中 C 是小写。IE 会给出消息“对象不支持该属性或方法”，而 Firefox 报告“myName.toUppercase 不是一个函数”。乍一看，这个小错误很容易改正，并开始在 JavaScript 参考手册中查找该方法。但为什么 JavaScript 参考手册中有这个方法，而代码不工作？再次重申，总是需要考虑大小写问题，因为即使是专家也常常犯这种错误。

4.1.3　不匹配的大括号

下面的代码定义了一个函数，然后调用它。但是，其中有一个故意制造的错误，它在哪里？

```
function myFunction()
{
x = 1;
y = 2;
if (x <= y)
{
if (x == y)
{
alert("x equals y");
}
}
myFunction();
```

这体现了设置代码格式的重要性——在下面的代码中找出错误要容易许多：

```
function myFunction()
{
    x = 1;
    y = 2;
    if (x <= y)
    {
       if (x == y)
       {
          alert("x equals y");
       }
    }
    myFunction();
```

现在，很容易看出遗漏了函数的结束大括号。当代码包含了多个 if、for 或者 do while 语句时，非常容易少写或者多写结束大括号。而在格式化的代码中，发现这类问题会容易得多。

4.1.4 不匹配的圆括号

看看下面的代码：

```
if (myVariable + 12) / myOtherVariable < myString.length)
```

找出其中的错误吗？这个错误是条件的开头少了一个圆括号。我们希望先计算 myVariable + 12，再计算该值除以 myOtherVariable，因此，需要将 myVariable + 12 放在圆括号中：

```
(myVariable + 12) / myOtherVariable
```

但是，if 语句的条件也必须放在圆括号中。不但少了开始的圆括号，而且结束的圆括号也比开始的圆括号多一个。与大括号一样，每个开始圆括号都必须对应一个结束圆括号。下面的代码是正确的：

```
if ((myVariable + 12) / myOtherVariable < myString.length)
```

如果有多个开始或结束的圆括号，则非常容易遗漏圆括号或者多写了圆括号。

4.1.5 赋值而不是相等

考虑下面的代码：

```
var myNumber = 99;
if (myNumber = 101)
{
   alert("myNumber is 101");
}
else
{
   alert("myNumber is " + myNumber);
}
```

乍一看，可能会认为 if 语句的 else 子句中的 alert()方法会执行，并提示 myNumber 中的数值为 99，但是事实并非如此。这段代码有一个典型错误：将相等运算符(==)写成了赋值运算符(=)。因此，这段代码并没有比较 myNumber 和 101，而是将 myNumber 设置为 101。如果使用 Visual Basic 等仅用一个等号表示赋值和比较操作的语言编程，在 JavaScript 中就会常犯这种错误。

找出这种错误的难处是，它不会产生任何错误消息，只是数据和逻辑会出问题。在 if 语句中给变量赋值可能是错误的，但完全合法，因此 JavaScript 不会给出错误消息。在较长的代码块中，这种错误很容易被忽略。所以，下次程序的逻辑出现混乱时，应检查一下这种错误。

4.1.6　将方法和属性混为一谈

另一个常见的错误是在无参数的方法名后没有括号，或者在属性名后带上了多余的括号。

当调用方法时，必须总是在方法名后加上括号；否则 JavaScript 会认为这是一个指向方法或属性的指针。例如，检查下面的代码：

```
var nowDate = new Date();
alert(nowDate.getMonth);
```

第一行创建了 Date 引用类型的一个实例，第二行尝试调用新建对象 Date 的 getMonth()方法，但是在方法名后少了括号。正确的代码如下所示：

```
var nowDate = new Date();
alert(nowDate.getMonth());
```

在方法名后必须有一对括号，但在属性名后不能添加括号，否则 JavaScript 会认为用户试图调用该对象的一个方法：

```
var myString = "Hello World!";
alert(myString.length());
```

第二行在 length 属性后添加一对括号，使 JavaScript 认为这是一个方法。应该将上面的代码改成：

```
var myString = new String("Hello");
alert(myString.length);
```

还有更棘手的问题。我们常常把一个函数作为参数传送给另一个函数(或一个属性，参见第 6 章对处理事件的讨论)。在这些情况下，传送函数时，函数名的后面没有开闭括号()。看看下面的代码：

```
function foo()
{
    alert("I' m in foo()!").
}

function bar(fpToCall)
```

```
{
    alert("Calling passed function").
    fpToCall();
}

bar(foo);
```

这段代码定义了两个函数 foo()和 bar()。foo()函数仅显示一个消息框，告诉用户 foo()函数当前正在运行。第二个函数 bar()接受一个函数实参，它显示一个消息，说明它正在调用所传送的函数，然后执行该函数。最后一行调用 bar()函数，并给它传送 foo()函数的指针。指针是对内存中某位置的引用(下一章将讨论内存引用)。

一般规则是：在执行函数时，应在函数名的后面加上括号；把一个函数传送给另一个函数或属性时，应省略括号。

4.1.7 在连接字符串时未使用加号(+)

下面的代码有一个故意制造的连接错误。

```
var myName = "Jeremy";
var myString = "Hello";
var myOtherString = "World";
myString = myName + " said " + myString + " " myOtherString;
alert(myString);
```

在第四行代码中，" "和 myOtherString 之间应该有一个 + 运算符。

在只有几行的代码中，很容易找出这个错误，但在较长的代码段中，很难发现这种错误。并且，这类问题的错误消息可能有误导作用。在浏览器中加载上面的代码，则 IE 将报告“Error: Expected”，而 Firefox 报告“Missing ; before statement”。这种错误出现频率之高令人惊讶。

这些是程序员最易犯的错误。还有其他类型的错误，称为运行时错误，在浏览器中执行代码时发生，它们不一定是由输入错误、遗漏了大括号、遗漏了圆括号或前面讨论的其他错误引起的。这类错误仍旧可以检查出来，如下一节所述。

4.2 错误处理

编写代码时，我们希望报告所有错误。但是我们最不希望的是当最后将代码部署到 Web 服务器上，供所有人访问时，用户会看到错误消息。当然，编写无错误的代码是一个良好的开端，但应该谨记以下几点：

- 有时超出控制范围的条件会导致错误。例如，代码依赖于某个基础条件，如 Java 小程序，而用户的计算机上没有安装它，且无法检查 Java 小程序是否安装时，就会导致错误。
- 墨菲定律指出，只要有可能出错，就一定会出错！

4.2.1　避免错误

处理错误的最佳办法就是从一开始就避免错误的产生。这听起来似乎是句空话，但要编写无错误的页面，有大量的工作要做。

- 在尽可能多的浏览器上全面检查页面。在一些操作系统上，是说易行难。另一个可选的办法是确定要在哪些浏览器上支持网页，再验证代码在这些浏览器上是否工作正常。使用本书前面的浏览器检查代码，给不支持的用户发送一个漂亮的、安全的、缩减了功能的网页，或者简单给用户提供一个消息，说明当前页面不支持它们的平台和(或)浏览器。
- 验证数据。如果用户输入的无效数据会导致程序失败，用户就有可能输入无效的数据。如果某个文本框为空将导致代码失败，就要确保在文本框中输入了数据。如果需要一个整数，就必须确保用户输入了整数。用户输入的日期是否有效？用户输入的 e-mail 地址 mind your own business 是否有效？它是无效的，所以必须检查它是否使用 something@ something.something 这样的格式。

我们仔细地检查了网页，其中没有语法错误或者逻辑错误。再加入数据验证，保证用户输入的每个数据都使用了有效格式。即便如此，仍有可能出错，可能出现无法处理的问题。下面就是一个真实的例子。

本书作者之一 Paul 创建了一个在线留言板，它依赖一个 Java 小程序在浏览器和服务器之间传送数据，而无需重载页面。Paul 检查了代码，确保它们都是正确的。留言板启动后，一直工作正常，但是有大约 5%的情况下，Java 小程序初始化后，因为用户位于某种防火墙(防火墙是一种禁止黑客侵入本地计算机网络的方式)之后，会产生一个错误。由于无法确定用户是否处于这种防火墙之后，因此对于此种特殊情况，我们无能为力。

实际上，JavaScript 包含 try...catch 语句，它可以尝试运行代码，如果失败，catch 子句就捕获错误，并根据需要加以处理。例如，对于上面的留言板，Paul 使用 try...catch 语句捕获 Java 小程序的故障，并将用户重定向到一个不需要使用 Java 小程序、仍然可以显示消息的简化页面。

4.2.2　try...catch 语句

try...catch 语句成对使用，不能仅使用 try 子句，或者仅使用 catch 子句。使用 try 子句，可以定义一个尝试运行的代码块，用 catch 子句来定义一个代码块，如果 try 子句定义的代码块在运行过程中产生异常，就执行 catch 子句定义的代码块。术语“异常(exception)”是这里的关键，它指的是不可预知的特殊情况。而“错误(error)”指的是编写不正确的代码。如果没有发生异常，就不执行 catch 子句中的代码。catch 子句还允许你获取异常消息的内容，如果没有使用 catch 子句来捕获异常，则这些异常消息会显示给用户。

下面创建一个 try...catch 子句的简单例子。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
    <title>Try/Catch</title>
</head>
<body>
<script type="text/javascript">
try
{
    alert('This is code inside the try clause');
    alert('No Errors so catch code will not execute');
}
catch(exception)
{
    alert("The error is " + exception.message);
}
</script>
</body>
</html>
```

将上面的代码保存为 trycatch.htm 页面。

这段代码首先定义 try 子句，与其他代码块一样，try 块也放在大括号中。

接下来定义 catch 语句。这段代码在 catch 关键字之后的圆括号中包含 exception。这个 exception 仅是一个变量名。它保存一个 Error 类型的对象，称为 exception 对象，其中包含执行 try 代码块时抛出的异常信息。此处使用了 exception，实际上可以使用任何合法的变量名。例如，catch(exceptionObject)也不错，且更明了。

exception 对象包含几个提供异常信息的属性。只是 IE 中的 exception 对象与其他浏览器中的 exception 对象有些不同(甚至 Firfox、Opera、Safari 和 Chrome 中的 exception 对象有不同的属性)，但它们是类似的，如果仅关心异常消息和类型，则编写跨浏览器的代码不会有问题。

所有的主流浏览器都支持 name 属性和 message 属性，name 属性包含错误的类型名，message 属性包含用户看到的错误消息。这些属性都是 ECMAScript 3 标准的一部分。

回头分析代码。在 catch 语句后面的大括号中，是发生异常时才执行的代码块。在这个例子中，try 代码块执行正常，所以不执行 catch 块中的 alert()方法。

故意插入一个错误：

```
try
{
    alert('This is code inside the try clause');
    ablert ('Exception will be thrown by this code');
}
catch(exception)
{
    alert("The error is " + exception.message);
}
```

重新保存文档，在浏览器中重载页面。try 代码块中的第一个 alert()方法会正常执行，将警告框显示给用户。但是，第二个 alert()语句会出错，代码将从 catch 块的第一个语句开

始执行。

如果使用 Internet Explorer，则所显示的错误说明是 Object expected，如果使用另一个浏览器，对该错误的解释就会不同，报告为 ablert is not defined。

如果再次修改代码，使之包含另一个错误，就会看到重要的消息。

```
try
{
    alert('This is code inside the try clause');
    alert('This code won't work');
}
catch(exception)
{
    alert("The error is " + exception.message)
}
```

在浏览器中加载修改过的代码，会显示一个通常的浏览器错误消息“Expected ')'，而不是显示 catch 块中的警告框。这是因为这段代码有一个语法错误，函数和方法是有效的，但有一个字符无效。单词 won't 中的单引号会结束传送给 alert()方法的字符串参数。JavaScript 的语法规则指定，此处应出现一个闭括号，但这段代码中没有闭括号。在执行代码之前，浏览器的 JavaScript 引擎会遍历所有代码，检查语法错误或违反 JavaScript 规则的代码。如果引擎找到了语法错误，浏览器就按通常方式处理它，try 子句不会运行，因此不处理语法错误。

1. 抛出错误

throw 语句可用于在 try 代码块中创建自定义的运行时错误。既然有错误的代码会生成错误，为什么还要创建一个语句来生成错误呢？

抛出错误非常有助于表示无效的用户输入等问题。为了指出问题(如用户输入无效)，错误抛出是非常有用的。与其使用许多 if...else 语句，不如检查用户输入的有效性，再使用 throw 在出错的地方停止代码的执行，并由 catch 代码块中的错误捕获代码接管。在 catch 子句中，可以判断错误是否源于用户的输入，如果是，则可以提示用户出了什么错，以及如何修改错误。如果是一个意外的错误，则也可以采用较正常的方式处理它，而不是报告一大堆 JavaScript 错误。

要使用 throw，可以输入 throw，并将错误消息放在其后。

```
throw "This is my error message";
```

注意，在 catch 子句中捕获 exception 对象时，可以获取已抛出的错误信息。在该例中，throw 语句包含一个字符串，但实际上可以抛出任何类型的数据，包括数值或者对象。

试一试　try...catch 与 throw 语句

下面的例子将创建一个简单的阶乘计算器。这个例子的重要部分是 try...catch 子句和 throw 语句。该例子是一个框架集页(frameset page)，用来演示无法处理的错误。在本例中，页面依赖于一个在框架集页面中定义的函数，因此，如果单独加载该页面，就会产生一个

错误。

首先创建一个页面，以便定义框架集，并包含一个重要的函数。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Example</title>
<script type="text/javascript">
function calcFactorial(factorialNumber)
{
    var factorialResult = 1;
    for (; factorialNumber > 0; factorialNumber-- )
    {
        factorialResult = factorialResult * factorialNumber;
    }
    return factorialResult;
}
</script>
</head>
<frameset cols="100%,*">
   <frame name="fraCalcFactorial" src="calcfactorial.htm" />
</frameset>
</html>
```

把这个页面保存为 calcfactorialtopframe.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Example</title>
<script type="text/javascript">
function butCalculate_onclick()
{
    try
    {
        if (window.top.calcFactorial == null)
            throw "This page is not loaded within the correct frameset";
        if (document.form1.txtNum1.value == "")
            throw "!Please enter a value before you calculate its factorial";
        if (isNaN(document.form1.txtNum1.value))
            throw "!Please enter a valid number";
        if (document.form1.txtNum1.value < 0)
            throw "!Please enter a positive number";

        document.form1.txtResult.value =
         window.parent.calcFactorial(document.form1.txtNum1.value);
    }
```

```
        catch(exception)
        {
            if (typeof(exception) == "string")
            {
                if (exception.charAt(0) == "!")
                {
                    alert(exception.substr(1));
                    document.form1.txtNum1.focus();
                    document.form1.txtNum1.select();
                }
                else
                {
                    alert(exception);
                }
            }
            else
            {
                    alert("The following error occurred " + exception.message);
            }
        }
    }

    </script>
    </head>
    <body>
    <form action=" " name="form1">
       <input type=" text" name=" txtNum1" size="3" /> factorial is
       <input type=" text" name=" txtResult" size="25" /><br />
       <input type=" button" value="Calculate Factorial"
         name="butCalculate" onclick="butCalculate_onclick()" />
    </form>
    </body>
    </html>
```

将这个页面保存为 calcfactorial.htm。然后在浏览器中加载第一个页面 calcfactorialtop-frame.htm。

该页面包含一个简单的表单，表单中有两个文本框和一个按钮。在第一个文本框中输入数字 4，并单击 Calculate Factorial 按钮，于是计算 4 的阶乘(即 24)，并显示在第二个文本框中，如图 4-3 所示。

数的阶乘是所有小于等于该数的正整数之积。例如，4 的阶乘(记为 4!)就是 1*2*3*4=24。阶乘用于数学的多个分支，包括统计学。本例仅希望创建一个函数，它执行足够复杂的操作，需要使用函数，但不至于令读者偏离本例的主要目标：try...catch 语句和 throw 语句。

如果清空第一个文本框，再单击 Calculate Factorial 按钮，则程序提示需要输入一个值。如果在第一个文本框中输入一个无效的非数字字符，则程序提示输入一个合法值。如果输入一个负数，则程序提示输入一个正数。

另外，如果试图在浏览器中加载 calcfactorial.htm 页面，并在文本框中输入值后，单击 Calculate Factorial 按钮，则程序提示当前页面没有加载到正确的框架集中。

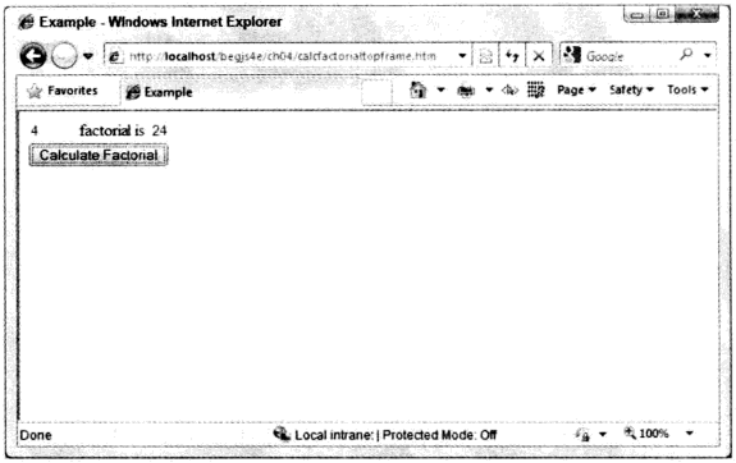

图 4-3

可以看到，所有的错误消息都是使用 try...catch 语句和 throw 语句创建的。

由于该示例主要是关于 try...catch 和 throw 的，因此下面仅分析 calcfactorial.htm 页面，特别是 butCalculate_onclick()函数，该函数连接到表单中按钮的 onclick 事件处理程序。

首先看看 try 子句及其包含的代码。这段代码包含 4 个 if 语句，最后一行代码将计算出来的阶乘放在第二个文本框中。每个 if 语句都检查一个条件，如果该条件为 true，则从代码中抛出一个异常。

第一个 if 语句检查位于顶层框架集窗口的 calcFactorial()函数是否真的存在。如果该函数不存在，则抛出一个错误，并由 catch 块捕获。如果没有这个 throw 语句，则当用户加载 calcfactorial.htm 页面(而不是框架集页面 calcfactorialtopframe.htm)时，代码就会失败。

```
try
{
    if (window.top.calcFactorial == null)
        throw "This page is not loaded within the correct frameset";
```

接下来的三个 if 语句检查用户在文本框中输入的数据的有效性。首先确保用户在文本框中输入了值，再确保用户输入的是一个数字，最后检查该值是否是一个非负数。如果任意一个 if 条件为 true，则抛出一个错误，并由 catch 块捕获。所定义的每个错误消息均以一个感叹号标记开头，以便将这些错误标识为用户输入错误，而不是其他错误(比如不在框架集中)。

```
if (document.form1.txtNum1.value == "")
    throw "!Please enter a value before you calculate its factorial";
if (isNaN(document.form1.txtNum1.value))
    throw "!Please enter a valid number";
if (document.form1.txtNum1.value < 0)
    throw "!Please enter a positive number";
```

如果一切顺利，则执行 calcFactorial()函数，用用户输入的数字的阶乘填充结果文本框。

```
    document.form1.txtResult.value =
      window.parent.calcFactorial(document.form1.txtNum1.value);
    }
```

最后，看看 try...catch 语句的 catch 部分。首先，try 代码抛出的任何消息都由变量 exception 捕获。

```
    catch(exception)
    {
```

错误的抛出方式决定了 exception 包含的数据类型。如果错误由浏览器抛出，而不是由代码抛出，则 exception 是一个对象。如果错误由代码抛出，例如在本例中，抛出的是一个简单的字符串，则 exception 包含该字符串。因此，首先要确定 exception 包含的数据类型。如果 exception 包含一个字符串，则说明该错误由代码抛出，并可以进行相应的处理。如果 exception 包含一个对象，说明代码并没有抛出对象，肯定是浏览器生成了这个异常，而且 exception 变量是一个 exception 对象。

```
    if (typeof(exception) == "string")
    {
```

如果代码使用 throw 语句抛出了异常(则 exception 是一个字符串)，就需要判断该错误是用户输入错误，例如文本框没有包含要计算的值；还是其他类型的错误，如页面没有加载到框架集中。所有用户输入异常消息都以一个感叹号开头，因此，使用 if 语句检查第一个字符。如果它是感叹号！，则提示用户输入有错误，并将焦点返回给控件。否则，就只显示错误消息。

```
        if (exception.charAt(0) == "!")
        {
            alert(exception.substr(1));
            document.form1.txtNum1.focus();
            document.form1.txtNum1.select();
        }
        else
        {
            alert(exception);
        }
    }
```

如果exception不是字符串，则说明exception包含一个exception对象，需要显示message属性：

```
        else
        {
            alert("The following error occurred " + exception.message);
        }
    }
```

2. 嵌套的 try...catch 语句

前面仅使用了一个 try...catch 语句，也可以把一个 try...catch 语句包含在另一个 try 语句中，甚至可以在这个内层的 try...catch 语句的 try 语句中包含另一个 try...catch 语句，嵌套的层数取决于代码的意义。

那么，为什么要使用嵌套的 try...catch 语句？因为可以处理内层 try...catch 语句中的某些错误。如果处理一个较严重的错误，内层的 catch 子句可以将错误抛出，把它传送给外层的 catch 子句。

下面是一个例子。

```
try
{
    try
    {
       ablurt ("This code has an error");
    }
    catch(exception)
    {
       var eName = exception.name;

       if (eName == "TypeError" || eName == "ReferenceError")
       {
           alert("Inner try...catch can deal with this error");
       }
       else
       {
           throw exception;
       }
    }
}
catch(exception)
{
    alert("Error the inner try...catch could not handle occurred");
}
```

上面的代码有两对 try...catch 语句，一个嵌套在另一个中。

内层的 try 子句中包含一行有错误的代码。内层 try...catch 语句的 catch 子句检查该错误的 name 属性值。如果该异常名是 TypeError 或 ReferenceError，则内层的 try...catch 语句通过警告框来处理该异常(请参见附录 B，来了解错误类型及其描述)。浏览器抛出的错误类型取决于浏览器本身。在上面的示例中，IE 把错误报告为 TypeError，其他浏览器报告为 ReferenceError。

如果内层catch子句捕获的异常是其他任何错误类型，则该异常会传送给外层try...catch语句的 catch 语句。

下面修改前一个例子 calcfactorial.htm 中的 butCalculate_onclick()函数，使之包含一个内层 try...catch 语句和一个外层 try...catch 语句：

```
function butCalculate_onclick()
```

```
{
    try
    {
        try
        {
            if (window.top.calcFactorial == null)
                throw ("This page is not loaded within the correct frameset");
            if (document.form1.txtNum1.value == "")
                throw("!Please enter a value before you calculate its factorial");
            if (isNaN(document.form1.txtNum1.value))
                throw("!Please enter a valid number");
            if (document.form1.txtNum1.value < 0)
                throw("!Please enter a positive number");
            document.form1.txtResult.value =
                window.parent.calcFactorial(document.form1.txtNum1.value);
        }
        catch(exception)
        {
            if (typeof(exception) == "string" && exception.charAt(0) == "!")
            {
                alert(exception.substr(1));
                document.form1.txtNum1.focus();
                document.form1.txtNum1.select();
            }
            else
            {
                throw exception;
            }
        }
    }
    catch(exception)
    {
        switch (exception)
        {
            case "This page is not loaded within the correct frameset":
            alert(exception);
            break;
            default :
            alert("The following critical error has occurred \n" + exception);
        }
    }
}
```

内层的 try...catch 语句处理用户输入错误。如果该错误不是用户输入错误，则将其抛给外层的 catch 子句。外层的 catch 子句使用一个 switch 语句检查所抛出的错误消息的值。如果该错误消息是由于未加载 calcfactorialtopframe.htm 而抛出，则 switch 语句在第一个 case 分支中处理它。任何其他错误都在 default 子句中加以处理。当然，有时存在多个要在 case 语句中处理的不同错误。

3. finally 子句

try...catch 语句可以包含一个 finally 子句，无论是否抛出异常，都会执行它定义的代码块。finally 子句不能独立使用，而必须放在 try 块的后面，如下面的代码所示：

```
try
{
   ablurt("An exception will occur");
}
catch(exception)
{
   alert("Exception occurred");
}
finally
{
   alert("Whatever happens this line will execute");
}
```

finally 子句适于放置一些清理代码，无论前面是否发生异常，都需要执行这些清理代码。

前面介绍了开发人员常犯的一些错误，还讨论了如何处理代码中的错误。但代码中仍有错误，所以下面介绍一种使用调试器修复错误的简便方法。

4.3 调试

传统上，JavaScript 被认为是一种难以编写和调试的语言，因为它缺乏合适的开发工具。现在情况有了改观，因为开发人员可以使用的工具有许多。最著名的是用于 Internet Explorer、Firefox、Safari 和 Opera 的调试工具。有了这些工具，就可以利用断点暂停脚本的执行，然后逐行执行代码，看看代码究竟执行了什么操作。

还可以随时查看变量中存储了什么数据，执行了什么语句。而如果没有调试器，就只能在代码中使用 alert()方法，在不同的地方显示变量状态。

调试过程对于所有浏览器(甚至所有语言)而言都是相同的，一些调试工具提供的功能可能比其他调试器多，但大多数情况下，下面的概念可用于所有调试器：

- 断点告诉浏览器应在某点中断或暂停代码的执行。可在 JavaScript 代码的任意地方设置断点，调试器会在遇到断点时暂停代码的执行。
- 观察窗口可以指定代码在断点处暂停时要查看的变量。
- 调用堆栈记录了到断点处已执行的函数和方法。
- 控制台可以在页面上下文和断点的作用域内执行 JavaScript 命令。另外，它还给页面上找到的所有 JavaScript 错误分类。
- 单步执行是调试过程中最常见的过程，它可以一次执行一行代码。单步执行代码可以通过 3 种方式进行：
  - Step Into 执行下一行代码。如果这行代码是一个函数调用，调试器就执行函数，并在函数的第一行处暂停。

- Step Over 与 Step Into 一样，也是执行下一行代码，如果这行代码是一个函数调用，Step Over 就执行整个函数，在函数外部的第一行代码处暂停。
- 在被调用的函数内部执行时，Step Out 会返回调用函数。Step Out 会恢复代码的执行，直到函数返回为止，接着在函数的返回点处中断。

在深入分析各种浏览器之前，先创建一个可供调试的页面。注意第 16 行故意输入了一个错误。如果从头开始创建该页面，应确保包含这个输入错误。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Debug: Times Table</title>
    <script type="text/javascript">
        function writeTimesTable(timesTable)
        {
            var counter;
            var writeString;
            for (counter = 1; counter < 12; counter++)
            {
                writeString = counter + " * " + timesTable + " = ";
                writeString = writeString + (timesTable * counter);
                writeString = writeString + "<br />";
                documents.write(writeString);
            }
        }
    </script>

</head>
<body>
        <script type="text/javascript">
            writeTimesTable(2);
        </script>
</body>
</html>
```

保存为 debug_timestable.htm。

下一节介绍用于 Firefox 的 Firebug 插件的特性和功能。由于调试器的调试功能比较通用，所以介绍 Internet Explorer、Safari 和 Opera 的各节仅讨论每个浏览器的调试器 UI，并指出它们之间的区别。

4.3.1　在 Firefox 中用 Firebug 调试

多年来，可用于 Firefox 的 JavaScript 调试器只有一个 Mozilla 项目，其代码名称是 Venkman，Venkman 的特性集类似于 Microsoft 的脚本编辑器，但许多开发人员觉得 Venkman 不大友好。其中一位开发人员 Joe Hewitt 决定使用 Firefox 中内置的调试 API(application programming interface)编写自己的调试器，他将这个调试器命名为 Firebug。目前，Firebug 是 Firefox 的首选 JavaScript 调试器，用于其他浏览器的所有其他 JavaScript(和 Web 开发)

工具实际上都基于 Firebug。

但默认情况下，Firebug 并不随 Firefox 一起安装，你必须安装 Firebug 插件。Firebug 的最新版本可以从 http://www.getfirebug.com、Joe Hewitt 的网站 http://www.joehewitt.com/software/firebug/，或 Mozilla 的插件站点 https://addons.mozilla.org/en-US/firefox/addon/1843 上下载。

要安装 Firebug，可打开 Firefox，进入上述 URL 中的一个，单击网页上的 Install 按钮，然后按照提示操作。在安装 Firebug 后确保重启 Firefox。

可以通过两种方式来访问 Firebug。在 Firefox 窗口的右下角可以单击状态栏中的 Firebug 图标。如果 Firefox 的状态栏不可见，可从 Firefox 的 Tools 菜单中选择 Firebug |Open Firebug，打开 Firebug。Firebug 默认打开为 Firefox 中的一个面板，如图 4-4 所示。

图 4-4

单击关闭按钮旁边的向上箭头，可使 Firebug 显示在它自己的窗口中。

在 Firefox 中打开 debug_timestable.htm。如果状态栏是可见的，在 Firefox 窗口的右下角就会显示红色文本“1 Error.”。单击该消息(或通过 Tools 菜单打开 Firebug)，Firebug 就会在控制台上打开。该控制台在 Firebug 中有多种用途，它可以列出 JavaScript 错误，还允许随时执行 JavaScript 代码。后面将使用控制台。

JavaScript 调试器包含在 Script 选项卡中，由两个面板组成。左面板包含源代码，右面板可显示 3 个不同的视图：Breakpoints、Watch 和 Stack。

- Breakpoints：列出在当前页面中给代码创建的所有断点。
- Watch：列出作用域内的变量及其在断点处的值，还可以添加其他要观察的变量。
- Stack：显示调用堆栈。

左面板中的源代码是只读的，如果想修改这些源代码，必须在文本编辑器中编辑它们。下面在文本编辑器中把第 16 行出错的 documents 改为 document，保存并重载该网页。

改正了错误，重载页面后，就应在网页上看到乘法表，如图 4-5 所示。

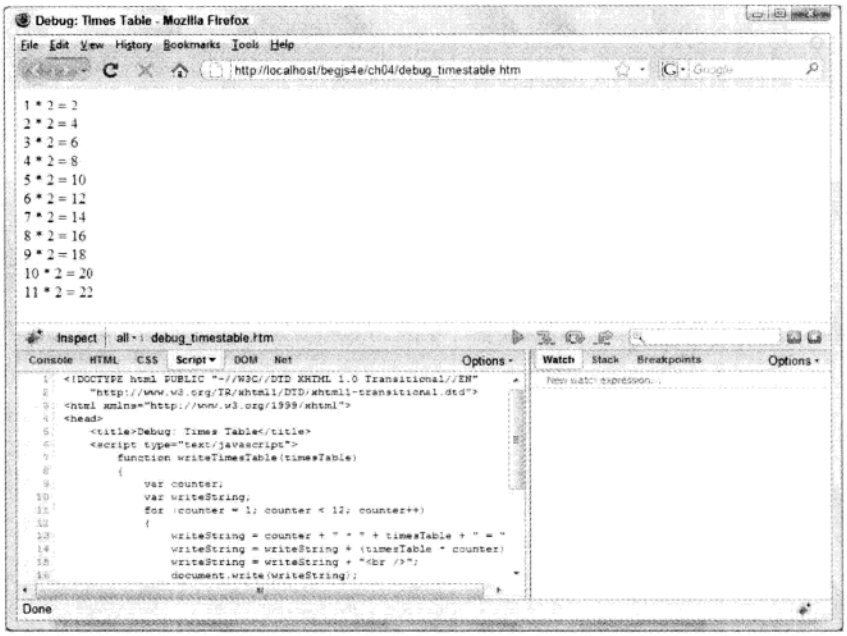

图　4-5

还要注意 Firebug 左面板上的源代码也更新显示前面进行的改变。

1. 设置断点

如前所述，断点告诉调试器在代码的指定位置暂停执行。如果需要在执行代码时检查代码，这就是很方便的。在 Firebug 中创建断点非常简单：只需用左键单击源代码行号左侧的灰色区域(页边，gutter)即可。单击页边时，断点用一个红色的圆圈表示。

在编写代码时，还可以使用 debugger 关键字来创建断点(后面将使用这种方法)。

把正确的 debug_timestable.htm 加载到 Firefox 中，在第 14 行上创建一个断点：

```
writeString = writeString + (timesTable * counter);
```

重载页面，注意 Firebug 在刚才创建的断点处停止执行代码。Firebug 用淡黄色突出显示当前代码行，并在页边显示一个黄色箭头。这行代码尚未执行。

单击右面板上的 Breakpoints 选项卡，就会显示出断点列表(本例只有一个断点)。列表中的项包括一个用于启用/禁用断点的复选框、包含该断点的函数名、源文件的文件名和行号、断点的源文本以及一个 Delete 按钮。

现在单击 Watch 选项卡。

2. 观察窗口

暂停执行代码时，Watch 选项卡会显示当前行中的变量及其在作用域中的值。图 4-6 显示了这个断点处的 Watch 选项卡中的内容。

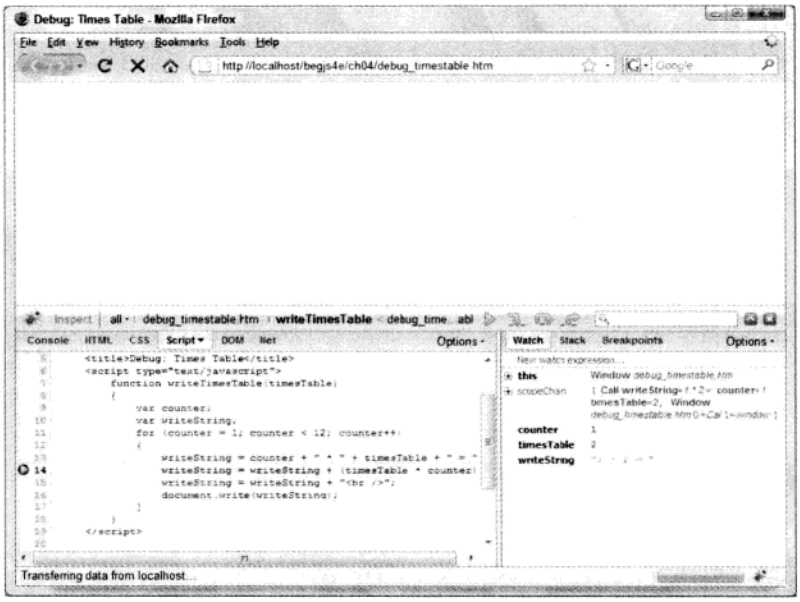

图 4-6

注意 counter、timesTable 和 writeString 变量是可见的。

在单步调试代码时，还可以添加自己需观察的变量，检查它们的值。要添加一个加以观察的变量，可以单击“New watch expression...”，输入要观察的变量名，并按下回车键。自己添加的要观察的变量有灰色的背景，把鼠标移动到该变量上，会显示一个红色的 Delete 按钮。

可以观察任意变量。如果该变量在作用域内，就会显示该变量的值。如果变量超出作用域范围，就显示 ReferenceError 作为其值。

当需要确定代码执行了什么操作时，这些信息是有帮助作用的，但如果不能控制代码的执行，这些信息的作用很小。设置一个断点，多次重载页面，以便执行到下一行代码的做法其实并不实用，所以我们使用“单步调试”过程。

3. 单步调试代码

代码的单步执行由窗口右上角源代码搜索框旁边的 4 个按钮控制，如图 4-7 所示。

图 4-7

- Continue(快捷键是 F8)：其功能是继续执行代码，直到下一个断点或到达程序的末尾。
- Step Into(快捷键是 F11)：执行当前代码行，并移动到下一个语句上，如果当前行是一个函数，就跳到该函数的第一行上。
- Step Over(快捷键是 F10)：与 Step Into 类似，Step Over 也执行当前代码行，并移动到下一个语句上。但如果当前语句是一个函数，Step Over 就执行函数，跳到函数调用后面的一行上。
- Step Out：返回调用函数。

下面执行一些单步调试，按照下面的步骤进行：

(1) 单击 Step Into 图标或按下 F11 键，逐行执行代码。调试器会执行当前突出显示的代码行，并移动到下一行代码上。

(2) 查看 Watch 选项卡中 writeString 的值，它是“1 * 2 = 2”。可以看出，显示在 Watch 选项卡中的值是实时更新的。

(3) Firebug 的一个功能是在单步调试代码时，可以根据需要更新页面。再单击两次 Step Into 按钮，就可以看到这个功能。图 4-8 显示了单步调试代码时更新的页面。

图　4-8

我们发现，单步调试的函数并非错误的根源，所以希望执行该函数剩余的代码行，从调用该函数的地方继续单步调试。为此，单击 Step Out 图标，跳出该函数。但是，如果在循环中执行代码，而在该循环中设置了一个断点，那么在执行完循环后才能跳出函数。

有时，在调用了许多函数的代码中有一个错误。如果事先知道某些函数是没有错误的，就可以一次执行完这些函数，而不是单步调试，查看它们的执行情况。在这些地方使用 StepOver 可以一次执行完函数中的代码，而不需要单步执行。

修改 debug_timestable.htm 中乘法表的代码，如下所示，以便使用它执行 3 种单步调试操作：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>Av
    <title>Debug: Times Table 2</title>

    <script type="text/javascript">
        function writeTimesTable(timesTable)
        {
            var counter;
            var writeString;
            for (counter = 1; counter < 12; counter++)
            {
                writeString = counter + " * " + timesTable + " = ";
                writeString = writeString + (timesTable * counter);
                writeString = writeString + "<br />";
                document.write(writeString);
            }
        }
    </script>

</head>
<body>
    <script type="text/javascript">
        var timesTable;
        for (timesTable = 1; timesTable <= 12; timesTable++)
        {
            document.write("<p>")
            writeTimesTable(timesTable)
            document.write("</p>")
        }
    </script>
</body>
</html>
```

保存为 debug_timestable2.htm。注意这个 HTML 文件中没有错误。

下面列出了单步调试代码的详细步骤：

(1) 在页面正文的 for 循环中的第 26 行上设置一个断点，重载页面。

(2) 单击 Step Into 图标，代码就会移动到下一个语句上执行。现在 for 循环中的第一个语句 document.write("<p>")等待执行。

(3) 再次单击 Step Into 图标，就会执行 writeTimesTable()函数的第一次调用。

(4) 为了查看函数内部的执行情况，再次单击 Step Into 按钮，单步执行该函数。屏幕应如图 4-9 所示。

(5) 单击几次 Step Into 按钮，查看函数的执行流。实际上，单步执行代码比较麻烦。所以假定该函数没有错误，我们希望一次运行完该函数的剩余代码。

(6) 使用 Step Out 执行函数的剩余代码。函数全部执行完毕后，就会返回调用该函数的代码行，如图 4-10 所示。

图　4-9

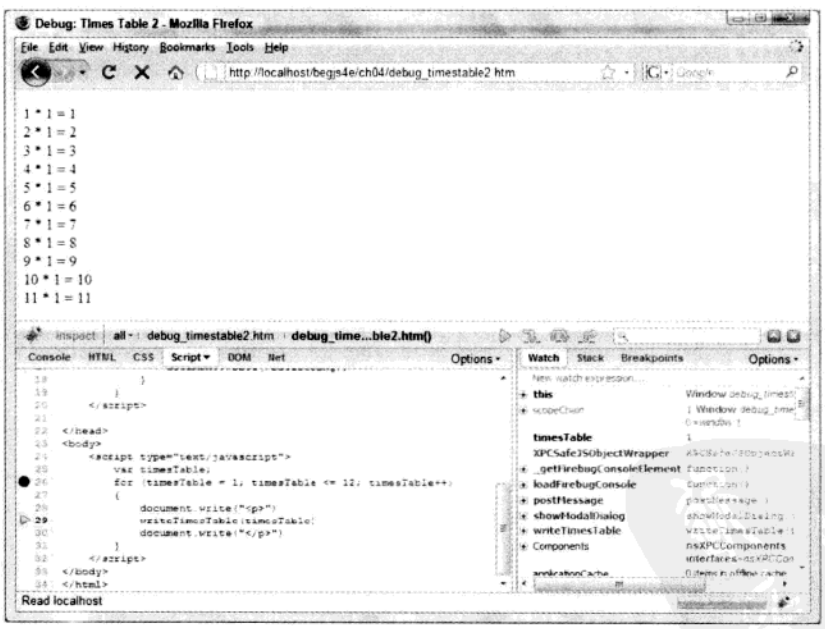

图　4-10

(7) 单击两次 Step Into 图标，执行 document.write()(它不可见，因为它是一个结束标记)。

(8) 再单击四次 Step Into 图标，继续执行 for 循环的条件判断和变量递增部分，在调用 writeTimesTable()函数的代码行处停止。

(9) 前面已经看到了这个过程，所以我们希望跳过它，进入下一行。此时需要使用 Step Over。单击 Step Over 图标(或者按下 F10 键)，执行函数，而不单步执行其中的语句。之后会回到 document.write("</p>")代码行。

如果完成了调试，就可以单击工具栏上的 Continue 图标(或者按下 F8 键)，执行剩余的代码，而无需单步执行代码。结果应是在浏览器页面上显示从 1*1=1 到 11*12=132 的乘法表。

4. 控制台

单步调试代码，检查其执行流时，真正有效的是计算条件以及随时改变代码的能力。这些工作可以用控制台完成。

执行下面的步骤：

(1) 单击源代码面板中的红圆圈，删除以前设置的断点，再在第 17 行设置一个新断点。

```
document.write(writeString);
```

(2) 看看 writeString 变量当前包含的值是什么。重载页面，当调试器在断点处停止时，单击 Console 选项卡，再单击">>>"字段，输入要检查的变量名，本例是 writeString，按下回车键。这会在命令窗口中把该变量包含的值显示在命令之下，如图 4-11 所示。

图 4-11

(3) 如果要修改变量，可在命令窗口中编写一行 JavaScript 代码，并按下回车键。尝试输入下面的代码：

```
writeString = "Changed on the Fly<br />"
```

(4) 单击 Script 选项卡，再单击红圆圈，删除断点，然后单击 Continue 图标。其结果是在本来应显示乘法表中 1*1 的地方，变成了刚才插入的文本。

注意：

这种修改不会改变实际 HTML 源文件，只改变当前加载到浏览器中的页面。

控制台还可以计算条件语句。再次在第 26 行创建断点，重载页面。在断点处停止执行代码，用 Step Into 单步执行到 for 循环的条件语句。

在命令窗口中输入下面的代码，并按下回车键：

```
timesTable <= 12
```

因为这是第一次运行循环，如图 4-12 所示，所以 timesTable 等于 1，于是条件 timesTable <= 12 是 true。

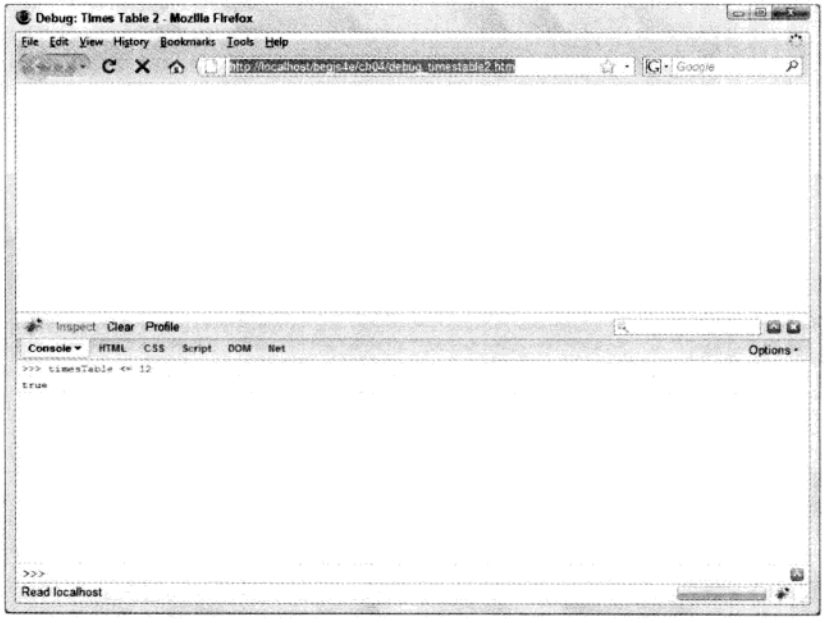

图　4-12

还可以使用控制台访问浏览器对象模型(Browser Object Model，详见第 6 章)的属性。例如，如果在命令窗口中输入 window.location.href，按下回车键，就会得到网页的 URL。

控制台并非仅允许输入单行 JavaScript 代码。单击屏幕右边的向上箭头，就可以输入要执行的多行代码。

5. 调用堆栈窗口

单步执行代码时，调用堆栈窗口会记录被调用的函数的运行列表，直到代码中当前执行的地方为止。

下面创建一个示例网页，来清晰地演示调用堆栈。

(1) 输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Debugging: Callstack</title>
    <script type="text/javascript">
        function firstCall()
        {
            secondCall();
        }
        function secondCall()
        {
            thirdCall();
        }
            function thirdCall()
        {
            //
        }
        function button1_onclick()
        {
            debugger
            firstCall();
        }
    </script>
</head>
<body>
    <input type="button" value="Button" name="button1"
        onclick="return button1_onclick()" />
</body>
</html>
```

(2) 把页面保存为 debug_callstack.htm，加载到 Firefox 中。结果是一个仅包含一个按钮的空白网页。

(3) 单击按钮，调试器就会打开 button1_ onclick()函数中的 debugger 语句，该语句连接到按钮的 onclick 事件处理程序上。

(4) 单击右面板上的 Call Stack 选项卡，调试器现在应如图 4-13 所示。

Firebug 会把每个函数调用都添加到调用堆栈的顶部。可以看出，第一个函数调用实际上是连接到按钮的 onclick 事件处理程序的代码。接着添加到调用堆栈中的是 onclick 事件处理程序调用的函数，即 button1_onclick()函数，它显示在调用堆栈的最上面。

(5) 如果希望查看每个函数第一次在哪里调用，就只需单击调用堆栈窗口中的函数名。单击 onclick，就会显示调用代码(即连接到<input/>元素的 onclick 属性的代码)。现在单击顶行中的 button1_onclick，就会返回当前的执行点。

图　4-13

(6) 单击 Step Into 两次。第一次单击 Step Into，会执行调用 firstCall()函数的代码，第二次单击 Step Into，会进入 firstCall()函数内部。该函数马上就添加到调用堆栈中，如图 4-14 所示。

图　4-14

(7) 再次单击 Step Into，进入第二个函数 secondCall()，这会把 secondCall()函数调用添加到调用堆栈中。下一次单击 Step Into 会进入第三个函数 thirdCall()，这个函数名也会添加到调用堆栈顶部。

(8) 再次单击 Step Into，会退出 thirdCall()函数，该函数名会从调用堆栈的顶部删除。再次单击 Step Into，会退出 secondCall()函数，该函数名也从调用堆栈的顶部删除了。每单击一次 Step Into，就会从一个函数退出，并从调用堆栈中删除相应的函数名，直到执行完全部代码，再次返回浏览器为止。

这个演示页面非常简单，但对于复杂页面，尤其是多框架页面，调用堆栈非常有助于跟踪当前位置、曾到达的位置以及到达指定位置的路径。

如前所述，用于其他浏览器的其他大多数开发工具都是基于的 Firebug，下面介绍 IE8 的内置工具。

4.3.2 在 Internet Explorer 中调试

在 IE 8 之前，开发人员必须为任意类型的脚本调试下载和安装 Microsoft 脚本调试器。而 Microsoft 现在在 IE8 中内置了一个调试器，但默认状态下将其关闭。要启用它，可执行如下步骤：

(1) 单击 Tools | Internet Options。

(2) 单击 Advanced 选项卡，取消 Browsing 部分中 Disable script debugging (Internet Explorer)复选框的选择，如图 4-15 所示。

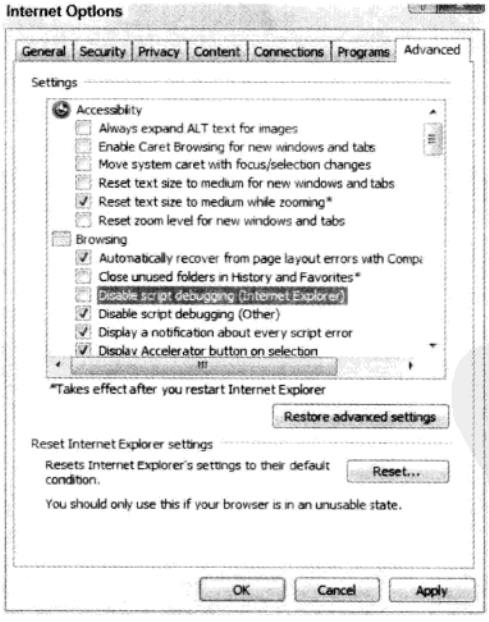

图 4-15

(3) 单击 OK 保存设置，退出 Internet Options 对话框。

启用了调试器后，可通过两种方式访问它：

- 单击 Tools | Developer Tools，可以打开调试器。Developer Tools 包含各种有用的工具(例如 Firebug，它不仅是一个 JavaScript 调试器)。为了便于访问，可以修改命令栏，以包含 Developer Tools 按钮。一旦显示出 Developer Tools 窗口，就单击 Script 选项卡，如图 4-16 所示。

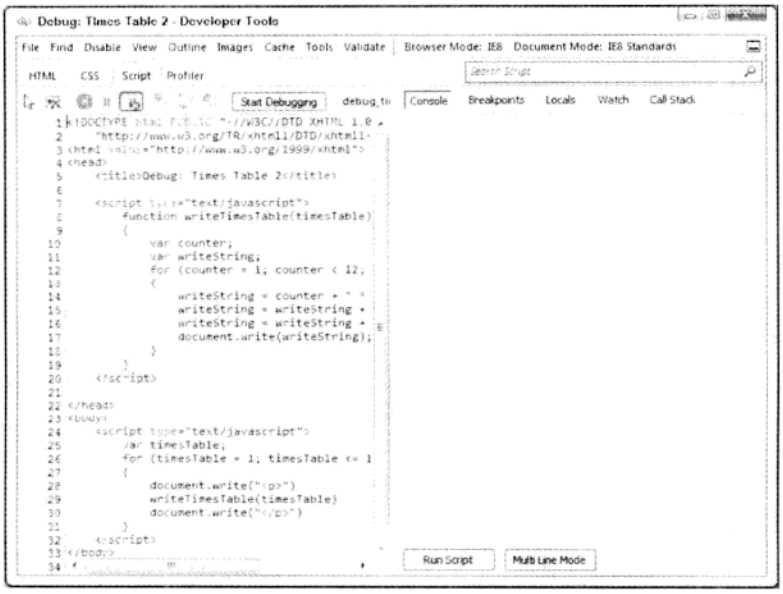

图　4-16

- 这可能是开始调试的最简单方法：导航到需要的页面，如果发生了错误，就会显示一个对话框，询问是否要进行调试，如图 4-17 所示。

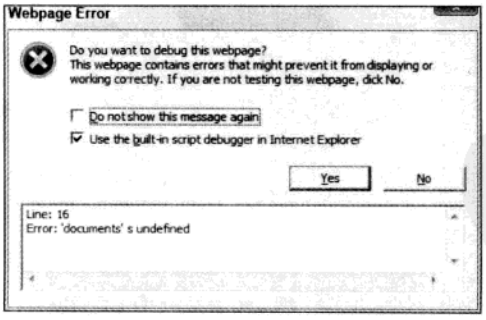

图　4-17

单击 Yes 按钮，Developer Tools 窗口就会显示出来，其中已选中了 Script 选项卡。调试器停在出错的代码行上，并用黄色突出显示该行，但在图 4-18 的黑白屏幕图上，这不太明显。在 IE8 中加载 debug_timestable.htm 的原始版本(有一行代码有错误)，观察这个过程。单击 Yes 开始调试。

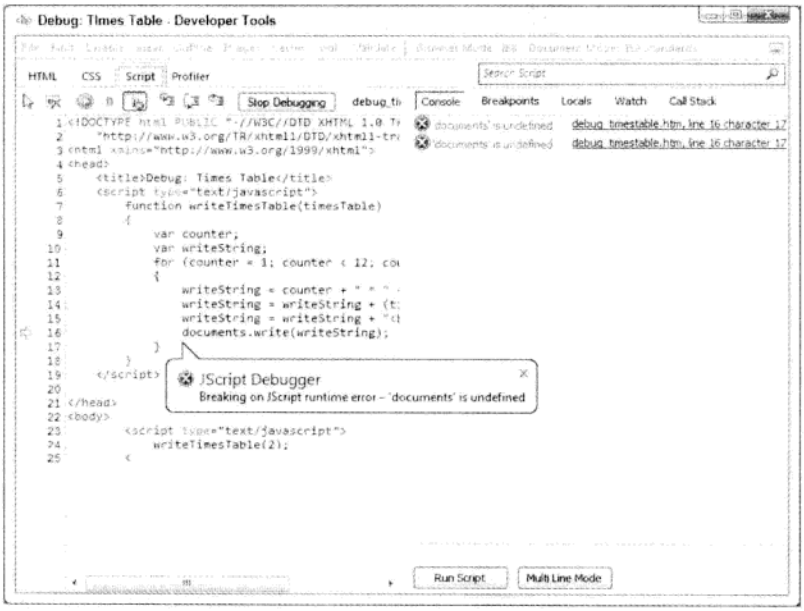

图 4-18

IE8 调试器的一个主要区别是，如果想调试，就必须进入调试模式。这似乎很明显，但 Firebug 没有调试模式。

从图 4-18 可以看出，调试器由两个面板组成。左面板显示存在错误的文件的源代码。如果多个文件包含 JavaScript，就可以使用 Stop Debugging 按钮旁边的下拉菜单选择这些文件。

右面板包含 5 个选项卡：

- Console：包括上面板和下面板。可在下面板中输入和运行 JavaScript 代码，它有单行模式和用于大代码块的多行模式。上面板记录了输入的命令及其结果。
- Breakpoints：列出为当前页面中的代码创建的所有断点。
- Locals：列出变量及其在断点的作用域中的值。
- Watches：列出要在断点处观察的变量及其值。
- Call Stack：显示调用堆栈。

IE8 的 Developer Tools 的另一个区别是增加了 Locals 选项卡。Microsoft 把 Firebug 的 Watch 选项卡分解为 Locals 和 Watches 选项卡。

左面板上的源代码是只读的，所以要修改源代码，就必须在文本编辑器中编辑文件。下面用文本编辑器把第 16 行中错误的 documents 改为 document。保存并尝试重载网页。注意无法重载网页，因为调试器当前正在运行，但因有一个错误而停止。为了重载网页，必须单击 Stop Debugging 按钮。调试器停止后，就可以重载网页了。

改正了错误，重载页面后，就可以在网页中看到乘法表。

1. 设置断点

在 IE8 中创建断点与在 Firebug 中一样简捷明了：在希望调试器中断的代码行上，单击该代码行的页边。创建断点后，该代码行的行号旁边就会出现一个红色的圆圈。

创建断点时，单击 Breakpoints 选项卡，会发现断点列表多了一项。该列表的每一项都包含一个用于启用/禁用断点的复选框、源文件的文件名以及断点所在的行号。图 4-19 显示，在 debug_timestable2.htm 的第 17 行有一个断点。

图　4-19

IE8 的 JavaScript 调试器也支持 debugger 关键字，但只有处于调试模式下，IE8 调试器才能在该行上暂停。否则，就会显示旧的选择屏幕。如果在代码中使用 debugger 关键字，并看到如图 4-20 所示的对话框，就需要启用调试模式。

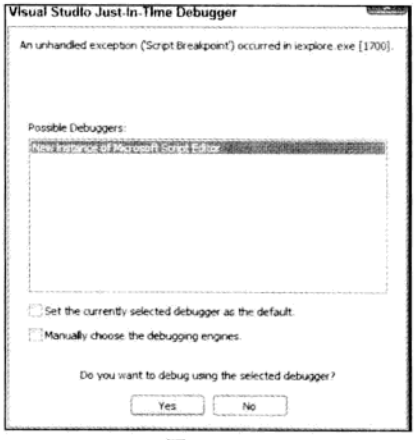

图　4-20

2. 添加要观察的变量

Watch 选项卡的唯一作用是列出要观察的变量，并显示其值和类型。在 IE8 中添加一个要观察的变量与 Firebug 稍有不同：只有在调试模式下，在某个断点处暂停，才能添加要观察的变量。

如果处于调试模式下，并在某个断点处暂停，只需单击“Click to Add...”，输入要观察的变量，按下回车键即可，如图 4-21 所示。

图 4-21

3. 单步调试代码

在调试窗口的顶部，Debugging 按钮的左边有 6 个控制代码执行的按钮，如图 4-22 所示。

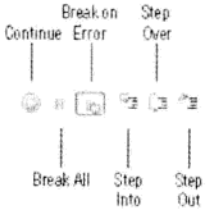

图 4-22

第一个 Continue 选项(快捷键是 F5)允许继续执行代码，直到遇到下一个断点为止，或者执行完所有的代码。第二个选项 Break All 确保调试器在执行下一个语句之前中断。Break on Error 告诉调试器只要发生了错误，调试器就中断。Step Into (F11)、Step Over (F10)和 Step Out (Shift+11)的行为方式与 Firebug 中的对应选项相同。

IE8 的调试器用黄色突出显示当前行，并在其页边添加一个黄色箭头。

与 Firefox 和 Firebug 不同的是，IE8 的调试器单步执行代码并不更新网页。会执行 JavaScript，但只有执行完所有代码后才能看到执行结果。

4. 控制台

与 Firebug 的控制器不同，IE8 的 Developer Tools 控制台与其他 JavaScript 工具放在一起，通过 Console 选项卡来访问，但区别也就仅此而已。

控制台会记录 JavaScript 错误，并允许在调试器暂停的代码行上下文中执行代码。图 4-23 显示了在 IE8 中重建的 Firebug 小节的 Changed on the Fly 示例。

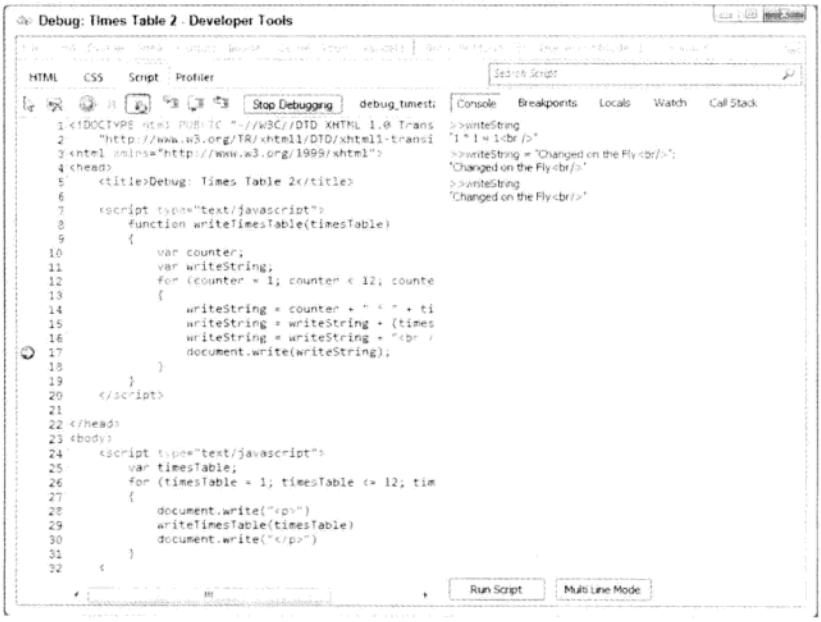

图　4-23

控制台默认仅接受单行 JavaScript 代码，单击 Multi Line Mode 按钮可以接受多行代码。

4.3.3　在 Safari 中调试

Safari 的调试工具类似于 IE 调试工具。Safari 的显示引擎称为 Webkit，编写和维护 Webkit 的开发人员建立了一个单独的工具，其代码名称是 Drosera，它包含的工具类似于 Firebug 和 IE8 的 Developer Tools。Drosera 需要单独下载，并关联到一个特定的 Safari/Webkit 窗口上。

Safari 3 包含一个工具 Web Inspector，但它不具备 JavaScript 调试功能。从 Safari 4 开始，Web Inspector 就有一个内置的 JavaScript 调试器，本节就介绍这个调试器。

注意：

Chrome 也使用 Webkit，但在编写本书时，只有测试版本 3 包含脚本调试器。

与 IE8 一样，默认也会禁用 Web Inspector。要启用它，可执行如下步骤：

(1) 单击 Settings 菜单按钮，选择 Preferences 选项，如图 4-24 所示。

图 4-24

(2) 在 Preferences 窗口中，单击 Advanced 选项卡，选择 Show Develop Menu in Menu Bar 选项，如图 4-25 所示。关闭 Preferences 窗口。

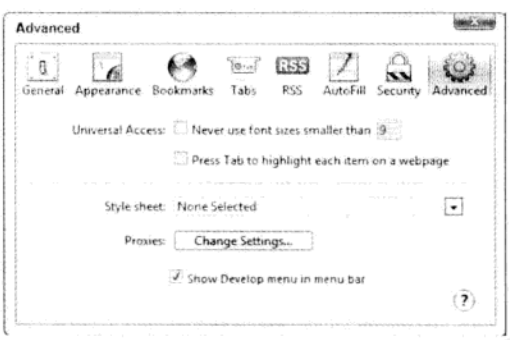

图 4-25

(3) 单击 Settings 菜单按钮，选择 Show Menu Bar 选项，这会在窗口顶部显示传统的菜单。

(4) 要打开调试器，可以选择菜单栏中的 Develop | Start Debugging JavaScript。

窗口打开后，可以看到一些肯定不是自己的代码，这就对了——可以修改这些代码。首先，看看该窗口，认识一下各个部分。图 4-26 显示了第一次打开 debug_timestable2.htm 文件时的 JavaScript 调试器。所显示的代码与自己计算机上的代码不同。

与前面介绍的工具不同，Safari 的 Web Inspector 不使用选项卡划分各个特性，而把所有功能都放在窗口上，以便访问它们。Call Stack 和 Scope Variables 不是选项卡，而是调试时可以同时看到的各个部分。单击 Console 按钮，会在窗口的底部添加一个面板，如图 4-27 所示。

文件选择器
Web Inspector — http://localhost/begjs4e/ch04/debug_timestable2.htm
Elements
Resources
Scripts
Profiles
Databases
Search Scripts
(program)
Call Stack
Scope Variables
显示/隐藏控制台

图　4-26

Web Inspector — http://localhost/begjs4e/ch04/debug_timestable2.htm
(program)
Call Stack
Not Paused
Scope Variables
Not Paused

图　4-27

所有的功能都在这里，以供查看和访问，所以在 Safari 中加载 debug_timestable2.htm，返回 Web Inspector 窗口。单击文件选择下拉菜单，选择 HTML 文件，以显示该文件的源代码。与前面的工具一样，源代码是只读的，但你可以设置断点。

1. 设置断点

在 Web Inspector 中创建断点的过程与其他工具相同：在希望调试器中断的代码行上单击页边，Web Inspector 中的断点用一个蓝色标记表示，如图 4-28 所示。在第 17 行创建一个断点。

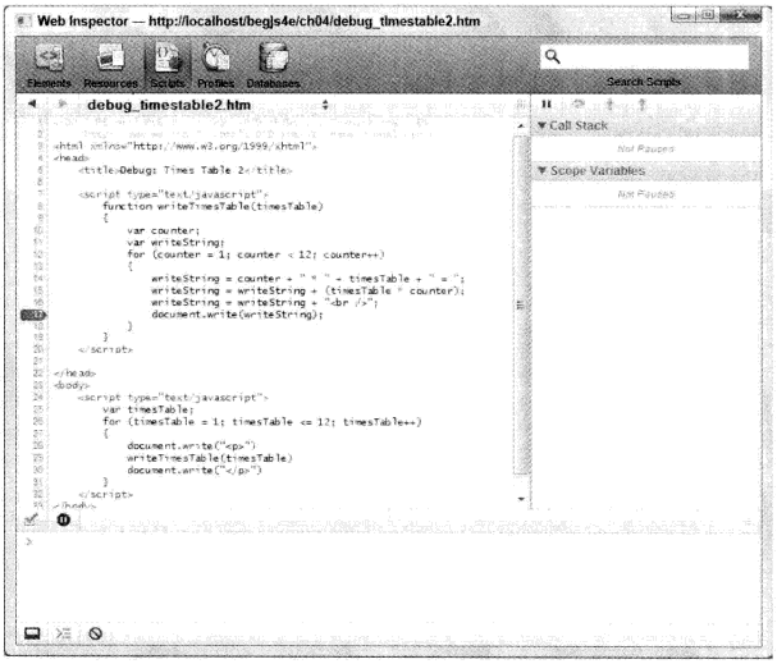

图 4-28

与 Firebug 和 IE 的 Developer Tools 工具不同，Web Inspector 不会在一个单独区域中列出已设置的所有断点，所以如果使用 Safari 和 Web Inspector 作为浏览器和调试器，就需要自己记住在哪里设置了断点。

重载页面，调试器就可以在断点处中断，下面介绍各种调试功能。

注意：

Web Inspector 支持 debugger 关键字。

2. 没有观察窗口

Web Inspector 不允许添加自己设置的要观察的变量，但 Scope Variables 部分会显示出作用域中的所有变量。图 4-29 说明了如何把变量分成局部变量和全局变量。

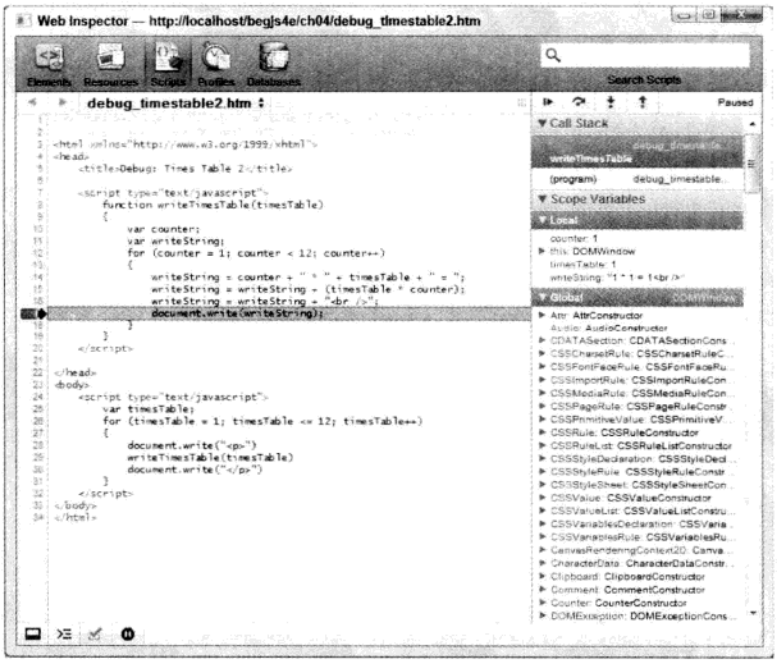

图　4-29

3. 单步调试代码

单步执行代码的按钮在右面板的顶部、搜索框的下面，如图 4-30 所示。

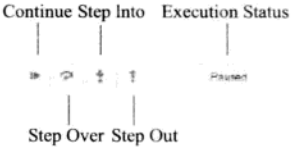

图　4-30

这些按钮的功能与 Firebug 和 IE 相同，但它们的顺序略有区别。第一个按钮继续执行代码，直到遇到下一个断点或者执行完所有代码为止。第二个按钮是 Step Over，第三个按钮是 Step Into，第四个按钮是 Step Out。

与 Firefox 和 Firebug 一样，Safari 和 Web Inspector 在单步调试代码时更新页面，所以你可以看到执行每一行代码的结果。

4. 控制台

控制器的作用与前面的工具相同，只要输入变量，并按下回车键，就可以查看该变量的值。也可以在当前代码行的上下文中执行代码。试试 Firebug 小节中的 Changed on the Fly 示例，看看其执行效果。

与前面的工具不同，Web Inspector 控制台不允许输入多行代码。

Web Inspector 的 UI 井然有序，没有选项卡(一些人觉得选项卡太多会显得凌乱)。下面看看 Opera 的 Dragonfly 中的选项卡。

4.3.4 使用 Opera 的开发工具 Dragonfly

Opera 的 Dragonfly 是基于浏览器的开发工具中的后起之秀。在编写本书时，它仍处于测试阶段，但会包含在 Opera 9.5 版本中。

可以通过两种方式打开 Dragonfly：

- 使用 Tools 菜单：Tools | Advanced | Developer Tools。
- 使用 Debug 菜单：在 Opera 中打开如下 URL 即可安装 Debug 菜单：http://dragonfly.opera.com/app/debugmenu/DebugMenu.ini

图 4-31 显示在 Opera 中加载 debug_timestable2.htm 时打开的 Dragonfly。

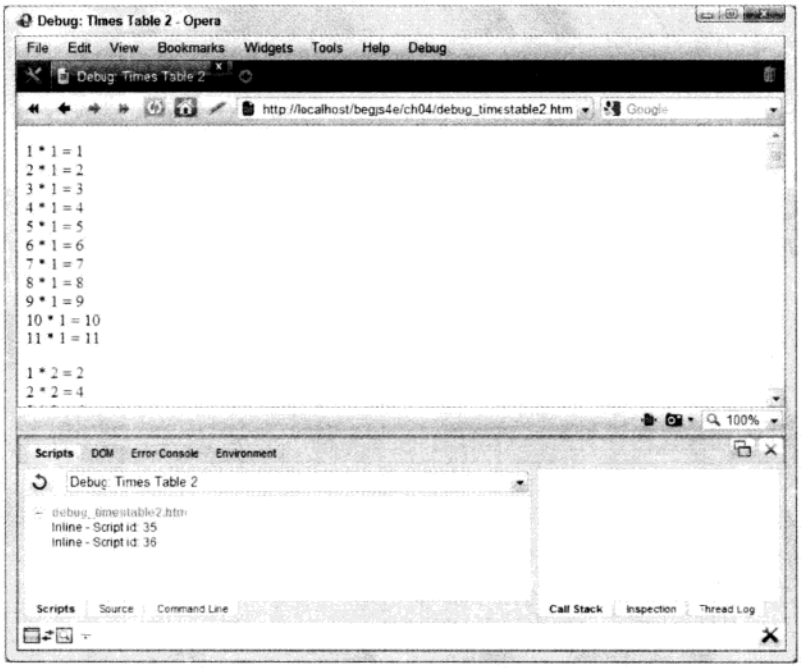

图 4-31

Dragonfly 采用流行的双面板布局。左面板的顶部包含几个选项卡，用于控制 Dragonfly 提供的几个不同工具。调试工具位于 Scripts 选项卡的下方，打开 Dragonfly 时，Scripts 是默认显示的选项卡。

在左面板的底部还有一些选项卡：Scripts、Source 和 Command Line。Source 和 Command Line 选项卡提供的功能与其他调试器相同：

- Scripts：显示当前加载到浏览器中的 HTML 文件，及其包含或引用的 JavaScript 代码。图 4-31 显示了两个嵌入式脚本，这表示它们未被包含在外部文件中。
- Source：显示源代码。与其他工具不同，它仅显示从 Script 选项卡中选择的代码。

进入 Script 选项卡，单击第一个嵌入式脚本，这会打开 Source 选项卡，显示 HTML 文件头部<script />元素的内容，如图 4-32 所示。

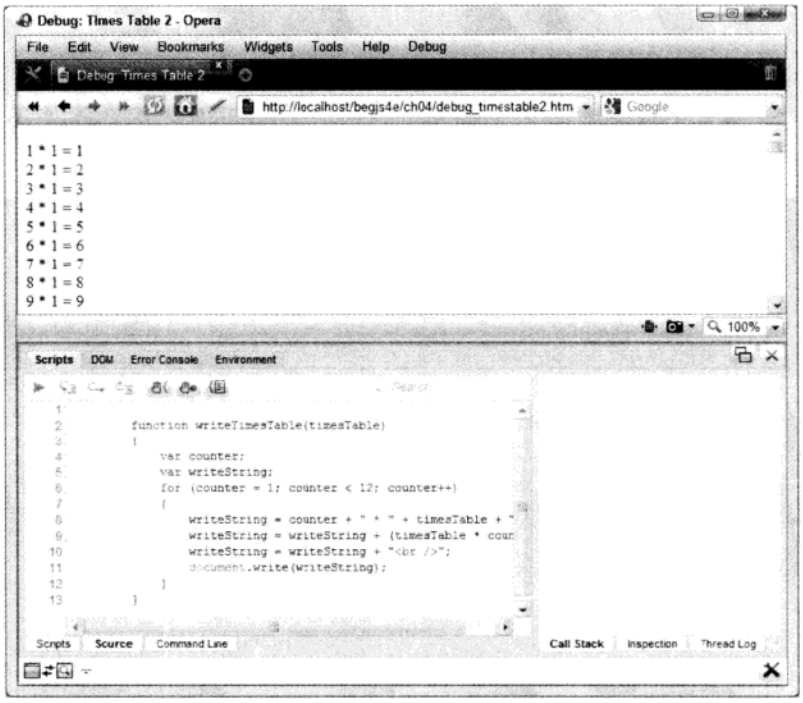

图　4-32

返回 Scripts 选项卡(窗口底部)，会看到被单击的嵌入式脚本放在一对方括号([])中。这表示该脚本是当前在 Source 选项卡中显示的脚本。单击第二个嵌入式脚本，会显示 HTML 文档体中<script />元素的内容。

- Command Line：这是 Dragonfly 的控制台，与 Firebug、IE8 的 Developer Tools 和 Safari 的 Web Inspector 的控制台一样，这个控制台也允许在当前暂停行的上下文中检查变量的值，执行 JavaScript 命令。

右面板包含 Call Stack、Inspection 和 Thread Log 选项卡：

- Call Stack：显示调用堆栈。
- Inspection：显示变量及其在当前暂停行的作用域中的值
- Thread Log：这是一个高级调试工具，提供了 JavaScript 代码的处理和执行信息。本节不介绍 Thread Log 选项卡。

1. 设置断点

在 Dragonfly 中设置断点与前面讨论的其他工具一样直接。在 Scripts 选项卡中单击一个嵌入式脚本，把它加载到 Source 面板中，再单击行号的左边。在该行号上会出现一个小黑点，表示这行代码设置了一个断点。图 4-33 显示在第一个嵌入式脚本的第 11 行设置了一个断点。

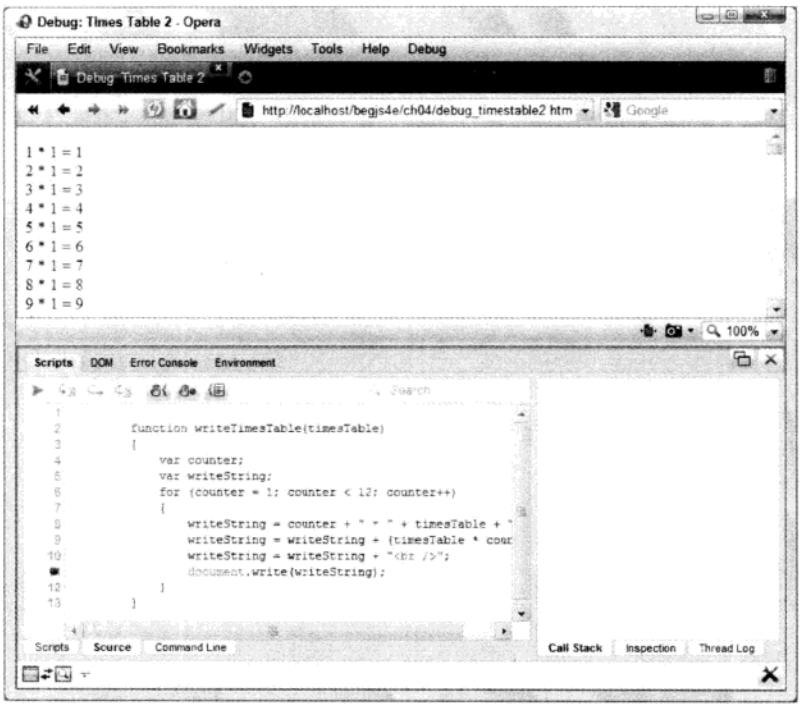

图 4-33

2. 单步调试代码

代码的单步调试由源代码上方的一行图标控制，如图 4-34 所示。

这些图标的功能与 Firebug、IE 的 Developer Tools 和 Web Inspector 相同，它们依次是：

- Continue (F8)
- Step Into (F11)
- Step Over (F10)
- Step Out (Shift + F11).

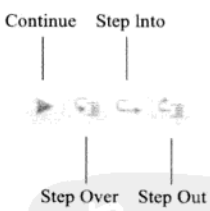

图 4-34

与 Firefox 和 Firebug 以及 Safari 和 Web Inspector 相同，Opera 和 Dragonfly 会在单步调试代码时更新页面，以便你查看每行代码的执行结果。

4.4 小结

本章讨论了代码中一个并不令人激动的话题：错误。在理想的世界中，我们可以使代码完美无瑕，永远正确，但在现实世界中，任何代码都难免出错。

- 本章首先介绍了一些较常见的错误，不仅 JavaScript 的初学者容易犯这些错误，即使是拥有丰富经验的专家也难免出错。
- 某些错误不一定是代码中的错误，而是会导致代码运行失败的异常(例如，当用户位于防火墙之后时，可能导致 Java 小程序运行失败)。try...catch 语句是用于处理异常的最佳途径，可以使用 throw 语句结合 catch 语句来处理诸如用户输入错误所导致的异常。最后，如果无论异常发生与否都需要执行一段代码时，可以使用 finally 子句。
- 本章介绍了用于 Firefox 的 Firebug，IE8 的 Developer Tools，用于 Safari 的 Web Inspector，以及 Opera 的 Dragonfly。使用这些工具可以从代码一开始运行就分析代码，一步步地查看代码的执行流，检查变量和条件。尽管这些调试器的界面不尽相同，但它们的原理和功能集是相同的。

4.5　习题

习题的参考答案在附录 A 中可以找到。

习题 1：

本章示例 debug_timestable2.htm 页面中包含一个故意设置的错误。对于每一个乘法表，它仅仅计算从 1 到 11 的乘法。

使用脚本调试器找出原因，并修正该错误。

习题 2：

下面的代码包含了多个常见错误，请找出这些错误。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 4, Question 2</title>
</head>
<body>
<script type="text/javascript">
function checkForm(theForm)
{
    var formValid = true;
    var elementCount = 0;
    while(elementCount =<= theForm.length)
    {
          if (theForm.elements[elementcount].type == "text")
          {
             if (theForm.elements[elementCount].value() = "")
                alert("Please complete all form elements")
             theForm.elements[elementCount].focus;
```

```
                formValid = false;
                break;
            }
        }
        return formValid;
    }
    </script>
    <form name="form1" onsubmit="return checkForm(document.form1) "action="">
        <input type="text" id="text1" name="text1" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox2" name="checkbox2" />
        <br />
        CheckBox 1<input type="checkbox" id="checkbox1" name="checkbox1" />
        <br />
        <input type="text" id="text2" name="text2" />
        <p>
            <input type="submit" value="Submit" id="submit1" name="submit1" />
        </p>
    </form>
    </body>
    </html>
```

第 5 章

JavaScript——基于对象的语言

本章介绍 JavaScript 的一个核心概念——对象。什么是对象？为什么对象很有用？

首先要说明的是，本书一直在使用对象，例如数组就是一种对象。JavaScript 是一种基于对象的语言，因此我们执行的大多数操作都涉及到操作对象。只要充分利用对象，JavaScript 的操作范围就会显著扩大。

本章首先介绍对象的含义及其重要性，接着讨论在 JavaScript 中可以使用哪些类型的对象，如何创建和使用它们，以及如何通过使用对象简化许多编程任务。最后详细探讨 JavaScript 提供的一些最有用对象，以及如何在实际中使用它们。

不仅 JavaScript 语言包含许多对象(也称为内部 JavaScript 对象)，浏览器本身也建模为一个对象集合，供我们使用。下一章将介绍这些对象。

5.1 基于对象的程序设计

“基于对象的编程”的另一种说法是“用对象编程”。但用于编程的这些对象是什么？它们存储在什么地方？为什么要使用它们编程？本节就用一般编程术语和 JavaScript 中的专用术语回答这些问题。

5.1.1 对象的含义

为了开始对象的介绍，考虑一下对象在计算机之外的“真实世界”中意味着什么。世界是由事物(即对象)组成的，例如桌子、椅子和轿车等。下面以轿车为例，看看对象究竟是什么。

如何定义轿车？可以说它是一辆四轮驱动的蓝色轿车，还可以指定它的车速。此时，指定的是对象的属性。例如，轿车有一个颜色属性，在本例中其值是蓝色。

如何使用轿车？可以转动点火钥匙、踩油门、按喇叭和换档(即选择 1、2、3、4 档，换为手动档，换为自动档等)。此时在使用对象的方法。

方法有点像函数。有时可能需要给方法使用一些信息，或者给方法传送一个参数，才能使方法工作起来。例如，在使用换档方法时，需要指定换到哪一档。一些方法可能会把

一些信息传送回来。例如，量油计方法会显示轿车的剩余油量。

有时使用一个或多个方法会改变对象的一个或多个属性。例如，使用加速器方法会改变轿车的车速属性。有些属性不能改变，例如汽车的外形属性(除非以 100 英里/小时的车速撞向一面砖墙)。

所以，轿车用它的方法和属性集合来定义。在基于对象的编程中，则用对象给真实的事物建模，而对象用其方法和属性来定义。

5.1.2 JavaScript 中的对象

读者现在应对对象的含义——拥有方法和属性的事物——有了基本的了解。那么，在 JavaScript 中如何使用这个概念？

前面的章节主要处理的是基本数据类型(即使用实际的数据)。此类数据并不复杂，处理起来相当容易。但并非所有信息都像基本数据类型那样简单。下面举例说明。

假定编写一个 Web 应用程序，来显示公共汽车或火车的时间表信息。一旦用户确定了某个旅程，程序就应告诉他该旅程所花费的时间。为此，需要从到达时间中减去出发时间。

但这并不像初看上去那么简单。例如，假定出发时间是 14:53 (即 2:53 p.m.)，到达时间是 15:10(即 3:10 p.m.)。如果让 JavaScript 计算表达式 15.10–15.53，结果就是 0.57，即 57 分钟。但实际的时间差是 17 分钟。对时间使用一般的数学运算符是无效的！

要计算出这个时间差，需要做什么？首先需要把每个时间中的小时数和分钟数分开，接着，为了得到这两个时间的分钟之差，需要检查到达时间的分钟数是否大于出发时间的分钟数。如果是，就可以简单地从到达时间的分钟数中减去出发时间的分钟数。否则，就需要给到达时间的分钟数加上 60，再从到达时间的小时数中减去 1。之后从到达时间的分钟数中减去出发时间的分钟数。接着需要从到达时间的小时数中减去到达时间的小时数，最后把得到的分钟数和小时数合在一起。

只要这两个时间在同一天，上述过程就是有效的，但对于 23:45 和 04:32 这两个时间，上述过程是无效的。

这种计算时间差的方式显然有问题，而且还很复杂。可以采用更简单的方法处理像时间和日期这样较复杂的数据吗？

此时就可以使用对象。可以把出发时间和到达时间定义为 Date 对象，这样它们就拥有各种属性和方法，在需要操作或计算时间时，就可以使用这些属性和方法。例如，使用 getTime()方法可以获得从 1970 年 1 月 1 日 00:00:00 到 Date 对象中的时间的毫秒数。一旦得到了到达时间和出发时间相对于 1970 年 1 月 00:00:00 的毫秒数，就可以对它们执行减法操作，把结果存储在另一个 Date 对象中。为了获得这个时间的小时数和分钟数，只需使用 Date 对象的 getHours()和 getMinutes()方法。本章稍后会介绍更多这方面的示例。

Date 对象并不是 JavaScript 提供的唯一对象类型，另一个对象类型是第 2 章介绍的 Array 对象，但为了简单起见，当时没有说明它是一个对象。数组是同时存储多项数据的一种方式。

Array 对象的一个属性是 length，它指定了数组包含了多少项数据或多少个元素。第 3 章曾使用这个属性确定数组需要循环多少次。

Array 对象还包含许多方法，其中一个方法是 sort()，它可用于把数组中的元素按字母顺序排序。

现在我们明白为什么对象在 JavaScript 中非常有用了。前面提到了 Date 和 Array 对象，而 JavaScript 还提供了许多其他类型的对象，这样你就可以利用代码完成更多任务。这些对象包括本章后面详细讨论的 Math 和 String。

5.1.3　使用 JavaScript 对象

明白了 JavaScript 为什么提供对象之后，就需要了解 JavaScript 有哪些对象，以及如何使用它们。

每个 JavaScript 对象都有一组相关属性和方法，可用于操作某种数据。例如，Array 对象包含可以操作数组的方法，以及可以检索数组信息的属性。在大多数情况下，要使用这些方法和属性，需要把数据定义为这些对象中的一个。换言之，需要创建对象。

本节将介绍如何创建对象，以及在创建好对象后如何使用它的属性和方法。

1. 创建对象

前面已经有一个创建 Array 对象的示例。要创建 Array 对象，可使用如下 JavaScript 语句：

```
var myArray = new Array();
```

这个语句有哪些部分组成？

我们很熟悉该语句的第一部分，即使用 var 关键字定义一个变量 myArray。这个变量使用等号赋值运算符(=)初始化为该语句的右半部分。

该语句的右半部分有两个组成部分。首先是运算符 new，它告诉 JavaScript 我们要创建一个新对象，接着是 Array()，它是 Array 对象的构造函数，它告诉 JavaScript 我们要创建的对象类型。大多数对象都有这样的构造函数。例如，Date 对象的构造函数是 Date()。本书中的唯一例外是 Math 对象，详见本章后面的内容。

如第 2 章所述，可以给构造函数 Array()传送参数，以给对象添加数据。例如，下面的代码创建了一个 Array 对象，该对象有 3 个元素，分别包含数据"Paul"、"Jeremy"和"Nick"：

```
var myArray = new Array("Paul", "Jeremy", "Nick");
```

下面是另一个示例，它创建了一个 Date 对象。最简单的方式是：

```
var myDate = new Date();
```

这行代码创建了一个 Date 对象，其中包含创建该对象的日期和时间。而下面的代码创建了一个包含 2010 年 1 月 1 日的 Date 对象：

```
var myDate = new Date("1 Jan 2010");
```

对象数据在变量中的存储方式与基本数据类型(例如文本和数字)不同。基本数据类型是 JavaScript 中最基本的数据。对于基本数据类型，变量存储数据的实际值。例如：

```
var myNumber = 23;
```

这行代码表示，myNumber 变量存储了数据 23。但赋予对象的变量不存储实际数据，而存储指向保存数据的内存地址的引用。这并不意味着可以获得该内存地址——这只是 JavaScript 在后台管理对象的方式，只有 JavaScript 才了解详情。只需记住，变量引用一个对象时，其含义是变量引用一个内存地址，如下例所示：

```
var myArrayRef = new Array(0, 1, 2);
var mySecondArrayRef = myArrayRef;
myArrayRef[0] = 100;
alert(mySecondArrayRef[0]);
```

上述代码首先把变量 myArrayRef 设置为引用新的数组对象，再把 mySecondArrayRef 设置为这个引用——例如，现在 mySecondArrayRef 设置为引用同一个数组对象。所以把数组的第一个元素设置为 100，如下所示：

```
myArrayRef [0] = 100;
```

再显示在 mySecondArrayRef 中引用的第一个数组元素的内容：

```
alert(mySecondArrayRef[0]);
```

它变成了 100！我们现在知道，这是因为这两个变量引用了同一个数组对象。对于对象来说，变量是对对象的引用，而不是存储在变量中的对象本身。在执行赋值操作时，并没有复制数组对象，而只复制了引用。与下面的代码进行比较：

```
var myVariable = "ABC";
var mySecondVariable = myVariable;
myVariable = "DEF";
alert(mySecondVariable);
```

这个示例处理的是字符串，它与数字一样是基本数据类型。这次把实际值存储在变量中，所以可以编写如下代码：

```
var mySecondVariable = myVariable;
```

mySecondVariable 得到了 myVariable 中数据的副本。所以最后的 alert 仍会把 mySecond Variable 显示为包含 ABC。

作为本节的小结，可以采用如下基本语法创建 JavaScript 对象：

```
var myVariable = new ObjectName(optional parameters);
```

2. 使用对象的属性

访问包含在对象属性中的值是非常简单的。只需写出包含(或引用)对象的变量名，后跟一个句点和对象属性名即可。

例如，如果定义了一个包含在 myArray 变量中的 Array 对象，就可以使用如下代码访问它的 length 属性：

```
myArray.length
```

现在，使用这个属性可以执行什么操作？可以像操作其他任何数据那样使用该属性，并把它存储在变量中：

```
var myVariable = myArray.length;
```

或者把它显示给用户：

```
alert(myArray.length);
```

在某些情况下，甚至可以修改该属性的值，如下所示：

```
myArray.length = 12;
```

但与变量不同，一些属性是只读的——只能从中获取信息，而不能修改属性中包含的信息。

3. 调用对象的方法

方法与函数非常类似，因为它们都可以用于执行有用的任务，例如获取某个日期的小时数，或者生成随机数。与函数一样，一些方法有返回值，例如 Date 对象的 getHours()方法，一些方法仅执行任务，不返回数据，例如 Array 对象的 sort()方法。

使用对象的方法与使用其属性非常类似，即把对象的变量名放在最前面，之后是一个句点和方法名。例如，要给 myArray 变量中的数组元素排序，可使用以下代码：

```
myArray.sort();
```

与函数一样，也可以给一些方法传送参数，这需要把参数放在方法名后面的括号中。但无论方法是否带参数，都必须在方法名之后加上括号，这与函数的做法是一样的。一般说来，可以使用函数的地方，就可以使用对象的方法。

5.1.4　基本数据类型与对象

我们现在了解了基本数据(如数字和字符串)与对象数据(例如 Date 和 Array)之间的区别。但如前所述，还有一个 String 对象，它是什么类型？

实际上，String、Number 和 Boolean 对象分别对应基本的字符串、数字和布尔值。例如，要创建一个包含文本"I'm a String object"的 String 对象，可以使用如下代码：

```
var myString = new String("I'm a String object");
```

与 Array 对象一样，String 对象也有 length 属性。它返回 String 对象中的字符数。例如：

```
var lengthOfString = myString.length;
```

上面的语句把值 19 存储在变量 lengthOfString 中(注意，字符串中的空格也是字符)。

但是，如果声明一个基本的字符串 mySecondString，来保存文本"I'm a primitive string"，

如下所示：

```
var mySecondString = "I'm a primitive string";
```

该怎样计算该字符串中的字符数呢？

JavaScript 可以完成这个任务。前面的章节介绍过，JavaScript 可自动将一种数据类型转换为另一种数据类型。例如，如果把字符串和数值加在一起，如下：

```
theResult = "23" + 23;
```

JavaScript 将假定我们要把数值当作字符串，并把两个字符串连接在一起，所以自动把数值转换为文本。变量 theResult 将包含"2323"，即把两个 23 连接起来，而不是把两个 23 加起来得到和 46。

这也适用于对象。如果声明一个基本的字符串，并把它视为一个对象，如访问它的某个方法或属性，JavaScript 就知道这个操作会失败，该操作只适用于对象，例如 String 对象。在这个例子中，JavaScript 把普通文本的字符串转换为一个临时的 String 对象，以进行这个操作，完成该操作后就释放该对象。

因此，对于基本的字符串 mySecondString，可以使用 String 对象的 length 属性来获得字符串中的字符数。例如：

```
var lengthOfSecondString = mySecondString.length;
```

这会把数据 22 存储在变量 lengthOfSecondString 中。

这也适用于数值类型与对应的 Number 对象，以及布尔类型与对应的 Boolean 对象。但是这些对象很少使用，所以本书不再进一步讨论。

5.2 JavaScript 的内置对象类型

前面讨论了对象的含义以及如何创建和使用它们。下面看看 JavaScript 内置的一些非常有用的对象，内置对象就是内置于 JavaScript 语言中的对象。

本节不全面介绍所有 JavaScript 内置对象，只讨论比较常用的对象，即 String 对象、Math 对象、Array 对象和 Date 对象。本书后面将用一整章的篇幅来介绍较复杂的对象，例如 String 对象(第 9 章)和 Date 对象(第 10 章)。

5.2.1 String 对象

与大多数对象一样，在使用 String 对象之前，必须先创建它。要创建 String 对象，可使用如下语句：

```
var string1 = new String("Hello");
var string2 = new String(123);
var string3 = new String(123.456);
```

但如前所述，也可以声明一个基本字符串，把它用作 String 对象，让 JavaScript 在后

台进行对象转换。例如：

```
var string1 = "Hello";
```

只要 JavaScript 能在后台准确地推断出我们要创建什么对象，就可以使用这种方法。如果基本数据类型是字符串，就没有问题，JavaScript 会进行正确的转换。其优点是不需要创建 String 对象，也避免了比较字符串对象的麻烦。在比较字符串对象与基本字符串值时，会比较实际值，但比较两个 String 对象时，比较的是对象引用。

String 对象有大量的方法和属性。本节只介绍一些比较简单的常用方法。第 9 章将介绍一些比较复杂、非常强大、与字符串和正则表达式对象(RegExp)相关的方法。正则表达式是一种按字符模式搜索字符串的强大方法。例如，假如要在字符串"Pauline，Paul，Paula"中查找完整的单词"Paul"，就需要使用正则表达式。但这需要一些技巧，本章不进一步讨论它们——把有趣的东西留在后面体验吧！

对于 String 对象的大部分方法而言，字符串仅是一系列单个字符，与数组类似，每个字符都有一个位置，或索引。第一个位置或索引是 0 而不是 1，这也与数组类似。因此，字符串"Hello World"的各字符位置如表 5-1 所示：

表　5-1

索引	0	1	2	3	4	5	6	7	8	9	10
字符	H	e	L	l	o		W	o	r	l	d

1. length 属性

length 属性返回字符串中的字符数。例如：

```
var myName = new String("Jeremy");
document.write(myName.length);
```

这行代码向页面输出字符串"Jeremy"的长度 6。

2. indexOf()和 lastIndexOf()方法——在一个字符串中查找另一个字符串

indexOf()方法和 lastIndexOf()方法用于查找一个字符串是否包含了另一个字符串。包含在另一个字符串中的字符串通常称为子字符串。仅需要某串信息中的一部分时，就可以使用这两个方法。例如，在“小测试”程序中，用户输入了一个文本答案，就需要检查该字符串是否包含了某些的关键字。

indexOf()和 lastIndexOf()方法都带两个参数：

- 需要查找的字符串
- 开始查找的字符位置(可选)

字符位置从 0 开始。如果不包含第二个参数，就从字符串开头开始搜索。

indexOf()和 lastIndexOf()方法的返回值是查找到的子串在字符串中的位置。它也是基于 0 的，因此，如果在字符串开头找到子串，则返回 0。如果没有找到子串，则返回 - 1。

例如，要在字符串"Hello jeremy. How are you Jeremy"中查找子串"Jeremy"，可以使用

如下代码：

```
<script type="text/javascript">
var myString = "Hello jeremy. How are you Jeremy";
var foundAtPosition;

foundAtPosition = myString.indexOf("Jeremy");
alert(foundAtPosition);
</script>
```

以上代码将弹出一个消息框，其中包含数值 26，这是"Jeremy"的字符位置。为什么是 26 呢？显然，它是字符串中第二个"Jeremy"出现的位置，而不是第一个"jeremy"出现的位置 6。这是由于 JavaScript 是区分大小写的，这一点已经反复强调过。JavaScript 对其语法和比较操作进行十分严格的区分大小写。如果输入的是 IndexOf()，而不是 indexOf()，则 JavaScript 会报告错误。同样，"jeremy"与"Jeremy"是不同的。大小写错误非常普遍，即使是编程专家也很容易犯此类错误，所以编程时最好特别注意。

前面介绍了 indexOf()方法，但 lastIndexOf()方法与之有什么区别呢？indexOf()方法从字符串的开头或第二个参数指定的位置开始向后查找，而 lastIndexOf()方法从字符串的结尾或指定的位置向字符串的开头查找。

> 在本例中，第一次查找使用 indexOf()，找到第一个"Jeremy"(把上一个例子中的"jeremy"改为正确的大小写)。警告框显示这个结果，即字符位置 6。接着使用 lastIndexOf()来查找，它从字符串的结尾开始查找，因此找到的第一个"Jeremy"是字符串中字符位置为 26 的最后一个"Jeremy"。因此，第二个警告框显示结果 26。

```
<script type="text/javascript">
var myString = "Hello Jeremy. How are you Jeremy";
var foundAtPosition;

foundAtPosition = myString.indexOf("Jeremy");
alert(foundAtPosition);

foundAtPosition = myString.lastIndexOf("Jeremy");
alert(foundAtPosition);
</script>
```

试一试 统计字符串中某个子串的出现次数

这个例子演示了如何使用 indexOf()方法的“起始字符位置”参数，以统计 Wrox 在字符串中的出现次数。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title>Chapter 5: Example 1</title>
</head>
<body>
<script type="text/javascript">
var myString = "Welcome to Wrox books. ";
myString = myString + "The Wrox website is www.wrox.com. ";
myString = myString + "Visit the Wrox website today. Thanks for buying Wrox";

var foundAtPosition = 0;
var wroxCount = 0;

while (foundAtPosition != -1)
{
    foundAtPosition = myString.indexOf("Wrox",foundAtPosition);
    if (foundAtPosition != -1)
    {
        wroxCount++;
        foundAtPosition++;
    }
}

document.write("There are " + wroxCount + " occurrences of the word Wrox");

</script>
</body>
</html>
```

将这个示例保存为 ch5_examp1.htm。在浏览器中加载该页面，页面上将显示信息：There are 4 occurrences of the word Wrox。

在脚本块的顶部，在变量 myString 中建立了一个字符串，后面将在这个字符串中查找 Wrox 的出现次数。还定义了两个变量：wroxCount 包含 Wrox 在字符串中出现的次数，foundAtPosition 包含在字符串中找到 Wrox 的当前位置。

接着使用一个 while 循环，只要在字符串中找到 Wrox，即变量 foundAtPosition 不等于 - 1，while 语句就继续循环。在 while 循环内使用如下代码：

```
foundAtPosition = myString.indexOf("Wrox",foundAtPosition);
```

这行代码在字符串 myString 中查找下一个 Wrox 子串。如何确保可以找到下一个 Wrox 子串？变量 foundAtPosition 用来给出查找的起始位置，因为它保存了上一次找到子串 Wrox 的索引位置后面的索引。给变量 foundAtPosition 赋予查找结果，即下一次查找 Wrox 子串的索引位置。

每次找到 Wrox 时(即 foundAtPosition 不是 - 1)，就递增变量 wroxCount，该变量保存了找到 Wrox 子串的次数；递增 foundAtPosition，就可以在字符串的下一个位置处继续查找。

```
if (foundAtPosition != -1)
{
    wroxCount++;
    foundAtPosition++;
}
```

最后用 document.write()方法将变量 wroxCount 的值输出到页面上。

第 3 章讨论了无限循环的危险性。在这个例子中也可能出现死循环：假如删除 foundAtPosition++，则每次都从开头位置开始查找，永远找不到下一个 Wrox 子串。

如果将 indexOf()和 lastIndexOf()方法与下一节介绍的 substr()和 substring()方法结合使用，效果将更好。联合使用这些方法，联合使用能够获取字符串中的子串。

3. substr()和 substring()方法——复制字符串的一个子串

substr()和 substring()方法可用来从字符串中提取一个子串，并赋予另一个变量，或者用在表达式中。这两个方法返回的结果相同，都是子串，但它们需要的参数不同。

substring()方法接受两个参数：子串的开始位置和子串中最后一个字符后面的字符位置。第二个参数是可选的，如果不包含它，则子串包含从开始位置到字符串末尾的所有字符。

例如，如果字符串为"JavaScript"(如表 5-2 所示)，要取出子串"Java"，则可以使用 substring()方法，如下所示：

```
var myString = "JavaScript";
var mySubString = myString.substring(0,4);
alert(mySubString);
```

表 5-2

位置	0	1	2	3	4	5	6	7	8	9
字符	J	a	v	a	S	C	r	i	p	t

与 substring()方法类似，substr()方法也带两个参数。第一个参数是子串中要包含的第一个字符的起始位置。但是，第二个参数是要从长字符串中提取的子串的长度。例如，可以将上面的代码改写为：

```
var myString = "JavaScript";
var mySubString = myString.substr(0,4);
alert(mySubString);
```

与 substring()方法类似，substr()方法的第二个参数是可选的。如果不包含它，则子串包含从开始位置到字符串末尾的所有字符。

注意：

版本 4 以前的浏览器支持 substring()方法，而版本 4 及以后版本的浏览器支持 substr()方法。大多数情况下使用的是 substr()方法。

下面结合使用 substr()与 lastIndexOf()方法。第 5 章将介绍如何获取当前所加载网页的文件路径和名称。但是，无法单独提取文件名。例如，如果文件是 http://mywebsite/temp/myfile.htm，而需要提取 myfile.htm 部分，就可以使用 substr()与 lastIndexOf()方法。

```
var fileName = window.location.href;
```

```
fileName = fileName.substr(fileName.lastIndexOf("/") + 1);
document.write("The file name of this page is " + fileName);
```

上面代码的第一行给变量 fileName 赋予当前的文件路径和文件名，如/mywebsite/temp/myfile.htm。现在不理解这行代码不要紧，第 6 章将介绍它。

我们感兴趣的是第二行。这行代码把 lastIndexOf()方法的返回值作为另一个方法的参数，这是正确的，也很有效。fileName.lastIndexOf("/")的作用是查找最后一个斜杠(/)出现的位置，斜杠(/)是文件名之前的最后一个字符。把斜杠出现的位置加 1，是因为不需要包含斜杠字符。接着把这个新值传递给 substr()方法。这里没有使用第二个参数(即长度)，因为我们不知道长度是多少。因此 substr()返回从该参数位置开始到字符串末尾的全部字符。

注意：

这个例子提取了本地计算机上的页面名称，原因是没有访问 Web 服务器上的页面。但是，请不要误以为仅使用 JavaScript 就可以从 Web 页面上访问本地硬盘上的文件。为了防止用户受到恶意攻击，JavaScript 访问用户系统的权限(如访问文件)十分有限。详见本书后面的内容。

4. toLowerCase()和 toUpperCase()方法——转换大小写

如果要改变字符串的大小写(例如比较字符串之前，先统一字符串的大小写形式)，就可以使用 toLowerCase()和 toUpperCase()方法。不难猜出这两个方法的作用。它们都返回 String 对象中的字符串，但根据所调用的方法不同，分别返回字符串的大写或小写形式，而非字母字符保持不变。

在下面的例子中，通过改变两个字符串的大小写，就可以在比较它们时忽略大小写：

```
var myString = "I Don't Care About Case"

if (myString.toLowerCase() == "i don't care about case")
{
   alert("Who cares about case?");
}
```

即使 toLowerCase()和 toUpperCase()不带任何参数，也必须在调用方法时，在方法名之后加上括号()。

5. charAt()和 charCodeAt()方法——从字符串中选取一个字符

如果要找出字符串中某个字符的信息，可以使用 charAt()和 charCodeAt()方法。这两个方法也可用于检查用户输入的有效性。第 7 章介绍 HTML 表单时，将介绍这方面的内容。

charAt()方法接受一个参数：所选字符在字符串中的索引位置。charAt()方法就返回该字符。字符串中字符的位置从 0 开始，因此，第一个字符的索引是 0，第二个字符的索引是 1，依此类推。

例如，要查找字符串的最后一个字符，可以使用如下代码：

```
var myString = prompt("Enter some text","Hello World!");
var theLastChar = myString.charAt(myString.length - 1);
```

```
document.write("The last character is " + theLastChar);
```

上面的第一行代码提示用户输入一个字符串，并把这个字符串保存在变量 myString 中。该字符串默认为"Hello World!"。

下一行代码使用 charAt()方法获取字符串的最后一个字符。该字符的索引位置是(myString.length - 1)。为什么？以字符串"Hello World!"为例，这个字符串的长度为 12，但最后一个字符的索引为 11，因为索引从 0 开始，所以需要把字符串的长度减去 1，得到最后一个字符的索引。

最后一行代码将字符串的最后一个字符输出在页面上。

charCodeAt()方法的用法与 charAt()方法类似，但它不返回字符本身，而是返回该字符在 Unicode 字符集中的 10 进制编码。计算机只能理解数字——对于计算机来说，字符串仅是数值数据。当需要文本(而不是数字)时，计算机将根据它对每个数字的内部理解进行转换，并显示对应的字符。

例如，要确定字符串中第一个字符的编码，可使用如下代码：

```
var myString = prompt("Enter some text","Hello World!");
var theFirstCharCode = myString.charCodeAt(0);
document.write("The first character code is " + theFirstCharCode);
```

上面的代码将在用户提供的字符串中，获取索引位置为 0 的字符的编码，并输出在页面上。

字符是按顺序编码的，例如，字母 A 的编码是 65，B 的编码是 66，依此类推。小写字母从 97 开始，即 a 的编码是 97，b 的编码是 98，依此类推。数字的编码从 48 开始(0 的编码是 48)到 57(9 的编码是 57)。这些信息可用于各种目的，例如下例就利用了它们。

试一试　检查字符的大小写

下例检查指定字符串的首字符是大写字母、小写字母、数字还是其他字符：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 2</title>
    <script type="text/javascript">
    function checkCharType(charToCheck)
    {
        var returnValue = "O";
        var charCode = charToCheck.charCodeAt(0);

        if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0))
        {
           returnValue = "U";
        }
        else if (charCode >= "a".charCodeAt(0) && charCode <= "z".charCodeAt(0))
        {
```

```
            returnValue = "L";
        }
        else if (charCode >= "0".charCodeAt(0) && charCode <= "9".charCodeAt(0))
        {
            returnValue = "N";
        }

        return returnValue;
    }
    </script>
</head>

<body>
<script type="text/javascript">

var myString = prompt("Enter some text","Hello World!");
switch (checkCharType(myString))
{
    case "U":
        document.write("First character was upper case");
        break;
    case "L":
        document.write("First character was lower case");
        break;
    case "N":
        document.write("First character was a number");
        break;
    default:
        document.write("First character was not a character or a number");
}
</script>

</body>
</html>
```

输入上面的代码，并保存为 ch5_examp2.htm。

在浏览器中加载该页面时，将提示用户输入字符串。之后，程序将在页面上显示一条消息，说明所输入的第一个字符的类型——即该字符是大写字母、小写字母、数字或其他字符，如标点符号。

本示例首先定义了一个在页面正文中使用的函数 checkCharType()。该函数首先声明了变量 returnValue，并将它初始化为字符"O"，表示除大写字母、小写字母、数字之外的其他字符。

```
function checkCharType(charToCheck)
{
    var returnValue = "O";
```

这个变量用作函数最后的返回值，它表示字符的类型：U 代表大写字母，L 代表小写字母，N 代表数字，O 代表其他字符。

函数的下一行代码使用 charCodeAt()方法获取 charToCheck 中存储的字符串首字符的

编码。charToCheck 是该函数唯一的参数。字符编码保存在变量 charCode 中。

```
var charCode = charToCheck.charCodeAt(0);
```

接下来的代码是一系列 if 语句，用于检查首字符的编码在哪个取值范围内。如果该编码在 A 和 Z 的编码之间，则它是一个大写字母，就给变量 returnValue 赋予 U。如果该编码在 a 和 z 的编码之间，则它是一个小写字母，就给变量 returnValue 赋予 L。如果该编码在 0 和 9 的编码之间，则它是一个数字，就给变量 returnValue 赋予 N。如果它不属于以上任一范围，则变量 returnValue 保持其初始值 O(代表其他字符)。

```
if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0))
{
   returnValue = "U";
}
else if (charCode >= "a".charCodeAt(0) && charCode <= "z".charCodeAt(0))
{
   returnValue = "L";
}
else if (charCode >= "0".charCodeAt(0) && charCode <= "9".charCodeAt(0))
{
   returnValue = "N";
}
```

这段代码看起来有点怪异，下面看看 JavaScript 是如何处理的。对于如下语句：

```
"A".charCodeAt(0)
```

这行代码似乎要在字符串字面值上使用 String 对象的一个方法，但字符串字面值与基本字符串一样，仅是字符，而不是一个对象。JavaScript 知道这行代码的含义，并把字符串字面值"A"转换为一个包含"A"的临时 String 对象。之后，JavaScript 在这个后台创建的 String 对象上执行 charCodeAt()方法。执行完毕后，就释放 String 对象。这基本上是下面代码的一种简写形式：

```
var myChar = new String("A");
myChar.charCodeAt(0);
```

无论采用哪种形式，都返回首字符(本字符串中仅有一个字符)的编码。例如"A".charCodeAt(0)返回数字 65。

在函数的最后，将变量 returnValue 返回到调用函数的代码。

```
    return returnValue;
}
```

为什么要使用变量 returnValue，而不直接返回其值呢？例如，可以编写如下代码：

```
if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0))
{
   return "U";
}
```

```
else if (charCode >= "a".charCodeAt(0) && charCode <= "z".charCodeAt(0))
{
   return "L";
}
else if (charCode >= "0".charCodeAt(0) && charCode <= "9".charCodeAt(0))
{
   return "N";
}
return "O";
```

这段代码可以正常运行，但为什么不采用这种方式呢？这种方式的缺点是，难以跟踪函数的执行流。在这样的小型函数中，这不是什么大问题，但在较大的函数中情况就比较棘手。在最初的代码中，总能准确地了解函数在何处停止，即在唯一的 return 语句后停止。而上述函数版本会在到达任何一个 return 语句时结束，因此函数有 4 个可能的停止位置。

在页面正文中，使用一些测试语句来检查函数的工作情况。首先用初始化为"Hello World!"的变量 myString 或用户在提示框中输入的内容作为测试字符串。

```
var myString = prompt("Enter some text","Hello World!");
```

接下来，switch 语句在比较表达式中使用了前面定义的函数 checkCharType()。根据该函数的返回值，执行某个 case 语句，并将字符的类型显示给用户。

```
switch (checkCharType(myString))
{
    case "U":
        document.write("First character was upper case");
        break;
    case "L":
        document.write("First character was lower case");
        break;
    case "N":
        document.write("First character was a number");
        break;
    default:
        document.write("First character was not a character or a number");
}
```

这就结束了这个例子，但在继续之前要注意，这仅是一个演示 charCodeAt()函数用法的例子。实际上，可以使用更简洁的代码：

```
if (char >= "A" && char <= "Z")
```

代替例子中使用的代码：

```
if (charCode >= "A".charCodeAt(0) && charCode <= "Z".charCodeAt(0))
```

3. fromCharCode()方法——将字符编码转换为字符串

fromCharCode()方法与 charCodeAt()方法正好相反，给它传送一系列用逗号分隔的、表

示字符编码的数字，该方法就会把它们转换为一个字符串。

但是，fromCharCode()方法有点不寻常，它是一个静态方法——无需创建 String 对象，就可以使用它，它总是可用的。

例如，下面的代码将字符串"ABC"保存在变量 myString 中：

```
var myString;
myString = String.fromCharCode(65,66,67);
```

fromCharCode()方法非常适于与变量一起使用。例如，要创建一个包含字母表中所有大写字母的字符串，可以使用如下代码：

```
var myString = "";
var charCode;

for (charCode = 65; charCode <= 90; charCode++)
{
    myString = myString + String.fromCharCode(charCode);
}

document.write(myString);
```

上述代码使用 for 循环依次选择从 A~Z 的每个字符，并连接起来，保存在变量 myString 中。注意，这只是一个例子，在实际编码时，可以采用以下更有效且更节省内存的代码：

```
var myString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

5.2.2 Array 对象

第 2 章介绍了如何创建和使用数组。本章前面提到，数组本质上是一种对象。

除了存储数据外，Array 对象还提供了大量有用的方法和属性，用于操纵数组中的数据，获取诸如数组长度等的信息。

这里不详细讲解 Array 对象的每个属性和方法，而只介绍其中一些常用方法和属性。

注意：

本书在创建数组时，使用了构造函数语法：var myArray = new Array()。也可以使用更简洁的语法创建数组：var myArray = []。即使用开闭方括号代替 new Array()。为简洁起见，本书继续使用构造函数。

1. length 属性——获得数组中元素的个数

length 属性提供了数组中元素的个数。第 3 章的“小测试”程序已经使用了这个 length 属性。有时数组的长度是已知的，但是有时我们会给数组添加新元素，但无法方便地跟踪已添加元素的数量。

length 属性可以用来确定最后一个数组元素的索引，例如下面的代码：

```
var names = new Array();

names[0] = "Paul";
```

```
names[1] = "Jeremy";
names[11] = "Nick";

document.write("The last name is " + names[names.length - 1]);
```

注意：

上面的代码在索引位置为 0、1、11 的元素中插入了数据。数组索引从 0 开始，因此，最后一个元素的索引是 length-1，即 11，而不是 length 属性的值 12。

当某个 JavaScript 方法返回它自己建立的数组时，length 属性也是非常有用的。例如，第 9 章介绍高级字符串处理时，会提到 String 对象有一个 split()方法，它把文本分为多个段，并将结果放在一个 Array 对象中。因为 JavaScript 创建了这个数组，如果没有 length 属性，就无法确定数组中最后一个元素的索引。

2. concat()方法——连接数组

如果要把两个单独数组连接在一起，组成一个大数组，就可以使用 Array 对象的 concat()方法。concat()方法返回一个新数组，它由两个数组结合而成：首先是第一个数组的所有元素，接着是第二个数组的所有元素。为此，只需在第一个数组上使用该方法，并把第二个数组名传送为其参数。

例如，假定有两个数组 names 和 ages，它们的元素如表 5-3 和表 5-4 所示。

表　5-3

names 数 组			
元 素 索 引	0	1	2
值	Paul	Jeremy	Nick

表　5-4

Ages 数 组			
元 素 索 引	0	1	2
值	31	30	31

如果使用 names.concat(ages)方法连接它们，将得到如表 5-5 所示的数组：

表　5-5

元 素 索 引	0	1	2	3	4	5
值	Paul	Jeremy	Nick	31	30	31

下面是该示例的代码：

```
var names = new Array("Paul","Jeremy","Nick");
var ages = new Array(31,30,31);

var concatArray = names.concat(ages);
```

也可以使用 names = names.concat(ages)，把两个数组连接为一个数组，但把新数组赋予已有的第一个数组。

如果使用的是 ages.concat(names)，结果会有什么区别呢？结果如表 5-6 所示，首先是 ages 数组的元素，然后是 names 数组的元素。

表 5-6

元素索引	0	1	2	3	4	5
值	31	30	31	Paul	Jeremy	Nick

3. slice()方法——复杂数组的部分

slice()方法可用于复制数组的一部分。使用 slice()方法可以提取出数组的一部分，并赋予一个新变量。slice()方法有两个参数：

- 欲复制的第一个元素的索引
- 表示所复制的部分末尾的元素索引(可选)

与使用 substring()进行字符串复制类似，起始元素包含在复制的部分中，而结束元素不在其中。另外，如果不包含第二个参数，则复制从起始索引往后的所有元素。

假如有一个如表 5-7 所示的 names 数组：

表 5-7

索引	0	1	2	3	4
值	Paul	Sarah	Jeremy	Adam	Bob

如果要创建一个新数组，它包含 names 数组中索引为 1(Sarah)和 2(Jeremy)的元素，则可以指定起始索引为 1，而结束索引为 3，代码如下所示：

```
var names = new Array("Paul","Sarah","Jeremy","Adam","Bob");
var slicedArray = names.slice(1,3);
```

JavaScript 复制数组时，会把新元素复制到新数组中，它们在新数组中的索引是 0 和 1，而不是原来的索引 1 和 2。

在执行 slice()方法后，slicedArray 如表 5-8 所示：

表 5-8

索引	0	1
值	Sarah	Jeremy

第一个数组 names 不受 slice()方法的影响。

5. join()方法——将数组转换为字符串

join()方法将数组中的所有元素连接起来，并返回为一个字符串。它还允许指定在连接数组元素时插入其间的任意字符。该方法仅有一个参数，即在元素之间插入的字符串。

下面通过实例予以说明。假如把每周的购物清单保存在一个数组中，如表 5-9 所示：

表　5-9

索　　引	0	1	2	3	4
值	Eggs	Milk	Potatoes	Cereal	Banana

现在，使用 document.write()将购物清单输出在页面上，每一项单独占一行，因此每个元素之间需要使用
标记。首先声明数组：

```
var myShopping = new Array("Eggs","Milk","Potatoes","Cereal","Banana");
```

接着使用 join()方法将数组转换为一个字符串：

```
var myShoppingList = myShopping.join("<br />");
```

现在变量 myShoppingList 包含以下文本：

```
"Eggs<br />Milk<br />Potatoes<br />Cereal<br />Banana"
```

使用 document.write()输出到页面上：

```
document.write(myShoppingList);
```

购物清单显示在网页上，每项各占一行，如图 5-1 所示。

图　5-1

5. sort()方法——对数组排序

如果数组包含类似的数据，例如姓名列表或年龄列表，就可以按字母或数值顺序排列它们。这时使用 sort()方法非常方便。下面的代码定义了数组 names，并使用 names.sort()按字母表的升序顺序来排列它。最后输出排序的数组。

```
var names = new Array("Paul","Sarah","Jeremy","Adam","Bob");
var elementIndex;

names.sort();
```

```
document.write("Now the names again in order" + "<br />");

for (elementIndex = 0; elementIndex < names.length; elementIndex++)
{
    document.write(names[elementIndex] + "<br />");
}
```

注意，排序是区分大小写的。因此，Paul 排在 paul 的前面。JavaScript 保存的是字符对应的 Unicode 编码，所以排序基于 Unicode 编码，而不是实际字母。Unicode 编码的顺序与字母表的顺序是相同的。但是，小写字母具有另一个编码系列，且位于大写字母的编码系列之后。因此，会将数组元素 Adam，adam，Zoë，zoë 的顺序排列为 Adam，Zoë，adam，zoë。

注意在 for 语句的循环条件中使用了 Array 对象的 length 属性，而没有插入数组的长度(5)，如下所示：

```
for (elementIndex = 0; elementIndex < 5; elementIndex++)
```

为什么要这么做？毕竟这个数组有 5 个元素是已知的，但是如果增加了 2 个元素，改变了数组中元素的个数，会发生什么情况？

```
var names = new Array("Paul","Sarah","Jeremy","Adam","Bob","Karen","Steve");
```

如果插入了 5，而没有使用 names.length，则循环代码就不会按期望的那样执行，它不会显示后两个元素，除非把 for 循环的条件部分改为 7。而使用 length 属性会很简单，因为如果添加了数组元素，就不必修改其他地方的代码。

前面按升序排列数组元素，但是如果需要按降序排列，该怎么办呢？可以使用 reverse()方法。

6. reverse()方法——反转数组元素的顺序

最后要介绍 Array 对象的方法是 reverse()，它用于反转数组中元素的顺序，即将数组中靠后的元素放在前面。还是以购物清单为例，如表 5-10 所示：

表 5-10

索　引	0	1	2	3	4
值	Eggs	Milk	Potatoes	Cereal	Banana

如果使用 reverse()方法

```
var myShopping = new Array("Eggs","Milk","Potatoes","Cereal","Banana");
myShopping.reverse();
```

则数组中的元素顺序如表 5-11 所示。

表 5-11

索　引	0	1	2	3	4
值	Banana	Cereal	Potatoes	Milk	Eggs

为了证明这一点，使用前面的 join()方法将数组输出到页面上：

```
var myShoppingList = myShopping.join("<br>")
document.write(myShoppingList);
```

试一试　**对数组元素进行排序**

当与 sort()方法结合使用时，reverse()方法可以把数组按字母或数字反序排列。例如下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 3</title>
</head>
<body>
<script type="text/javascript">

var myShopping = new Array("Eggs","Milk","Potatoes","Cereal","Banana");

var ord = prompt("Enter 1 for alphabetical order, and -1 for reverse order", 1);

if (ord == 1)
{
    myShopping.sort();
    document.write(myShopping.join("<br />"));
}
else if (ord == -1)
{
    myShopping.sort();
    myShopping.reverse();
    document.write(myShopping.join("<br />"));
}
else
{
    document.write("That is not a valid input");
}
</script>
</body>
</html>
```

将这个示例保存为 ch5_examp3.htm。在浏览器中加载该文件，页面将提示用户指定是按升序还是降序排列。如果用户输入 1，则数组按升序排列，如果输入 - 1，则数组按降序排列。如果输入了其他值，则页面将提示输入无效。

脚本块顶部定义了包含购物清单的数组 myShopping。接着定义了变量 ord，用于保存用户在提示框中输入的值：

```
var ord = prompt("Enter 1 for alphabetical order, and -1 for reverse order", 1);
```

这个值将用在其后 if 语句的条件中。第一个 if 语句将判断 ord 的值是否为 1——即用户是否想按字母顺序排列数组。如果是，则执行下面的代码：

```
myShopping.sort();
document.write(myShopping.join("<br>"));
```

数组排序后，使用 join()方法输出在页面的多个行上。接着，在 else if 语句中，检查 ord 的值是否为 - 1，即用户是否想按字母反序排列数组，如果是，就执行下面的代码：

```
myShopping.sort();
myShopping.reverse();
document.write(myShopping.join("<br>"));
```

排好序数组后，反转其顺序，接着用 join()方法将数组显示给用户。

最后，如果 ord 的值既不是 1，也不是 - 1，则告知用户输入是无效的。

```
document.write("That is not a valid input");
```

5.2.3 Array 的新方法

2005 年，Mozilla 更新了 Firefox 中的 JavaScript 引擎。为此，他们给 Array 对象添加了 7 个新方法。可将这 7 个方法分为两类：位置方法和迭代方法。

下面的 7 个方法都不能用于 Internet Explorer，但可以用于 Firefox、Safari、Opera 和 Chrome。

1. indexOf()和 lastIndexOf()方法——查找数组元素

顾名思义，这两个方法的功能类似于 String 对象的 indexOf()和 lastIndexOf()方法——它们返回某元素在数组中的第一个出现位置和最后一个出现位置的索引。考虑下面的代码：

```
var colors = new Array("red", "blue", "green", "blue");
alert(colors.indexOf("red"));
alert(colors.lastIndexOf("blue"));
```

第一行代码创建了一个数组 colors，该数组有 4 个元素(其中有两个是 blue)。第二行代码给用户返回 0，因为 red 是数组的第一个元素。

String 对象的 lastIndexOf()方法会向后搜索字符串，返回第一个匹配字符的索引。Array 对象的 lastIndexOf()方法也以类似的方式执行，所以第三行代码给用户显示 3。

与 String 对象的同名方法类似，如果在数组中没有找到元素，这两个方法就返回-1。

2. 迭代数组，但不使用循环

剩余的 5 个方法都称为迭代方法，因为它们会迭代(或循环)数组。另外，这些方法在迭代数组时，会在每个元素上执行用户定义的函数。这些方法使用的函数必须遵循一个规则：该函数必须接受 3 个参数，如下面的代码所示：

```
function functionName(value, index, array) {
    // do something here
}
```

执行这个函数时，JavaScript 会把 3 个参数传送给函数。第一个是元素的值，第二个是元素的索引，最后一个是数组本身。有了这些参数，就可以执行任何需要的操作或比较。

every()、some()和 filter()方法——测试每个元素

先来分析 every()和 some()方法。它们是测试方法。every()方法测试数组中的所有元素是否通过了函数中的测试。考虑下面的代码：

```
var numbers = new Array(1, 2, 3, 4, 5);

function isLessThan3(value, index, array)
{
    var returnValue = false;

    if (value < 3)
    {
        returnValue = true;
    }

    return returnValue;
}

alert(numbers.every(isLessThan3));
```

第一行代码创建了一个数组 numbers，其元素包含从 1 到 5 的数字。下一行代码定义了 isLessThan3()函数，它接受 3 个必选参数，确定每个元素的值是否小于 3。最后一行输出 every()测试的结果。因为数组中的值并非都小于 3，所以 every()测试的结果是 false。

将其与 some()方法进行比较。与 every()不同，some()方法仅测试数组中的某些元素是否通过了函数中的测试。仍使用 numbers 数组和 isLessThan3()函数，考虑下面的代码行：

```
alert(numbers.some(isLessThan3));
```

结果是 true，因为数组中的一些元素小于 3。很容易把这两个方法区分开。只有数组中的所有元素都通过了函数中的测试，every()方法才返回 true；而只要数组中的一些元素通过了函数中的测试，some()方法就返回 true。

假定要检索出数组中值小于 3 的元素。已知一些元素满足这个条件，但如何找出并检索出这些元素？此时可以使用 filter()方法。

filter()方法对数组中的每个元素执行某函数，如果该函数对某个元素返回 true，就把该元素添加到 filter()方法返回的另一个数组中。分析下面的代码：

```
var numbers = new Array(1, 2, 3, 4, 5);

function isLessThan3(value, index, array)
{
    var returnValue = false;

    if (value < 3)
```

```
        {
            returnValue = true;
        }

        return returnValue;
    }

    if (numbers.some(isLessThan3))
    {
        var result = numbers.filter(isLessThan3);
        alert("These numbers are less than 3: " + result);
    }
```

这段代码定义了前面使用的 numbers 数组和 isLessThan3 函数。新代码确定 numbers 数组中的元素是否包含小于 3 的值，如果是，就调用 filter()方法，在一个新数组中返回这些元素。这段代码的结果如图 5-2 所示。

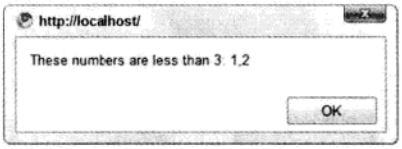

图 5-2

可以看出，结合使用 filter()方法与 some()方法的效果相当不错！

在继续之前，再看看 isLessThan3()函数。这个函数的用法与往常一样，没有对它进行任何修改。但可以简化函数体，以提高效率。下面的代码列出了简化版本：

```
function isLessThan3(value, index, array)
{
    return (value < 3);
}
```

简单了不少。JavaScript 与大多数其他语言一样，也把条件语句返回为 true 或 false，还可以把返回值赋予变量。在这个函数中，条件语句的返回值返回给了调用代码。

为了使叙述更清晰，作者选择使用较长、但易于理解的方法来编码，而不是这个简化版本。

forEach()和 map()方法——操作元素

最后两个方法是 forEach()和 map()。与前面的迭代方法不同，这两个方法不用函数测试数组中的每个元素，而是以某种方式使用数组中的元素执行某类操作。看看下面的代码：

```
var numbers = new Array(1, 2, 3, 4, 5);

for (var i = 0; i < numbers.length; i++ )
{
    var result = numbers[i] * 2;
    alert(result);
}
```

程序员经常看到和使用这类代码。它定义了一个数字数组，并迭代它，以便对每个元素执行某种操作。在这个示例中，给每个元素的值加倍，结果显示在一个警告框中。如果可以不编写循环就完成这个操作，效果真的非常不错。

有了 forEach()方法，就可以做到这一点。只需编写一个函数，为给定的值加倍，把结果输出到警告框中即可，如下所示：

```
var numbers = new Array(1, 2, 3, 4, 5);

function doubleAndAlert(value, index, array)
{
    var result = value * 2;
    alert(result);
}

numbers.forEach(doubleAndAlert);
```

注意，doubleAndAlert()函数不像测试方法那样有返回值，它不能返回任何值，它的唯一作用是对数组中的每个元素执行一个操作。这在某些情况下是有效的，但如果需要操作的结果，这种方法就没有什么用了。此时应使用 map()方法。

map()方法的作用与 forEach()类似，但每个操作的结果会存储在 map()方法返回的另一个数组中。

下面修改前面的示例。doubleAndAlert()函数仍需要给数组的元素值加倍，但现在它需要返回该操作的结果，并按顺序存储在 map()返回的数组中。

```
var numbers = new Array(1, 2, 3, 4, 5);

function doubleAndAlert(value, index, array)
{
    var result = value * 2;
    return result;
}

var doubledNumbers = numbers.map(doubleAndAlert);
alert("The doubled numbers are: " + doubledNumbers);
```

图 5-3 显示了这段代码的执行结果。可以看出，这段代码与 forEach()方法的代码区别很小。doubleAndAlert()函数现在返回元素值与 2 的乘积(而不是把结果输出到警告框中)，再调用 map()方法，给用户显示完整的结果集。

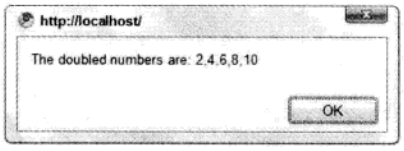

图　5-3

可以看出，需要查找数组中的元素时，或者要对所有元素执行相同操作时，这 7 个方法很方便。

这些方法的唯一缺点是 Internet Explorer 不支持它们，Microsoft 尚未说明是否或何时添加它们，只有一件事可以肯定：如果 Microsoft 添加了对它们的支持，开发人员的工作会更加轻松。

5.2.4 Math 对象

Math 对象提供了大量有效的数学函数和数值操作方法。下面将介绍其中的一些，其余方法的详细描述可以参考 W3C 的站点：www.w3schools.com/ jsref/default.asp。

Math 对象有点与众不同，因为 JavaScript 会自动创建它。在使用 Math 对象之前，不需要将变量声明为 Math 对象，也不需要定义新的 Math 对象，所以 Math 对象使用起来很容易。

Math 对象的属性包含一些有用的数学常量，如 PI 属性(其值是 3.14159)。可以采用通常的方式访问这些属性，即对象名 Math 后加一个句点(.)，然后写出属性名。例如，要计算圆面积，可以使用如下代码：

```
var radius = prompt("Give the radius of the circle", "");
var area = (Math.PI)*radius*radius;
document.write("The area is " + area);
```

Math 对象的方法包含一些难以用标准数学运算符(+、－、*、/)完成的操作，甚至包含无法用数学运算符完成操作。例如 cos()方法返回所传入参数值的余弦。下面将介绍其中的一些方法。

1. abs()方法

abs()方法返回所传入参数的绝对值。实际上，它返回该参数的正值。例如，－1 的绝对值是 1，－4 的绝对值是 4。而 1 的绝对值就是 1，因为 1 本来就是正数。

例如：下面的代码把数字 101 输出在页面上：

```
var myNumber = -101;
document.write(Math.abs(myNumber));
```

2. min()和 max()方法——查找最大值和最小值

假定有两个数字，要确定哪个较大，哪个较小。为此，Math 对象提供了 min()和 max()方法。这两个方法都接受至少两个参数，它们都必须是数字。看看下面的示例代码：

```
var max = Math.max(21,22); // result is 22
var min = Math.min(30.1, 30.2); // result is 30.1
```

min()方法返回值最小的数字，max()返回值最大的数字。传送给这两个方法的数字可以是整数或浮点数。

注意：

min()和 max()方法可以接受多个数字，而不仅仅是两个。

3. 舍入方法

Math 对象提供了几个数字舍入方法，每个方法都有各自的用途。

ceil()方法

ceil()方法总是把数值向上修整到最接近的最小整数，例如，10.01 向上圆整后为 11，－9.99 向上圆整后为－9(这是因为－9 大于－10)。ceil()方法只有一个参数，即要向上修整的数值。

ceil()的用法与第 2 章介绍的 parseInt()函数不同。parseInt()方法只是截断小数点之后的所有数字，把整数部分保留下来。而 ceil()是向上修整数值。

例如，下面的代码在页面上输出两行，第一行包含数值 102，第二行包含数值 101：

```
var myNumber = 101.01;
document.write(Math.ceil(myNumber) + "<br />");
document.write(parseInt(myNumber));
```

floor()方法

与 ceil()方法类似，floor()方法也是舍去小数点后的数字，返回一个整数。所不同的是，floor()总是向下修整。例如，10.01 向下修整后为 10，而－9.99 向下修整后为－10。

4. round()方法

round()方法非常类似于 ceil()和 floor()，但它不总是向上修整或向下修整，而是当小数部分大于等于 0.5 时向上修整，小于等于 0.5 时向下修整。

例如：

```
var myNumber = 45.5;
document.write(Math.round(myNumber) + "<br>");

myNumber = 45.49;
document.write(Math.round(myNumber));
```

这段代码在页面上输出数字 45 和 44。

舍入方法的小结

如前所述，ceil()、floor()和 round()方法都是舍去小数点后的数字，返回一个整数。但返回哪个整数取决于所使用的方法：floor(n)返回最接近的最大整数；ceil(n)返回最接近的最小整数；而 round(n)返回最接近的整数。这几个方法可能容易混淆，所以表 5-12 显示了一列数值，以及将这些数值分别传给 parseInt()函数、ceil()、floor()和 round()时的返回值。

表　5-12

参　　数	parseInt()返回	ceil()返回	floor()返回	round()返回
10.25	10	11	10	10
10.75	10	11	10	11
10.5	10	11	10	11

(续表)

参　　数	parseInt()返回	ceil()返回	floor()返回	round()返回
–10.25	–10	–10	–11	–10
–10.75	–10	–10	–11	–11
–10.5	–10	–10	–11	–10

注意：

parseInt()是一个 JavaScript 内置函数，而不是 Math 对象的方法，与表中其他方法不同。

试一试　使用舍入方法的计算器

如果还不清楚这些舍入方法，下面的例子将有助于理解它们。这个计算器先从用户处获得一个数值，然后将该数值分别传入 parseInt()、ceil()、floor()和 round()，并在页面上显示各个方法的结果。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 4</title>
</head>
<body>
<script type="text/javascript">

var myNumber = prompt("Enter the number to be rounded"," ");

document.write("<h3>The number you entered was " + myNumber + "</h3><br />");
document.write("<p>The rounding results for this number are</p>");
document.write("<table width='150' border='1'>");
document.write("<tr><th>Method</th><th>Result</th></tr>");
document.write("<tr><td>parseInt()</td><td>"+ parseInt(myNumber) +"</td></tr>");
document.write("<tr><td>ceil()</td><td>" + Math.ceil(myNumber) + "</td></tr>");
document.write("<tr><td>floor()</td><td>"+ Math.floor(myNumber) + "</td></tr>");
document.write("<tr><td>round()</td><td>" + Math.round(myNumber) +"</td></tr>");
document.write("</table>")

</script>
</body>
</html>
```

保存为 ch5_examp4.htm，并在 Web 浏览器中加载它。在提示框中输入一个数值，如 12.354，然后单击 OK。这个数值将分别传给 parseInt()、ceil()、floor()和 round()，各方法的舍入结果以表格形式显示在页面上，如图 5-4 所示。

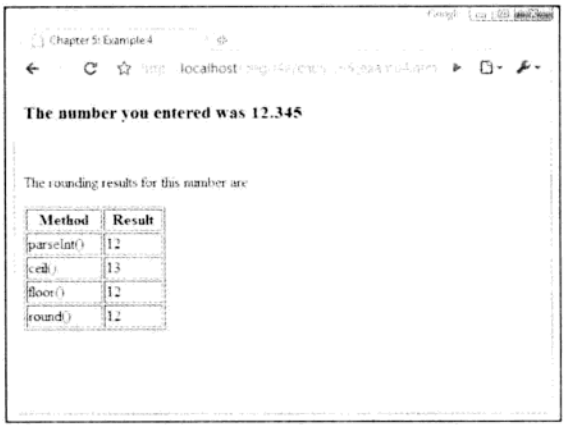

图　5-4

上述代码首先从用户处获取要舍入的数值：

```
var myNumber = prompt("Enter the number to be rounded","");
```

然后输出该数值和一些描述文本：

```
document.write("<h3>The number you entered was " + myNumber + "</h3><br/>");
document.write("<p>The rounding results for this number are</p>");
```

注意，这次包含了一些用于格式化的 HTML 标记——主标题放在<h3>标记中，表格的描述放在段落标记<p>中。

接着创建显示结果的表格：

```
document.write("<table width=150 border=1>");
document.write("<tr><th>Method</th><th>Result</th></tr>");
document.write("<tr><td>parseInt()</td><td>"+ parseInt(myNumber)
                                                    + "</td></tr>");
document.write("<tr><td>ceil()</td><td>" + Math.ceil(myNumber)
                                                    + "</td></tr>");
document.write("<tr><td>floor()</td><td>"+ Math.floor(myNumber)
                                                    + "</td></tr>");
document.write("<tr><td>round()</td><td>" + Math.round(myNumber)
                                                    + "</td></tr>");
document.write("</table>")
```

首先创建表头，然后在单独的一行上显示每个舍入方法的结果。仅使用 JavaScript 很容易动态地创建 Web 页面中的 HTML。其原则与 Web 页面中的 HTML 相同：必须确保标记的语法是有效的，否则会显示错误的信息或者什么都不显示。

每一行的创建方法都相同，只是使用了不同的舍入方法。第一行显示了 parseInt()的结果：

```
document.write("<tr><td>parseInt()</td><td>"+ parseInt(myNumber)
                                                    + "</td></tr>");
```

在向页面输出的字符串中，首先使用<tr>标记来创建表行，然后使用<td>标记创建一个单元格，并将插入要在该表行上显示其结果的方法名，最后以</td>结束这个单元格。在

下一个单元格中放置 parseInt()函数的实际结果。尽管 parseInt()返回一个数值，由于把它与一个字符串连接在一起，所以 JavaScript 在连接之前，自动把 parseInt()返回的数值转换为字符串。这是在后台实现的，我们无需执行任何操作。最后用</td></tr>结束单元格和表行。

6. random()方法

random()方法返回一个 0～1 之间的随机浮点数，包含 0 但不包括 1。它非常适于显示随机的横幅图像或编写 JavaScript 游戏。

下面说明如何模拟掷骰子的游戏。下列代码将在页面上显示 10 个随机数，单击浏览器的刷新按钮，就将显示另一组随机数：

```
<html>
<body>
<script type="text/javascript">
var throwCount;
var diceThrow;
for (throwCount = 0; throwCount < 10; throwCount++)
{
    diceThrow = (Math.floor(Math.random() * 6) + 1);
    document.write(diceThrow + "<br>");
}
</script>
</body>
</html>
```

我们希望 diceThrow 在 1～6 之间。random()方法返回一个 0～1(不包含 1)之间的随机浮点数。把这个数值乘以 6，就得到一个 0～6(不包含 6)的随机数，然后加 1，就得到一个 1～7(不包含 7)的随机数。接着用 floor()方法向下修整为最近的整数，便得到一个 1～6(包含 6)的随机整数。

如果想获得 1～100 的随机数，只须修改代码，使 Math.random()乘以 100，而不是 6。

7. pow()方法

pow()方法计算一个数的指定幂。它带两个参数，第一个参数是底数，第二个参数是指数。例如，要求出 2 的 8 次幂，即计算 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2，可以编写 Math.pow(2,8)——结果为 256。一些数学方法在 Web 编程中并不常用，例如 sin()、cos()和 acos()等，除非编写的是科学应用程序，但 pow()方法非常有用。

试一试　使用 pow()方法

下面的例子使用 pow()方法编写一个函数，用于固定数字中的小数位数——JavaScript 的早期版本没有提供这个函数，而在 JScript 5.5 和 JavaScript 1.5 中增加了它，如本章后面所述。本示例说明，即使 JavaScript 没有提供某函数，也总是可以利用现有的函数来创建该函数。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 5</title>
</head>
<head>
<script type="text/javascript">

function fix(fixNumber, decimalPlaces)
{
    var div = Math.pow(10,decimalPlaces);
    fixNumber = Math.round(fixNumber * div) / div;
    return fixNumber;
}
</script>
</head>
<body>
<script type="text/javascript">

var number1 = prompt("Enter the number with decimal places you want to fix","");
var number2 = prompt("How many decimal places do you want? ", "");

document.write(number1 + " fixed to " + number2 + " decimal places is: ");
document.write(fix(number1,number2));

</script>
</body>
</html>
```

将该页面保存为 ch5_examp5.htm。在浏览器中加载该页面，将会看到两个提示框。在第一个提示框中输入一个要固定小数位数的数字，例如输入 2.2345。在第二个提示框中，输入要固定的小数位数，例如 2。程序把输入的数字变成具有指定小数位数的数，并显示在页面上。如图 5-5 所示，对于本例输入的 2.2345，结果是 2.23。

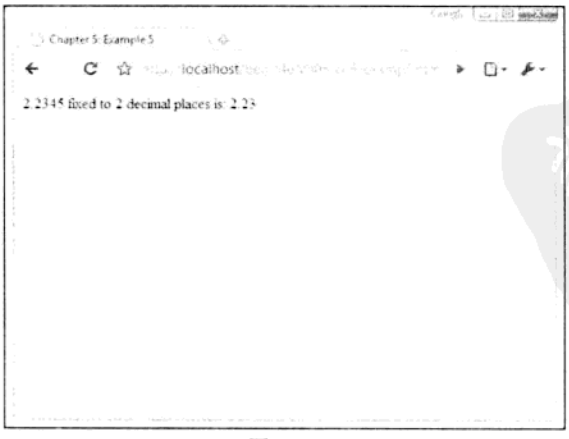

图　5-5

页面开头定义了 fix()函数。这个函数把其 fixNumber 参数变成小数点后有 decimalPlaces 位的数字。例如，把 35.76459 变成舍入后的 3 位小数的数字，结果是 35.765。

该函数的第一行代码把变量 div 设置为 10 的 decimalPlaces 次幂，decimalPlaces 是指定的小数位数。

```
function fix(fixNumber, decimalPlaces)
{
    var div = Math.pow(10,decimalPlaces);
```

下一行代码计算新的数值：

```
fixNumber = Math.round(fixNumber * div) / div;
```

代码 Math.round(fixNumber * div)的作用是将要转换的数字中的小数点移动到要保留的位数之后。对于 2.2345，如果要保留两位小数，就将它转换为 223.45。然后，Math.round()方法把这个数值舍入为最近的整数(本例是 223)，并去掉多余的小数部分。

然后，把这个整数转换回相应的小数，当然仅有指定的小数位保留下来。为此，需要除以 div，即刚才的乘数。在这个例子中，将 223 除以 100，得到 2.23。这就是将 2.2345 变成有 2 位小数的数字的结果。下面的代码把这个值返回给调用代码。

```
    return fixNumber;
}
```

在页面正文中，使用两个提示框获取用户的输入，然后用 document.write()给用户显示在 fix()函数中处理这些数字的结果。

5.2.5 Number 对象

与 String 对象类似，只有先创建 Number 对象，才能使用它。要创建 Number 对象，可以使用如下代码：

```
var firstNumber = new Number(123);
var secondNumber = new Number('123');
```

也可以声明一个基本数值，再将其用作 Number 对象，让 JavaScript 在后台实现对象的转换。例如：

```
var myNumber = 123.765;
```

与 String 对象类似，只要 JavaScript 能在后台推断出要创建什么对象，就优先使用这个技巧。例如：

```
var myNumber = "123.567";
```

JavaScript 能正确地推断出这是一个字符串。因此，使用 Number 对象的方法的任何尝试都会失败。

下面将介绍 Number 对象的一个最常用的方法——toFixed()方法。

5.2.6　toFixed()方法

toFixed()方法可以在指定点处截断一个数字。假定要显示某个税后价格。假如该价格是 9.99 美元，而营业税是 7.5%，则税后价格是 10.739 25 美元。然而，对于货币交易来说这是个很古怪的数值——税后价格的小数位数不应超过两位。下面是一个例子：

```
var itemCost = 9.99;
var itemCostAfterTax = 9.99 * 1.075;
document.write("Item cost is $" + itemCostAfterTax + "<br>");
itemCostAfterTax = itemCostAfterTax.toFixed(2);
document.write("Item cost fixed to 2 decimal places is " + itemCostAfterTax);
```

第一个 document.write()在页面上输出如下信息：

```
Item cost is 10.73925
```

但这并不是我们需要的格式，我们只需两位小数，所以使用下面的代码：

```
itemCostAfterTax = itemCostAfterTax.toFixed(2);
```

使用 Number 对象的 toFixed()方法，使变量 itemCostAfterTax 中的值保留两位小数。toFixed()方法的唯一参数是要保留的小数位数。于是，下一个 document.write()语句显示：

```
Item cost fixed to 2 decimal places is 10.74
```

为什么结果是 10.74 而不是 10.73 呢？toFixed()方法不仅截去多余的小数位，还会进行四舍五入操作。在本例中，数值是 10.739，向上修整为 10.74。如果数值是 10.732，就向下修整为 10.73。

注意，只能修整小数位数为 0～20 的数字。

5.2.7　Date 对象

在 JavaScript 中，Date 对象用于处理日期和时间。使用 Date 对象可以获取当前的日期和时间，存储自己的日期和时间，计算这些日期，把日期转换为字符串。

Date 对象包含大量方法，但其用法有点难度，所以第 10 章专门介绍 JavaScript 中的日期、时间和计时器。第 12 章还将介绍如何使用日期判断自用户上一次访问网站以来是否添加了新内容。本节将重点介绍如何创建 Date 对象以及 Date 对象的一些常用方法。

1. 创建 Date 对象

可以使用 4 种方法来声明和初始化 Date 对象。第一种方法仅声明一个新的 Date 对象，而不初始化其值。此时，日期和时间值设置为运行该脚本的 PC 机的当前日期和时间。

```
var theDate1 = new Date();
```

第二种方法是在定义 Date 对象时，传入从 GMT(格林威治标准时间)1970 年 1 月 1 日 00:00:00 开始所经过的毫秒数。在下面的例子中，日期是 GMT 2000 年 1 月 31 日 00:20:00(午

夜后 20 分钟)：

```
var theDate2 = new Date(949278000000);
```

很少使用这种方法来定义 Date 对象，但 JavaScript 实际上使用这种方式来保存日期。提供日期的其他格式仅是为了方便起见。

接下来，可以传入一个表示日期或者日期和时间的字符串。在下面的示例中，传入字符串"31 January 2010"：

```
var theDate3 = new Date("31 January 2010");
```

但是，在编写日期时，可以写成"31 Jan 2010"、"Jan 31 2010"，或者其他的有效形式。如果对此有怀疑，就试一试。注意，Firefox 不支持"01-31-2010"格式的日期。

如果为美国之外的国际用户编写网页，就需要知道指定日期的不同方式。在英国和其他很多地方，日期的标准格式为“日、月、年”，然而在美国，日期的标准格式为“月、日、年”。如果仅指定数字，就可能出问题，JavaScript 可能把用户定义的“月”理解为“日”。避免这个问题的最简单办法就是总是使用月份的名称，以免造成混乱。

第四种定义 Date 对象的方法是，初始化时传入用逗号分隔的参数：年,月,日,小时,分钟,秒,毫秒，例如：

```
var theDate4 = new Date(2010,0,31,15,35,20,20);
```

这个日期是 2010 年 1 月 31 日 15:35:20，20 毫秒，如果忽略时间部分，也可以仅声明日期部分。

注意，在这个示例中，一月用 0 表示，而不是 1，十二月用 11 表示。在指定月份时这很容易出错。

2. 获得日期值

保存了日期后，如何获取日期信息呢？可以使用 get 方法，如表 5-13 所示：

表 5-13

方　法	返　回　值
getDate()	月份中的第几天
getDay()	表示星期几的整数，其中，0 表示星期日，1 表示星期一，依此类推
getMonth()	表示月份的整数，其中，0 表示一月，1 表示二月，依此类推
getFullYear()	以 4 位数表示的年份
toDateString()	基于当前时区，返回一个人们便于阅读的完整日期字符串，如"Wed 31 Dec 2003"

例如，要获取 ourDateObj 中的月份，可以使用如下代码：

```
theMonth = myDateObject.getMonth();
```

所有方法都以非常类似的方式工作，并且所有的返回值都基于本地时间，这里，本地

时间是运行代码的计算机的本地时间。也可以使用世界标准时间，以前称为 GMT，参见第 9 章。

试一试　使用 Date 对象获得当前日期

这个示例使用 get date type 方法，将当前的月、日、年输出到网页上：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 6</title>
</head>
<body>

<script type="text/javascript">

var months = new Array("January","February","March","April","May","June","July",
                       "August", "September", "October", "November","December");

var dateNow = new Date();
var yearNow = dateNow.getFullYear();
var monthNow = months[dateNow.getMonth()];
var dayNow = dateNow.getDate();
var daySuffix;

switch (dayNow)
{
    case 1:
    case 21:
    case 31:
       daySuffix = "st";
       break;
    case 2:
    case 22:
       daySuffix = "nd";
       break;
    case 3:
    case 23:
       daySuffix = "rd";
       break;
    default:
       daySuffix = "th";
       break;
}

document.write("It is the " + dayNow + daySuffix + "day");
document.write("in the month of " + monthNow);
document.write(" in the year " + yearNow);

</script>
</body>
```

```
</html>
```

将以上代码保存为 ch5_examp6.htm。如果加载该页面，页面将用一个格式正确的语句显示当前日期。

上面的代码首先声明一个数组，并填充一年的各个月份。为什么要这样做呢？因为 Date 对象的方法都不返回月份的名称，而返回以数字表示的月份。但这并不是什么问题，只要声明一个月份数组，并用月份数字作为该数组的索引，就能选择正确的月份名称。

```
var months = new Array("January","February","March","April","May","June",
            "July","August","September","October","November","December");
```

接下来，创建一个新的 Date 对象，但不用自己的值初始化它，这样就可以把它初始化为当前的日期和时间：

```
var dateNow = new Date();
```

然后，把变量 yearNow 设置为 getFullYear()方法的返回值，即当前年份：

```
var yearNow = dateNow.getFullYear();
```

之后，使用 getMonth()返回的索引数，把对应数组元素包含的值填充到 monthNow 变量中。getMonth()方法返回的月份是一个整数值，并且用 0 表示一月，这是一个额外的好处，因为数组的索引也是从 0 开始，所以不需要任何调整，就可以找到正确的数组元素。

```
var monthNow = months[dateNow.getMonth()];
```

最后，把月份中的当前日期赋给变量 dayNow：

```
var dayNow = dateNow.getDate();
```

之后使用第 3 章介绍的 switch 语句，可以非常方便地为日期加上正确的后缀。毕竟，输出“it is the 1st day”的应用程序要比输出“it is the 1 day”看起来更专业。这有点麻烦，因为要添加的后缀依赖于该后缀前面的数字。所以，对于 1 号、21 号或 31 号，代码如下：

```
switch (dayNow)
{
    case 1:
    case 21:
    case 31:
        daySuffix = "st";
        break;
```

对于 2 号、22 号，代码如下：

```
case 2:
case 22:
    daySuffix = "nd";
    break;
```

对于 3 号、23 号，代码如下：

```
case 3:
case 23:
    daySuffix = "rd";
    break;
```

最后，其他的日期需要在 default 中处理，只需要加上后缀 th：

```
default:
    daySuffix = "th";
    break;
}
```

在最后几行代码中，使用 document.write()把信息输出到 HTML 页面上。

3. 设置日期值

要修改 Date 对象中的日期部分，可以使用一组 set 方法，它们与前面介绍的 get 函数相对应，只不过这里是设置日期值，而不是获取日期值。这些 set 方法如表 5-14 所示：

表　5-14

方　法	说　明
setDate()	月中的某一天作为参数传入，以设置日期
setMonth()	年中的某一月作为整数参数传入，其中 0 表示一月，1 表示二月，依此类推
setFullYear()	把年份设置为作为参数传入的 4 位整数

注意：
出于安全方面的考虑，基于 Web 的 JavaScript 无法修改用户计算机上的当前日期和时间。

要将年份设置为 2009，代码如下：

```
myDateObject.setFullYear(2009);
```

将日期和月份设置为 2 月 27 号，代码如下：

```
myDateObject.setDate(27);
myDateObject.setMonth(1);
```

这里要注意，没有与 getDay()方法直接对应的 set 方法。确定年、月、日后，会自动设置星期几。

5. 日期的计算

先看看下面的代码：

```
var myDate = new Date("1 Jan 2010");
myDate.setDate(32);
document.write(myDate);
```

这段代码有错误吗——1 月份并没有 32 号呀？这段代码当然没有什么错误，JavaScript 知道 1 月份并没有 32 号，因此会把日期设置为 1 月 1 号之后的 32 天，即把日期设置为 2 月 1 号。

这也适用于 setMonth()方法。如果设置的值大于 11，则日期将自动滚动到下一年。所以，如果使用 setMonth(12)，日期就设置为下一年的一月。同样，setMonth(13)表示下一年的二月。

如何使用 setDate()方法和 setMonth()方法的这一特性？例如需要确定从现在开始的第 28 天的日期。由于不同的月份有不同的天数，28 天后可能滚动到下一年，因此这并不像乍看起来那么简单。或者至少如果没有 setDate()，这个问题就没有那么简单，完成这个任务的代码如下：

```
var nowDate = new Date();
var currentDay = nowDate.getDate();
nowDate.setDate(currentDay + 28);
```

首先将变量 nowDate 定义为一个不带初始值的 Date 对象，以获取当前系统的日期。第二行代码将月中的当前天数放在变量 currentDay 中。为什么？在使用 setDate()时，如果传入的值大于该月的天数，则该方法将从该月的第 1 天开始，向前计数传入的天数。所以，如果今天是 1 月 15 号，使用 setDate(28)方法，则并不是表示 1 月 15 号之后的 28 天，而是 1 月 1 日之后的 28 天。如果要计算的是当前日期之后的 28 天，就需要给当前的天数加上 28。即使用 setDate(15 + 28)。第三行把日期设置为当前天数加上 28 天。该月的当前天数保存在 currentDay 中，因此给它加上 28，计算当前日期之后的 28 天。

如果想获得当前日期之前第 28 天的日期，只需使用当前日期减去 28。注意这大多是一个负值。需要将上面代码的第三行修改为：

```
nowDate.setDate(currentDay - 28);
```

setDate()的这个原则也适用于 setMonth()方法。

5. 获得时间值

用于获取时间数据中各个部分的方法，类似于获取日期值的 get 方法。这些方法是：

- getHours()
- getMinutes()
- getSeconds()
- getMilliseconds()
- toTimeString()

这些方法分别返回指定 Date 对象的小时、分钟、秒、毫秒和整个时间，这里的时间是 24 小时制，即 0 表示午夜，23 表示晚上 11 点。最后一个方法与 toDateString()方法类似，它返回一个易于理解的字符串，只是这里它包含时间(例如："13:03:51 UTC")。

试一试　**在网页上输出当前时间**

下面的例子把当前时间输出在页面上。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 7</title>
</head>
<body>
<script type="text/javascript">

var greeting;

var nowDate = new Date();
var nowHour = nowDate.getHours();
var nowMinute = nowDate.getMinutes();
var nowSecond = nowDate.getSeconds();

if (nowMinute < 10)
{
    nowMinute = "0" + nowMinute;
}

if (nowSecond < 10)
{
    nowSecond = "0" + nowSecond;
}

if (nowHour < 12)
{
    greeting = "Good Morning";
}
else if (nowHour < 17)
{
    greeting = "Good Afternoon";
}
else
{
    greeting = "Good Evening";
}

document.write("<h4>" + greeting + " and welcome to my website</h4>")
document.write("According to your clock the time is ");
document.write(nowHour + ":" + nowMinute + ":" + nowSecond);

</script>
</body>
</html>
```

将这个页面保存为 ch5_examp7.htm 文件。在 Web 浏览器中加载它时，页面将根据当

前的时间显示一条问候语和当前的时间，如图 5-6 所示。

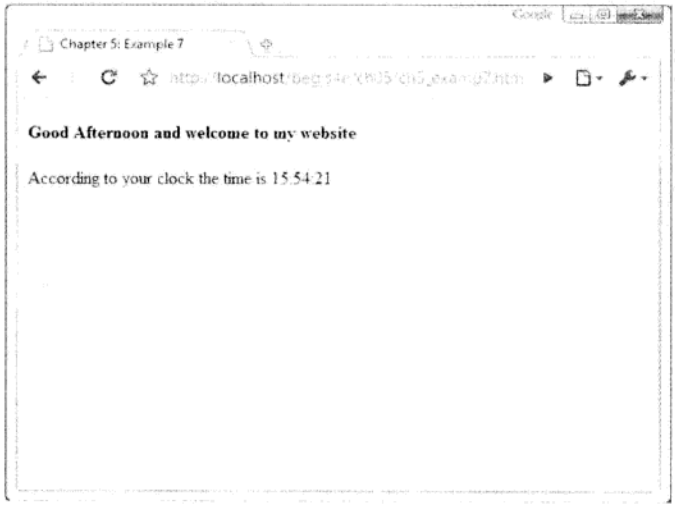

图 5-6

上述代码的前两行声明了两个变量 greeting 和 nowDate：

```
var greeting;
var nowDate = new Date();
```

稍后将使用变量 greeting 来保存网站上的欢迎信息，例如："Good Morning"、"Good Afternoon"或者"Good Evening"。变量 nowDate 初始化为一个新的 Date 对象。注意，该 Date 对象的构造函数是空的，因此 JavaScript 在其中存储当前日期和时间。

接下来从 nowDate 中获取当前时间的信息，并将其保存在不同的变量中。可以看到，获取时间数据与获取日期数据是非常类似的，只是使用了不同的方法而已。

```
var nowHour = nowDate.getHours();
var nowMinute = nowDate.getMinutes();
var nowSecond = nowDate.getSeconds();
```

为什么示例包含下面的代码行呢？

```
if (nowMinute < 10)
{
    nowMinute = "0" + nowMinute;
}

if (nowSecond < 10)
{
    nowSecond = "0" + nowSecond;
}
```

这段代码仅用于格式化。如果时间为 10 点 9 分，则应显示 10:09，而不是 10:9，但如果使用 getMinutes()方法，就会得到 10:9，没有额外的 0。秒数同样如此。如果仅在计算中

使用该数据，则不需要关心格式化问题——这里要考虑格式化，是因为要把执行代码所得的时间插入到网页上。

下面的一系列 if 语句根据一天中的时间，判断创建什么问候语，并显示给用户。

```
if (nowHour < 12)
{
    greeting = "Good Morning";
}
else if (nowHour < 17)
{
    greeting = "Good Afternoon";
}
else
{
    greeting = "Good Evening";
}
```

最后，将问候语和当前时间显示在页面上：

```
document.write("<h4>" + greeting + " and welcome to my website</h4>");
document.write("According to your clock the time is ");
document.write(nowHour + ":" + nowMinute + ":" + nowSecond);
```

第 10 章将在 Web 页面上输出持续更新的时间，让它看起来像一个时钟。

6. 设置时间的值

要设置 Date 对象的时间，也可以使用一系列与获取时间对应的方法：

- setHours()
- setMinutes()
- setSeconds()
- setMilliseconds()

这几个方法与设置日期的方法很类似，如果设置的时间参数是不合法的值，则 JavaScript 将向前或向后滚动相应的时间。例如，如果现在是 9:57，把分钟设置为 64，则时间会设置为 10:04——即 9:00 之后的 64 分钟。

如下面的代码所示：

```
var nowDate = new Date();
nowDate.setHours(9);
nowDate.setMinutes(57);
alert(nowDate);

nowDate.setMinutes(64);
alert(nowDate);
```

首先声明变量 nowDate，给它赋予一个包含当前日期和时间的新的 Date 对象。接下来的两行代码分别把小时设置为 9，把分钟设置为 57。然后用一个警告框显示日期和时间，

即 9:57。随后，把分钟设置为 64，并再次使用警告框将日期和时间显示给用户。此时，由于分钟数使小时数向前滚动，因此显示的时间是 10:04。

如果小时数设置为 23 而不是 9，同时将分钟数设置为 64，那么不仅会向前滚动 1 小时，还将向前滚动到下一天。

5.3 创建对象的新类型(引用类型)

这一节介绍一些高级内容。这并不是必需的，因此可以跳过这一节，以后再阅读。

如前所述，JavaScript 提供了大量内置于该语言的对象，供我们使用。这有点像建好的房子，我们只需搬进去住即可。但是，如果我们想根据特定的需求来建造自己的房子，该怎么办呢？此时需要让建筑师绘制技术图样和设计图，以提供新房子的模板，建筑施工人员使用设计图确定如何造房子。

这些与 JavaScript 和对象有什么关系？JavaScript 允许像建筑师那样，根据自己的需求创建特定对象的模板。例如创建一个电影票预订系统。JavaScript 没有任何内置的电影预订对象，因此必须设计自己的电影票预订对象，这需要以现实世界为模型来创建对象。对于简单的电影票预订系统，需要一个代表客户预订信息的对象，和一个表示已预订电影票的对象。除了保存信息之外，还可以为对象建立自己的方法。对于预订系统，需要一个“新增预订”方法，或者一个获取当前所有预订信息的方法。

如果只需要某个功能，而不需要保存数据——例如前面介绍的 fix()函数，则通常只需编写一个代码库，而不创建特定的对象。

房屋建筑人员需要建筑师的设计图纸，来确定建造什么房子、房子如何布局等。与此类似，我们需要提供蓝图，告诉 JavaScript 如何构建对象。例如，需要定义对象的方法，提供方法的代码。其关键是 JavaScript 支持引用类型的定义。引用类型实际上是对象的模板，就像建筑师的图纸是建造房子的模板一样。在使用新的对象类型之前，需要先定义对象类型及其方法和属性。一个重要的区别是，在定义引用类型时，不会创建基于该类型的对象。只有使用 new 关键字创建该引用类型的实例，才会根据蓝图或原型(prototype)创建该类型的对象。

在开始之前，要理清一个重要的区别。许多开发人员把引用类型称为类，并交替使用这两个术语。这在许多基于对象的语言中都是正确的，例如 Java、C#和 C++，但在 JavaScript 中不正确。JavaScript 没有正式的类结构，但该语言完全支持对应的逻辑，即引用类型。

还必须指出，本章前面讨论的内置对象也是引用类型。String、Array、Number、Date 和 Object 都是引用类型，用户创建的对象是这些类型的实例。

引用类型由以下三部分组成：

- 构造函数
- 方法定义
- 属性

构造函数是一个方法，每次根据这个引用类型创建对象时，都会调用构造函数。用某种方式初始化属性或对象时，就可以使用构造函数。即使不向构造函数传递参数，或者构

造函数不包含任何代码(其定义为空)，也需要创建一个构造函数。与函数类似，构造函数可以有 0 个或多个参数。

使用 JavaScript 的内置引用类型时，可以使用其方法。现在可以构建自己的类型，定义自己的方法，以便执行指定任务。自己的引用类型可以指定要包含什么方法，以及每个方法执行的代码。以前，我们使用了内置对象的属性，现在可以定义自己的属性。不需要声明引用类型的属性，而可以直接使用属性，JavaScript 无需事先知道。

下面根据现实世界中的电影票预订系统来创建一个简单的引用类型。

1. 定义引用类型

首先创建一个表示客户预订信息的类型，称为 CustomerBooking。第一步是创建构造函数，如下所示：

```
function CustomerBooking (bookingId, customerName, film, showDate)
{
    this.customerName = customerName;
    this.bookingId = bookingId;
    this.showDate = showDate;
    this.film = film;
}
```

这段代码似乎只是一个函数。没错，在开始定义属性和方法之前，它就只是一个函数。这与其他编程语言相反，其他编程语言有定义类型的更正式方式。

注意：

通常，引用类型用一个大写字母定义，这有助于简捷地区分函数和引用类型。

查看上面的代码，一定要注意，构造函数的名称必须匹配当前定义的类型名称——本例是 CustomerBooking。这样，把该类型的一个新实例创建为对象(称为对象实例)时，就会自动调用该构造函数。这个构造函数有 4 个参数，它们都在定义中使用。注意，此处使用了 this 关键字，例如：

```
this.customerName = customerName;
```

在构造函数或方法内，this 关键字表示引用类型的当前对象实例。这行代码引用了当前对象实例的 customerName 属性，将它设置为参数 customerName。如果读者使用过其他面向对象的编程语言，就可能想知道这个 customerName 属性是在何处定义的。答案是我们没有定义它，只为属性赋值，JavaScript 将自动创建该属性。这里没有检查该属性是否存在，JavaScript 将根据需要创建它。如果给对象使用了从未在类型定义中提及的属性，JavaScript 也将自动创建该属性。这种自由创建属性的方式看起来很好，但是它有一些缺点，一个主要的缺点是，如果不小心拼错了属性名，JavaScript 不会给出提示，而是用拼错的名称创建一个新属性，这将难以跟踪错误。解决这个问题的一个方法是创建方法，来获取属性的值并设置属性的值。这听起来很麻烦，但可以减少错误，至少更容易发现错误。下面为 CustomerBooking 引用类型创建几个属性 get 和 set 方法：

```
CustomerBooking.prototype.getCustomerName = function()
{
    return this.customerName;
}

CustomerBooking.prototype.setCustomerName = function(customerName)
{
    this.customerName = customerName;
}

CustomerBooking.prototype.getShowDate = function()
{
    return this.showDate;
}

CustomerBooking.prototype.setShowDate = function(showDate)
{
    this.showDate = showDate;
}

CustomerBooking.prototype.getFilm = function()
{
    return this.film;
}

CustomerBooking.prototype.setFilm = function(film)
{
    this.film = film;
}

CustomerBooking.prototype.getBookingId = function()
{
    return this.bookingId;
}

CustomerBooking.prototype.setBookingId = function(bookingId)
{
    this.bookingId = bookingId;
}
```

现在已经为 4 个属性 bookingId、film、customerName 和 showDate 定义了相应的 get 和 set 方法。下面看看如何定义其中一个方法 getCustomerName()：

```
CustomerBooking.prototype.getCustomerName = function()
{
    return this.customerName;
}
```

首先需要注意的是，这是定义函数的一种非常古怪的方式。它把该类型的 prototype 属性的 getCustomerName 设置为等于一个函数，紧接着定义了该函数。实际上，JavaScript 提供的大部分引用类型都有 prototype 属性，通过它可以创建新的属性和方法。因此，每当需要为类型创建方法时，都可以使用如下代码：

```
typeName.prototype.methodName = function(method parameter list)
{
     // method code
}
```

创建了类型后，如何根据它创建新对象呢？

2. 创建和使用引用类型的实例

创建引用类型的实例与创建 JavaScript 内置类型的实例一样：使用 new 关键字。所以，要创建 CustomerBooking 的一个新实例，可以使用如下代码：

```
var firstBooking = new
    CustomerBooking(1234, "Robert Smith","Raging Bull", "25 July 2004 18:20");

var secondBooking = new
    CustomerBooking(1244, "Arnold Palmer","Toy Story", "27 July 2004 20:15");
```

与 String 对象一样，这里创建了两个新对象，并保存在变量 firstBooking 和 secondBooking 中，但这次这两个新对象基于 CustomerBooking 类型。

注意：

使用构造函数创建对象时，使用 new 关键字是非常重要的。如果没有使用 new 关键字，浏览器不会抛出错误，但脚本无法正常工作。此时不会创建新对象，而是给全局窗口对象添加属性。不使用 new 关键字导致的问题难以诊断，所以在使用构造函数创建对象时，一定要指定 new 关键字。

下面调用了这两个对象的 getCustomerName()方法，并将结果输出到页面上：

```
document.write("1st booking person's name is " +
                firstBooking.getCustomerName() + "<br/>");
document.write("2nd booking person's name is " +
                secondBooking.getCustomerName());
```

下面是输出到页面上的、包含在这些对象中的信息：

```
1st booking person's name is Robert Smith
2nd booking person's name is Arnold Palmer
```

将所有代码放在一个页面上：

```
<html>
<body>

<script type="text/javascript">

// CustomerBooking type

function CustomerBooking(bookingId, customerName, film, showDate)
{
    this.customerName = customerName;
    this.bookingId = bookingId;
```

```
    this.showDate = showDate;
    this.film = film;
}

CustomerBooking.prototype.getCustomerName = function()
{
    return this.customerName;
}

CustomerBooking.prototype.setCustomerName = function(customerName)
{
    this.customerName = customerName;
}

CustomerBooking.prototype.getShowDate = function()
{
    return this.showDate;
}

CustomerBooking.prototype.setShowDate = function(showDate)
{
    this.showDate = showDate;
}

CustomerBooking.prototype.getFilm = function()
{
    return this.film;
}

CustomerBooking.prototype.setFilm = function(film)
{
    this.film = film;
}

CustomerBooking.prototype.getBookingId = function()
{
    return this.bookingId;
}

CustomerBooking.prototype.setBookingId = function(bookingId)
{
    this.bookingId = bookingId;
}

var firstBooking = new CustomerBooking(1234,
                        "Robert Smith","Raging Bull", "25 July 2004 18:20");
var secondBooking = new CustomerBooking(1244,
                        "Arnold Palmer","Toy Story", "27 July 2004 20:15");
document.write("1st booking persons name is " +
                firstBooking.getCustomerName() + "<br />");
document.write("2nd booking persons name is " +
                secondBooking.getCustomerName());
```

```
</script>

</body>
</html>
```

页面的顶部是<script />元素，其中是定义引用类型的代码。在每个使用该类型以创建对象的页面中，都必须包含类型定义代码。为简便起见，可以把这个定义单独放在一个文件中，并在每个使用该引用类型的页面中导入该文件。为此要使用<script />元素，但不是把类型定义代码放在开闭标记之间，而使用<script>元素的 src 特性来指定包含 JavaScript 的文件。例如，如果创建一个 MyCinemaBookingTypes.js 文件，以保存类型代码，则可以在页面中使用如下语句导入该文件：

```
<script src="MyCinemaBookingTypes.js"></script>
```

src 特性指向包含类型定义的 JavaScript 文件的 URL，这个例子假设.js 文件与页面位于同一目录中。

3. 数据项的数组

到现在为止，我们有了数据项的引用类型，来表示单个预订项目，但还没有表示某电影院的所有预订项的类型。如何创建一个可以包含 0 个或多个预订项的电影院类型呢？答案就是使用本章前面和第 3 章介绍的数组。

首先定义这个新类型，命名为 Cinema 类，并添加到包含 CustomerBooking 定义的脚本块中。

```
// Cinema type

function Cinema()
{
    this.bookings = new Array();
}
```

上面的代码定义了构造函数。在该构造函数中，将 bookings 属性初始化为包含所有的 CustomerBooking 实例对象。

接下来添加用于预订电影票的方法，为此创建 addBooking()方法：

```
cinema.prototype.addBooking = function(bookingId,customerName,film,showDate)
{
    this.bookings[bookingId] = new CustomerBooking(bookingId,
                                                   customerName,film, showDate);
}
```

该 addBooking()方法接受 4 个参数，即创建一个新的预定项所需的信息。接着在该方法中，创建 CustomerBooking 类型的一个新对象，对这个对象的引用保存在 bookings 数组中，并使用唯一的 bookingId 关联存储新对象的位置。

下面看看如何访问数组中的元素。在下面的 getBookingsTable()方法中，遍历电影院中的每个预订项，并创建相应的 HTML，在表格中显示所有的预订项：

```
Cinema.prototype.getBookingsTable = function()
{
    var booking;
    var bookingsTableHTML = "<table border=1>";

    for (booking in this.bookings)
    {
       bookingsTableHTML += "<tr><td>";
       bookingsTableHTML += this.bookings[booking].getBookingId();
       bookingsTableHTML += "</td>";

       bookingsTableHTML += "<td>";
       bookingsTableHTML += this.bookings[booking].getCustomerName();
       bookingsTableHTML += "</td>";

       bookingsTableHTML += "<td>";
       bookingsTableHTML += this.bookings[booking].getFilm();
       bookingsTableHTML += "</td>";

       bookingsTableHTML += "<td>";
       bookingsTableHTML += this.bookings[booking].getShowDate();
       bookingsTableHTML += "</td>";
       bookingsTableHTML += "</tr>";
    }

    bookingsTableHTML += "</table>";
    return bookingsTableHTML;
}
```

可以通过唯一的 bookingId 访问每个预订项。但我们只想简单地遍历电影院的所有预订项，所以使用 for...in 循环，遍历 items 数组的每个元素。每次执行循环时，JavaScript 都会把 booking 设置为包含下一个预订项的 bookingId。它不包含元素本身，而包含对应的关键字。

通过对应的关键字，就可以访问数组中的元素对象，如下所示：

```
this.bookings[booking]
```

其中 this 引用的是引用类型的对象实例。接着使用 CustomerBooking 对象的 get 方法来获得每个预订项的信息。最后一行将包含所有预订项的小结的 HTML 返回给调用代码。

把全部代码放在一个页面上，并将其保存为 ch5_examp8.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Example 8</title>
</head>
<body>

<h2>Summary of bookings</h2>
```

```
<script type="text/javascript">

// CustomerBooking type

function CustomerBooking(bookingId, customerName, film, showDate)
{
    this.customerName = customerName;
    this.bookingId = bookingId;
    this.showDate = showDate;
    this.film = film;
}

CustomerBooking.prototype.getCustomerName = function()
{
    return this.customerName;
}

CustomerBooking.prototype.setCustomerName = function(customerName)
{
    this.customerName = customerName;
}

CustomerBooking.prototype.getShowDate = function()
{
    return this.showDate;
}

CustomerBooking.prototype.setShowDate = function(showDate)
{
    this.showDate = showDate;
}

CustomerBooking.prototype.getFilm = function()
{
    return this.film;
}

CustomerBooking.prototype.setFilm = function(film)
{
    this.film = film;
}

CustomerBooking.prototype.getBookingId = function()
{
    return this.bookingId;
}

CustomerBooking.prototype.setBookingId = function(bookingId)
{
    this.bookingId = bookingId;
}

// Cinema type
```

```
function Cinema()
{
    this.bookings = new Array();
}

Cinema.prototype.addBooking = function(bookingId,customerName,film,showDate)
{
   this.bookings[bookingId] = new CustomerBooking(bookingId,
                                                  customerName,film,showDate);
}

Cinema.prototype.getBookingsTable = function()
{
    var booking;
    var bookingsTableHTML = "<table border=1>";

    for (booking in this.bookings)
    {
        bookingsTableHTML += "<tr><td>";
        bookingsTableHTML += this.bookings[booking].getBookingId();
        bookingsTableHTML += "</td>";

        bookingsTableHTML += "<td>";
        bookingsTableHTML += this.bookings[booking].getCustomerName();
        bookingsTableHTML += "</td>";

        bookingsTableHTML += "<td>";
        bookingsTableHTML += this.bookings[booking].getFilm();
        bookingsTableHTML += "</td>";

        bookingsTableHTML += "<td>";
        bookingsTableHTML += this.bookings[booking].getShowDate();
        bookingsTableHTML += "</td>";
        bookingsTableHTML += "</tr>";
    }
    bookingsTableHTML += "</table>";
    return bookingsTableHTML;
}

var londonOdeon = new Cinema();
londonOdeon.addBooking(342,"Arnold Palmer","Toy Story","15 July 200920:15");
londonOdeon.addBooking(335, "Louise Anderson",
                       "The Shawshank Redemption", "27 July 2009 11:25");
londonOdeon.addBooking(566, "Catherine Hughes",
                       "Never Say Never", "27 July 2009 17:55 ");
londonOdeon.addBooking(324, "Beci Smith", "Shrek", "29 July 2009 20:15");

document.write(londonOdeon.getBookingsTable());
</script>

</body>
</html>
```

新增代码如下：

```
var londonOdeon = new cinema();
londonOdeon.addBooking(342,"Arnold Palmer", "Toy Story", "15 July 2009 20:15");
londonOdeon.addBooking(335,"Louise Anderson",
                       "The Shawshank Redemption", "27 July 2009 11:25");
londonOdeon.addBooking(566,"Catherine Hughes",
                       "Never Say Never", "27 July 2009 17:55");
londonOdeon.addBooking(324, "Beci Smith", "Shrek", "29 July 2009 20:15");

document.write(londonOdeon.getBookingsTable());
```

这段代码创建了一个新的 cinema 对象，并将该对象的引用保存在变量 londonOdeon 中。接着使用 Cinema 类型的 addBooking()方法创建了 4 个新的预订项。最后一行代码将 getBookingsTable()方法返回的 HTML 输出到页面上。

页面如图 5-7 所示。

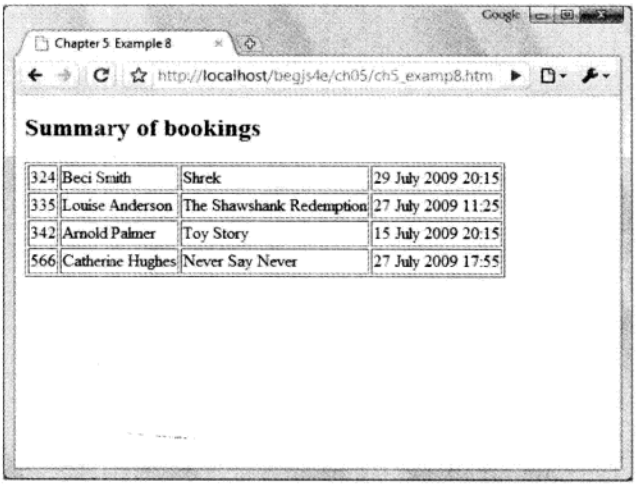

图　5-7

尽管前面创建的电影票预订系统只是一个十分简单的系统，但它说明通过创建自己的引用类型，有助于对现实世界中的问题和事物建模，并使代码更便于维护。

5.4　小结

本章介绍了对象的概念，这些概念对于理解 JavaScript 是至关重要的，对象几乎代表了 JavaScript 的一切。另外，还介绍了 JavaScript 语言用于增强其功能的一些原生引用类型。

本章的主要内容如下：

- JavaScript 基于对象，用对象的概念来表示事物，如字符串、日期和数组等。
- 对象有属性和方法。例如，Array 对象有 length 属性和 sort()方法。
- 要创建新对象，只需编写 new ObjectType()。在创建对象时，可以选择初始化对象。

- 要获取或设置对象的属性值，只需编写 ObjectName.ObjectProperty。
- 调用对象的方法与调用函数非常类似。可以传递参数，也可以传回返回值。访问对象的方法与访问属性相同，但即使方法没有参数，也必须在方法名之后加上括号。例如可以编写 ObjectName.ObjectMethod()。
- String 类型为文本提供了大量便捷的方法，如计算文本的长度、在字符串中查找文本以及选择文本的一部分等。
- Math 类型是自动创建的，它提供了大量数学属性和方法。例如，使用 Math.random()方法可以获得 0～1 之间的随机数。
- Array 类型提供了操作数组的方式。例如，获得数组的长度、排列数组元素的顺序以及合并两个数组等。
- Date 类型提供了保存、计算以及获得日期和时间的方法。
- JavaScript 允许使用引用类型来定义自己的对象类型。引用类型可以用于对现实世界中的事物建模，使代码更容易创建和维护，但开始时需要做额外的工作。

第 5 章将讨论 Web 浏览器本身，尤其是浏览器为 JavaScript 编程提供的各种对象。使用浏览器对象是创建功能强大的网页的关键所在。

5.5 习题

在附录 A 中可以找到本章习题的参考答案。

习题 1：

使用 Date 类型，计算距今 12 个月后的日期，并输出在网页上。

习题 2：

让用户输入一个名字列表，并将名字保存在数组中。继续获取下一个名字，直到用户输入为空为止。然后按升序排列名字顺序，并输出在页面上，每个名字各占一行。

习题 3：

本章学习了如何使用 pow()方法来修整数值的小数位数。但是这个函数有一个缺陷。要把 2.1 修整为 3 位小数，正确的 fix()函数应该返回 2.100，但前面的 fix()函数返回 2.1。修改这个 fix()函数，补上所需的 0。

第 6 章

浏览器程序设计

前面几章讨论了核心 JavaScript 语言，介绍了如何使用变量和数据，在这些数据上执行操作，在代码中做出判断，循环执行同一段代码，以及编写自定义函数。JavaScript 是一种基于对象的语言，第 4 章介绍了如何使用 JavaScript 的内置对象。然而，我们不仅对该语言本身感兴趣，还希望学习如何编写 Web 浏览器的脚本。通过这个功能，就可以创建更富有感染力的网页。

不仅 JavaScript 是基于对象的，浏览器也是由对象组成的。JavaScript 在浏览器中运行时，可以访问浏览器的对象，其方式与使用上一章的 JavaScript 内置对象一样。那么，浏览器提供了哪些对象呢？

浏览器提供了许多对象。例如，window 对象对应浏览器的窗口。前面已经使用过这个对象的两个方法，即 alert()和 prompt()方法。为了简便起见，前面把它们称为函数，但它们实际上都是浏览器的 window 对象的方法。

浏览器提供的另一个对象是页面本身，用 document 对象表示。前面也使用过该对象的方法和属性。第 1 章曾使用 document 对象的 bgColor 属性来改变页面的背景颜色。还使用 document 对象的 write()方法向页面写入信息。

还有许多其他对象可表示页面上写入的 HTML。例如，每个<img/>元素都对应一个 img 对象，用于在文档中插入一幅图片。

浏览器为 JavaScript 提供的对象集合通常称为浏览器对象模型(Browser Object Model，BOM)。

注意：

另一个常见的术语是文档对象模型(Document Object Model，DOM)。本书使用术语 DOM 表示 W3C 的标准文档对象模型，详见第 12 章。

JavaScript 的这些附加功能存在一个潜在的弊端：BOM 没有标准的实现方式。可以使用哪个对象集合高度依赖于当前使用的浏览器的类型和版本。一些对象可以在某些浏览器上使用，但不能在其他浏览器上使用，另一些对象在不同的浏览器上有不同的属性和方法。幸运的是，浏览器的编写者一般不修改浏览器的 BOM，因为这么做会降低交互操作性。

这意味着，如果仅使用 BOM 的核心功能(所有浏览器都有的对象)，代码就能更好地在不同的浏览器和版本中正常运行。本章介绍 BOM 的核心功能，在 JavaScript 中仅使用这些核心功能，就能完成大量工作。要了解核心对象的更多信息，请访问 http://www.w3schools.com/dhtml/dhtml_domreference.asp 和 http://msdn.microsoft.com/en-us/library/Library/ms952605.aspx。

6.1 浏览器对象简介

本节介绍对所有浏览器都适用的 BOM 对象。

第 5 章提到，JavaScript 有许多可访问和使用的内置对象。大多数对象都需要我们自己创建，例如 String 和 Date 对象，而其他某些对象，如 Math 对象，无需创建，在页面开始加载时就可以直接使用。

JavaScript 在页面中运行时，可以访问大量由 Web 浏览器提供的其他对象。与 Math 对象类似，会自动创建这些对象，不需要显式地创建它们。如前所述，对象及其方法、属性和事件都在 BOM 中映射好了。

BOM 非常庞大，但是，初学者只需要使用 BOM 中不到 10%的对象、方法和属性。本章首先介绍 BOM 中较常用的部分，如图 6-1 所示。从某种程度上说，BOM 的这些部分是所有浏览器通用的。后面的章节以本章为基础，学完本书之后，读者就能使用 BOM 来工作了。

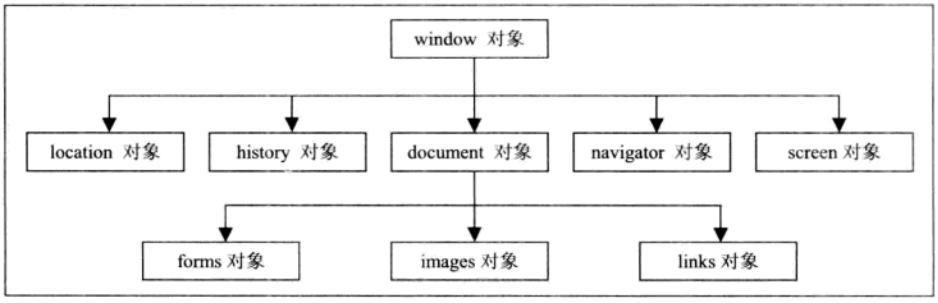

图 6-1

BOM 有一个层次结构，在层级的顶端是 window 对象，它表示浏览器的框架以及与其相关的所有内容，如滚动条和导航栏按钮等。

页面包含在窗口框架中。在 BOM 中，页面用 document 对象表示。这两个对象如图 6-2 所示。

下面逐一介绍这些 BOM 对象。

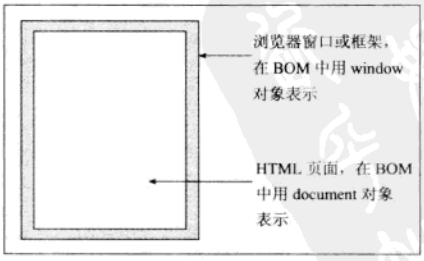

图 6-2

6.1.1　window 对象

window 对象代表浏览器的框架或窗口，其中包含了网页。在某种程度上，该对象也表示浏览器本身，它包含大量属性，因为这些属性不适合放在其他地方。例如，通过 window 对象的属性，可以确定正在运行什么浏览器、用户访问过的页面，浏览器窗口的大小和用户屏幕的大小等。还可以使用 window 对象来访问或修改浏览器状态栏中的文本、修改加载的页面甚至打开新窗口。

window 对象是一个全局对象，因此不需要使用其名称来访问其属性和方法。实际上，全局函数和全局变量(可以在页面的任何位置访问)都创建为该全局对象的属性。例如，从本书开始就经常使用的 alert()函数，就是 window 对象的 alert()方法。尽管前面使用如下代码：

```
alert("Hello!");
```

下述代码的效果与其完全相同：

```
window.alert("Hello!");
```

由于 window 对象是全局对象，因此使用第一个语句是完全正确的。

window 对象的某些属性也是对象。所有浏览器通用的对象包括 document、navigator、history、screen 和 location。document 对象表示页面；history 对象包含用户访问页面的历史信息；navigator 对象包含浏览器的信息；screen 对象包含了客户端显示能力的信息；location 对象包含了当前页面的位置信息。本章后面将介绍这几个重要的对象。

下面列举一个简单的例子：改变浏览器状态栏中显示的默认文本。状态栏(通常位于浏览器窗口的左下角)通常由浏览器用于显示正在加载的文档的状态。例如，在 IE 或 Firefox 中，文档加载完毕后，状态栏一般显示 Done。下面把它改为 Hello and Welcome。

要改变窗口状态栏中的默认信息，需要使用 window 对象的 defaultStatus 属性。为此，可使用如下代码：

```
window.defaultStatus = "Hello and Welcome";
```

或者，由于 window 对象是全局对象，因此也可以使用如下代码：

```
defaultStatus = "Hello and Welcome";
```

这两个语句都是正确而有效的。但是，加上 window 清楚地表明 defaultStatus 属性来自 window 对象，否则 defaultStatus 看起来像是一个变量名。对于像 defaultStatus 这样不常用的属性和方法，加上 window 尤其重要。对于我们非常熟悉的属性和方法，如 document 和 alert()，则可以省略 window。

把代码放在页面上：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
        <title>Chapter 6: Example 1</title>
        <script type="text/javaScript">
            window.defaultStatus = "Hello and Welcome";
        </script>
    </head>
    <body>
    </body>
    </html>
```

将该页面保存为 ch6_examp1.htm，并加载到浏览器中。状态栏中就显示了指定的信息。

注意，在网页中，自定义函数或变量不应与 BOM 对象或其属性及方法同名。否则，代码并不会出错，但可能得到意想不到的结果。例如，下面的代码声明了 defaultStatus 变量，并试图把 window 对象的 defaultStatus 属性设置为 Welcome to my website。但这不会改变状态栏中的默认信息，而改变了变量 defaultStatus 的值。

```
var defaultStatus;
defaultStatus = "Welcome to my website";
```

此时需要使用另一个变量名。出现这种情况，是因为在全局作用域中定义的任何函数或变量实际上都会添加到 window 对象中。看看下面的示例代码：

```
var myVariable = "Hello, World!";
alert(window.myVariable);
```

如果在浏览器中执行这段代码，警告框就会显示消息“Hello, World” 。

与所有 BOM 对象一样，window 对象有很多属性和方法。但是本章重点介绍其 history、location、navigator、screen 和 document 属性。这 5 个属性都包含相应的对象，即 history、location、navigator、screen 和 document 对象，每个对象都有各自的属性和方法。下面将依次学习这些对象，看看如何通过它们充分使用 BOM。

6.1.2 history 对象

history 对象跟踪用户访问的每个页面。这个页面列表常称为浏览器的历史栈(history stack)。它允许用户单击浏览器的“后退”或“前进”按钮，来重新访问页面。通过 window 对象的 history 属性可以访问 history 对象。

与 JavaScript 内置的 Array 类型类似，history 对象也有 length 属性。使用它可以获得历史栈中的页面数量。

history 对象有 back()和 forward()方法。调用它们时，浏览器当前加载的页面位置就变成用户访问过的前一个页面或后一个页面。

history 对象还有 go()方法。它带一个参数，该参数指定在历史栈中前进或后退几个页面。例如，如果想返回上一个页面之前的页面，可以使用如下代码：

```
history.go(-2);
```

要前移 3 个页面，可以使用如下代码：

```
history.go(3);.
```

注意，go(－1)等价于 back()，而 go(1)等价于 forward()。

6.1.3　location 对象

location 对象包含大量有关当前页面位置的有用信息。它不仅包含了页面的统一资源定位器(Uniform Resource Locator，URL)，还包含保存该页面的服务器、连接服务器的端口号及所使用的协议。通过 location 对象的 href、hostname、port 和 protocol 属性，就可以获得这些信息。但这些信息与页面的访问位置相关：是从服务器上加载页面，还是直接从本地硬盘上加载页面(如前面的示例)。

除了获得当前页面的位置之外，还可以使用 location 对象的方法来改变当前页面的位置，或者刷新当前页面。

可以采用两种方式导航到另一个页面。一是将 location 对象的 href 属性设置为指向另一个页面，二是使用 location 对象的 replace()方法。这两种方式的效果相同，页面都改变了位置。但是它们的区别在于：replace()方法将从历史栈中移除当前页面，代之以新页面；而使用 href 属性仅把新页面加在历史栈的顶部。这意味着如果使用了 replace()方法，用户单击浏览器中的“后退”按钮，将无法返回到最初加载的页面。但如果使用 href 属性，则用户可以正常使用“后退”按钮。

例如，要用新页面 myPage.htm 替换当前页面，使用 replace()方法的代码如下：

```
window.location.replace("myPage.htm");
```

这行代码将加载 myPage.htm，并用 myPage.htm 替换历史栈中当前页面的所有信息。

如果要加载新页面，并把它添加到历史栈的顶部，则可以使用 href 属性：

```
window.location.href = "myPage.htm";
```

当前加载的页面会添加到历史栈中。在这两种情况下，表达式的前面都加上了 window，但由于 window 对象是全局的，因此也可以将代码改写为：

```
location.replace("myPage.htm");
location.href = "myPage.htm";
```

6.1.4　navigator 对象

navigator 对象是 window 对象的一个属性，可用于所有浏览器。该名称更具有历史性而不是描述性，更恰当的名称是“浏览器对象”。因为 navigator 对象包含浏览器和运行浏览器的操作系统的大量信息。

navigator 对象最常见的用途是消弥浏览器之间的差异。使用其属性，可以确定用户的浏览器、浏览器的版本和操作系统。接着可以操作这些信息，确保将用户定向到与其浏览器兼容的页面上。本章最后一节将专门讨论这个重要问题，所以这里不再讨论。

6.1.5 screen 对象

window 对象的 screen 对象属性包含了大量有关客户机显示能力的信息。其属性包括 height 和 width，分别表示屏幕的垂直和水平尺寸，单位为像素。

后面的一个例子将使用 screen 对象的另一个属性 colorDepth，它表示客户机屏幕使用的色彩位数。

6.1.6 document 对象

与 window 对象一样，document 对象可能是 BOM 中最重要和最常用的对象之一。通过这个对象，可以访问到页面上的 HTML 元素及其属性和方法。

遗憾的是，不同浏览器的 document 对象存在较大的差异。本章重点介绍所有浏览器通用的属性和方法。有关 document 对象的高级操作的信息，请参见第 12 章。

document 对象有很多关联的属性，它们也是类似于数组的结构，称为集合。主要的集合有 forms、images 和 links。IE 还支持其他很多集合属性，如 all 集合属性，它是页面上用对象表示所有元素的数组。然而，本章重点学习使用具有跨浏览器支持功能的对象，这样网页就不会局限用于一种浏览器。

稍后将介绍 images 和 links 集合。第三个集合 forms 是下一章(讨论 Web 浏览器中的表单)的一个主题。下面先看一个简明扼要的例子，说明如何使用 document 对象的方法和属性。

6.1.7 使用 document 对象

前面遇到过 document 对象的一些属性和方法，如 write()方法和 bgColor 属性。

试一试　根据用户屏幕的色彩深度设置颜色

这个例子根据用户屏幕支持的色彩位数来设置页面的背景色。屏幕支持的色彩位数称为色彩深度。如果用户的显示器仅支持黑白两色，将背景色设置为鲜红色就毫无意义。使用 JavaScript，可以将背景色设置为用户实际可见的颜色。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 2</title>
</head>
<body>
<script type="text/javaScript">
switch (window.screen.colorDepth)
{
    case 1:
    case 4:
        document.bgColor = "white";
        break;
```

```
        case 8:
        case 15:
        case 16:
            document.bgColor = "blue";
            break;
        case 24:
        case 32:
            document.bgColor = "skyblue";
            break;
        default:
            document.bgColor = "white";
    }

    document.write("Your screen supports " + window.screen.colorDepth +
                   "bit color");
    </script>
    </body>
    </html>
```

将该页面保存为 ch6_examp2.htm。在浏览器中加载它时，页面的背景色将由当前的屏幕色彩深度决定。另外，页面上的一条信息指出当前色彩深度是多少。

可以通过改变屏幕支持的色彩深度，来测试上面的代码。在 Windows XP 上，可以在桌面上右击，选择“属性”选项。在“设置”选项卡中，可以在“颜色质量”区域改变所支持的色彩数。刷新浏览器，就能看到刚才的修改给页面的背景色带来的变化。

注意：

在 Firefox、Safari 和 Chrome 浏览器上，只有关闭并重启浏览器，才能看到上述代码的效果。

如前所述，window 对象包含 screen 对象属性。screen 对象的一个属性是 colorDepth，它返回 1、4、8、15、16、24 或 32。这表示赋予屏幕上每个像素的位数(像素仅是构成屏幕的很多个点中的一个)。要确定屏幕有多少种颜色，只需计算 2 的 colorDepth 次方。例如，colorDepth 为 1，表示只有两种颜色；而 colorDepth 为 8，表示有 256 种颜色。目前大部分屏幕的色彩深度都至少是 8，通常是 24 或 32。

脚本块的第一个任务是根据用户实际可见的色彩数，来设置页面的背景色。为此使用了一个庞大的 switch 语句，switch 语句检查的条件是 window.screen.colorDepth 的值。

```
switch (window.screen.colorDepth)
```

不需要为每个 colorDepth 设置不同的颜色，因为对于一般的 Web 应用来说，许多颜色都是相近的。而应给不同但相近的 colorDepth 值设置相同的背景色。当 colorDepth 为 1 或 4 时，把背景设置为白色。为此声明了 case 1:语句，但是未给出任何代码。如果 colorDepth 匹配该 case 语句，则执行流会进入后面的 case 4:语句，把背景色设置为白色。接着调用 break 语句，这样就不检查该 case 语句之后的其他条件。

```
{
    case 1:
```

```
    case 4:
        document.bgColor = "white";
        break;
```

colorDepth 为 8、15 或 16 时，将背景设为蓝色，代码如下所示：

```
case 8:
case 15:
case 16:
    document.bgColor = "blue";
    break;
```

最后，colorDepth 为 24 或 32 时，将背景设为天蓝色：

```
case 24:
case 32:
    document.bgColor = "skyblue";
    break;
```

用 default 语句结束 switch 语句，处理不匹配前面 case 语句的其他情况。在这个 default 语句中，再次把背景设为白色。

```
    default:
        document.bgColor = "white";
}
```

接下来的脚本使用 document 对象的 write()方法，前面的例子多次使用过该方法。这里用它在页面上输出当前设置的色彩深度位数，代码如下所示：

```
    document.write("Your screen supports " + window.screen.colorDepth +
                   "bit color")
```

本书的例子一直在使用 document 对象。第 1 章使用它的 bgColor 属性来改变页面的背景色，并且在多个例子中使用它的 write()方法，向页面输出 HTML 和文本。

下面看看 document 对象的几个稍复杂的属性。这几个属性的共同点是它们都包含集合。首先介绍的是一个集合，它包含页面上每幅图片对应的对象。

6.1.8 images 集合

可以使用以下标记在 HTML 页面中插入一幅图片：

```
<img alt="USA" name=myImage src="usa.gif">
```

浏览器通过创建一个 img 对象 myImage，使这个图像可以通过 JavaScript 来操作。实际上，页面上的每幅图片都有一个对应的 img 对象。

页面上的每个 img 对象都保存在 images 集合中，该集合是 document 对象的一个属性。这个集合以及其他集合的用法与数组相同。页面上的第一幅图片在 document.images[0]元素中，第二幅图片在 document.images[1]中，依此类推。

如有必要，可以指定一个变量来引用 images 集合中的 img 对象，以使代码更容易阅读。例如，下面的代码给变量 myImage2 赋予索引位置为 1 的 img 对象的引用：

```
var myImage2 = document.images[1];
```

现在就可以在代码中使用 myImage2 来代替 document.images[1]，二者的效果相同。

也可以通过名称来访问 images 集合中的 img 对象。例如，可以在 document 对象的 images 集合属性中访问通过<img />元素创建的 img 对象 myImage，代码如下所示：

```
document.images["myImage"]
```

由于 document.images 属性是一个集合，因此它也拥有类似于 JavaScript 内置类型 Array 的属性，如 length 属性。例如，代码 document.images.length 可以获得页面包含的图片数量。

试一试　选择图片

img 对象有许多非常有用的属性。其中最重要的是 src 属性。改变它就可以改变所加载的图片。下面的例子演示了这个过程。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 3</title>
</head>
<body>
<img name="img1" src="" border="0" width="200" height="150" />
<script type="text/javaScript">
   var myImages = new Array("usa.gif","canada.gif"," jamaica.gif", "mexico.gif");
   var imgIndex = prompt("Enter a number from 0 to 3","");
   document.images["img1"].src = myImages[imgIndex];
</script>
</body>
</html>
```

保存为 ch6_examp3.htm。另外还需要 4 个图片文件 usa.gif、canada.gif、jamaica.gif 和 mexico.gif。可以自行创建这些图片，也可以从本书的代码下载中获得。

在浏览器中加载该页面时，弹出的提示框要求输入一个 0～3 的数字。根据所输入的数字，将显示不同的图片。

页面顶部是 HTML<img />元素。注意，src 属性是空的，name 的值设置为 img1。

```
<img name="img1" src="" border="0" width="200" height="150">
```

接下来的脚本块用于决定显示哪幅图片。第一行定义了一个包含图片源列表的数组。这个例子把图片与 HTML 文件放在同一个目录中，因此没有指定路径。否则，务必输入完整路径，例如 C:\myImages\mexico.gif。

接着，要求用户输入一个 0~3 的数字，用作数组的索引，以访问 myImages 数组中的

图片源。

```
var imgIndex = prompt("Enter a number from 0 to 3","");
```

最后，把 img 对象的 src 属性设置为 myImages 数组元素(其索引号由用户提供的源文本)。

```
document.images["img1"].src = myImages[imgIndex];
```

注意，编写 document.images["img1"]，就会访问存储在 images 集合中的 img 对象。这里使用图片的名称，它在<img />元素的 name 属性中定义。也可以使用 document.images[0]，它的索引为 0，因为它是页面上的第一幅(也是唯一的)图片。

6.1.9 links 集合

对于每个有 href 属性的超链接元素<a/>，浏览器都会创建一个 a 对象。a 对象最重要的属性是 href，它对应该标记的 href 属性。使用该属性，可以确定该链接指向何处，在页面加载完后，还可以修改它。

就像前面的 img 对象包含在 images 集合中那样，页面上所有 a 对象的集合包含在 links 集合中。

6.2 用事件响应用户的操作

JavaScript 无疑是一个有效的 Web 编程工具。如前所述，它可以进行有限的数据处理。但在大多数 Web 应用程序中，数据处理一般提交给服务器，因为服务器比较适合完成这种任务。用户通过浏览器使用 Web 应用程序，浏览器也响应用户的操作。如果可以给用户的特定操作执行代码，就太好了。这可以使用事件来实现。

6.2.1 事件的含义

当特定的操作发生时，就会发生事件。例如，用户单击页面、单击超链接，或者把鼠标指针移过某段文本时，都会发生事件。另一个常用事件是页面的 load 事件，当页面完全加载到浏览器中时，窗口将引发(或触发)一个通知。

我们为什么对事件感兴趣呢？

例如，用户单击网页上的任意位置时，我们希望弹出一个菜单。假定可以编写一个弹出菜单的函数，那么如何确定何时弹出菜单？换句话说，何时调用这个函数？这需要以某种方法捕获用户单击文档的事件，并且确保该事件发生时，调用上述函数。

为此，需要使用事件处理程序(event handler)或监听器(Listener)，并把它关联到事件发生时要执行的代码上。这就提供了在事件发生时捕获事件并执行相应代码的方式。在代码中加入事件处理程序称为“将代码连接到事件”。这有点像设置闹钟——当某个事件发生时，使闹钟振铃。对于闹钟来说，事件就是到达某一时间点。

6.2.2　将代码连接到事件

第 5 章介绍了通过属性和方法来定义对象。对象还有与之相关的事件。前面没有提到事件，因为 JavaScript 的内置对象没有这些事件。但是，BOM(和 DOM)对象有相关的事件。

事件处理程序由 on 和要处理的事件组成。例如，click 事件的处理程序为 onclick，load 事件的处理程序为 onload。

通过事件处理程序，可以使用多种方法将代码连接到事件。本章将介绍两种添加事件最简单的办法。这两种方法由来已久，甚至得到更早的浏览器的支持，当前的浏览器也支持它们。第 12 章将介绍较新的、易于标准化的事件添加方法。

1. 通过 HTML 属性处理事件

第一种方法是最常见的，即把事件处理程序名和要执行的代码添加到 HTML 元素的属性上。

下面创建一个简单的 HTML 页面，该页面用元素<a/>创建了一个超链接。与这个元素关联的是 a 对象。a 对象的一个事件是 click，当用户单击超链接时，将触发 click 事件。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Connecting Events Using HTML Attributes</title>
</head>
<body>
<a href="somepage.htm" name="linkSomePage">
    Click Me
</a>
</body>
</html>
```

显然，这个页面只有一个超链接。单击它，将导航到另一个页面 somepage.htm(尚未创建)。目前还没有给链接添加事件处理程序。

如前所述，要将事件连接到代码，一个常见的简便方法是直接将事件添加到捕获了该事件的元素对象的开始标记上。在本例中，该事件是<a/>标记定义的 a 对象的 click 事件。一旦单击了这个链接，就需要捕获该事件，并连接到代码上。因此，需要给<a>开始标记添加事件处理程序 onclick 作为一个属性，该属性的值设置为事件触发时要执行的代码。

重写<a>开标记：

```
<a href="somepage.htm" name="linkSomePage" onclick="alert('You Clicked? ')">
  Click Me
</a>
```

这段代码为<a>开始标记的定义添加了 onclick="alert('You Clicked?')"。此时，单击链接时，将显示一个警告框。之后，超链接才会执行通常的功能，进入 href 属性定义的页面。

如果只有一行代码连接到事件处理程序上，这就是可行的。但如果单击链接时，要执

行多行代码，该怎么办呢？

只需定义要执行的函数，并在 onclick 代码中调用它即可。请看下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Connecting Events using HTML Attributes</title>
</head>
<body>
<script type="text/javascript">
function linkSomePage_onclick()
{
    alert('You Clicked?');
    return true;
}
</script>
<a href="somepage.htm" name="linkSomePage"
 onclick="return linkSomePage_onclick()">
      Click Me
</a>
</body>
</html>
```

上面的脚本块创建了一个标准函数，并给它取了一个描述性的名字，以提高代码的可读性。此处使用 ObjectName_event()的形式作为函数名。这样可以立即看出，这个函数与页面上的哪个对象关联，连接了哪个事件。因此，在上面的例子中，函数命名为 linkSomePage_onclick()，因为它给名为 linkSomePage 的 a 对象关联了 onclick 事件处理程序。这种命名规范是作者建立的，不是强制性的。用户也可以使用自己喜欢的其他命名方式，只要命名规范一致即可。

现在 onclick 属性连接到调用 linkSomePage_onclick()函数的代码上。因此，当用户单击超链接时，就执行这个函数。

这个函数返回一个值，本例返回 true。此外，在定义 onclick 属性时，在函数名前使用 return 语句，返回了该函数的返回值，这是为什么呢？

由 onclick="return linkSomePage_onclick()"返回的值由 JavaScript 用来决定是否执行链接的正常操作，即进入一个新页面。如果返回 true，则继续执行该操作，进入 somepage.htm。如果返回 false，则不执行正常的事件链(即进入 somepage.htm)，即取消与事件关联的操作。尝试修改函数，如下所示：

```
function linkSomePage_onclick()
{
    alert("This link is going nowhere");
    return false;
}
```

现在，会显示一个消息，而不会尝试进入 somepage.htm。

并不是所有的对象及其事件都使用返回值，因此有时返回值是多余的。另外，返回 false 也并不总是取消操作。由于浏览器的历史而不是逻辑原因，有时返回 true 将取消操作。通常，最好返回 true，并在发现异常时进行处理。

某些事件并不直接链接到用户的操作上。例如，window 对象的 load 事件在页面加载时触发，unload 事件在页面卸载(用户关闭浏览器，或者转向另一个页面)时触发。

window 对象的事件处理程序实际上放在<body>开标记中。例如，要为 load 和 unload 事件添加事件处理程序，可以使用如下代码：

```
<body onload="myOnLoadfunction()"
onunload="myOnUnloadFunction()">
```

2. 通过对象属性处理事件

下面看看第二种连接事件的方法。

这种方法首先需要定义事件发生时执行的函数。然后，把对象的事件处理程序属性设置为前面定义的函数。

下面举例说明：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 4</title>
</head>
<body>
<script type="text/javascript">
function linkSomePage_onclick()
{
    alert('This link is going nowhere');
    return false;
}
</script>
<a href="somepage.htm" name="linkSomePage">
   Click Me
</a>
<script type="text/javaScript">
 window.document.links[0].onclick = linkSomePage_onclick;
</script>
</body>
</html>
```

保存为 ch6_examp4.htm。

与前面类似，我们定义了 linkSomePage_onclick()函数。还可以返回一个值，表示是否要执行该对象的正常操作。

接下来是<a/>元素，我们要连接该对象的事件。注意，在该标记的属性中并没有提及事件处理程序或函数。

脚本的最后几行建立了对象的事件与函数的连接，如下面的代码所示：

```
<script type="text/javaScript">
   document.links[0].onclick = linkSomePage_onclick;
</script>
```

如前所述，document.links[0]返回对应于网页上第一个链接的 a 对象，即 linkSomePage 超链接。将该对象的 onclick 属性设置引用前面定义的函数——这样就建立了对象的事件处理程序与函数的连接。注意，函数名之后没有添加括号。只要用户单击链接，就会执行函数。

将代码连接到事件的第一种方法比较简单，为什么还要使用第二种方法呢？

使用第二种方法最常见的情况是，要捕获某个事件，但没有 HTML 元素把事件处理程序编写为它的一个属性。另外，如果希望代码关联到要动态改变的事件处理程序上，第二种方法也比较有效。

试一试　在页面加载时显示一幅随机图片

下面的例子将连接超链接的 click 事件，以随机地改变页面上加载的图片。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 5</title>
    <script type="text/javascript">
    var myImages = new Array("usa.gif","canada.gif","jamaica.gif", "mexico.gif");

    function changeImg(that)
    {
        var newImgNumber = Math.round(Math.random() * 3);

        while (that.src.indexOf(myImages[newImgNumber]) != -1)
        {
            newImgNumber = Math.round(Math.random() * 3);
        }

        that.src = myImages[newImgNumber];

        return false;
    }
    </script>
</head>
<body>
    <img name="img0" src="usa.gif" border="0"
        onclick="return changeImg(this)" />

    <img name="img1" src="mexico.gif" border="0"
        onclick="return changeImg(this)" />
</body>
</html>
```

将该页面保存为 ch6_examp6.htm。本例仍需要 4 个图像文件，可以自己创建，也可以从本书的代码下载中获得。

在浏览器中加载该页面，页面如图 6-3 所示：

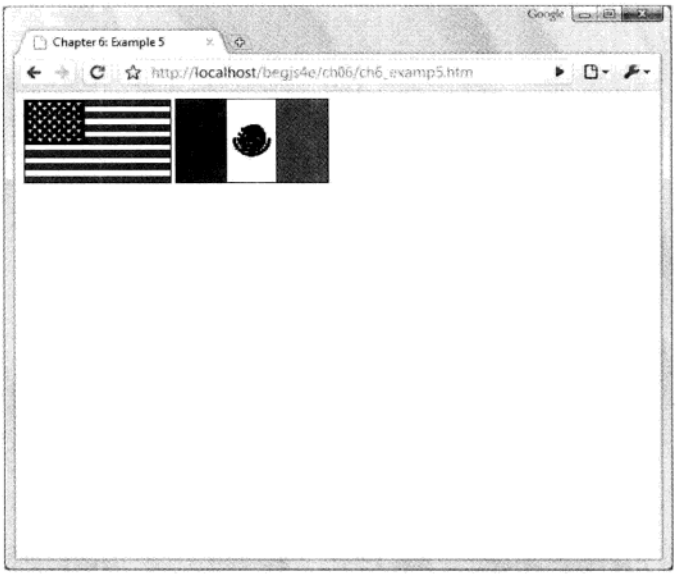

图　6-3

如果单击一幅图片，它就会变成另一幅随机选取的图片。

在页面顶部，脚本块的第一行定义了一个页面级变量 myImages，它是一个包含图片源列表的数组。

```
var myImages = new Array("usa.gif","canada.gif","jamaica.gif","mexico.gif");
```

接着定义了 changeImg()函数，该函数会连接到页面上定义的<img/>元素的 onclick 事件处理程序上。这里给两个图片的 onclick 事件处理器使用了同一个函数，实际上，可以将任意多个事件处理程序连接到同一个函数。这个函数接受一个参数 that，称之为 that，是因为要把 this 关键字传送给该函数，以便即刻访问所单击的 img 对象。可以给这个参数指定其他名称，但在该参数引用 this 时，大多数开发人员都使用 that 作为其名称。

函数的第一行把变量 newImgNumber 设置为一个 0～3 的随机整数：

```
function changeImg(that)
{
    var newImgNumber = Math.round(Math.random() * 3);
```

Math.random()方法提供了一个 0～1 之间的随机数，再乘以 3，就得到一个 0～3 之间的随机数。然后，使用 Math.round()把它舍入为最近的整数(即 0、1、2 或 3)。这个整数用作从 myImages 数组中所选图片 src 的索引。

接下来是一个 while 循环，其作用是确保不会选择当前的图片。如果 myImages [newImg Number]包含的字符串是当前图片的 src 属性，则表示选中的图片与当前的图片相同，需要

获取另一个随机数。继续循环，直到选出一幅新图片为止，此时 myImages [newImgNumber] 不是现有的 src，indexOf()方法返回 - 1，从而结束循环。

```
while (imgClicked.src.indexOf(myImages[newImgNumber]) != -1)
{
    newImgNumber = Math.round(Math.random() * 3);
}
```

接着将 img 对象的 src 属性设置为 myImages 数组包含的新值，并返回 false，以阻止链接导航到另一个页面的尝试。注意，这里的 HTML 链接仅提供了一种捕获 onclick 事件处理程序的方式。

```
    that.src = myImages[newImgNumber];

    return false;
}
```

现在，把第一个<img/>元素的 onclick 事件连接到 changeImg()函数上。

```
<img name=img0 src="usa.gif" border="0" onclick="return changeImg(this)">
```

接着连接第二个<img/>元素：

```
<img name="img1" src="mexico.gif" border="0" onclick="return changeImg(this )">
```

向 changeImg()函数传入 this，使该函数能直接访问这个<img/>元素的对应对象。给 HTML 元素的属性事件处理程序传入 this 时，该元素的对应对象就会传入函数。这是在 JavaScript 中访问元素的对象的一种简洁方式。

事件对 Web 开发人员非常重要——可以肯定，许多代码都是在处理事件。第 12 章将再次介绍事件。下面讨论另一个主题：用户的浏览器。

6.2.3 确定用户的浏览器

Internet 上充斥着各种操作系统和各种版本的浏览器，每种浏览器都有自己的 BOM 和独有的特性。因此，保证页面在所有浏览器上正常运行是非常重要的，至少可以正常退出，例如显示一个消息，建议用户升级浏览器。

尽管使用跨浏览器兼容的代码大有帮助，但是有时需要添加仅一种浏览器支持的额外功能。因此，需要编写脚本来确定用户的浏览器，然后执行与该浏览器兼容的脚本。

可以通过两种方法检查浏览器的详细信息。第一种方法是检查代码中使用的对象和属性是否可实际用于用户的浏览器。例如，假定代码需要使用 IE 中 document 对象的 all 集合。如果编写如下代码：

```
if (window.all)
{
    // our code using the document.all collection
}
```

如果该属性返回一个有效值，if 语句的条件就等于 true。如果浏览器不支持该属性，其值就是 undefined，if 语句就是 false。

要检查浏览器是否支持某个方法，可以使用如下代码：

```
if (document.getElementById)
{
   // code using document.getElementById()
}
else
{
   // code for browsers that do not have that method
}
```

“测试”某个方法是否存在的方式与测试属性相同。注意，不要在方法名之后包括开闭括号，即使这个方法有许多参数也是如此。例如 getElementById 方法有一个参数，参见第 12 章。

注意：

在 JavaScript 语言中，函数(和方法)实际上是对象。本书不讨论这个高级主题，但 Nicholas Zakas 著的 *Professional JavaScript for Web Developers* 一书(Wrox 出版)深入探讨了这个主题。

下面的例子演示了如何使用对象检测技术，来确保为不同的浏览器执行正确的代码。这个例子并不是无懈可击的，但非常有用。

试一试　检查浏览器支持的属性

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 6</title>
</head>
<body>

<script type="text/javascript">
var browser = "Unknown";
var version = "0";

if (window.opera)
{
    browser = "Opera";
    version = "5+";

    if (window.opera.setPreference)
    {
        version = "9";
    }
}
```

```
else if (document.all)
{
    browser = "Internet Explorer";
    version = "6-";+

    if (window.XMLHttpRequest)
    {
        version = "7+";
    }
}
else if (window.sidebar)
{
    browser = "Firefox";
    version = "1+";
}

document.write(browser + " " + version);
</script>

</body>
</html>
```

将这个示例保存为 ch6_examp6.htm。

页面检查浏览器支持哪些 BOM 和 JavaScript 属性，以此来初步判断浏览器的类型。因此第一行检查浏览器的 windows 对象是否有 opera 属性。

```
if (window.opera)
{
    browser = "Opera";
    version = "5+";

    if (window.opera.setPreference)
    {
        version = "9";
    }
}
```

由于只有 Opera 支持 opera 属性，所以可以确信这至少是 Opera 5 版本。内层的 if 语句检查 window.opera 对象是否支持 setPreference 方法；只有 Opera 9 版本支持该方法。

接着测试 document 对象的 all 属性，IE4+和 Opera 7+支持这个属性。

```
else if (document.all)
{
    browser = "Internet Explorer";
    version = "6-";+

    if (window.XMLHttpRequest)
    {
        version = "7+";
    }
}
```

由于已经测试了 Opera，就不必担心这个脚本的 false 结果了。要确定浏览器是否是 IE 7+，可检查 window 对象的 XMLHttpRequest 属性。支持该属性的 Microsoft 浏览器只有 IE 7+(IE6 及更早版本支持 XMLHttpRequest 的功能，但不支持 window.XMLHttpRequest 属性)。

最后检查 Firefox 支持的 sidebar 属性，该属性可处理边栏的工具。

```
// Firefox
else if (window.sidebar)
{
    browser = "Firefox";
    version = "1+";
}
```

最后一行输出浏览器对象检测的结果：

```
document.write(browser + " " + version);
```

这个例子演示了如何使用对象检测技术，来确定某浏览器是否支持某特性。本例并没有实际使用各种特性，只是简单地演示了如何检查浏览器特定的对象。在编写自己的代码时，要确保复查用于访问网站的所有浏览器是否支持代码使用的某个特性。如果某种浏览器不支持该特性，就应测试该特性，并为不支持它的浏览器编写替代代码。

第 12 章将介绍高级的对象检测技术。

1. 浏览器不支持脚本

有时用户关闭了浏览器中的 JavaScript，或者使用不支持 JavaScript 的浏览器。虽然在目前这属于罕见情况，但是为了兼顾这种情形，可以使用<noscript/>元素。仅在浏览器不支持 JavaScript 或者禁用了 JavaScript 时，才显示位于该标记内的任何 HTML。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>No Script</title>
</head>
<body>
<noscript>
    This website requires JavaScript to be enabled.
</noscript>
</body>
<html>
```

2. 使用 navigator 对象检查浏览器

第二种检查浏览器信息的方法是使用 window 对象的 navigator 对象属性。通常使用 navigator 对象的 appName 和 userAgent 属性。这种方法的主要缺陷是，少数浏览器可能宣

称自己是某主流浏览器的某一特定版本，但实际上不支持该主流浏览器的所有 JavaScript 或 BOM 对象、属性或方法。因此，这种“浏览器嗅探”技术并不是人们的首选，不推荐用这种方法检测浏览器的兼容性。当所有其他方法无效时，才采用这种方法。例如两种不同的浏览器支持相同的对象和属性，但实现方式不同时，对象检测技术对此将无能为力，此时只能使用 navigator 对象。

appName 属性将返回浏览器的模型，如对 IE 返回 Microsoft Internet Explorer，对 Opera 返回 Opera，对 Firefox、Safari 和 Chrome 返回 Netscape。

userAgent 属性返回一个包含多段信息的字符串，如浏览器的版本、操作系统和浏览器模型。但是，这个属性的返回值因浏览器而异，因此使用它时要倍加小心。例如，浏览器的版本嵌入在该字符串的不同位置。

试一试　不同浏览器的检测和处理

下面的例子将创建一个页面，并使用前面提到的属性来检测客户浏览器的类型及版本。根据客户浏览器的不同，页面可以执行不同的操作。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example 7</title>
    <script type="text/javaScript">

    function getBrowserName()
    {
        var lsBrowser = navigator.userAgent;
        if (lsBrowser.indexOf("MSIE") >= 0)
        {
            lsBrowser = "MSIE";
        }
        else if (lsBrowser.indexOf("Firefox") >= 0)
        {
            lsBrowser = "Firefox";
        }
        else if (lsBrowser.indexOf("Chrome") >= 0)
        {
            lsBrowser = "Chrome";
        }
        else if (lsBrowser.indexOf("Safari") >= 0)
        {
            lsBrowser = "Safari";
        }
        else if (lsBrowser.indexOf("Opera") >= 0)
        {
            lsBrowser = "Opera";
        }
        else
```

```
        {
            lsBrowser = "UNKNOWN";
        }
        return lsBrowser;
    }

    function getBrowserVersion()
    {
        var findIndex;
        var browserVersion = 0;
        var browser = getBrowserName();

        browserVersion = navigator.userAgent;
        findIndex = browserVersion.indexOf(browser) + browser.length + 1;
        browserVersion = parseFloat(browserVersion.substring(findIndex,
                findIndex + 3));

        return browserVersion;
    }
    </script>
</head>
<body>
<script type="text/javaScript">

var browserName = getBrowserName();
var browserVersion = getBrowserVersion();

if (browserName == "MSIE")
{
    if (browserVersion < 7)
    {
        document.write("Your version of Internet Explorer is too old");
    }
    else
    {
        document.write("Your version of Internet Explorer is fully supported");
    }
}
    else if (browserName == "Firefox")
{
    document.write("Firefox is fully supported");
}
else if (browserName == "Safari")
{
    document.write("Safari is fully supported");
}
    else if (browserName == "Chrome")
{
    document.write("Chrome is fully supported");
}
else if (browserName == "Opera")
{
```

```
        document.write("Opera is fully supported");
    }
    else
    {
        document.write("<h2>Sorry this browser version is not supported</h2>");
    }
    </script>
    <noscript>
        <h2>This website requires a browser supporting scripting</h2>
    </noscript>
    </body>
    </html>
```

将这个脚本保存为 ch6_examp7.htm。

如果浏览器是 Firefox、Internet Explorer 7+、Safari、Chrome 或 Opera，则脚本会显示一条消息，告诉用户该页面支持当前的浏览器。如果是早期版本的 IE，则消息显示，页面不支持该浏览器的版本。

如果浏览器不属于上述几种类型，则消息显示，页面不支持当前的浏览器。这种方法不够友好，因此实际上可以提供一个无脚本的简单页面，并尽量保留无须 JavaScript 脚本处理的功能。

如果浏览器不支持 JavaScript，或者用户禁用了 JavaScript 支持，则消息显示，该网站需要 JavaScript 的支持。

页面顶部的脚本块定义了两个重要的函数。getBrowserName()函数确定浏览器的名称，getBrowserVersion()函数确定浏览器版本。

浏览器检测代码的关键在于 navigator.userAgent 属性的返回值。下面是几个当前浏览器的 userAgent 字符串示例：

(1) Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 1.0.3705; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

(2) Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10 (.NET CLR 3.5.30729)

(3) Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Safari/530.17

(4) Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/531.0 (KHTML, like Gecko) Chrome/3.0.183.1 Safari/531.0

(5) Opera/9.63 (Windows NT 5.1; U; en) Presto/2.1.1

这里给出了每个 userAgent 字符串的编号。仔细观察每一行，很容易猜出它们对应的浏览器：

(1) Microsoft Internet Explorer 8

(2) Firefox 3.0.10

(3) Safari 4.0

(4) Chrome 3.0.183.1

(5) Opera 9.63

根据这些信息，先看看第一个函数 getBrowserName()。首先获得浏览器的名称，把 navigator.userAgent 的值保存在变量 lsBrowser 中。lsBrowser 也用于存储函数的返回值。

```
function getBrowserName()
{
    var lsBrowser = navigator.userAgent;
```

这个属性返回的字符串通常很长，有时其功能也略有区别。但是，通过检测是否存在某些关键字，如 MSIE 或者 Firefox，就可以确定浏览器的名称。首先分析下面的代码行：

```
if (lsBrowser.indexOf("MSIE") >= 0)
{
    lsBrowser = "MSIE";
}
```

这段代码查找 lsBrowser 字符串中的 MSIE。如果这个子字符串的 indexOf 值大于等于 0，则表示找到了 MSIE，因此将返回值设置为 MSIE。

接下来的 else if 子句完成相同的功能，只不过这次查找 Firefox。

```
else if (lsBrowser.indexOf("Firefox") >= 0)
{
    lsBrowser = "Firefox ";
}
```

其他三个 if 语句的原理与此相同，分别查找 Chrome、Safari 和 Opera。如果还需要查找其他类型的浏览器，应在这里增加相应的 if 语句。只需观察 navigator.userAgent 返回的字符串，查找浏览器名称或者唯一标识浏览器的子串。

如果任何 if 语句都不匹配，则返回 UNKNOWN 作为浏览器的名称。

```
else
{
    lsBrowser = "UNKNOWN";
}
```

接着把 lsBrowser 的值返回给调用代码：

```
    return lsBrowser;
}
```

下面看看最后一个函数 getBrowserVersion()。

在 userAgent 字符串中，浏览器的版本信息通常紧跟在浏览器名称之后。因此，在这个函数中，首先要确定当前使用的浏览器。声明 browser 变量，使用刚才编写的 getBrowser-Name()，把它初始化为浏览器的名称。

```
function getBrowserVersion()
{
    var findIndex;
    var browserVersion = 0;
```

```
var browser = getBrowserName();
```

如果浏览器是 MSIE(Internet Explorer)，则需要再次使用 userAgent 属性。在 IE 下，userAgent 属性总是包含 MSIE，后跟浏览器的版本号。因此，只需查找关键字 MSIE，然后获取其后的版本号。

将 findIndex 设置为浏览器名称的字符位置加上该名称的长度，再加 1。这样做可确保获得该名称后的字符，在名称后面的空格或/字符之后就是版本号。browserVersion 设置为版本号的浮点值，该值是用 substring()方法获得的。这个方法选择的字符从 findIndex 开始，包含版本号，结束位置为 findIndex+3。这样就只选择了表示版本号的 3 个字符。

```
browserVersion = navigator.userAgent;
findIndex = browserVersion.indexOf(browser) + browser.length + 1;
browserVersion = parseFloat(browserVersion.substring(findIndex,findIndex + 3));
```

看看前面的 userAgent 字符串，会发现 IE8 的 userAgent 如下所示：

```
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; .NET CLR 2.0.40607)
```

所以 findIndex 设置为浏览器名称后的数字 8 的字符索引，browserVersion 设置为从数字 8 开始的 3 个字符，即浏览器的版本号 8.0。

在该函数的最后，把 browserVersion 返回给调用代码，如下所示：

```
   return browserVersion;
}
```

前面是支持函数的定义，但如何使用它们呢？下面的代码在加载页面时执行，这些代码获得了浏览器的名称和版本信息，并使用这些信息过滤用户当前运行的浏览器。

```
var browserName = getBrowserName();
var browserVersion = getBrowserVersion();

if (browserName == "MSIE")
{
    if (browserVersion < 7)
    {
        document.write("Your version of Internet Explorer is too old");
    }
    else
    {
        document.write("Your version of Internet Explorer is fully supported");
    }
}
```

在上面的代码中，第一个 if 语句检查用户是否有 IE。如果是，则接着检查其版本号是否低于 7。如果是，则提示用户浏览器的版本太低。如果 IE 的版本是 7+，则告诉用户，其浏览器会得到全面支持。

对 Firefox、Chrome、Safari 和 Opera 执行相同的操作。本例没有检查这些浏览器的版本，如果需要，也可以予以检查。

```
else if (browserName == "Firefox")
{
    document.write("Firefox is fully supported");
}
else if (browserName == "Safari")
{
    document.write("Safari is fully supported");
}
else if (browserName == "Chrome")
{
    document.write("Chrome is fully supported");
}
else if (browserName == "Opera")
{
    document.write("Opera is fully supported");
}
else
{
    document.write("<h2>Sorry this browser version is not supported</h2>");
}
```

if 语句的最后部分是 else 语句，它涵盖其他所有的浏览器，告诉用户页面不支持当前的浏览器。

最后，给早期的浏览器(或选择禁用 JavaScript 的用户)使用了<noscript/>元素，并显示一条消息，提示用户其浏览器不支持 JavaScript。

```
<noscript>
    <h2>This website requires a browser supporting scripting</h2>
</noscript>
```

如前所述，尽管目前这段脚本运行正常，但是浏览器可能会改变 userAgent 字符串，因此需要更新函数，以跟踪这个变化。另外，一些浏览器可能会冒充其他浏览器，却不完全支持所冒充浏览器的功能，这也可能使上面的代码产生错误。

因此，应该尽量使用本章前面介绍的对象检测方法。

6.3　小结

本章介绍了不少内容，为进一步学习更有效、更高级的内容打好了基础，例如表单、用户输入、文本和日期处理等。本章的主要内容如下：

- 浏览器是 JavaScript 运行的环境，也是本章的重点。与 JavaScript 有内置对象类似，Web 浏览器也有内置对象。Web 浏览器中的这些对象以层次方式组织起来，称为浏览器对象模型(Browser Object Model，BOM)，它其实是浏览器对象的映射。使用 BOM，就可以查看浏览器中的每个对象及其属性、方法和事件。

- 第一个主要对象是 window，它位于 BOM 层次结构的顶端。window 对象包含一些重要的子对象，如 location 对象、navigator 对象、history 对象、screen 对象以及 document 对象。
- location 对象包含了当前页面的位置信息，如文件名、包含页面的服务器和所使用的协议。这些信息都是 location 对象的属性，某些属性是只读的，而另一些属性是可读写的，如 href 属性不仅可以用来找到页面的位置，还可以修改它，从而将页面导航到一个新位置。
- history 对象记录了自用户打开浏览器以来访问的所有页面。有时页面不会记录下来(如使用 location 对象的 replace()方法来导航时，就不会记录加载的页面)。可以在浏览器的历史栈中向前向后移动，查看用户访问了哪些页面。
- navigator 对象表示浏览器自身，它包含了浏览器的类型、版本号和用户的操作系统等有用信息。这些信息可用于编写页面，处理类型不同甚至不兼容的浏览器。
- screen 对象包含用户计算机显示能力的信息。
- document 对象是最重要的对象之一。它是页面的对象表示，包含了页面上的所有元素，这些元素也用对象表示。不同浏览器在这方面的差异特别突出，但浏览器之间的相似性允许我们编写跨浏览器的代码。
- document 对象包含三个实际上是集合的属性，即 links、images 和 forms 集合。它们分别包含页面上由<a/>、<img/>和<form/>元素创建的所有对象，也可以通过这些集合访问这些元素。
- images 集合包含了页面上用于每个<img/>元素的 img 对象。即使在页面加载后，仍可以修改图片的属性。例如，可以在单击某幅图片时改变该图片。使用 images 集合的规则同样也适用于 links 集合。
- BOM 对象不但有属性和方法，还有事件。在 JavaScript 中通过事件处理程序来处理这些事件，事件处理程序会连接到事件发生时用执行的代码上。可使用的事件取决于当前处理的对象。
- 只要给指定对象的元素添加一个属性，就可以把编写好的函数连接到事件处理程序上。该属性的名称是要捕获的事件处理程序名称，其值是要连接的函数名称。
- 在某些情况下，例如 document 对象，需要采用第二种方法来连接事件处理程序与相应代码。即把对象的事件处理程序属性设置为要执行的函数名，其效果与把事件处理程序用作属性相同。
- 在某些情况下，事件函数的返回值可以用来取消与该事件关联的操作。例如，要禁止被单击的链接导航到新页面上，可以从事件处理程序的代码中返回 false。
- 最后介绍了如何检查用户的浏览器类型，以确保用户仅能浏览与其浏览器兼容的页面或部分页面。navigator 对象提供了浏览器检测的详细信息，尤其是 appName 属性和 userAgent 属性。还可以检查特定的 BOM 属性，以确定它们是否获得支持，之后再使用它们。如果浏览器不支持代码正常运行所需的特定属性，则可以编写替代代码，或者提示用户升级浏览器。

这就是本章介绍的全部内容。下一章将介绍令人更加激动的表单脚本技术，通过这种技术，可以在页面上添加各种控件，以便帮助收集用户的信息。

6.4　习题

在附录 A 中可以找到本章习题的参考答案。

习题 1：

创建一个包含多个链接的页面，并编写代码，触发 window 的 load 事件，以显示页面上每个链接的 href (提示：事件处理程序以 on 开头)。

习题 2：

创建两个页面 ieonly.htm 和 notieonly.htm。每个页面都包含一个标题，以说明当前加载的是哪个页面。例如：

```
<H2>Welcome to the Internet Explorer only page</H2>
```

使用检查浏览器类型的函数，并将该函数连接到 window 对象的 onload 事件处理程序，以检测用户浏览器的类型。如果浏览器加载的不是对应的页面，则重定向到另一个页面。

习题 3：

在页面上用<img/>元素插入一幅图片。当鼠标指针移到图片上时，该图片应切换成另一幅图片；当鼠标指针从图片上移开时，再切换回原来的图片(提示：使用 mouseover 和 mouseout 事件)。

第 7 章

HTML 表单：与用户交互

如果网页不能与用户交互，或者不能获取用户的信息，如文本、数字或日期，这样的网页就无聊至极！幸运的是，JavaScript 提供了这些功能。我们可以在网页上使用这些信息，或者将它们传送给 Web 服务器，在服务器中对数据进行处理，如有必要，还可以将数据保存到数据库中。本章重点介绍如何在 Web 浏览器中使用这些信息，这称为“客户端处理”。

我们习惯于使用各种用户界面元素。例如，Windows 操作系统有大量的标准元素，如按钮、列表、下拉列表框、单选按钮和复选框。任何图形用户界面(Graphical User Interface，GUI)操作系统都是这样，包括 Mac、Unix 或 Linux 系统。这些元素是用户与应用程序交互的方式。在网页中也可以包含许多这类元素，而且非常简单。在页面上使用了某个元素，如按钮时，还可以将代码连接到该元素的事件。例如，单击按钮时，可以触发一个 JavaScript 函数。

注意，本章讨论的元素都是 HTML 中常见的元素，不包括 ActiveX 元素、Java Applet 或插件。这些内容请参见第 13 章。

所有用于交互操作的 HTML 元素都应放在 HTML 表单(form)中。下面先讨论 HTML 表单，以及在 JavaScript 中如何与表单交互。

7.1 HTML 表单

通常情况下，表单提供了一种把 HTML 交互元素组合起来的方式。例如，表单包含的元素允许用户输入在网站上注册的数据，另一个表单包含的元素允许用户查询汽车的保险报价。一个页面可以包含多个独立的表单。我们不必关心页面是否包含多个表单，除非要向 Web 服务器提交信息——注意，一次只能提交页面中一个表单的信息。

要创建表单，可使用<form>和</form>标记来声明表单的开始和结束位置。<form/>元素包含很多属性，如 action 属性可确定把表单提交到什么地方；method 属性可确定如何提交信息；target 属性可确定将表单的响应加载到哪个框架上。

客户端脚本编程通常不需要给服务器提交信息，所以不需要上述属性。后面的章节介

绍服务器页面的编程时，才会使用这些属性。现在，在<form/>元素中唯一要设置的属性是 name，以便引用表单。

要创建空表单，需要的标记如下所示：

```
<form name="myForm">
</form>
```

这些标记创建了一个 Form 对象，该对象可用来访问这个表单。访问这个对象有两种方法：

第一，可以直接使用名称(本例是 document.myForm)访问该对象。也可以通过 document 对象的 forms 集合属性来访问这个对象。注意，上一章讨论了 document 对象的 images 集合，可以像使用其他数组那样操作该集合。这也适用于 forms 集合，只不过集合中的每个元素不是包含 IMG 对象，而是包含 Form 对象。例如，如果这是页面上的第一个表单，则可以使用 document.forms[0]来引用它。

<form/>元素的许多属性都可以作为 Form 对象的属性访问。特别是，Form 对象的 name 属性镜像<form/>元素的 name 属性。

试一试　Forms 集合

下面是一个使用 forms 集合的例子，其页面包含三个表单。使用 forms 集合依次访问每个 Form 对象，并在消息框中显示其 name 属性值。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 1</title>
    <script type="text/javascript">
    function window_onload()
    {
        var numberForms = document.forms.length;
        var formIndex;
        for (formIndex = 0; formIndex < numberForms; formIndex++)
        {
            alert(document.forms[formIndex].name);
        }
    }
    </script>
</head>
<body onload="window_onload()">
    <form action=" "name="form1">
        <p>
            This is inside form1.
        </p>
    </form>
    <form action=" "name="form2">
        <p>
            This is inside form2
```

```
        </p>
    </form>
    <form action=" "name="form3">
        <p>
            This is inside form3
        </p>
    </form>
</body>
</html>
```

保存为 ch7_examp1.htm。在浏览器中加载该页面时，将看到三个警告框，每个警告框显示了一个表单的名称。

在页面体中定义了三个表单。每个表单都指定了名称，并包含了一段文本。

在<body/>元素的定义中，将 window_onload()函数连接到 window 对象的 onload 事件处理程序上。

```
<body onload="window_onload()">
```

这意味着，在加载页面时，将调用 window_onload()函数。

在页面顶部的脚本块中定义 window_onload()函数。在这个函数中遍历 forms 集合，与其他任意 JavaScript 数组类似，forms 集合有 length 属性，可用于决定需要循环的次数。实际上，表单的数量是已知的，因此可以直接写出该次数。但是，这个示例使用了 length 属性，是因为这样便于给集合添加元素，而无须修改函数。推广这种编码方法是应遵循的一个最佳实践。

该函数首先确定 forms 数组中 Form 对象的数量，并将其保存在变量 numberForms 中。

```
function window_onload()
{
    var numberForms = document.forms.length;
```

接着定义了一个变量 formIndex，在 for 循环中使用。

```
    var formIndex;
    for (formIndex = 0; formIndex < numberForms; formIndex++)
    {
        alert(document.forms[formIndex].name);
    }
}
```

数组的索引从 0 开始，因此循环需要应从 0 到 numberForms - 1。将变量 formIndex 初始化为 0，把 for 循环的条件设置为 formIndex < numberForms。

在 for 循环的代码中，把表单的索引 formIndex 传给 document.forms[]，以获得 forms 集合中该索引对应的 Form 对象。要访问 Form 对象的 name 属性，可以加上句点和属性名 name。

7.2　Form 对象的其他属性和方法

HTML 表单中的常用控件(稍后详细介绍)也有对应的对象。访问这些对象的一种方式是使用 Form 对象的 elements 属性，elements 属性也是集合，它包含表单中对应于 HTML 交互元素的所有对象，除了很少用到的<input type="image"/>元素之外。如后面所述，这个属性非常适用于遍历表单中的每个元素。例如，在提交表单之前，可以遍历每个元素，以便检查它们是否包含有效的数据。

Form 对象的 elements 属性是一个集合，因此它也有 length 属性，用于确定表单中的元素数量。Form 对象也有 length 属性，它也用以提供表单中的元素个数。这两个属性是等效的，使用哪一个都可以。但 document.myForm.length 比 document.myForm.elements.length 短，输入的字符少，代码也更简短。

将表单中的数据提交给服务器时，通常会用到 Submit 按钮(稍后介绍)。Form 对象也有 submit()方法，它的作用与 Submit 按钮类似。

注意：

使用 submit()方法提交表单时，不会触发 Form 对象的 submit 事件，也不会调用 onsubmit 事件处理程序。

第 6 章介绍过，从事件处理程序代码传送回的返回值如何决定事件的正常操作是继续执行还是取消。例如，如果超链接的 onclick 事件处理程序返回 false，就取消该链接的导航操作。这个原则也适用于 Form 对象的 onsubmit 事件处理程序。用户提交表单时，将触发 onsubmit 事件处理程序。如果给这个事件处理程序返回 true，就提交表单；如果返回 false，则取消提交操作。因此 onsubmit 事件处理程序的代码非常适用于进行表单验证——即检查用户在表单中输入的数据是否有效。例如，假如要求用户输入年龄，但用户输入了 mindyour own business，就可以发现这是文本，而不是有效的数字，并禁止用户继续操作。

另一个常见的控件是本章后面介绍的 Reset 按钮。Form 对象也有 reset()方法，该方法将清空表单，或者还原默认值(如果有)。

创建空表单似乎没有什么意义，下面讨论在表单中提供交互功能的 HTML 元素。

7.2.1　表单中的 HTML 元素

<form/>元素中大约有 10 个常见元素。图 7-1、图 7-2、图 7-3 和图 7-4 按类型显示了最常见的元素。图中给出了每个类型名称，并在圆括号中给出了创建它所需的 HTML，注意这不是完整的 HTML，而只是其中的一部分。

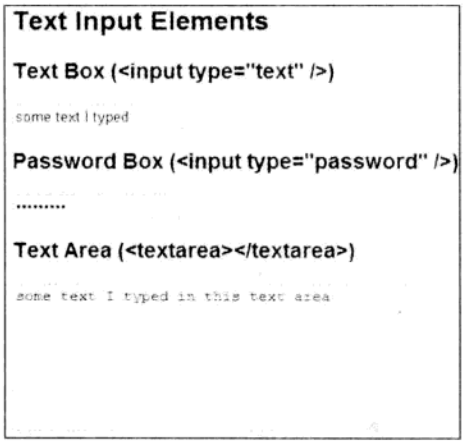

图 7-1

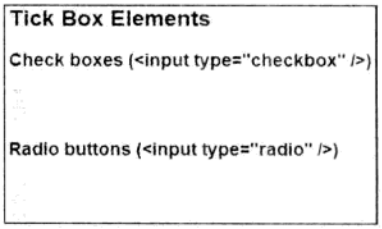

图 7-2

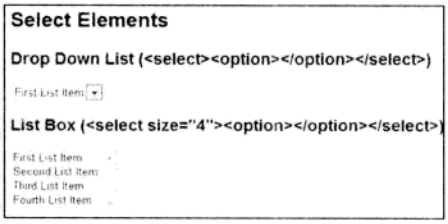

图 7-3

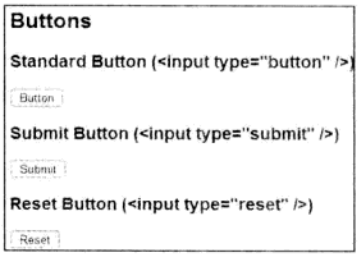

图 7-4

可以看出，大部分表单元素都通过<input/>元素创建。<input/>元素的一个属性是 type，该属性决定了<input/>元素创建哪种表单元素。这个属性的值可以是 button(创建按钮)和 text(创建文本框)。

网页中的每个表单元素都可用作对象。与其他对象类似，每个元素的对象都有各自的属性、方法和事件。接下来将依次介绍每个表单元素，以及如何使用它的属性、方法和事件。但在此之前，先讨论表单元素对象共有的属性和方法。

7.2.2 共有的属性和方法

大多数元素都是通过<input/>元素创建的，所以所有的表单元素都共享几个属性和方法。下面介绍其中的几个。

1. name 属性

所有表单元素的对象都有 name 属性。在脚本中，可以使用这个属性的值来引用元素。另外，如果把表单中的信息提交给服务器，元素的 name 属性将与表单元素的值一起发送，以告知服务器该值与哪个元素相关。

2. value 属性

大多数表单元素对象都有 value 属性，它返回元素的值。例如，文本框的 value 属性返回用户在文本框中输入的文本。另外，设置 value 属性的值可以把文本放在文本框中。但是，每个元素的 value 属性的用法都是特定的，所以应根据每个元素确定该属性的含义。

3. form 属性

所有表单元素对象都有 form 属性，它返回包含当前元素的 Form 对象。用通用的例程检查表单中数据的有效性时，就可以使用这个属性。例如，用户单击 Submit 按钮时，可以把 Submit 按钮的 form 属性引用的 Form 对象传给数据检查程序，数据检查程序使用 Form 对象依次遍历表单中的每个元素，检查元素中数据的有效性。如果一个页面有多个表单，或者将通用的数据检查程序剪切并粘贴到不同的页面上，这就很方便——这样，不必事先知道表单的名称。

4. type 属性

有时，需要知道当前处理的元素类型，特别是使用 elements 集合属性来遍历表单中的元素时，就可以通过每个元素对象都有的 type 属性，获得元素的类型信息。type 属性返回元素的类型(例如 button 或 text)。

5. focus()和 blur()方法

所有表单元素对象都有 focus()和 blur()方法。焦点(focus)是一个新概念。如果一个元素获得焦点，则用户按下的任何键都直接传递给该元素。例如，如果文本框获得了焦点，就在该文本框中输入值。另外，如果按钮获得了焦点，则按回车键将触发按钮的 onclick 事件处理程序的代码，就像用户用鼠标单击了该按钮一样。

要设置哪个元素当前有焦点，用户可以单击该元素，或者使用 Tab 键来选择它。而程序员也可以使用表单元素对象的 focus()方法，来确定哪个元素有焦点。例如，假如有一个文本框供用户输入年龄，但用户输入了一个无效的值，如输入了字母而不是数字，则可以提示用户输入无效，并返回该文本框，以便用户修改错误。

Blur(最好称为失去焦点(lost focus))与 focus 相反。如果要把用户的焦点从某个表单元素上移出，则可以使用 blur()方法。在某个表单元素上使用该方法时，该方法通常会把焦点转移到包含该表单的页面上。

除了 focus()和 blur()方法之外，所有的表单元素对象还有 onfocus 和 onblur 事件处理程序。根据用户的操作或 focus()和 blur()方法，当元素获得或失去焦点时，将触发这两个事件处理程序。onblur 事件处理程序非常适于检查刚失去焦点的元素数据的有效性。如果数据无效，就可以将焦点设置回该元素，并提示用户输入数据错误的原因。

注意：

submit()方法的执行方式与 focus()和 blur()不同，因为 submit()方法不会触发 submit 事件和 onsubmit 事件处理程序。

在 onfocus 或 onblur 事件处理程序代码中使用 focus()和 blur()方法时，要格外小心，以免出现死循环。例如考虑两个元素，它们的 onfocus 事件把焦点传送给另一个元素。如果一个元素获得了焦点，其 onfocus 事件就把焦点传送给第二个元素，第二个元素的 onfocus 事件又把焦点传送回第一个元素，唯一的退出方式是关闭浏览器。用户可不希望出现这种情况。

如果某字段或其他字段依赖于用户输入，则使用 focus()和 blur()方法把焦点设置回有问题的字段时也要小心。例如，假定有两个文本框，一个文本框要求用户输入城市，另一个文本框要求用户输入州。再假定要检查州文本框的输入，以确保指定的城市在该州中。如果州不包含该城市，就把焦点设置回州文本框，以便用户修改州名。但是，如果用户输入的城市是错误的，而州是正确的，就不能返回城市文本框，改正错误了。

7.2.3 按钮元素

下面从标准的按钮元素开始，因为按钮最常用，且相当简单。创建按钮的 HTML 元素是<input/>。例如，要创建显示 Click Me 的按钮 myButton，<input/>元素应如下所示：

```
<input type="button" name="myButton" value="Click Me" />
```

type 属性设置为 button，value 属性设置为要在按钮上显示的文本。可以关闭 value 属性，但这样会得到一个无文本的按钮，用户只能猜测该按钮的用途。

这个元素创建了一个关联的 Button 对象，在本例中是 myButton 对象。这个对象拥有前面介绍的所有公共属性和方法，包括 value 属性。这个属性允许使用 JavaScript 改变按钮上的文本，但常常不需要这么做。真正需要关注的是按钮的 click 事件。

连接按钮的 onclick 事件处理程序的方式，与连接其他 HTML 元素(如<a/>)的 onclick 事件一样。只需定义一个要在单击按钮时执行的函数(如 button_onclick())，再把 onclick 事件处理程序添加为<input/>元素的一个属性，如下所示：

```
<input type="button" onclick="button_onclick()">
```

试一试　计算按钮的单击次数

下面的示例使用前面介绍的方法，记录按钮的单击频率。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 2</title>
    <script type="text/javascript">
    var numberOfClicks = 0;
    function myButton_onclick()
    {
        numberOfClicks++;
        document.form1.myButton.value = "Button clicked " +
            numberOfClicks + "times";
```

```
        }
        </script>
    </head>
    <body>
        <form action=" " name="form1">
            <input type="button" name="myButton" value="Button clicked 0 times"
                onclick="myButton_onclick()" />
        </form>
    </body>
    </html>
```

将这个页面保存为 ch7_examp2.htm。如果在浏览器中加载这个页面，将看到按钮上的文本显示为 Button clicked 0 times。如果重复单击该按钮，按钮上的文本就会显示按钮的单击次数。

页面顶部的脚本块首先定义了一个可在页面的任何地方访问的全局变量numberOf Clicks，在这个变量中记录按钮的单击次数，并使用该信息更新按钮上的文本。

脚本块中的另一段代码是 myButton_onclick()函数的定义。这个函数连接到页面体中<input/>元素的 onclick 事件处理程序上。这个<input/>元素是一个按钮，它包含在 form1 表单中。

```
<form action=" " name="form1">
    <input type="button" name="myButton" value="Button clicked 0 times"
        onclick="myButton_onclick()" />
</form>
```

下面详细分析 myButton_onclick()函数。首先该函数给 numberOfClicks 变量的值加 1。

```
function myButton_onclick()
{
    numberOfClicks++;
```

接着使用 Button 对象的 value 属性更新按钮上的文本：

```
    document.form1.myButton.value = "Button clicked" +
        numberOf Clicks +"times";
}
```

本例的函数专用于这个表单和按钮，而不是可用于其他情况的通用函数。因此，本例的代码直接使用 document.form1.myButton 引用表单和按钮。注意 document 对象包含页面上的所有元素，包括<form/>元素，而按钮嵌入在表单中。

试一试　onmouseup 和 onmousedown

Button 对象支持两个不常用的事件，即 mousedown 事件和 mouseup 事件。下面的例子演示了这两个事件的用法：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 3</title>
    <script type="text/javascript">
    function myButton_onmouseup()
    {
        document.form1.myButton.value = "Mouse Goes Up"
    }
    function myButton_onmousedown()
    {
        document.form1.myButton.value = "Mouse Goes Down"
    }
    </script>
</head>
<body>
    <form action=" "name="form1">
        <input type="button" name="myButton" value="Mouse Goes Up"
            onmouseup="myButton_onmouseup()"
            onmousedown="myButton_onmousedown()" />
    </form>
</body>
</html>
```

将这个页面保存为 ch7_examp3.htm，并加载到浏览器中。如果用鼠标左键单击按钮并按住不放，按钮上的文本就变为 Mouse Goes Down。只要释放鼠标左键，该文本就变为 Mouse Goes Up。

在页面正文中，在 form1 表单中定义了一个 myButton 按钮；然后把 myButton_onmouseup()函数关联到 onmouseup 事件处理程序上，myButton_onmousedown()函数关联到 onmousedown 事件处理程序上。

```
<form action=" "name="form1">
    <input type="button" name="myButton" value="Mouse Goes Up"
        onmouseup="myButton_onmouseup()"
        onmousedown="myButton_onmousedown()" />
</form>
```

myButton_onmouseup()和 myButton_onmousedown()函数在页面开头的脚本块中定义。每个函数仅包含一行代码，该代码使用 Button 对象的 value 属性改变按钮上显示的文本。

注意，仅在鼠标指针位于当前元素上时才触发 mouseup 和 mousedown 事件。例如，如果单击并按住鼠标左键移过按钮，再在按钮的外部释放鼠标左键，就不会触发 mouseup 事件，按钮上的文本也不会改变。此时会触发 document 对象的 onmouseup 事件处理程序(假定给它连接了代码)。

与其他的表单元素对象一样，Button 对象也有 onfocus 和 onblur 事件，但它们在按钮的上下文中很少使用。

另外两个按钮类型是 Submit 和 Reset。这两种按钮的定义与标准按钮类似，但<input>

标记的 type 属性设置为 submit 或 reset，而不是 button。例如，下面的代码创建了如图 7-4 所示的 Submit 和 Reset 按钮：

```
<input  type="submit" value="Submit"  name="submit1">
<input  type="reset"  value="Reset"  name="reset1">
```

这两个按钮有特定的用途，且不需要编写脚本。

单击 Submit 按钮时，按钮所在表单中的数据将自动提交给服务器，无需编写任何脚本。

单击 Reset 按钮时，会清空按钮所在表单中的所有元素，并重置为其默认值(即页面首次加载时的元素值)。

Submit 和 Reset 按钮有对应的 Submit 和 Reset 对象，它们的属性、方法和事件与标准 Button 对象相同。

7.2.4　文本元素

标准的文本元素允许用户输入单行文本。这些信息可以用在 JavaScript 代码中，也可以提交给服务器，以便在服务器端进行处理。

1. 文本框

与按钮一样，文本框也用<input/>元素创建，但 type 属性要设置为 text。也可以选择不包含 value 属性，但如果包含它，这个值就会在页面加载时显示在文本框中。

在下面的例子中，<input/>元素有另外两个属性 size 和 maxlength。size 属性确定文本框的字符宽度，maxlength 确定用户在文本框中可以输入的最大字符数。这两个属性都是可选的，都使用由浏览器确定的默认值。

例如，要创建一个 10 字符宽、最长 15 个字符、初始值为 Hello World 的文本框，可以使用如下所示的<input/>元素：

```
<input type="text" name="myTextBox" size=10 maxlength=15 value="Hello World">
```

这个元素创建的 Text 对象有一个 value 属性，可用于在脚本中设置或读取文本框中的文本。除了前面讨论的共有属性和方法之外，Text 对象还有 select()方法，它可以选择或加亮文本框中的所有文本。如果用户输入了无效的值，就可以把焦点设置到文本框上，并选择其中的文本。这会把用户的光标放在修改数据的正确位置上，并使用户非常清楚地了解无效的数据在哪里。Text 对象的 value 属性总是返回字符串数据类型，即使输入了数字字符，也是如此。如果把该属性值用作数字，JavaScript 一般会自动把它从字符串类型转换为数字类型，但并不总是进行转换。例如，如果所执行的操作对字符串而言是有效的，JavaScript 就不执行转换。如果表单上有两个文本框，要把它们返回的值相加，JavaScript 就会连接这两个值，而不是相加，所以 1 加 1 得到 11，而不是 2。要修正这个错误，需要使用 parseInt()、parseFloat()或 Number()把这两个值转换为数字。但如果对两个值相减，这是仅对数字有效的操作，JavaScript 就会认为，这是只能对数字进行的操作，所以把值转换为数字类型。因此 1 减 1 得 0，而无需使用 parseInt()或 parseFloat()。这是一个很难查找的错误，所以最好养成显式转换数据类型的习惯，以免以后出问题。

除了共有的事件处理程序，如 onfocus 和 onblur 之外，Text 对象还有 onchange、onselect、onkeydown、onkeypress 和 onkeyup 事件处理程序。

用户选择文本框中的文本时，将触发 onselect 事件。

最有用的是 onchange 事件，当且仅当文本框失去焦点时的值不同于它获得焦点时的值，才触发该事件。因此，该事件可以仅在文本框的值发生了变化时才检查其有效性。

使用<input/>元素的 readonly 属性或 Text 对象的 readOnly 属性，可以防止修改文本框的内容。

```
<input type="text" name="txtReadonly" value="Look but don't change"
    onfocus="document.form1.txtReadonly.blur()"
    readonly="readonly">
```

顾名思义，用户按一个键时，触发 onkeypress 事件；用户按下键时，触发 onkeydown 事件，按下的键被释放时，触发 onkeyup 事件。

试一试　包含验证功能的简单表单

下面的例子综合运用了按钮和文本框。这个例子有一个简单的表单，它包含两个文本框和一个按钮。第一个文本框供用户输入名字，第二个文本框输入年龄。然后进行多项有效性检查。当年龄文本框失去焦点时，检查年龄是否有效。但是，单击按钮时，仅检查名字和年龄文本框是否为空。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 4</title>
    <script type="text/javascript">
    function btnCheckForm_onclick()
    {
        var myForm = document.form1;
        if (myForm.txtAge.value == "" || myForm.txtName.value == "")
        {
            alert("Please complete all the form");
            if (myForm.txtName.value == "")
            {
                myForm.txtName.focus();
            }
            else
            {
                myForm.txtAge.focus();
            }
        }
        else
        {
            alert("Thanks for completing the form" + myForm.txtName.value);
        }
    }
```

```
        function txtAge_onblur()
        {
            var txtAge = document.form1.txtAge;
            if (isNaN(txtAge.value) == true)
            {
                alert("Please enter a valid age");
                txtAge.focus();
                txtAge.select();
            }
        }

        function txtName_onchange()
        {
            window.status = "Hi " + document.form1.txtName.value;
        }
        </script>
    </head>
    <body>
        <form action=" " name="form1">
            Please enter the following details:
            <p>
                Name:
                <br />
                <input type="text" name="txtName" onchange="txtName_onchange()" />
            </p>
            <p>
                Age:
                <br />
                <input type="text" name="txtAge" onblur="txtAge_onblur()"
                    size="3" maxlength="3" />
            </p>
            <p>
                <input type="button" value="Check Details"
                    name="btnCheckForm" onclick="btnCheckForm_onclick()">
            </p>
        </form>
    </body>
    </html>
```

输入上面的文本后，保存为 ch7_examp4.htm。在 Web 浏览器中加载该文件。

在如图 7-5 所示的文本框中，输入名字。退出该文本框时，窗口底部的状态栏会显示 Hi yourname。

在年龄文本框中输入一个无效的值，尝试退出该文本框，代码将提示出错，并返回该文本框，以改正错误。

最后，单击 Check Details 按钮，检查两个文本框是否输入了值。如果其中一个为空，就显示一条消息，提示完成这个表单的填写，并返回输入为空的文本框。

如果正确填写了两个文本框，则显示一条感谢消息，如图 7-5 所示。

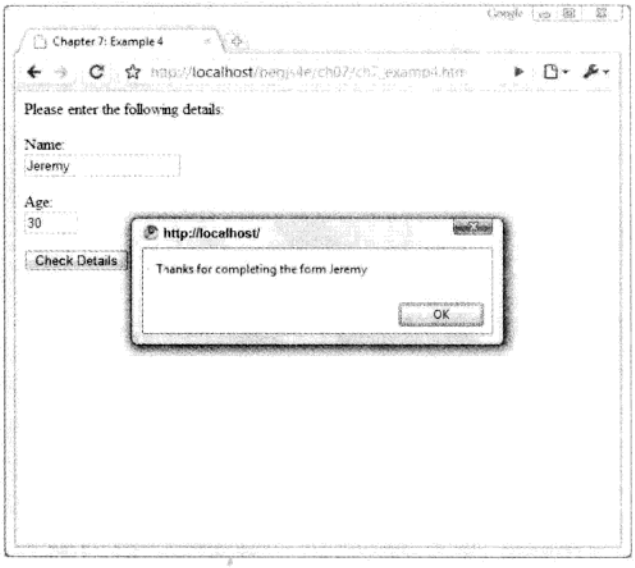

图 7-5

注意：

这个示例在 Firefox 上不能正常工作，稍后讨论这个问题。

在页面正文中，创建了定义 form1 表单的 HTML 元素。在这个表单中，创建了三个表单元素 txtName、txtAge 和 btnCheckForm。

```
<form action=" "name="form1">
    Please enter the following details:
    <p>
        Name:
        <br />
            <input type="text" name="txtName" onchange="txtName_onchange()" />
    </p>
    <p>
        Age:
        <br />
        <input type="text" name="txtAge" onblur="txtAge_onblur()"
            size="3" maxlength="3" />
    </p>
    <p>
       <input type="button" value="Check Details"
           name="btnCheckForm" onclick="btnCheckForm_onclick()">
    </p>
</form>
```

对于第二个文本框 txtAge，在<input/>元素中包含了 size 和 maxlength 属性。size 属性设置为 3，以提示用户在年龄文本框中能输入的位数；将 maxlength 属性设置为 3，以确保用户不会输入过大的年龄数字。

第一个文本框的 onchange 事件处理程序连接到函数 txtName_onchange()上，第二个文本框的 onblur 事件处理程序连接到函数 txtAge_onblur()上，按钮的 onclick 事件处理程序连接到函数 btnCheckForm_onclick()上。这三个函数都在页面顶部的脚本块开头处定义。下面依次讨论它们，从 btnCheckForm_onclick()开始。

首先定义一个变量 myForm，并设置为引用页面后面<form/>元素创建的 Form 对象：

```
function btnCheckForm_onclick()
{
    var myForm = document.form1;
```

这样可以减少代码量，每次要使用 form1 对象时，只需输入 myForm，而无须输入 document.form1。所以提高了代码的可读性，使代码更易于调试，减少了输入量。把变量设置为等于一个已有的对象时，并没有创建新的 form1 对象(本例便是如此)，仅是把变量指向已有的 form1 对象。因此，当输入 myForm.name 时，JavaScript 将检查该变量，发现它实际上存储了对象 form1 在内存中的位置，于是使用这个对象。这都在后台完成，无需关心它，而可以直接使用 myForm，就像它是 document.form1 一样。

获得了 Form 对象的引用后，就可以在 if 语句中使用它来检查 txtAge 文本框或 txtName 文本框是否包含文本。

```
if (myForm.txtAge.value == "" || myForm.txtName.value == "")
{
    alert("Please complete all the form");
    if (myForm.txtName.value == "")
    {
        myForm.txtName.focus();
    }
    else
    {
        myForm.txtAge.focus();
    }
}
```

如果未填全表单，就警告用户。接下来在内层的 if 语句中，检查哪个文本框未填写，并把焦点设置回未填写的文本框。这样，用户就可以直接开始填写它，而无须自己把焦点移动到该文本框上。这也告诉用户，程序要求填写哪个文本框。为了避免使用户感到疑惑，确保用页面上的文本指出哪些字段是必填的。

如果外层的 if 语句发现表单已填全，就向用户显示一条感谢消息。

```
    else
    {
        alert("Thanks for completing the form " + myForm.txtName.value);
    }
}
```

在这种情况下，常常是将表单提交给服务器，而不是显示一条感谢消息。为此，可以使用 Form 对象的 submit()方法，或者使用标准的 Submit 按钮。

第二个函数是 txtAge_onblur()，它连接到 txtAge 文本框的 onblur 事件上。这个函数的

作用是检查用户在年龄文本框中输入的字符串是否由数字字符组成：

```
function txtAge_onblur()
{
    var txtAge = document.form1.txtAge;
```

该函数的开头也声明了一个变量，并设置为引用一个对象；这次是为页面后面定义的 txtAge 文本框创建 Text 对象。现在，无需每次输入 document.form1.txtAge，而只需输入 txtAge，这有助于减少输入量，尤其是这个函数要多次用到 txtAge 对象时，输入量就更少。

接下来的 if 语句检查 txtAge 文本框中的输入能否转换为数值。为此使用了 isNaN()函数。如果 txtAge 文本框中的值不能转换为数值，则提示用户输入无效，并使用 focus()方法将焦点设置回该文本框。另外，这次使用 Text 对象的 select()方法来加亮文本。这样用户将很清楚需要修改什么地方，并且用户可以改正错误，而无需先删除文本。

```
    if (isNaN(txtAge.value) == true)
    {
        alert("Please enter a valid age");
        txtAge.focus();
        txtAge.select();
    }
}
```

可以进一步检查文本框中的数值是否确实是有效年龄——例如，191 或 255 都不是有效年龄。为此，只需添加另一个 if 语句进行检查。

这个函数连接到 txtAge 文本框的 onblur 事件处理程序上，为什么不使用 onchange 事件处理程序？因为 onchange 事件仅在文本框的值发生了改变时，才复查该值。而在文本框获得焦点之前和失去焦点之后，如果文本框为空，onchange 事件就不会触发。但最好在提交表单之前检查表单是否填全，因为某些用户可能不按顺序填写表单。

最后一个函数用于 txtName 文本框的 onchange 事件，它仅用于演示 onchange 事件。

```
function txtName_onchange()
{
    window.status = "Hi " + document.form1.txtName.value;
}
```

当焦点移出名字文本框，且其内容有了改变时，将触发 onchange 事件。这时，提取 txtName 文本框的值，并放在窗口底部的状态栏上，显示“Hi 你的名字”。使用 window 对象的 status 属性可以访问该状态栏。也可以仅输入：

```
status = "Hi " + document.form1.txtName.value;
```

把 window 放在前面，清楚地说明当前访问的对象。在阅读代码时，很容易将 status 误认为是一个变量。因此在这种情况下，虽然把 window 放在前面不是严格必须的，但会使代码更易于阅读、理解和调试。

2. Firefox 和 blur 事件的问题

如果上面的例子在 Firefox 中运行，在名字文本框中输入名字，在年龄文本框中输入一个无效的年龄，如 abc，然后单击 Check Form 按钮，该示例就会失败。在 IE 中，如果年龄无效，会触发 blur 事件，并显示一个警告框，但按钮的 click 事件不会触发。但是在 Firefox 中，这两个事件都会触发，导致年龄无效警告框被“感谢完成表单”警告框遮住。

另外，如果在 IE 和 Firefox 浏览器中都输入一个无效的年龄，然后切换到另一个程序，“年龄无效”警告框将会显示出来，这会让用户感到迷惑。用户可能会在另一个程序中了解详情。

尽管这是一个很好的例子，但并不能用于实际。一个较好的选项是在最后提交表单时检查表单，而不是在用户输入数据时检查。也可以在用户输入数据时检查，但不使用警告框显示错误。而是在输入错误的控件旁显示红色警告，以提示用户输入无效。然后代码在提交表单时检查数据。第 12 章将介绍在页面加载后如何在页面上输出信息。

3. 密码文本框

使用密码框的唯一目的就是用户在页面上输入密码时，隐藏输入的密码字符，以防别人窥探到用户的密码。但是，这种保护仅仅是做个样子。发送给服务器时，密码中的文本以明文发送——没有加密或隐藏文本(除非页面通过与服务器的安全连接来传送)。

密码框的定义与文本框相同，只是 type 属性要设置为 password。

```
<input name="password1" type="password">
```

这个表单元素创建一个关联的 Password 对象，其属性、方法和事件与 Text 对象相同。

4. 隐藏文本框

隐藏文本框可以像普通文本框那样保存文本和数字，不同之处在于隐藏文本框对用户来说是不可见的。隐藏的元素？听起来与不可见的图画一样，但实际上却非常有用。

要定义一个隐藏文本框，可以使用如下 HTML：

```
<input type="hidden" name="myHiddenElement" />
```

隐藏文本框会创建一个 Hidden 对象。与其他任何对象一样，可以通过 Form 对象的 elements 数组属性来访问 Hidden 对象，并在 JavaScript 中操作它。但只能在其 HTML 定义中或通过 JavaScript 设置隐藏文本框的值。与一般的文本框一样，用户提交表单时，隐藏文本框的值会提交给服务器。

隐藏文本框有什么作用呢？假如需要从用户处获得很多信息，但是为了避免页面充斥着各种元素，看起来像是航天飞机的控制面板，应通过多个页面来获取信息。问题在于，如何保存前面页面中输入的内容？很简单——只需将这些信息保存在隐藏文本框中。接着，在最终页面上，将所有信息提交给服务器——其中的一些信息是隐藏的。

7.2.5 textarea 元素

<textarea/>元素允许输入多行文本。除此之外，它与文本框元素非常类似。

但与文本框不同，textarea 元素有自己的标记<textarea>。它另有两个属性：cols 和 rows。cols 属性定义了文本区域的字符宽度；rows 属性定义了文本区域的字符行数。设置该元素中的文本时，应将文本放在开闭标记之间，而不是使用 value 属性。所以，如果需要一个<textarea/>元素，其宽度为 40 个字符、高度为 20 行，第一行为 Hello World，第二行为 Line 2，可定义为如下代码：

```
<textarea name="myTextArea" cols="40" rows="20">Hello World
Line 2
</textarea>
```

<textarea/>元素的另一个属性是 wrap，它确定了用户输入到行末时如何处理。其默认值是 soft，所以用户在行末处不需要按下回车键，但这因浏览器而异。要打开自动换行功能，可以使用 soft 或 hard 这两个值。对于客户端处理而言，这两个值的作用相同：打开自动换行功能。但在进行服务器端处理时，这两个值的区别是：提交表单时传送给服务器的信息不同。

如果 wrap 属性设置为 soft，就打开自动换行功能，客户端会进行自动换行，但回车符不会传送给服务器，而只传送文本。如果 wrap 属性设置为 hard，则由自动换行产生的所有回车符都转换为硬回车符，就好像用户按下了回车键一样，这些回车符也会发送给服务器。另外要注意，回车符由运行浏览器的操作系统确定，例如，Windows 上的回车符是\r\n，Macintosh 上的回车符是\r，Unix 上的回车符是\n。要关闭客户端的自动换行功能，可把 wrap 设置为 off。

<textarea/>元素创建的 Textarea 对象与前面介绍的 Text 对象拥有相同的属性、方法和事件。但是，文本区域没有 maxlength 属性。注意，虽然<textarea/>元素没有 value 属性，但是 Textarea 对象有 value 属性，它仅返回<textarea>和</textarea>标记之间的文本。Textarea 对象支持的事件包括 onkeydown、onkeypress、onkeyup 和 onchange 事件处理程序。

试一试　事件观察

为了演示 keydown、keypress、keyup 和 change 事件的工作过程，特别是这些事件的触发顺序，下面创建一个例子来了解触发的事件。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 5</title>
    <script type="text/javascript">

    function DisplayEvent(eventName)
    {
        var myMessage = window.document.form1.textarea2.value;
        myMessage = myMessage + eventName;
```

```
            document.form1.textarea2.value = myMessage;
        }
        </script>
    </head>
    <body>
        <form action=" "name="form1">
            <textarea rows="15" cols="40" name="textarea1"
                onchange="DisplayEvent('onchange\n'); "
                onkeydown="DisplayEvent('onkeydown\n'); "
                onkeypress="DisplayEvent('onkeypress\n'); "
                onkeyup="DisplayEvent('onkeyup\n\n');"></textarea>

            <textarea rows="15" cols="40" name="textarea2"></textarea>
            <br />
            <br />
            <input type="button" value="Clear Event TextArea" name="button1"
                onclick="document.form1.textarea2.value=' '" />
        </form>
    </body>
    </html>
```

将这个页面保存为 ch7_examp5.htm。在浏览器中加载该页面，并在第一个文本区域中输入任意字符，观察页面的变化。在第二个文本区域中会列出触发的事件(onkeydown、onkeypress 和 onkeyup)，如图 7-6 所示。单击第一个文本区域的外部，会触发 onchange 事件。

图　7-6

通过该例子来观察触发了什么事件和事件的触发时间。

在页面正文的 form1 表单中，定义了两个文本区域和一个按钮。我们将监视第一个文本区域的事件。将调用 DisplayEvent()函数的代码分别连接到 onchange、onkeydown、onkeypress 和 onkeyup 事件处理程序上。传送给函数的值反映了所触发事件的名称。

```
<textarea rows="15" cols="40" name="textarea1"
    onchange="DisplayEvent('onchange\n');"
    onkeydown="DisplayEvent('onkeydown\n');"
    onkeypress="DisplayEvent('onkeypress\n');"
    onkeyup="DisplayEvent('onkeyup\n\n');"></textarea>
```

接下来的空文本区域与第一个文本区域大小相同。

```
<textarea rows="15" cols="40" name="textarea2"></textarea>
```

最后是按钮元素：

```
<input type="button" value="Clear Event TextArea" name="button1"
    onclick="document.form1.textarea2.value='' " />
```

注意，该按钮的 onclick 事件处理程序没有调用函数，而仅执行一行 JavaScript 代码。尽管通常情况是调用函数，但这不是强制的，如果仅执行一行代码，就只需插入该代码，而不是创建一个函数，再调用它。在本例中，onclick 事件处理程序连接到一行代码上，该代码将第二个文本区域的 value 属性设置为空串('')，清空了其内容。

下面看看 displayEvent()函数。该函数在页面开头的脚本块中定义。它把作为参数传入的事件处理程序名称追加到第二个文本区域已包含的文本后面。

```
function displayEvent(eventName)
{
    var myMessage = document.form1.textarea2.value;
    myMessage = myMessage + eventName;
    document.form1.textarea2.value = myMessage;
}
```

7.2.6 复选框和单选按钮

将复选框和单选按钮放在一起讨论，因为它们的对象有相同的属性、方法和事件。复选框允许用户选中它或者取消选中状态。这类似于问卷调查中的“请选择合适的选项”。单选按钮基本上是一组一次只能选择一项的复选框。当然，它们的外观不同，其分组性质也意味着它们要分别处理。

创建复选框和单选按钮需要我们熟悉的<input/>元素。其 type 属性设置为"checkbox"或"radio"，以确定创建复选框还是单选按钮。要在页面加载时选中某个复选框或单选按钮，只需在<input>标记中插入属性 checked，并给它赋值 checked。如果要设置默认选项，这种方法就很方便。例如，网页上常见的表单“如果你需要我们的邮件，请选中该复选框”，通常是默认选中的，强制用户取消它们。要创建一个已选中的复选框，<input>标记如下所示：

```
<input type="checkbox" name="chkDVD" checked="checked" value="DVD" />
```

要创建一个选中的单选按钮，<input>标记应如下所示：

```
<input type="radio" name="radCPUSpeed" checked="checked" value="1 GHz" />
```

前面提到，单选按钮是分组的元素。实际上，页面上只放置一个单选按钮是没有意义的，因为用户将无法选择其他选项。

要创建一组单选按钮，只需给每个单选按钮指定相同的 name，这会创建一个使用该名称的单选按钮数组，使用其索引可以访问该数组，就像访问普通的数组那样。

例如，要创建一个包含三个单选按钮的组，可使用如下的 HTML：

```
<input type="radio" name="radCPUSpeed" checked="checked" value="800 MHz" />
<input type="radio" name="radCPUSpeed" value="1 GHz" />
<input type="radio" name="radCPUSpeed" value="1.5 GHz" />
```

可以在表单上放置任意多个单选按钮组，只要为每个组定义唯一的名称即可。注意，一个组中只能使用一个 checked 属性，因为只能选中组中的一个单选按钮。如果在多个单选按钮中使用了 checked 属性，则只有最后一个单选按钮被选中。

复选框和单选按钮元素的 value 属性与前面介绍的元素不同。该属性与用户和元素的交互无关，因为它是在 HTML 或 JavaScript 中预定义的。无论是否选中复选框或单选按钮，它都返回相同的值。

每个复选框都有一个对应的 Checkbox 对象，组中的每个单选按钮都有一个独立的 Radio 对象。如前所述，单选按钮有相同的名称，所以可以访问组中的每个 Radio 对象，方法是把单选按钮组当作数组，该数组名就是组中单选按钮的名称。与数组一样，该数组也有 length 属性，它指定了组中的单选按钮数量。

为了确定用户是选中了某复选框，还是取消了该复选框，需要使用 Checkbox 对象的 checked 属性。如果复选框当前被选中，该属性就返回 true，否则返回 false。

单选按钮则稍有区别。因为同名的单选按钮组合在一起，所以需要依次测试每个 Radio 对象，看看它是否被选中。组中只能选中一个单选按钮，所以如果选中组中的另一个单选按钮，以前选中的单选按钮就会取消选中，新的单选按钮处于选中状态。

Checkbox 和 Radio 都有 onclick、onfocus 和 onblur 事件处理程序，它们的操作与其他元素相同，而且它们还可用于取消默认操作，例如单击复选框或单选按钮。

试一试　**复选框和单选按钮**

下面的例子使用了前面讨论的所有属性、方法和事件。这个例子是一个简单的表单，它允许用户建立计算机系统。也许它可以用在电子商务网站中，以便完全根据客户指定的配置来销售计算机。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 6</title>
    <script type="text/javascript">

    var radCpuSpeedIndex = 0;
```

```
    function radCPUSpeed_onclick(radIndex)
    {
        var returnValue = true;
        if (radIndex == 1)
        {
            returnValue = false;
            alert("Sorry that processor speed is currently unavailable");
            // Next line works around a bug in IE that doesn't cancel the
            // Default action properly
            document.form1.radCPUSpeed[radCpuSpeedIndex].checked = true;
        }
        else
        {
            radCpuSpeedIndex = radIndex;
        }
        return returnValue;
    }

    function btnCheck_onclick()
    {
        var controlIndex;
        var element;
        var numberOfControls = document.form1.length;
        var compSpec = "Your chosen processor speed is ";

        compSpec = compSpec + document.form1.radCPUSpeed[radCpuSpeedIndex].value;
        compSpec = compSpec + "\nWith the following additional components\n";
        for (controlIndex = 0; controlIndex < numberOfControls; controlIndex++)
        {
            element = document.form1[controlIndex];
            if (element.type == "checkbox")
            {
                if (element.checked == true)
                {
                    compSpec = compSpec + element.value + "\n";
                }
            }
        }
        alert(compSpec);
    }
    </script>
</head>
<body>
    <form action=" " name="form1">
        <p>
            Tick all of the components you want included on your computer
        </p>
        <table>
            <tr>
                <td>
                    DVD-ROM
```

```
            </td>
            <td>
                <input type="checkbox" name="chkDVD" value="DVD-ROM" />
            </td>
        </tr>
        <tr>
            <td>
                CD-ROM
            </td>
            <td>
                <input type="checkbox" name="chkCD" value="CD-ROM" />
            </td>
        </tr>
        <tr>
            <td>
                Zip Drive
            </td>
            <td>
                <input type="checkbox" name="chkZip" value="ZIP Drive" />
            </td>
        </tr>
    </table>
    <p>
        Select the processor speed you require
    </p>
    <table>
        <tr>
            <td>
                <input type="radio" name="radCPUSpeed" checked="checked"
                    value="3.8 GHz" onclick="return radCPUSpeed_onclick(0)" />
                </td>
                <td>
                    3.8 GHz
                </td>
                <td>
                    <input type="radio" name="radCPUSpeed" value="4.8 GHz"
                        onclick="return radCPUSpeed_onclick(1) " />
                </td>
                <td>
                    4.8 GHz
                </td>
                <td>
                    <input type="radio" name="radCPUSpeed" value="6 Ghz"
                        onclick="return radCPUSpeed_onclick(2)" />
                </td>
                <td>
                   6 GHz
                </td>
            </tr>
        </table>
```

```
            <input type="button" value="Check Form" name="btnCheck"
                onclick="return btnCheck_onclick()" />
    </form>
</body>
</html>
```

将该页面保存为 ch7_examp6.htm，并在 Web 浏览器中加载它。表单如图 7-7 所示。

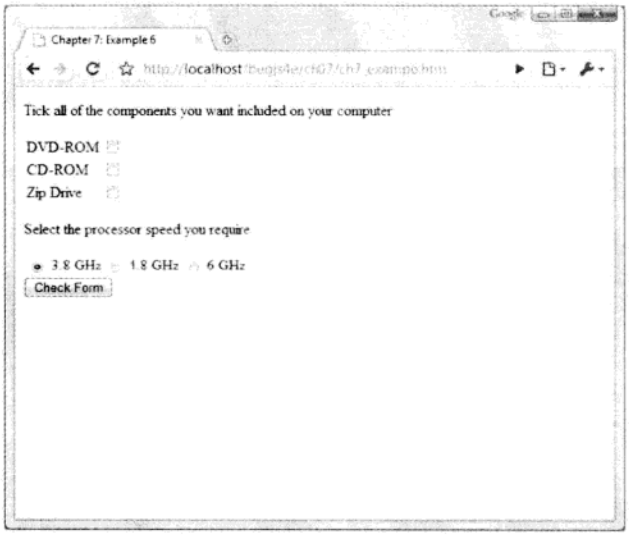

图 7-7

选择一些复选框，修改处理器的速度，然后单击 Check Form 按钮。页面将会弹出一个消息框，其中列出所选的配件和处理器的频率。例如，如果选择了 DVD-ROM、Zip drive 以及频率为 6 GHz 的处理器，则消息框如图 7-8 所示。

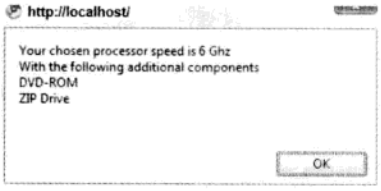

图 7-8

注意，4.8 GHz 的处理器脱销，因此如果尝试选择它，弹出的消息框会说明该处理器脱销，并且 4.8 GHz 处理器的单选按钮将不会选中。用户退出该消息框时，会恢复以前的设置。

在页面正文上，在 form1 表单中定义了复选框、单选按钮和一个标准按钮。首先看看复选框。它们放在一个表格中，以进行格式化。没有调用函数，也没有连接任何事件。

```
<table>
    <tr>
        <td>
            DVD-ROM
        </td>
        <td>
            <input type="checkbox" name="chkDVD" value=" DVD-ROM" />
        </td>
    </tr>
    <tr>
```

```
            <td>
                CD-ROM
            </td>
            <td>
                <input type="checkbox" name="chkCD" value="CD-ROM" />
            </td>
        </tr>
        <tr>
            <td>
               Zip Drive
            </td>
            <td>
                <input type="checkbox" name="chkZip" value="ZIP Drive" />
            </td>
        </tr>
    </table>
```

接下来是用于选择所需 CPU 速度的单选按钮，这些内容稍复杂些。它们也放在表格中，以进行格式化。

```
    <table>
        <tr>
            <td>
                <input type="radio" name="radCPUSpeed" checked="checked"
                    value="3.8 GHz" onclick="return radCPUSpeed_onclick(0)" />
            </td>
            <td>
                3.8 GHz
            </td>
            <td>
                <input type="radio" name="radCPUSpeed" value="4.8 GHz"
                    onclick="return radCPUSpeed_onclick(1)" />
            </td>
            <td>
                4.8 GHz
            </td>
            <td>
                <input type="radio" name="radCPUSpeed" value="6 Ghz"
                    onclick="return radCPUSpeed_onclick(2) " />
            </td>
            <td>
                6 GHz
            </td>
        </tr>
    </table>
```

单选按钮组名称为 radCPUSpeed。其中，默认选中第一个单选按钮，因为在其<input/>元素的定义中包含 checked。最好默认选中一个单选按钮，因为如果没有选中某个单选按钮，用户也没有选择一项，提交表单时就不会为该组提交值。

上述代码使用了每个对象的 onclick 事件，每个按钮都连接到同一个函数 radCPUSpeed_

onclick()上。但为每个单选按钮传送了不同的值——该按钮在 radCPUSpeed 单选按钮组集合中的索引。这便于确定选中了哪个单选按钮。稍后介绍这个函数，现在先看看完成表单的标准按钮。

```
<input type="button" value="Check Form" name=btnCheck
    onclick="return btnCheck_onclick()"/>
```

该按钮的 onclick 事件处理程序连接到 btnCheck_onclick()函数，用户填写完表单时，就单击该按钮。

因此，本例有两个函数 radCPUSpeed_onclick()和 btnCheck_onclick()。它们都在页面顶部的脚本块中定义。现在分析这个脚本块。它首先声明了一个变量 radCpuSpeedIndex，用于存储单选按钮组中当前选中的索引。

```
var radCpuSpeedIndex = 0;
```

接下来是 radCPUSpeed_onclick()函数，它由每个单选按钮的 onclick 事件处理程序调用。这个函数有一个参数，即所选单选按钮在 radCPUSpeed 集合中的索引位置。

```
function radCPUSpeed_onclick(radIndex)
{
    var returnValue = true;
```

在这个函数中，首先声明变量 returnValue，并设置为 true。从函数中返回时，要返回这个变量。本例的返回值非常重要，因为它确定了用户单击某单选按钮时，是否选中它。如果 returnValue 返回 false，表示取消用户的操作，该单选按钮就不选中。此时就没有选中任何单选按钮，因此要跟踪所选单选按钮的索引，以跟踪上一次选中的单选按钮。为了允许执行用户的操作，应返回 true。

为了演示上述情况，下一行使用了 if 语句。如果所传送的单选按钮索引值是 1(即用户选中了 4.8 GHz 的处理器，就告诉用户该处理器已脱销，并把 returnValue 设置为 false，取消单击操作。

```
if (radIndex == 1)
{
    returnValue = false;
    alert("Sorry that processor speed is currently unavailable");
    // Next line works around a bug in IE that doesn't cancel the
    // Default action properly
    document.form1.radCPUSpeed[radCpuSpeedIndex].checked = true;
}
```

如前所述，取消单击操作，会导致没有选中任何单选按钮。为了修正这种情况，下面的代码再次选中上一次选中的单选按钮：

```
document.form1.radCPUSpeed[radCpuSpeedIndex].checked = true;
```

这里使用的是 radCpuSpeed 单选按钮组的集合。集合中的每个元素都包含一个对象，因此集合包含 3 个 Radio 对象。把 radCpuSpeedIndex 变量用作上次选中的 Radio 对象的索

引，因为它包含这个索引。

最后，在 else 子句中，将 radCpuSpeedIndex 设置为新选中的单选按钮的索引值：

```
else
{
    radCpuSpeedIndex = radIndex;
}
```

在函数的最后一行，将 returnValue 的值返回给调用方，以允许或取消单击操作：

```
    return returnValue;
}
```

第二个函数 btnCheck_onclick()连接到按钮的 onclick 事件。在实际的电子商务中，该按钮将检查表单，并将表单提交给服务器进行处理。这里仅使用该表单显示一个消息框，来确认用户选中的复选框。

首先声明函数中使用的 4 个本地变量。变量 numberOfControls 设置为表单的 length 属性，即表单中的元素个数。compSpec 变量用于建立要在消息框中显示的字符串。

```
function btnCheck_onclick()
{
  var controlIndex;
  var element;
  var numberOfControls = document.form1.length;
  var var compSpec = "Your chosen processor speed is ";
  compSpec = compSpec + document.form1.radCPUSpeed[radCpuSpeedIndex].value;
  compSpec = compSpec + "\nWith the following additional components\n";
```

下面的代码把用户选中的单选按钮值添加到消息字符串中：

```
compSpec = compSpec + document.form1.radCPUSpeed[radCpuSpeedIndex].value;
```

全局变量 radCpuSpeedIndex 由单选按钮组的 onclick 事件设置，它包含当前所选单选按钮的索引。

确定选中哪个单选按钮的另一种方法是遍历单选按钮组的集合，依次测试每个单选按钮是否被选中。代码如下：

```
var radIndex;
for (radIndex = 0; radIndex < document.form1.radCPUSpeed.length; radIndex++)
{
    if (document.form1.radCPUSpeed[radIndex].checked == true)
    {
        radCpuSpeedIndex = radIndex;
        break;
    }
}
```

但在实际的代码中，注意在消息字符串中使用了几个换行符(\n)，以进行格式化。

接下来遍历表单的元素：

```
for (controlIndex = 0; controlIndex < numberOfControls; controlIndex++)
```

```
    {
        element = document.form1[controlIndex];
        if (element.type == "checkbox")
        {
            if (element.checked == true)
            {
                compSpec = compSpec + element.value + "\n";
            }
        }
        alert(compSpec);
    }
```

这段代码使用 document.form1[controlIndex]遍历表单上的每个元素，返回一个对索引为 controlIndex 的元素对象的引用。

在这个示例中，把 element 变量设置为对 form1 集合中索引为 controlIndex 的对象的引用，以便于速记。现在，要使用该对象的属性或方法，只需输入 element、一个句点以及方法或对象名，使代码更便于阅读和调试，输入也更精简。

本例只想看看选中了哪些复选框，所以使用每个 HTML 表单元素对象都有的 type 属性，来确定当前处理的元素是什么类型。如果 type 是 checkbox，就继续检查它是否被选中。如果是，就把其值追加到 compSpec 中的消息字符串末尾。如果不是复选框，就可以安全地忽略它。

最后使用 alert()方法来显示消息字符串的内容。

7.2.7 选择框

尽管下拉列表和列表框看上去大不相同，但实际上二者都是由<select>标记创建的，严格地讲，它们都是选择元素。选择元素在列表中有供用户选择的一项或多项，每一项都是由<select>开闭标记中的一个或多个<option/>元素定义的。

<select/>元素的 size 属性用来指定用户能够直接看到的选项数。

例如，要创建一个包含 7 项、5 项直接可见的列表框，则 HTML 如下所示：

```
<select name="theDay" size="5">
    <option value="0" selected="selected">Monday</option>
    <option value="1">Tuesday</option>
    <option value="2">Wednesday</option>
    <option value="3">Thursday</option>
    <option value="4">Friday</option>
    <option value="5">Saturday</option>
    <option value="6">Sunday</option>
</select>
```

注意，Monday 的<option/>元素还包含了属性 selected，它使该选项在页面加载时被默认选中。上面把选项的值定义为数字，但实际上文本也是有效的。

如果想定义下拉列表框，只需将<select/>元素的 size 属性设置为 1，上面的列表框马上就变成下拉列表框。

如果允许用户从列表中一次选择多项，只需在<select/>定义中添加 multiple 属性。

<select/>元素会创建一个 Select 对象。这个对象有一个集合属性 options，它由 Option 对象组成，每个 Option 对象都对应<select/>元素中的一个<option/>元素。例如，在前面的例子中，如果<select/>元素位于 theForm 表单中，则下面的代码：

```
document.theForm.theDay.options[0]
```

可访问为 Monday 创建的选项。

如何确定用户选择了哪些选项？很简单，使用 Select 对象的 selectedIndex 属性。使用这个属性返回的索引值和 options 集合就可以访问选中的选项。

Option 对象还有 index 属性、text 属性和 value 属性。index 属性返回选项在 options 集合中的索引，text 属性是列表中显示的内容，value 属性是为选项定义的值。如果提交表单，value 属性会传送给服务器。

如果要确定所选元素包含的选项数，可以使用 Select 对象的 length 属性或 options 集合属性。

下面分析如何遍历上述选择框的 options：

```
var theDayElement = window.document.form1.theDay;
document.write("There are " + theDayElement.length + "options<br />");
var optionCounter;
for (optionCounter = 0; optionCounter < theDayElement.length; optionCounter++)
{
    document.write("Option text is " +
        theDayElement.options[optionCounter].text)
    document.write(" and its value is ");
    document.write(theDayElement.options[optionCounter].value);
    document.write("<br />")
}
```

首先把变量 theDayElement 设置为引用 Select 对象。然后在页面上输出选项的个数，本例是 7。

接着用一个 for 循环遍历 options 集合，显示每个选项的文本，如 Monday、Tuesday 等，以及每个选项的值，如 0、1 等。如果基于上面的代码来创建页面，则页面必须放在<select/>元素的定义后。

也可以在页面加载完后，给选择元素添加选项，如下所述。

1. 添加和删除选项

要在选择元素中添加新选项，只需使用 new 运算符创建新的 Option 对象，再把它插入到 Select 对象的 options 集合中的空索引位置上。

创建新的 Option 对象时，要传入两个参数：第一个是要在列表框中显示的文本，第二个是赋予该选项的值。

```
var myNewOption = new Option("TheText","TheValue");
```

然后把该 Option 对象赋给一个空的数组元素，例如：

```
document.theForm.theSelectObject.options[0] = myNewOption;
```

如果要移除一个选项，只需将 options 集合中的该选项设置为 null。例如，要移除刚才插入的选项，可使用如下代码：

```
document.theForm.theSelectObject.options[0] = null;
```

从 options 集合中移除 Option 对象时，options 集合会重新排序，每个排在被移除项之后的选项的索引将自动减 1。

在某个索引位置上插入新选项时，注意它会覆盖原来位于该索引位置上的 Option 对象。

试一试　添加和移除列表项

下面使用前面的星期列表示例来演示如何添加和移除列表项。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 7</title>
    <script type="text/javascript">
    function btnRemoveWed_onclick()
    {
        if (document.form1.theDay.options[2].text == "Wednesday")
        {
            document.form1.theDay.options[2] = null;
        }
        else
        {
            alert('There is no Wednesday here!');
        }
    }

    function btnAddWed_onclick()
    {
        if (document.form1.theDay.options[2].text != "Wednesday")
        {
            var indexCounter;
            var days = document.form1.theDay;
            var lastoption = new Option();
            days.options[days.options.length] = lastoption;
            for (indexCounter = days.options.length - 1;
                 indexCounter > 2; indexCounter-- )
            {
                days.options[indexCounter].text =
                    days.options[indexCounter - 1].text;
                days.options[indexCounter].value =
                    days.options[indexCounter - 1].value;
            }
            var option = new Option("Wednesday", 2);
            days.options[2] = option;
```

```
            }
            else
            {
                alert("Do you want to have TWO Wednesdays?????");
            }
        }
        </script>
    </head>
    <body>
        <form action=" " name="form1">
            <select name=" theDay" size="5">
                <option value="0" selected="selected">Monday</option>
                <option value="1">Tuesday</option>
                <option value="2">Wednesday</option>
                <option value="3">Thursday</option>
                <option value="4">Friday</option>
                <option value="5">Saturday</option>
                <option value="6">Sunday</option>
            </select>
            <br />
            <input type="button" value="Remove Wednesday" name="btnRemoveWed"
                onclick="btnRemoveWed_onclick()" />
            <input type="button" value="Add Wednesday" name="btnAddWed"
                onclick="btnAddWed_onclick()" />
            <br />
        </form>
    </body>
    </html>
```

保存为 ch7_examp7.htm。如果在浏览器中加载该文件，表单将如图 7-9 所示。单击 Remove Wednesday 按钮，列表中的 Wednesday 将消失。单击 Add Wednesday 按钮，会把该项添加回来。如果试图添加第二个 Wednesday，或者试图移除不存在的 Wednesday，警告框将会给出无法操作的提示。

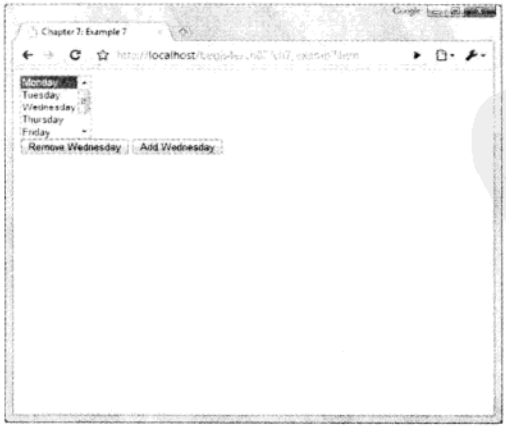

图　7-9

在页面正文中定义了一个表单 form1，其中的选择元素包含了前面示例中的星期选项。该表单还包含两个按钮，如下所示：

```
<input type="button" value="Remove Wednesday" name=butRemoveWed
    onclick="btnRemoveWed_onclick()">
<input type="button" value="Add Wednesday" name=butAddWed
    onclick="btnAddWed_onclick()">
```

这两个按钮的 onclick 事件处理程序都连接到一段代码上，以分别调用 btnRemoveWed_onclick()和 btnAddWed_onclick()函数。这些函数都在页面顶部的脚本块中定义，下面分别来看一看这两个函数。

页面的开头是第一个函数 btnRemoveWed_onclick()，它可移除 Wednesday 选项。

```
function btnRemoveWed_onclick()
{
    if (document.form1.theDay.options[2].text == "Wednesday")
    {
        document.form1.theDay.options[2] = null;
    }
    else
    {
        alert('There is no Wednesday here!');
    }
}
```

该函数首先进行完整性检查：只有在 Wednesday 选项存在的情况下，才能移除它。为此，要检查集合中的第三项(索引为 2，因为数组索引从 0 开始)的文本是否是 Wednesday。如果是，就可以把该选项设置为 null，以移除它。如果集合中的第三项不是 Wednesday，就向用户发送消息，指出没有要移除的 Wednesday。这段代码在 if 语句的条件中使用了 text 属性，也可以使用 value 属性，二者没有区别。

下面是 btnAddWed_onclick()函数，顾名思义，它可以添加 Wednesday 选项。这比移除该选项的代码复杂。首先使用 if 语句来确认并不存在 Wednesday 选项。

```
function btnAddWed_onclick()
{
    if (document.form1.theDay.options[2].text != "Wednesday")
    {
        var indexCounter;
        var days = document.form1.theDay;
        var lastoption = new Option();
        days.options[days.options.length] = lastoption;
        for (indexCounter = days.options.length - 1;
             indexCounter > 2; indexCounter-- )
        {
             days.options[indexCounter].text =
                 days.options[indexCounter - 1].text;
             days.options[indexCounter].value =
                 days.options[indexCounter - 1].value;
        }
```

如果没有 Wednesday 选项，则还需要为新的 Wednesday 选项的插入腾出位置。

在此之前，定义两个变量 indexCounter 和 days(它引用 theDay 选择元素，是一种简写方式)。目前有 6 个选项(最后一个选项的索引是 5)，所以接着用变量名 lastoption 创建一个新选项，并把它赋予集合末尾的元素。这个新元素用 options 集合的 length 属性指定其索引位置 6，这个位置以前没有内容。接着把从 Sunday 到 Thursday 的每个对象的 text 和 value 属性赋予集合中索引值比当前对象大 1 的对象，在数组的索引位置 2 处腾出空间，以便在其中插入 Wednesday。这是使用 if 语句中的 for 循环完成的。

接着向 Option 构造函数传入文本"Wednesday"和值 2，创建了一个新的 Option 对象。将这个 Option 对象插入 options 集合中索引为 2 的位置，它就马上出现在选择框中。

```
        var option = new Option("Wednesday",2);
        days.options[2] = option;
    }
```

最后，如果 if 语句中的条件是 false，就警告用户，指出列表中已有一个 Wednesday 选项。

```
        else
        {
           alert("Do you want to have TWO Wednesdays?????");
        }
    }
```

这个示例可在所有的浏览器中工作，所有的现代浏览器都提供了更便于添加和移除选项的其他方法。

2. 用标准方法添加新选项

我们关注的 Select 对象有其他的 add()和 remove()方法，用于添加和删除选项，简化了编码工作。

在添加选项之前，需要先创建它。为此可以使用 new 运算符。

Select 对象的 add()方法可以插入已建好的 Option 对象，它接受两个参数。第一个参数是要添加的 Option 对象，但第二个参数因浏览器而异。在 Firefox、Safari、Chrome、Opera 和 IE8 的标准模式下，第二个参数是要放在新 Option 对象之前的 Option 对象。在 IE7(或 IE8 的非标准模式)下，第二个参数是要添加选项的索引位置。在所有浏览器中，第二个参数都可以是 null，此时，新 Option 对象添加到 options 集合的末尾。

add()方法不会覆盖索引位置上已有的 Option 对象，而是向后移动集合中 Option 对象的位置，以腾出空间。这基本上与 btnAddWed_onclick()函数中使用 for 循环实现的功能相同。

使用 add()方法，可将 ch7.examp7.htm 例子中的 btnAddWed_onclick()函数改为：

```
function btnAddWed_onclick()
{
    var days = document.form1.theDay;
```

```
        if (days.options[2].text != "Wednesday")
        {
            var option = new Option("Wednesday", 2);
            var thursdayOption = theDay.options[2];

            try
            {
                days.add(option, thursdayOption);
            }
            catch (error)
            {
                days.add(option, 2);
            }
        }
        else
        {
            alert("Do you want to have TWO Wednesdays?????");
        }
    }
```

在 IE7(或 IE8 的非标准模式)中，如果把 Option 对象传送为第二个参数，浏览器会抛出一个错误。所以使用 try...catch 语句捕获该错误，并给第二个参数传送一个数字，如上述代码所示。

Select 对象的 remove()方法只接受一个参数，即要删除的选项的索引。删除一个选项后，集合中该选项之后的所有选项都会向前移动，以填补删除选项后的空白。

使用 remove()方法，可以将 ch7_examp7.htm 例子中的 btnRemoveWed_onclick()函数改写为：

```
function btnRemoveWed_onclick()
{
    var days = document.form1.theDay;

    if (days.options[2].text == "Wednesday")
    {
        days.remove(2);
    }
    else
    {
        alert("There is no Wednesday here!");
    }
}
```

修改前面的例子，并保存为 ch7_examp8.htm。然后在浏览器中加载它，效果与上一个例子完全相同。

3. 选择元素的事件

选择元素有 3 个事件处理程序：onblur、onfocus 和 onchange。前面介绍过它们。对于文本框元素，当焦点移出文本框，且文本框的内容发生了改变时，就触发 change 事件。但

是对于选择元素，用户改变了列表中选择的选项时，将触发 change 事件。

试一试　使用选择元素计算日期差

下面的例子在下拉列表表单中使用了选择元素及 change 事件。此例的目的是计算用户通过下拉列表框设置的两个日期相差的天数。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Example 8</title>
    <script type="text/javascript">
    function writeOptions(startNumber, endNumber)
    {
        var optionCounter;
        for (optionCounter = startNumber;
            optionCounter <= endNumber; optionCounter++)
        {
            document.write("<option value=" + optionCounter + ">" +
                optionCounter);
        }
    }

    function writeMonthOptions()
    {
        var theMonth;
        var monthCounter;
        var theDate = new Date(1);
        for (monthCounter = 0; monthCounter < 12; monthCounter++)
        {
            theDate.setMonth(monthCounter);
            theMonth = theDate.toString();
            theMonth = theMonth.substr(4, 3);
            document.write("<option value=" + theMonth + ">" + theMonth);
        }
    }

    function recalcDateDiff()
    {
        var myForm = document.form1;
        var firstDay =
            myForm.firstDay.options[myForm.firstDay.selectedIndex].value;
        var secondDay =
            myForm.secondDay.options[myForm.secondDay.selectedIndex].value;
        var firstMonth =
            myForm.firstMonth.options[myForm.firstMonth.selectedIndex].value;
        var secondMonth =
            myForm.secondMonth.options[myForm.secondMonth.selectedIndex].value;
        var firstYear =
            myForm.firstYear.options[myForm.firstYear.selectedIndex].value;
```

```
        var secondYear =
            myForm.secondYear.options[myForm.secondYear.selectedIndex].value;
        var firstDate = new Date(firstDay + " " + firstMonth + " " + firstYear);
        var secondDate = new Date(secondDay + " " + secondMonth + " " +
            secondYear);
        var daysDiff = (secondDate.valueOf() - firstDate.valueOf());

        daysDiff = Math.floor(Math.abs((((daysDiff / 1000) / 60) / 60) / 24));
        myForm.txtDays.value = daysDiff;
    }

    function window_onload()
    {
        var theForm = document.form1;
        var nowDate = new Date();
        theForm.firstDay.options[nowDate.getDate() - 1].selected = true;
        theForm.secondDay.options[nowDate.getDate() - 1].selected = true;
        theForm.firstMonth.options[nowDate.getMonth()].selected = true;
        theForm.secondMonth.options[nowDate.getMonth()].selected = true;
        theForm.firstYear.options[nowDate.getFullYear()-1970].selected = true;
        theForm.secondYear.options[nowDate.getFullYear()-1970].selected=true;
    }
    </script>
</head>
<body onload="window_onload()">
    <form action=" " name="form1">
        <p>
            First Date<br />
            <select name="firstDay" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeOptions(1, 31);
                </script>
            </select>
            <select name="firstMonth" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeMonthOptions();
                </script>
            </select>
            <select name="firstYear" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeOptions(1970, 2020);
                </script>
            </select>
        </p>
        <p>
            Second Date<br />
            <select name="secondDay" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeOptions(1, 31);
                </script>
            </select>
```

```
            <select name="secondMonth" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeMonthOptions();
                </script>
            </select>
            <select name="secondYear" onchange="recalcDateDiff()">
                <script type="text/javascript">
                writeOptions(1970, 2020);
                </script>
            </select>
        </p>
        <p>
            Total difference in days:
            <input type="text" name="txtDays" value="0" readonly="readonly" />
        </p>
    </form>
</body>
</html>
```

将这个示例命名为 ch7_examp9.htm，并在 Web 浏览器中加载它。表单应如图 7-10 所示，但两个日期框都设置为当前日期。

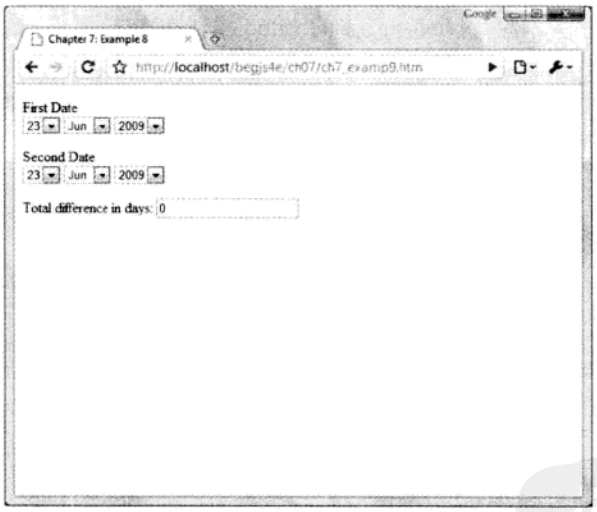

图　7-10

如果改变任意选择框，就会再次计算两个日期相差的天数，并显示在文本框中。

在页面正文的表单中建立了 6 个下拉列表框和一个文本框。以其中一个选择元素为例：第一个<select/>元素，它允许用户选择第一个日期的天数。

```
<select name="firstDay" onchange="recalcDateDiff()">
    <script type="text/javascript">
    writeOptions(1, 31);
    </script>
</select>
```

这个选择框在浏览器中显示为下拉列表框，<select/>元素的 size 属性默认设置为 1。onchange 事件处理程序连接到稍后介绍的 recalcDateDiff()函数上。

但是，在<select/>元素中没有定义任何<option/>元素。这个下拉列表框需要手工填写许多选项，所以这里使用函数来填写选项，该函数使用了 document.write()方法。

天数和年份选项用页面开头声明的 writeOptions()函数来填写。需要给该函数传送两个值：需要在选择元素中填写的选项的起始值和结束值。下面分析 writeOptions()函数：

```
function writeOptions(startNumber, endNumber)
{
    var optionCounter;
    for (optionCounter = startNumber;
        optionCounter <= endNumber; optionCounter++)
    {
        document.write("<option value=" + optionCounter + ">" +
            optionCounter);
    }
}
```

这个函数实际上很简单，它仅包含一个 for 循环，该 for 循环使用变量 optionCounter，从第一个数字 startNumber 循环到最后一个数字 endNumber，并给每个<option/>元素输出所需的 HTML。选项的文本和<option/>元素的 value 属性都指定为 optionCounter 变量的值。这肯定比输入 31 个<option/>元素来表示一月的天数快许多。

对于年份选择框，可以重用这个函数。只需将 1970 和 2020 作为参数传入 writeOptions()函数，来填写年份选择框即可。

```
<select name="firstYear" onchange="recalcDateDiff()">
    <script type="text/javascript">
    writeOptions(1970, 2020);
    </script>
</select>
```

为了用每个月的名字填充月份选择框，需要另一个函数。但填充<select/>元素的原则是相同的：使用 document.write()方法。这个函数是 writeMonthOptions()，如下面的月份选择元素所示：

```
<select name="firstMonth" onchange="recalcDateDiff()">
    <script type="text/javascript">
    writeMonthOptions();
    </script>
</select>
```

这个新函数 writeMonthOptions()在页面开头定义。下面分析这个函数。该函数先定义了三个变量，并将变量 theDate 初始化为当前日期：

```
function writeMonthOptions()
{
   var theMonth;
   var monthCounter;
```

```
var theDate = new Date();
```

然后使用 theDate 变量包含的 Date 对象，获得文本形式的月份(即 Jan、Feb…Dec)。为了获得这些月份，在 for 循环中使用 setMonth()方法，把 theDate 变量中的月份设置为 0 到 11 中的某个数字。尽管 Date 对象只提供了把日期返回为数字的方法，但它提供了 toString()方法，可以把存储在变量中的日期值返回为字符串。它返回的日期格式是星期、月份、日期、时间和年份，例如 Wed Jul 15 2009 16:11:10 GMT-0500。这个字符串因浏览器而异，但它们都从第 5 个字符开始表示月份。有了这个信息，就很容易使用 String 对象的 substr()方法提取出月份：

```
    for (monthCounter = 0; monthCounter < 12; monthCounter++)
    {
       theDate.setMonth(monthCounter);
       theMonth = theDate.toString();
       theMonth = theMonth.substr(4,3);
       document.write("<option value=" + theMonth + ">" + theMonth);
    }
}
```

有了用 3 个字符表示的月份后，就可以创建<option/>元素，用月份填写其文本和值了。

为了方便用户，最好在加载页面的过程中，把两个选择元素中的日期都设置为当前日期。这是在 window_onload()函数中实现的，它通过<body>开始标记处理窗口的 load 事件。

```
<body onload="window_onload()">
```

window_onload()函数在页面的开头定义。它首先把 theForm 变量设置为引用 Form 对象，因为它缩短了代码中需要的引用。接着创建一个保存 Date 对象的变量，以存储当前日期。

```
function window_onload()
{
    var theForm = document.form1;
    var nowDate = new Date();
```

设置每个<select/>元素的初始值是非常简单的，可以修改 Date 对象 nowDate 的返回值，以便提供 options 集合需要的索引。对于月份中的天数，正确的索引是月份中的天数减 1，因为数组索引是从 0 开始的，所以某月第一天的索引是 0。selected 属性设置为 true，使该天是列表中当前选中的选项。

```
theForm.firstDay.options[nowDate.getDate() - 1].selected = true;
theForm.secondDay.options[nowDate.getDate() - 1].selected = true;
```

月份更容易设置，因为 getMonth()函数给月份返回 0 到 11，正好是 options 集合所需的索引值。

```
theForm.firstMonth.options[nowDate.getMonth()].selected = true;
theForm.secondMonth.options[nowDate.getMonth()].selected = true;
```

对于年份，由于要把 1970 年作为第一年，所以需要从当前年份中减去 1970，以获得正确的索引值：

```
    theForm.firstYear.options[nowDate.getFullYear() - 1970].selected = true;
    theForm.secondYear.options[nowDate.getFullYear() - 1970].selected = true;
}
```

需要分析的最后一部分代码是连接到每个选择元素的 change 事件上的 recalcDateDiff()函数。该函数首先根据用户在下拉列表框中做出的选择，建立两个日期。

```
function recalcDateDiff()
{
    var myForm = document.form1;
    var firstDay = myForm.firstDay.options[myForm.firstDay.selectedIndex].value;
    var secondDay =
        myForm.secondDay.options[myForm.secondDay.selectedIndex].value;
    var firstMonth =
        myForm.firstMonth.options[myForm.firstMonth.selectedIndex].value;
    var secondMonth =
        myForm.secondMonth.options[myForm.secondMonth.selectedIndex].value;
    var firstYear =
        myForm.firstYear.options[myForm.firstYear.selectedIndex].value;
    var secondYear =
        myForm.secondYear.options[myForm.secondYear.selectedIndex].value;
```

上面的代码遍历每个选择元素，并提取所选 Option 对象的值。Select 对象的属性 selectedIndex 提供了引用所选 Option 对象在 options 集合中的索引。例如在下面的代码中，索引是由 myForm.firstDay.selectedIndex 提供的。

```
var firstDay = myForm.firstDay.options[myForm.firstDay.selectedIndex].value;
```

接着使用方括号中的值作为 firstDay 选择元素的 options 集合中的索引值。这就提供了对所选 Option 对象的引用，其 value 属性保存在变量 firstDay 中。

对所有其他选择元素也采用这种方法来处理。

然后，根据从选择元素中获取的值，创建新的 Date 对象，并保存在变量 firstDate 和 secondDate 中：

```
var firstDate = new Date(firstDay + " " + firstMonth + " " + firstYear);
var secondDate = new Date(secondDay + " " + secondMonth + " " + secondYear);
```

最后计算两个日期相差的天数：

```
var daysDiff = (secondDate.valueOf() - firstDate.valueOf());
daysDiff = Math.floor(Math.abs((((daysDiff / 1000) / 60) / 60) / 24));
```

Date 对象的 valueOf()方法返回从 1970 年 1 月 1 日到 Date 对象存储的日期所经过的毫秒数。从 secondDate 的 valueOf()属性值中减去 firstDate 的 valueOf 属性值，并将结果存储在变量 daysDiff 中。此时，它存储的是两个日期相差的毫秒数。于是下一行代码把这个值转换为天数。把该值除以 1000，得到秒数，再除以 60，得到分钟数，再除以 60，得到小

时数，最后除以 24，就得到了相差的天数。Math 对象的 abs()方法可以把负数变成正数。用户设置的第一个日期可能比第二个日期晚，而本例只希望得到两个日期相差的天数，而不管日期的先后，所以把负数变成正数。Math.floor()方法可去掉结果的小数部分，并向下修整为最接近结果的最大整数。

最后，把相差的天数输出在页面的 txtDays 文本框中：

```
myForm.txtDays.value = daysDiff;
```

7.3　小结

本章介绍了如何在 JavaScript 中添加用户界面元素，以便与用户交互，获取用户输入的信息。下面是本章内容的小结：

- 在页面上，HTML 表单用于放置构成界面的元素。
- 每个 HTML 表单都把一组 HTML 元素组合在一起。把表单提交给服务器进行处理时，该表单中的所有数据都将发送给服务器。一个页面可以包含多个表单，但是只有一个表单的信息能提交给服务器。
- 表单用开始标记<form>和结束标记</form>来创建。表单中包含的所有元素都放在<form>开闭标记之间。<form/>元素有各种属性，对于客户端脚本编程来说，最重要的是 name 属性。可以通过表单的 name 属性或 ID 属性来访问表单。
- 每个<form>元素都将创建一个 Form 对象，它包含在 document 对象中。要访问 myForm 表单，只需使用 document.myForm 即可。document 对象还有 forms 属性，它是一个包含文档中所有表单的集合。页面中的第一个表单是 document.forms[0]，第二个表单是 document.forms[1]，依此类推。Forms 属性的 length 属性(document.forms.length)表示页面上的表单数。
- 讨论了表单后，本章介绍了可放在表单中的各种 HTML 元素，以及如何创建它们以及如何在 JavaScript 中使用它们。
- 与表单元素关联的对象有许多通用的属性、方法和事件。这些对象都有 name 属性，该属性可用于在 JavaScript 中引用对象。这些对象也都有 form 属性，该属性是对包含该元素的 Form 对象的引用。type 属性返回一个表示元素类型的文本字符串，如 text、button 和 radio。
- 每个表单元素对象都有 focus()和 blur()方法，以及 focus 和 blur 事件。当用户选中了某元素，或者使用了它的 focus()方法，使该元素成为表单中的激活元素，就称为该元素获得了焦点。但一旦元素获得了焦点，就触发其 focus 事件。把另一个元素设置为当前激活元素时，前一个元素就失去了焦点(blur)，这是因为用户选中了另一个元素，或者使用了 blur()方法，此时会触发 blur 事件。如果小心使用，focus 和 blur 的触发可以用于检查用户输入数据的有效性。
- 所有元素都返回一个值，它是赋予该元素的字符串数据。该值的含义取决于元素，对于文本框，它是文本框中的值，对于按钮，它是显示在按钮上的文本。

- 讨论了元素的通用属性后，就从按钮元素开始，依次讨论了常用的元素。
- 按钮元素的作用是供用户单击，这个单击会触发执行用户编写的一些脚本。连接到按钮的 click 事件上，就可以捕获这个单击。创建按钮时，应使用<input/>元素，并将 type 属性设置为 button。其 value 属性确定了在按钮上显示的文字。按钮的两个变体是 submit 和 reset 按钮。除了用作按钮之外，它们还提供了不连接到代码上的特殊功能。submit 按钮可自动把表单提交给服务器，reset 按钮可清除表单的内容，把表单恢复到加载页面时的默认状态。
- 文本元素允许用户输入单行的简单文本。要创建文本框，只需使用<input/>元素，并将 type 属性设置为 text。通过<input/>元素的 maxlength 和 size 属性，可以分别设置文本框允许输入的字符数和文本框宽度。文本框有一个关联对象 Text，以及 select 和 change 事件。当用户选择了框中的文本时，将触发 select 事件。如果文本框失去焦点，且其内容自文本框获得焦点以来发生了变化，则触发 change 事件。change 事件的触发可用于检查用户输入的数据。如果输入的值无效，例如在需要数字时中输入了字母，就提示用户，并使用户返回文本框，以改正错误。密码框是文本框的一种变体，它与文本框几乎完全相同，只是将输入的值隐藏起来，并显示为星号。另外，文本框还有 keydown、keypress 和 keyup 事件。
- 接下来讨论的是 textarea 元素，它与文本框类似，但允许输入多行文本。这个元素用开始标记<textarea>和结束标记</textarea>创建，文本框的宽度和高度分别由 cols 和 rows 属性确定，且以字符为单位。wrap 属性指定文本在到达行末时是否自动换行，以及把内容提交给服务器时是否发送换行符。如果没有设置 wrap 属性，或者将其设置为 off，就不自动换行。如果设置为 soft，就在客户端自动换行，但换行符不发送给服务器。如果设置为 hard，就在客户端自动换行，且把换行符发送给服务器。关联的 Textarea 对象与 Text 对象拥有相同的属性、事件和方法。
- 复选框和单选按钮是一起讨论的。它们基本上是同类元素，只不过单选按钮是一种组合元素，即组中一次只能选择一个单选按钮。如果选择另一个单选按钮，就取消对上一个单选按钮的选择。复选框和单选按钮都是用<input/>元素创建的，其 type 属性分别设置为 checkbox 和 radio。如果在<input>标记中加入了 checked，则页面加载时就选中该元素。用相同的名称创建多个单选按钮，就会创建一个单选按钮组。单选按钮的名称实际上引用一个数组，该数组中的每个元素是在表单的该单选按钮组中定义的单选按钮。这些元素有关联的对象 Checkbox 和 Radio。使用这些对象的 checked 属性，可以确定某复选框或单选按钮当前是否选中。这两个对象都有 click 事件，以及通用事件 focus 和 blur。
- 之后讨论的元素是列表框和下拉列表框，它们实际上是相同的选择元素，其 size 属性决定了它是列表框还是下拉列表。<select>标记用于创建这些元素，其 size 属性确定一次可以看到几个选项。如果 size 属性为 1，就创建下拉列表框，而不是列表框。选择元素的每个选项都用<option/>元素创建，或者以后通过 Select 对象的 options 集合属性添加，options 集合是一个类似于数组的结构，包含了该元素的所有 Option 对象。但是，在页面加载后添加元素在 IE7 和其他浏览器中略有区别。Select 对象的 selectedIndex 属性指定选中了哪个选项，接着可以使用该值来访问

options 集合中的对应对象，使用 Option 对象的 value 属性。Option 对象也有 text 和 index 属性，text 是列表中显示的文本，index 是该选项在 Select 对象的 options 集合属性中的位置。可以遍历 options 集合，通过 Select 对象的 length 属性确定其长度。Select 对象具有 change 事件，用户在列表中选择另一项时，将触发该事件。

第 8 章将讨论，一旦在页面中创建了框架集(frameset)，就可以在各个框架(frame)之间访问代码和变量。我们还将介绍如何使用 JavaScript 打开新窗口，以及操作新窗口的方法。你将看到，小测试程序演变成一个基于框架的应用程序。

7.4　习题

本章习题的参考答案在附录 A 中。

习题 1：

使用第 2 章中温度转换例子的代码，创建一个用户界面，并将其连接到已有的代码上，以便用户输入一个华氏温度值，并转换为摄氏温度。

习题 2：

创建一个用户界面，以允许用户根据自己的爱好挑选计算机系统，其原则类似于在因特网上销售计算机的电子商务网站。例如，允许用户选择处理器的类型、速度、内存和硬盘容量，并添加 DVD-ROM、声卡等附件。用户改变其选择时，系统的价格应自动更新，并通过使用警告框或更新文本框的内容，来通知用户所指定的系统的价格。

第 8 章

窗口和框架

前面介绍的都是单个页面，但许多 Web 应用程序都使用框架(frame)分隔浏览器的窗口，就像用玻璃窗格分隔真实的窗户一样。我们也很可能需要建立使用这种框架的网站。JavaScript 允许操纵框架，在一个框架中创建的函数和变量还可以在另一个框架中使用。其优点之一是可以在一个地方保存通用变量和函数，并在许多地方使用它们。本章首先介绍如何对这种框架编写脚本。

访问另一个框架中的变量和函数还有许多其他原因。其中两个重要的原因是代码的模块化，以及在页面之间保存信息。

模块化(modular)是什么意思？在其他编程语言，例如 C、C++或 Visual Basic 中，可以创建一个模块(一个保存通用函数和变量的区域)，并在程序的不同地方重用它。使用框架时，可以把所有的通用函数和变量都放在一个区域，例如顶层框架，这个区域就可以看作代码模块。接着就可以在不同的页面和不同的框架中重复调用这些函数。

如果把通用函数和变量放在定义框架的页面中(即定义框架集的页面)，则以后修改框架中的页面时，框架集页面中定义的变量值保持不变。这是用户浏览网站时保存信息的一个非常有效的方式。另一个优点是在框架集页面中定义的函数只需在浏览器中加载一次，就可以由后续的页面调用，所以页面的加载速度会更快。

本章的第二个主题是如何打开并操纵新的浏览器窗口。新窗口有许多用途。例如，可以在网站的一个新窗口中打开一个“外部”网站，而第一个网站仍是打开的。这里的“外部网站”是指由另一人或公司创建和维护的网站。例如在一个汽车网站上可能需要一个指向外部站点的链接，例如汽车厂商(例如福特或通用汽车)的网站。而把小窗口用作对话框会更有效，用于获得用户的信息。就像可以在框架之间编写脚本一样，也可以在某些窗口之间编写脚本。本章后面将介绍其过程，但首先看看如何在框架之间编写脚本。

8.1 框架和窗口对象

框架是把浏览器窗口分隔为各个窗格的方式，在各个窗格中可以加载不同的 HTML 文

档。在定义框架集的页面中，用<frameset/>和<frame/>元素定义框架。<frameset/>元素包含<frame/>元素，并指定框架在页面上的显示方式。然后使用<frame/>元素指定每个框架，以及要在页面中包含的文档。

第 6 章提到，window 对象表示页面或文档上的浏览器框架。如果页面没有框架，就只有一个 window 对象。但如果页面上有多个框架，则每个框架都对应一个 window 对象。除了框架集中的顶层窗口之外，每个 window 对象都包含在另一个 window 对象中。

描述这个概念的最简单方式是列举一个例子，这个例子将创建 3 个框架，其顶层框架包含了另外两个框架。

试一试　多个框架

在这个多框架的示例中，需要创建 3 个 HTML 文件。第一个是定义框架集的页面。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 1</title>
</head>
<frameset row="50%, *" id="topWindow">
    <frame name="upperWindow" src="ch08_examp1_upper.htm" />
    <frame name="lowerWindow" src="ch08_examp1_lower.htm" />
</frameset>
</html>
```

保存为 ch08_examp1.htm。注意在这个页面中，两个<frame />元素的 src 属性是 ch08_examp1_upper.htm 和 ch08_examp1_lower.htm。下面创建这两个文件。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 1 Upper Frame</title>
    <script type="text/javascript">
    function window_onload()
    {
        alert("The name of the upper frame's window object is " +
            window.name);
        alert("The location of upperWindow's parent is " +
            window.parent.location.href);
    }
    </script>
</head>
<body onload="window_onload()">
    <p>
        Upper Frame
    </p>
```

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

上面的代码块是顶部框架的源页面，名为 upperWindow，需要另存为 ch08_examp1_upper.htm。最后一个页面与它非常类似：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 1 Lower Frame</title>
    <script type="text/javascript">
    function window_onload()
    {
        alert("The name of the lower frame's window object is " +
            window.name);

        alert("The location of lowerWindow's parent is " +
            window.parent.location.href);
    }
    </script>
</head>
<body onload="window_onload()">
    <p>
        Lower Frame
    </p>
</body>
</html>
```

这就是下部框架的源页面，保存为 ch08_examp1_lower.htm。

因为这 3 个页面放在一起，所以 ch08_examp1_upper.htm 和 ch08_examp1_lower.htm 包含在 ch08_examp1.htm 页面中。

把它们加载到浏览器中时，会得到 3 个 window 对象。一个是父 window 对象，包含 ch08_examp1.htm 文件，另外两个是子 window 对象，包含了 ch08_examp1_upper.htm 和 ch08_examp1_lower.htm 文件。两个子 window 对象包含在父 window 对象中，如图 8-1 所示。

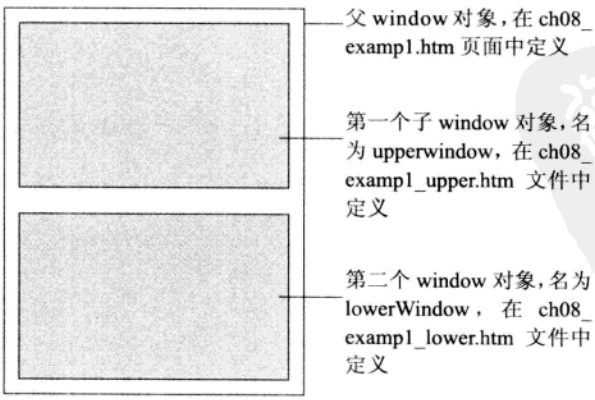

图 8-1

如果框架包含了子框架，则这些子框架对应的 window 对象就是父框架对应的 window 对象的子对象。

在浏览器中加载 ch08_examp1.htm 时，会看到 4 个消息框，如图 8-2 到图 8-5 所示。它们使用 window 对象的属性获取信息，指出了 window 对象在层次结构中的位置。

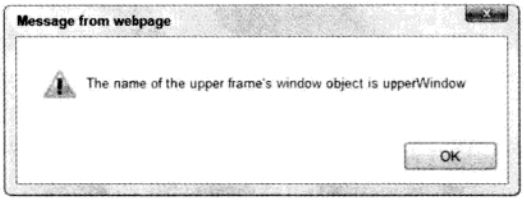

图　8-2

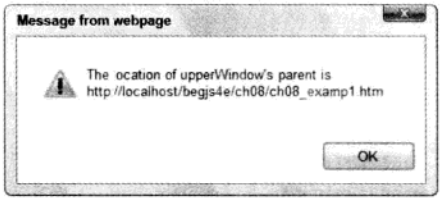

图　8-3

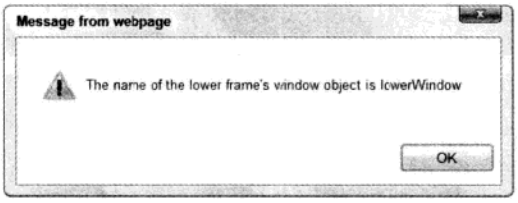

图　8-4

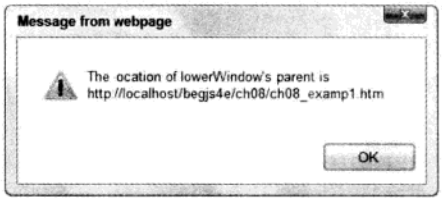

图　8-5

图 8-3 到图 8-5 中的路径因文件在计算机中的存储位置而异。

下面查看定义框架集的页面 ch08_examp1.htm，该页面如下列代码段所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 1</title>
</head>
<frameset rows="50%, *" id="topWindow">
    <frame name="upperWindow" src="ch08_examp1_upper.htm" />
    <frame name="lowerWindow" src="ch08_examp1_lower.htm" />
</frameset>
</html>
```

用<frameset />元素定义框架集。这里使用了两个属性 rows 和 id。rows 属性的值是“50%,*”，表示第一个框架应占据窗口的一半高度，第二个框架应占据剩余的空间。id 属性则指定一个名称，用于引用该页面。

用<frame />元素创建两个子窗口，每个子窗口都包含 name 和 src 属性，name 属性指定使用哪个 window 对象，src 属性指定把哪个页面加载到新建的窗口中。

接下来看看 ch08_examp1_upper.htm 文件。在<body />元素中，把 window_onload()函数关联到 window 对象的 onload 事件处理程序上。在浏览器加载完窗口、窗口中的文档以及文档中的所有对象后，就调用这个事件处理程序。这里非常适合放置初始化代码，或者在页面加载完后、把控制权传送回用户前需要修改数据的代码。

```
<body onload="window_onload()">
```

这个函数在页面顶部的脚本块中定义，如下所示：

```
function window_onload()
{
    alert("The name of the upper frame's window object is " + window.name);
    alert("The location of UpperWindow's parent is " +
        window.parent.location.href);
}
```

window_onload()函数给加载页面的框架使用了 window 对象的两个属性：name 和 parent。name 属性很简单，是框架集页面中定义的名称，在本例中是 upperWindow。

第二个属性 parent 非常有用，它允许访问当前框架的父框架的 window 对象。这表示，可以访问父 window 对象的所有属性和方法。通过这个属性，可以访问父 window 对象中的文档以及父对象定义的其他框架。这里显示的消息框使用 location 对象的 href 属性(location 本身是 window 对象的一个属性)给出了父框架的文件名或 URL。

ch08_examp1_lower.htm 的代码与 ch08_examp1_upper.htm 完全相同，但结果不同，因为我们访问了另一个 window 对象。这次，window 对象的 name 属性是 lowerWindow。但它与 upperWindow 的父 window 对象相同，所以访问 window 对象的 parent 属性时，会引用与 upperWindow 相同的 window 对象。消息框验证了这一点，它显示文件名/URL 或 href 属性，结果匹配 upperWindow 框架中显示的页面的文件名。

注意：

消息的显示顺序在不同类型的浏览器、甚至不同的操作系统上可能不同。在这里，这并不重要，但有时事件的触发顺序很重要，会影响代码的工作方式。应重视这种不兼容性，并在自己的程序中注意这一点。

8.1.1 在框架之间编写代码

每个框架都是一个不同的窗口，都有自己的 window 对象。另外，在框架集定义页面指定的任意框架中，使用 window 对象的 parent 属性，都可以访问框架集页面对应的 window 对象。如果有对父窗口的 window 对象的引用，就可以访问其属性和方法，其方式与访问当前页面的 window 对象相同。还可以访问该页面定义的所有 JavaScript 变量和函数。

试一试　将框架集页面用作模块

下面是一个比较复杂的例子，该例子使用顶层框架跟踪用户浏览网站时的页面。本例将创建 5 个页面，其中 4 个页面几乎相同。第一个需要创建的是框架集定义页面：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 2</title>
```

```
    <script type="text/javascript">
    var pagesVisited = new Array();
    function returnPagesVisited()
    {
        var returnValue = "So far you have visited the following pages\n";
        var pageVisitedIndex;
        var numberOfPagesVisited = pagesVisited.length;
        for (pageVisitedIndex = 0; pageVisitedIndex < numberOfPagesVisited;
            pageVisitedIndex++)
        {
            returnValue = returnValue + pagesVisited[pageVisitedIndex] + "\n";
        }
        return returnValue;
    }

    function addPage(fileName)
    {
        var fileNameStart = fileName.lastIndexOf("/") + 1;
        fileName = fileName.substr(fileNameStart);
        pagesVisited[pagesVisited.length] = fileName;

        return true;
    }
    </script>
</head>
<frameset cols="50%,*">
    <frame name="fraLeft" src="ch08_examp2_a.htm">
    <frame name="fraRight" src="ch08_examp2_b.htm">
</frameset>
</html>
```

把这个页面保存为 ch08_examp2.htm。

注意两个框架的 src 属性初始化为 ch08_examp2_a.htm 和 ch08_examp2_b.htm。还需要创建 ch08_examp2_c.htm 和 ch08_examp2_d.htm，因为用户可以从这 4 个页面中选择要加载到每个框架中的页面。先创建 page_a.htm 页面，如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 2 Page A</title>
    <script type="text/javascript">
    function btnShowVisited_onclick()
    {
        document.form1.txtaPagesVisited.value =
            window.parent.returnPagesVisited();
    }
    </script>
</head>
<body onload="window.parent.addPage(window.location.href)">
    <h2>This is Page A</h2>
    <p>
```

```
            <a href="ch08_examp2_a.htm">Page A</a>
            <a href="ch08_examp2_b.htm">Page B</a>
            <a href="ch08_examp2_c.htm">Page C</a>
            <a href="ch08_examp2_d.htm">Page D</a>
        </p>
    <form name="form1" action=" ">
            <textarea rows="10" cols="35" name="txtaPagesVisited"></textarea>
            <br />
            <input type="button" value="List Pages Visited" name="btnShowVisited"
                onclick="btnShowVisited_onclick()" />
        </form>
    </body>
    </html>
```

把这个页面保存为 ch08_examp2_a.htm。

其他 3 个页面都与 ch08_examp2_a.htm 相同，但页面的标题和元素<h2/>不同，所以只需剪切并粘贴 ch08_examp2_a.htm 中的文本。把显示所加载页面名称的 HTML 改为：

```
<h2>This is Page B</h2>
```

接着保存为 ch08_examp2_b.htm。

同样，创建第三个页面(页面 C)：

```
<h2>This is Page C</h2>
```

保存为 ch08_examp2_c.htm。

最后一个页面也是 ch08_examp2_a.htm 的一个副本，但下面的代码行除外：

```
<h2>This is Page D</h2>
```

保存为 ch08_examp2_d.htm。

在浏览器中加载 ch08_examp2.htm，单击链接，浏览各个页面。接着单击左边框架中的 List Pages Visited 按钮，屏幕应如图 8-6 所示。

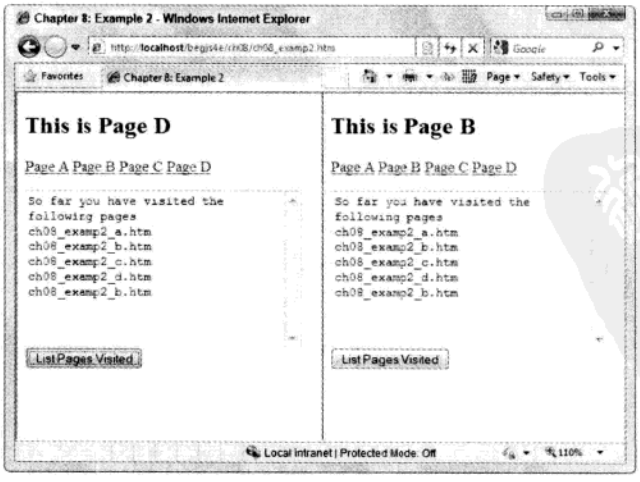

图 8-6

单击左右框架中的链接，导航到新位置上。例如，单击右框架中的 Page C 链接，再单击左框架中的 Page D 链接。单击左框架的 List Pages Visited 按钮，会看到 ch08_examp2_c.htm 和 ch08_examp2_d.htm 都添加到列表中。

一般在加载新页面时，上一个页面中的变量及其值就会丢失，但把框架集页面用作模块时，每个框架加载什么页面并不重要——顶层框架一直处于加载状态，其变量总是保有其值。在这个示例中，无论在每个框架中加载什么页面，顶层框架中的一些全局变量总是跟踪着已访问过的页面，加载到左右框架中的任何页面都可以访问顶层框架的变量和函数。

注意：

加载到框架中的页面来自外部源时，是有限制的，详见本章后面的内容。

先来分析框架集定义页面 ch08_examp2.htm 中的 JavaScript 代码。该页面的开头包含一个脚本块。这个脚本块首先声明了 pagesvisited 变量，把它设置为引用一个新的 Array 对象。这个数组中存储了用户浏览站点时访问的每个页面的文件名。

```
var pagesVisited = new Array();
```

接着定义了两个函数。顾名思义，第一个函数 returnPagesVisited()返回一个字符串，其中包含一个消息和所访问的每个页面。为此，该函数遍历 pagesVisited 数组，在 returnValue 变量中建立消息字符串，然后返回给调用函数。

```
function returnPagesVisited()
{
    var returnValue = "So far you have visited the following pages\n";
    var pageVisitedIndex;
    var numberOfPagesVisited = pagesVisited.length;
    for (pageVisitedIndex = 0; pageVisitedIndex < numberOfPagesVisited;
        pageVisitedIndex++)
    {
        returnValue = returnValue + pagesVisited[pageVisitedIndex]
            + "\n";
    }
    return returnValue;
}
```

第二个函数 addPage()把页面名称添加到 pagesVisited 数组中：

```
function addPage(fileName)
{
    var fileNameStart = fileName.lastIndexOf("/") + 1;
    fileName = fileName.substr(fileNameStart);
    pagesVisited[pagesVisited.length] = fileName;

    return true;
}
```

传送给这个函数的 fileName 参数是所访问页面的完整文件名和路径，所以需要对路径进行修剪，提取出文件名。该字符串的格式是 file:///D:/myDirectory/ch08_examp2_b.htm，

我们只需最后一个/字符后面的内容。所以在第一行代码中，确定/字符的位置，再给它加上 1，因为应从下一个字符开始提取。

接着在下一行代码中使用 String 的 substr()方法，从字符位置 fileNameStart 开始向右提取字符，直到字符串的末尾。注意 substr()方法接受两个参数，分别是需要的起始字符和要提取的字符串长度，但如果没有提供第二个参数，就提取从起始位置到字符串末尾的所有字符。

接着把文件名添加到数组中，数组的 length 属性提供了下一个空白的索引位置。

现在看看各个框架页面 ch08_examp2_a.htm、ch08_examp2_b.htm、ch08_examp2_c.htm 和 ch08_examp2_d.htm。在这些页面中，都创建了一个窗体 form1：

```
<form name="form1" action=" ">
    <textarea rows="10" cols="35" name="txtaPagesVisited"></textarea>
    <br />
    <input type="button" value="List Pages Visited"
        name="btnShowVisited" onclick="btnShowVisited_onclick()" />
</form>
```

其中包含一个 textarea 控件和一个按钮。Textarea 列出了访问过的页面，用户可以单击按钮，以填充<textarea />元素。

加载其中一个页面时，window 对象的 onload 事件处理程序把该页面的名称放在 ch08_examp2.htm 定义的 pagesVisited 数组中。onload 事件处理程序连接到 ch08_examp2.htm 中创建的 addPage()函数上。在页面的<body />元素中连接了事件处理程序的代码，如下所示：

```
<body onload="window.parent.addPage(window.location.href)">
```

与页面中的其他对象一样，在页面中声明的所有函数都包含在该页面的 window 对象中；因为 window 是全局对象，所以不需要在变量或函数的名称前面加上 window。

但这次，函数不在当前页面中，而在 ch08_examp2.htm 页面上。包含这个页面的 window 是包含当前页面的 window 的父对象。因此需要使用 window 对象的 parent 属性引用父框架的 window 对象。window.parent 代码给出了指向 ch08_examp2.htm 的 window 对象的引用。通过这个引用，就可以访问 ch08_examp2.htm 中包含的变量和函数。指定了要引用哪个 window 对象后，就只需添加所调用的函数名即可，这里是 addPage()函数。给这个函数传送 location.href 字符串，作为它的一个参数值，其中包含页面的完整路径和文件名。

如前所述，页面上的按钮把其 onclick 事件处理程序连接到 btnShowVisited_onclick()函数上。该函数在页面开始处定义：

```
function btnShowVisited_onclick()
{
    document.form1.txtaPagesVisited.value =
        window.parent.returnPagesVisited();
}
```

在这个函数中，调用了父 window 对象的 returnPagesVisited()函数，如前所述，这个函数返回一个字符串，其中包含所访问页面的列表。textarea 对象的 value 属性设置为这个文本。

这就讨论完了框架页面中的代码。可以看出，代码并不多，因为所有的公用函数都放在框架集页面中了。这种代码重用方式不仅可以减少输入量，而且所有函数都放在一个地方。如果函数有错误，则修改一个页面上的错误，也就意味着为使用该函数的其他所有页面修改了该错误。当然，这仅适用于放在一个地方的公用函数，专用于某个页面的函数和从来不在外部使用的函数最好保存在此页面上。

8.1.2　框架之间的代码访问

前面介绍了子窗口如何访问父窗口的变量和函数，框架集中的框架如何互相访问？

本章前面有一个简单的示例，所以这里讨论一个较复杂的例子。该示例创建的页面如图 8-7 所示。

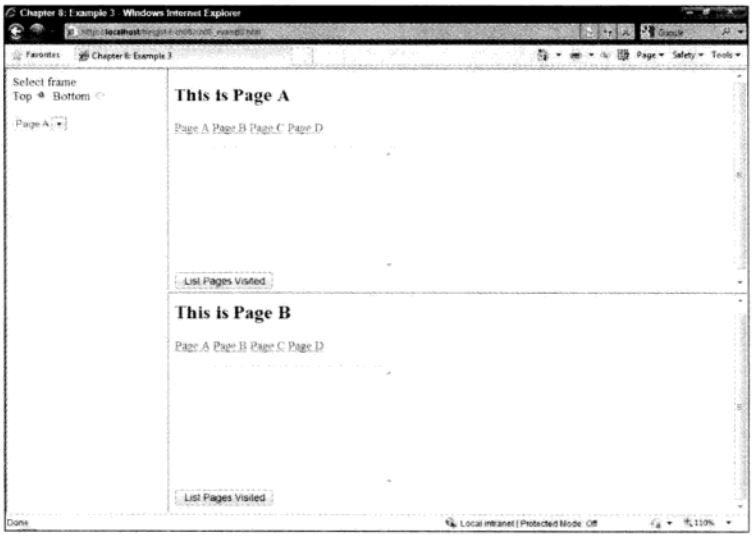

图　8-7

框架布局如图 8-8 所示。文本标签指定了在<frameset/>和<frame/>元素中给定的每个框架的名称，但顶层框架例外，它仅是位于框架集层次结构顶部的窗口。

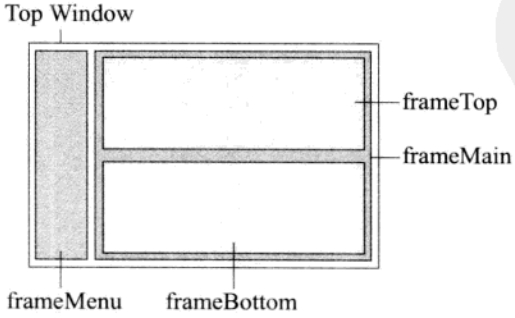

图　8-8

理解这种基于框架的网页的最简单方式是与家族关系做类比，家族关系可以用家族树表示。如果用这种方式表示框架集，结果就应如图 8-9 所示。

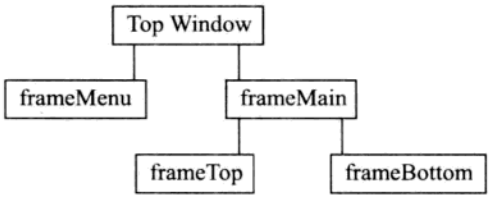

图 8-9

从图中可以看出，右侧的底层框架是 frameBottom，其父框架是 frameMain，frameMain 的父框架是顶层窗口。所以，如果要从 frameBottom 窗口中访问顶层窗口的函数，就需要访问 frameBottom 的父框架的父框架的 window 对象。window 对象拥有 parent 属性，它引用该 window 对象的父窗口。下面就使用该属性创建代码，访问顶层窗口中的函数 myFunction()。

```
window.parent.parent.myFunction();
```

现在进行分析。下面的代码引用了运行代码的窗口的父 window 对象：

```
window.parent
```

这行代码在 frameBottom 中，所以 window.parent 是 frameMain。但是我们需要访问的是顶层窗口，即 frameMain 的父窗口，所以给上面的代码添加 parent 属性：

```
window.parent.parent
```

现在就有了一个对顶层窗口的引用。最后，在表达式的末尾添加 myFunction()，以调用它：

```
window.parent.parent.myFunction();
```

如果要在 frameBottom 中访问 frameMenu 的 window 对象，该怎么办？其实前面已经编写出了访问该对象所需的大部分代码。window.parent.parent 表示顶层窗口，而现在需要访问该窗口的子 window 对象 frameMenu。可以采用 3 种方式，它们的结果都相同。

可以在 window 对象的 frames 集合属性中使用该对象的索引，如下所示：

```
window.parent.parent.frames[0]
```

其次，还可以在 frames 集合中使用该对象的名称，如下所示：

```
window.parent.parent.frames["frameMenu"]
```

最后，可以像处理其他 window 对象那样使用该对象的名称，直接引用它：

```
window.parent.parent.frameMenu
```

第三种方法最简单，除非不知道框架的名称，才需要通过 frames 集合中的索引值来访问它，或者需要遍历每个子框架。

window.parent.parent.frameMenu 提供了与 frameMenu 关联的 window 对象的引用，所以要访问函数 myFunction()或变量 myVariable，只需输入一行代码：

```
window.parent.parent.frameMenu.myFunction
```

或者：

```
window.parent.parent.frameMenu.myVariable
```

如果要在框架内访问的不是页面中的函数或变量，而是窗体上的控件或该页面上的链接，该怎么办？假定要在 frameBottom 页面上访问控件 myControl，该控件在 frameMenu 页面的窗体 myForm 上。

在 frameBottom 中，使用 window.parent.parent.frameMenu 可以引用 frameMenu 的 window 对象，但如何引用窗体？

这基本上与在脚本中访问同一个页面中的窗体一样，只是需要引用的不是该页面的 window 对象，而是要访问的页面 frameMenu 的 window 对象。

一般只编写 document.myForm.myControl.value，因为 window 是一个全局对象。严格来说，应是 window.document.myForm.myControl.value。

现在我们访问的是另一个窗口，所以只需引用该窗口，再使用相同的代码。如果要在 frameBottom 中访问 myControl 的 value 属性，就需要如下代码：

```
window.parent.parent.frameMenu.document.myForm.myControl.value
```

可以看出，引用其他框架的代码相当长，此时最好把引用保存到变量中。例如，如果多次访问 myForm，可以编写如下代码：

```
var myFormRef = window.parent.parent.frameMenu.document.myForm;
```

之后，就可以编写：

```
myFormRef.myControl.value;
```

来替代：

```
window.parent.parent.frameMenu.document.myForm.myControl.value;
```

1. top 属性

从框架层次结构靠近底层的框架访问顶层的窗口时，使用 window 对象和 parent 属性会比较麻烦。另一种方法是使用 window 对象的 top 属性，它返回对框架层次结构中顶层窗口的 window 对象的引用。在当前示例中，就是顶层窗口。

例如，在刚才的示例中，下面的代码：

```
window.parent.parent.frameMenu.document.myForm.myControl.value;
```

可以改写为：

```
window.top.frameMenu.document.myForm.myControl.value;
```

由于 window 是全局对象，所以可以缩短上述代码：

```
top.frameMenu.document.myForm.myControl.value;
```

那么，何时应使用 top，何时应使用 parent？

这两个属性各有优缺点。parent 属性允许指定与当前窗口对应的对象，该窗口上一层的窗口是 window.parent，其父窗口是 window.parent.parent，依此类推。top 属性则通用得多，无论使用什么框架集布局，它都是顶层窗口。总是有 top 窗口，但 parent.parent 不一定总是有效。如果把希望能在框架集的任意页面访问的所有全局函数和变量都放在顶层窗口中，则无论下面的框架集如何变化，window.top 都总是有效的，而 parent 属性依赖于其上的框架集结构。但如果有人把网站加载到他自己的一个框架集页面中，则 top 窗口就不是原来的顶层窗口了，而是这个人的框架集页面，且 window.top 不再有效。

一个技巧是检查 top 窗口是否包含自己的页面，如果不包含，就重载 top 页面，并指定要加载哪个页面作为 top 页面。例如，检查 top 页面的文件名是否匹配期望的文件名。window.top.location.href 给出了文件名和路径——如果它们不匹配期望的页面，就使用 window.top.location.replace("myPagename.htm")加载正确的 top 页面。但如后面所述，如果他人把自己的页面加载到他创建的框架集中，就会出问题——此时应使用同源策略(same-origin policy)，详见本章后面的内容。

试一试　框架的脚本编程

下面根据 ch08_examp2.htm 中的框架集，把前面所学的框架和脚本编程内容放在一个示例中。本例将重用本章上一个示例中的许多页面和代码。

要创建的第一个页面是顶层窗口页面，下面突出显示的代码行是对 ch08_examp2.htm 的修改：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 3</title>
    <script type="text/javascript">
    var pagesVisited = new Array();
    function returnPagesVisited()
    {
        var returnValue = "So far you have visited the following pages\n";
        var pageVisitedIndex;
        var numberOfPagesVisited = pagesVisited.length;
        for (pageVisitedIndex = 0; pageVisitedIndex < numberOfPagesVisited;
            pageVisitedIndex++)
        {
            returnValue = returnValue + pagesVisited[pageVisitedIndex] + "\n";
        }
        return returnValue;
```

```
        }

        function addPage(fileName)
        {
            var fileNameStart = fileName.lastIndexOf("/") + 1;
            fileName = fileName.substr(fileNameStart);
            pagesVisited[pagesVisited.length] = fileName;

            return true;
        }
        </script>
    </head>
    <frameset cols="200,*">
        <frame name="frameMenu" src="ch08_examp3_menu.htm">
        <frame name="frameMain" src="ch08_examp3_main.htm">
    </frameset>
    </html>
```

可以看出，本例重用了 ch08_examp2.htm 中的许多代码，所以可以剪切并粘贴 ch08_examp2.htm 的脚本块。并修改上述突出显示的代码。把这个页面保存为 ch08_examp3.htm。

接着创建要加载到 frameMenu 中的页面 ch08_examp3_menu.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 3 Menu</title>
    <script type="text/javascript">
    function choosePage_onchange()
    {
        var choosePage = document.form1.choosePage;
        var windowobject;
        if (document.form1.radFrame[0].checked == true)
        {
            windowobject = window.parent.frameMain.frameTop;
        }
        else
        {
            windowobject = window.parent.frameMain.frameBottom;
        }
        windowobject.location.href =
            choosePage.options[choosePage.selectedIndex].value;
        return true;
    }
    </script>
</head>
<body>
    <form name="form1" action=" ">
        Select frame
        <br />
        Top
```

```
        <input name="radFrame" checked="checked" type="radio" />
        Bottom
        <input name="radFrame" type="radio" />
        <br />
        <br />
        <select name="choosePage" onchange="choosePage_onchange()">
            <option value="ch08_examp3_a.htm">Page A</option>
            <option value="ch08_examp3_b.htm">Page B</option>
            <option value="ch08_examp3_c.htm">Page C</option>
            <option value="ch08_examp3_d.htm">Page D</option>
        </select>
    </form>
</body>
</html>
```

保存为 ch08_examp3_menu.htm。

frameMain 框架包含一个页面，该页面只是 frameTop 和 frameBottom 页面的框架集。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 3 Main</title>
</head>
<frameset rows="50%,*">
   <frame name="frameTop" src="ch08_examp3_a.htm">
   <frame name="frameBottom" src="ch08_examp3_b.htm">
</frameset>
</html>
```

保存为 ch08_examp3_main.htm。

接下来的 4 个页面基本上是第二个示例中 ch08_examp2_a.htm、ch08_examp2_b.htm、ch08_examp2_c.htm 和 ch08_examp2_d.htm 这 4 个页面的副本。但需要做几处修改，如下面突出显示的代码所示(这些页面是相同的，只是页面上显示的文本不同，所以下面仅显示了对 ch08_examp2_a.htm 所做的修改。其他页面的修改与此类似)。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 3 Page A</title>
    <script type="text/javascript">
    function btnShowVisited_onclick()
    {
        document.form1.txtaPagesVisited.value =
            window.top.returnPagesVisited();
    }

    function setFrameAndPageControls(linkIndex)
```

```
        {
            var formobject = window.parent.parent.frameMenu.document.form1;
            formobject.choosePage.selectedIndex = linkIndex;

            if (window.parent.frameTop == window.self)
            {
                formobject.radFrame[0].checked = true;
            }
            else
            {
                formobject.radFrame[1].checked = true;
            }

            return true;
        }
        </script>
    </head>
    <body onload="window.top.addPage(window.location.href)">
        <h2>This is Page A</h2>
        <p>
            <a href="ch08_examp3_a.htm" name="pageALink"
                onclick="return setFrameAndPageControls(0)">Page A</a>
            <a href="ch08_examp3_b.htm" name="pageBLink"
                onclick="return setFrameAndPageControls(1)">Page B</a>
            <a href="ch08_examp3_c.htm" name="pageCLink"
                onclick="return setFrameAndPageControls(2)">Page C</a>
            <a href="ch08_examp3_d.htm" name="pageDLink"
                onclick="return setFrameAndPageControls(3)">Page D</a>
        </p>
        <form name="form1" action=" ">
            <textarea rows="10" cols="35" name="txtaPagesVisited"></textarea>
            <br />
            <input type="button" value="List Pages Visited" name="btnShowVisited"
                onclick="btnShowVisited_onclick()" />
        </form>
    </body>

    </html>
```

把页面保存为 ch08_examp3_a.htm、ch08_examp3_b.htm、ch08_examp3_c.htm 和 ch08_examp3_d.htm。

把 ch08_examp3.htm 加载到浏览器中，屏幕应如图 8-7 所示。

单选按钮允许用户确定用哪个框架浏览新页面。用户在下拉列表中改变当前选中的页面时，该页面就会加载到通过单选按钮选中的框架内。

如果在 frameTop 和 frameBottom 框架中使用页面上的链接来导航，所选框架左侧的 frameMenu 中的 Selectframe 单选按钮和下拉列表就会自动更新为刚才浏览的页面和框架。注意如本例所示，如果用户把 ch08_examp3_a.htm 加载到一个框架中，则选择列表不允许在另一个框架中加载这个页面。可以改进这个示例，即添加一个按钮，把当前选中的页面加载到选中的框架中。

List Pages Visited 按钮显示了已访问过的页面列表，因为它们和上一个示例中的对应按钮相同。

前面讨论过 ch08_examp3.htm 中定义顶层窗口的代码如何工作，它非常类似于上一个示例。快速查看一下<frameset/>元素，可以看出，窗口的名称在<frame/>元素的名称中定义。

```
<frameset cols="200,* ">
    <frame name="frameMenu" src="ch08_examp3_menu.htm">
    <frame name="frameMain" src="ch08_examp3_main.htm">
</frameset>
```

还要注意，<frameset/>元素的 cols 属性设置为"200,*"，这表示第一个框架占用 200 像素的列宽，另一个框架占用剩余的宽度。

下面详细论述包含 ch08_examp3_menu.htm 的 frameMenu 框架。在页面的顶部是主脚本块。该脚本块包含 choosePage_onchange()函数，这个函数连接到页面下部的选择框的 onchange 事件处理程序上。选择框的 options 包含了各个页面的 URL。

该函数首先定义了两个变量，一个是 choosePage，它是页面上 choosePage Select 对象的快捷引用。

```
function choosePage_onchange()
{
    var choosePage = document.form1.choosePage;
    var windowobject;
```

接着，if...else 语句把 windowobject 变量设置为引用用户在 radFrame 单选按钮组中选中的框架的 window 对象。

```
if (document.form1.radFrame[0].checked == true)
{
    windowobject = window.parent.frameMain.fraTop;
}
else
{
    windowobject = window.parent.frameMain.fraBottom;
}
```

如前所述，使用 window.parent 就可以引用父 window 对象。这里使用 window.top 也可以进行引用。于是 window.parent.frameMain 引用了 frameMain 框架的 window 对象。最后，根据要浏览的框架，使用 window.parent.frameMain.frameTop 或 window.parent.frameMain.frameBottom 来引用 frameMain 包含的 frameTop 或 frameBottom 的 window 对象。

有了要浏览的框架的 window 对象引用之后，就可以把其 location.href 属性改为所选下拉列表项的值，让框架加载该页面。

```
    windowobject.location.href =

        choosePage.options[choosePage.selectedIndex].value;
    return true;
}
```

如前所述，ch08_examp3_main.htm 只是 frameTop 和 frameBottom 的框架集定义页面。现在分析加载到 frameTop 和 frameBottom 中的页面。因为它们完全相同，所以只讨论 ch08_examp3_a.htm。

首先分析顶部的脚本块，其中包含两个函数 btnShowVisited_onclick()和 setFrameAnd PageControls()。上一个示例讨论过 btnShowVisited_onclick()。

```
function btnShowVisited_onclick()
{
    document.form1.txtaPagesVisited.value = window.top.returnPagesVisited();
}
```

但是，因为框架集的布局有变化，所以需要修改代码。前面的 returnPagesVisited()函数放在父窗口中，现在它移到顶层窗口中。可以看出，只需把引用从 window.parent. return PagesVisited();改为 window.top.returnPagesVisited();即可。

在前面的示例中，parent 窗口也是 top 窗口，所以如果一开始以这种方式编写代码，这里就不需要进行任何修改。通常最好把所有的通用函数都放在顶层的框架集页面中。这样，即使以后改变了框架集布局，所有的引用也可以是 window.top。

这个页面中的新函数是 setFrameAndPageControls()，它连接到页面下部定义的链接的 onclick 事件处理程序上。这个函数的作用是，如果用户使用链接(而不是 frameMenu 窗口中的控件)导航到另一个页面上，这些控件就会更新，以反映用户所做的操作。

首先设置 formobject 变量，以引用 frameMenu 页面中的 form1，如下所示：

```
function setFrameAndPageControls(linkIndex)
{
    var formobject = window.parent.parent.frameMenu.document.form1;
```

下面进行分析：

```
window.parent
```

引用 frameMain 对应的 window 对象。沿着层次结构向上移动，使用下面的代码引用顶层窗口的 window 对象：

```
window.parent.parent
```

这是正确的。可以使用 window.top 来替代，而且 window.top 是比较好的方法。这里使用较长的代码只是为了演示层次结构的工作方式。

现在沿着层次结构向下移动，在树图的另一端，引用 frameMenu 的 window 对象。

```
window.parent.parent.frameMenu
```

最后，我们只对窗体及其控件感兴趣，所以用如下代码引用该对象：

```
window.parent.parent.frameMenu.document.form1
```

有了对该窗体的引用后，就可以使用它，就好像在 frameMenu 中编码一样。

该函数的 linkIndex 参数指定单击了 4 个链接中的哪一个，函数的下一行代码将使用这

个值，在 frameMenu 的窗体上设置下拉列表框中选择的选项。

```
formobject.choosePage.selectedIndex = linkIndex;
```

if...else 语句把 frameMenu 的单选按钮组设置为用户刚才单击的框架，但如何确定这是哪个框架？

```
if (window.parent.frameTop == window.self)
{
    formobject.radFrame[0].checked = true
}
else
{
    formobject.radFrame[1].checked = true
}
```

检查当前的 window 对象是否与 frameTop 的 window 对象相同。为此使用 window 对象的 self 属性和 window.parent.frameTop，self 属性返回对当前 window 对象的引用，window.parent.frameTop 返回对 frameTop 的 window 对象的引用。如果两者相等，说明当前窗口是 frameTop。此时，frameMenu 框架中的 radFrame 单选按钮组会选中其第一个单选按钮。否则，应选中表示 frameBottom 的另一个单选按钮。

在函数中，最后应返回 true。注意这个函数连接到 A 对象上，所以返回 false 会取消链接的动作，而 true 允许继续执行，这是我们期望的操作。

```
    return true;
}
```

2. IFrames 的脚本编写

内联框架(iframes)是 Microsoft 在 Internet Explorer (IE) 3 中引入的，是 HTML 4 中 HTML 标准的一部分。它们是比较独特的元素，因为通过它们可以给网页添加框架，而无需使用框架集，所以把内联框架添加到页面中会简单许多。例如：

```
<iframe name="myIFrame" src="child_frame.htm" />
```

这行 HTML 把框架 myIFrame 添加到页面中，该框架加载了 child_frame.htm 文件。可以看出，这简化了 JavaScript。在加载到其中的页面上访问 iframe 的 document 对象是很简单的。例如：

```
window.myIFrame.document.bgColor = "red";
```

这显然非常类似于框架集中的常规框架(也可以使用 frames 集合，例如 window. frames ["myIFrame"])。在 iframe 中访问父 window 对象也很类似：使用 parent 属性，例如：

```
window.parent.document.bgColor = "yellow";
```

8.2　打开新窗口

本章前面介绍了框架，并在框架之间编写了脚本。本节会略微改变方向，讨论如何打开其他浏览器窗口。

为什么要打开新窗口？因为新窗口在各种情况下都很有用，例如：

- 希望有一个页面包含网站的链接，单击一个链接，会打开一个包含新网站的新窗口。
- 新窗口可以用于显示信息。例如，如果有一个包含产品的页面，用户就可以单击产品图像，打开一个新的小窗口，列出该产品的信息。如果从现有的窗口导航到包含产品信息的新页面上，再要求用户单击“后退”，返回产品列表，会比较麻烦。本章后面将创建一个示例来演示这种情形。
- 对话框窗口非常适用于获取用户的信息，但过度使用会令用户生气。

注意：

所有现代浏览器的最新版本都包含阻止弹出窗口的功能。默认情况下，浏览器会阻止弹出在页面加载时自动创建的新窗口。但一般不会阻止在用户必须执行某个操作(例如单击链接或按钮)时打开的窗口，但用户可以修改浏览器设置，阻止弹出这种窗口。

8.2.1　打开新的浏览器窗口

window 对象的 open()方法会打开新窗口，它接受 3 个参数，其中第三个参数是可选的。该方法返回对新浏览器窗口的 window 对象的引用。

open()方法的第一个参数是要在新窗口中打开的页面的 URL。但可以给这个参数传送空字符串，得到一个空白页面，再使用 document.write()方法在新窗口中动态插入 HTML。本章后面会介绍一个例子。

第二个参数是要分配给新窗口的名称。这不是用于脚本编程的名称，而用于超链接和窗体等的 target 属性。例如，如果把这个参数设置为 myWindow，把最初页面上一个超链接的 target 属性设置为相同的值(如下面的代码所示)，则单击该超链接会打开新窗口。

```
<a href="test3.htm" target="myWindow">Test3.htm</a>
```

这表示用户单击链接时，test3.htm 会加载到新窗口，而不是当前窗口中。<form />元素的 target 属性也是这样。对于该属性，如果从最初的窗口中提交表单，服务器的响应就会显示在新窗口中。

打开新窗口时，该窗口会默认带一些属性，例如 width 和 height，以及浏览器窗口的一般特性，包括位置输入字段和带导航按钮的菜单栏等。

open()方法的第三个参数可用于指定 width 和 height 属性的值。另外，因为浏览器窗口的大多数特性都默认为关闭，所以可以使用 open()方法的第三个参数打开它们。稍后将详细介绍浏览器窗口的特性。

下面先介绍一个打开基本窗口所需的代码示例。这个窗口命名为 myWindow，其 width 和 height 属性都设置为 250 像素。新窗口应打开 test2.htm 页面。

```
var newWindow = window.open("test2.htm","myWindow","width=250,height=250");
```

test2.htm 传送为第一个参数，它是要打开的页面的 URL。窗口在第二个参数中命名为 myWindow。在第三个参数中，把 width 和 height 属性都设置为 250 像素。

还要注意，newWindow 变量设置为 open()方法的返回值，即新打开窗口的 window 对象的引用。现在就可以使用 newWindow 操作新窗口，使用 newWindow.document 属性访问其中包含的 document 了。使用这个引用可以执行与处理框架及其 window 对象所能进执行的所有操作。例如，如果要修改新窗口包含的 document 的背景色，就可以输入下面的代码：

```
newWindow.document.bgColor = "red";
```

如何关闭刚才打开的窗口？只需使用 window 对象的 close()方法，如下所示：

```
newWindow.close();
```

试一试　打开新窗口

下面是前面提及的产品页面示例，单击一个产品会打开一个窗口，其中列出该产品的信息。下面使用两本 Wrox 的图书来作为例子。在这个页面上只有两个产品，但可将这个例子扩展为一个联机图书的目录。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 4</title>
    <script type="text/javascript">
    var detailsWindow;
    function showDetails(bookURL)
    {
        detailsWindow = window.open(bookURL, "bookDetails",
            "width=400,height=350");
        detailsWindow.focus();
    }
    </script>
</head>
<body>
    <h2>Online Book Buyer</h2>
    <p>
        Click any of the images below for more details
    </p>
    <h4>Professional Ajax</h4>
    <p>
        <img src="pro_ajax.jpg" alt="Professional Ajax, 2nd Edition" border="0"
            onclick="showDetails('pro_ajax_details.htm')" />
    </p>
    <h4>Professional JavaScript for Web Developers</h4>
    <p>
```

```
        <img src="pro_js.jpg" alt="Professional JavaScript, 2nd Edition"
            border="0" onclick="showDetails('pro_js_details.htm')" />
    </p>
</body>
</html>
```

将这个页面保存为 ch08_examp4.htm。还需要创建两个图像，命名为 pro_ajax.jpg 和 pro_js.jpg。也可以在下载代码中找到这些文件。

注意，如果用户禁用了 JavaScript，窗口就不会打开——这实际上会使网页崩溃。要解决这个问题，可以用<a/>元素把<img/>元素括起来，把 href 属性指定为图书的详细页面，再使用<a/>元素的 onclick 事件处理程序，在启动新窗口后返回 false，如下所示：

```
<a href="pro_ajax_details.htm"
 onclick="showDetails(this.href); return false; ">
    <img src="pro_ajax.jpg" alt="Professional Ajax" />
</a>
```

在支持 JavaScript 的浏览器中，单击链接会打开一个包含 pro_ajax_ details.htm 页面的新窗口，因为 onclick 处理程序返回 false，所以浏览器不会把主窗口导航到链接的 href 属性定义的页面上。但是，在不支持 JavaScript 的浏览器上，浏览器会忽略 JavaScript，不执行链接的 onclick 事件处理程序中的代码，因此会把用户的浏览器导航到图书详细页面上，因为该详细页面在 href 属性中定义。

现在需要创建两个详细页面，它们都是普通的 HTML。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Professional ASP.NET 2.0</title>
</head>
<body>
    <h3>Professional Ajax, 2nd Edition</h3>
    <strong>Subjects</strong><br />
    Ajax<br />
    Internet<br />
    JavaScript<br />
    ASP.NET<br />
    PHP<br />
    XML<br />
    <hr color="#cc3333" />
    <h3>Book overview</h3>
    <p>
        A comprehensive look at the technologies and techniques used in Ajax,
        complete with real world examples and case studies. A must have for
        any Web professional looking to build interactive Web sites.
    </p>
</body>
</html>
```

保存为 pro_ajax_details.htm。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Professional JavaScript</title>
</head>
<body>
    <h3>Professional JavaScript, 2nd Edition</h3>
    <strong>Subjects</strong><br />
    ECMAScript<br />
    Internet<br />
    JavaScript
    <br />
    XML and Scripting<br />
    <hr color="#cc3333" />
    <p>
        This book takes a comprehensive look at the JavaScript language
        and prepares the reader in-depth knowledge of the languages.
    </p>
    <p>
        It includes a guide to the language - when where and how to get
        the most out of JavaScript - together with practical case studies
        demonstrating JavaScript in action. Coverage is bang up-to-date,
        with discussion of compatability issues and version differences,
        and the book concludes with a comprehensive reference section.
    </p>
</body>
</html>
```

最后这个页面保存为 pro_js_details.htm。

把 ch08_examp4.htm 加载到浏览器中，单击两个图像中的任一个。在现有的浏览器窗口上面就会显示一个列出图书信息的新窗口。单击另一本书的图像，新窗口就会被列出该书信息的窗口替代。

pro_ajax_details.htm 和 pro_js_details.htm 文件都是普通的 HTML 文件，所以这里不介绍它们。但在 ch08_examp4.htm 中，有一些脚本动作，下面讨论它们。

在页面顶部的脚本块中，首先定义了 detailsWindow 变量。

```
var detailsWindow;
```

接着是打开新窗口的函数：

```
function showDetails(bookURL)
{
    detailsWindow = window.open(bookURL,"bookDetails", "width=400,height=350");
    detailsWindow.focus();
}
```

这个函数连接到在页面后面显示的图书图像的 onclick 事件处理程序上。参数 bookURL 由 onclick 事件处理程序中的代码传送，其值是 pro_ajax_details.htm 或 pro_js_details.htm。

使用 window.open()方法创建新窗口，把 bookURL 参数传送为要打开的 URL，把 bookDetails 传送为要应用于新窗口的名称。如果已存在该窗口，就不会打开另一个新窗口，而是把现有的窗口导航到所传送的 URL 上。这是因为给每本书打开窗口时，都使用了相同的名称(bookDetails)。如果使用不同名称，就会打开新窗口。

在 detailsWindow 变量中存储对刚才创建的 window 对象的引用，就可以访问其方法和属性了。在下一行代码上，使用 window 对象(用 detailsWindow 引用)把焦点设置到新窗口上——否则，多次单击主窗口中的同一个图像时，新窗口就会显示在现有窗口的后面。

尽管给每个图像的 onclick 事件处理程序使用了相同的函数，但传送的一个参数不同，即传送了每本书的详细页面的 URL。

```
<h4>Professional Ajax</h4>
<p>
    <img src="pro_ajax.jpg" alt="Professional Ajax, 2nd Edition" border="0"
        onclick="showDetails('pro_ajax_details.htm')" />
</p>
<h4>Professional JavaScript for Web Developers</h4>
<p>
    <img src="pro_js.jpg" alt="Professional JavaScript, 2nd Edition" border="0"
        onclick="showDetails('pro_js_details.htm')" />
</p>
```

1. 在新窗口中添加 HTML

前面提到，可以把一个空字符串传送给 window 对象的 open()方法，作为第一个参数，再使用 HTML 给页面写入信息。下面分析如何完成这个写入操作。

首先需要把一个空值赋予第一个参数(它指定要加载的文件名)，打开一个空窗口。

```
var newWindow = window.open("","myNewWindow","width=150,height=150");
```

接着就可以打开窗口的 document，接收 HTML：

```
newWindow.document.open();
```

在打开新窗口时，这不是必须的，因为页面是空白的，但对于已包含 HTML 的文档而言，这样做可以清除已有的 HTML，清空页面，以便写入信息：

现在可以使用 document.write()方法写入任意有效的 HTML：

```
newWindow.document.write("<h4>Hello</h4>");
newWindow.document.write("<p>Welcome to my new little window</p>");
```

每次使用 write()方法时，文本都会添加到已有内容的后面，直到使用 document.close()方法关闭文档为止：

```
newWindow.document.close();
```

如果再次使用 document.write()方法，所传送的文本就会替代已有的 HTML，而不是添加进去。

2. 给窗口添加特性

如前所述，window.open()方法带 3 个参数，本节介绍其中的第三个参数。使用第三个参数可以控制新建窗口的大小、在屏幕上的初始位置、用户能否重置其大小以及窗口是否有工具栏等。

菜单栏、状态栏和工具栏等特性可以打开或关闭，其中 yes 或 1 表示打开，no 或 0 表示关闭。如果不指定这些值，还可以通过包含它们的名称，来打开它们。

表 8-1 的选项列表并不完整，而且并不是所有的选项都可用于 IE 和 Firefox 浏览器。

表 8-1

窗 口 特 性	可能的取值	描　　述
copyHistory	yes 或 no	当打开新窗口时复制当前窗口的历史记录
directories	yes 或 no	显示目录按钮
height	整数	新窗口的高度
left	整数	新窗口距屏幕左边界的距离
location	yes 或 no	显示地址文本字段
menubar	yes 或 no	显示菜单栏
resizable	yes 或 no	打开新窗口之后，允许用户重置新窗口的大小
scrollbars	yes 或 no	如果页面太大，在新窗口中放不下，就显示滚动条
status	yes 或 no	显示状态栏
toolbar	yes 或 no	显示工具栏
top	整数	新窗口距屏幕顶部的距离
width	整数	新窗口的宽度，以像素为单位

如前所述，第三个参数是可选的。如果不包含它，则所有的窗口特性都默认为 yes，但窗口的大小和位置属性除外，它们默认为预置的值。例如，如果使用下面的代码，窗口就如图 8-10 所示。

图　8-10

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <script type="text/javascript">
    var newWindow;
    newWindow = window.open("","myWindow");
    </script>
</head>
<body>
</body>
</html>
```

注意：

图 8-10 是 IE 8 中的窗口。默认的 UI 隐藏了菜单和工具栏。所以只要默认设置起作用，打开的窗口就不显示菜单和工具栏。

但是，只要指定了一个特性，其他所有特性(窗口大小和位置属性除外)就都默认设置为 no。例如，尽管定义了窗口的大小，但下面的代码生成了没有特性的窗口，如图 8-11 所示。

```
var newWindow = window.open("","myWindow","width=200,height=120")
```

较大的窗口是初始页面，其上较小的窗口(如图 8-11 所示)是弹出窗口。

图　8-11

下面是另一个例子，它创建了可重置大小的 250*250 像素的窗口，带有位置字段和菜单栏：

```
var newWindow = window.open("","myWindow",
                    "width=250,height=250,location,menubar,resizable");
```

但注意，切不可在特性字符串中包含空格，否则某些浏览器会认为该字符串无效，而忽略其设置。

8.2.2 窗口之间的脚本编程

前面简要介绍了如何使用 window.open()方法的返回值操作新窗口的属性和方法，访问其 document 对象。下面讨论新打开的窗口如何访问打开它的窗口(源窗口)，以及如何像框架那样使用源窗口的函数。

要访问打开新窗口的源窗口的 window 对象，关键是 window 对象的 opener 属性，它返回对源窗口的 window 对象的引用。所以下面的代码会把 opener 窗口的背景色改为红色：

```
window.opener.document.bgColor = "red";
```

该引用的用法与在框架中使用 window.parent 和 window.top 属性类似。

试一试　窗口之间的脚本编程

下面例子将打开一个新窗口，在新窗口中访问源窗口中的表单。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 5</title>
    <script type="text/javascript">
    var newWindow;
    function btnOpenWin_onclick()
    {
        var winTop = (screen.height / 2) - 125;
        var winLeft = (screen.width / 2) - 125;
        var windowFeatures = "width=250,height=250, ";
        windowFeatures = windowFeatures + "left=" + winLeft + ",";
        windowFeatures = windowFeatures + "top=" + winTop;
        newWindow = window.open("ch08_examp5_popup.htm", "myWindow",
            windowFeatures);
    }
    function btnGetText_onclick()
    {
        if (typeof (newWindow) == "undefined" || newWindow.closed == true)
        {
            alert("No window is open");
        }
        else
        {
            document.form1.text1.value = newWindow.document.form1.text1.value;
        }
    }

    function window_onunload()
    {
```

```
        if (typeof (newWindow) != "undefined" && newWindow.closed == false)
        {
            newWindow.close();
        }
    }
    </script>
</head>
<body onunload="window_onunload()">
    <form name="form1" action=" ">
        <input type="button" value="Open newWindow" name="btnOpenWin"
            onclick="btnOpenWin_onclick()" />
        <p>
            newWindow's Text:<br />
            <input type="text" name="text1" />
            <input type="button" value="Get Text" name="btnGetText"
                onclick="btnGetText_onclick()" />
        </p>
    </form>
</body>
</html>
```

这是源窗口的代码，保存为 ch08_examp5.htm。现在看看源窗口加载的页面：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Example 5 Popup</title>
    <script type="text/javascript">
    function btnGetText_onclick()
    {
        document.form1.text1.value = window.opener.document.form1.text1.value;
    }
    </script>
</head>
<body>
    <form name="form1" action=" ">
        Opener window's text<br />
        <input type="text" name="text1" />
        <input type="button" value="Get Text" name="btnGetText"
            onclick="btnGetText_onclick()" />
    </form>
</body>
</html>
```

保存为 ch08_examp5_popup.htm。

在浏览器中打开 ch08_examp5.htm，会看到带简单表单的页面，如图 8-12 所示。

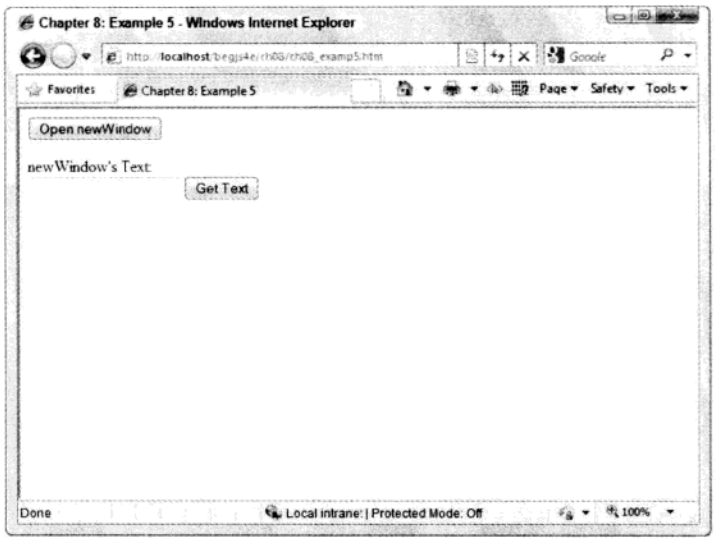

图 8-12

单击 Open newWindow 按钮，会在源页面上打开如图 8-13 所示的窗口。

图 8-13

在新窗口的文本框中输入一些文本。然后返回到源窗口中，单击 Get Text 按钮，则 newWindow 中输入的文本出现在源窗口表单的文本框中。

在源窗口的文本框中修改文本，然后返回 newWindow，并单击 Get Text 按钮。此时在源窗口的文本框中输入的文本将出现在 newWindow 的文本框中。

首先分析源窗口。该页面开头是一个脚本块，它定义了一个变量和三个函数。首先声明了一个新变量 newWindow，用来保存后面使用的 window.open()方法返回的 window 对象引用。该变量定义于函数之外，是一个全局变量，所以可在页面的任何函数中访问它。

```
var newWindow;
```

接下来，看看这个页面上三个函数中的第一个函数 btnOpenWin_onclick()，它连接到页面后面 Open newWindow 按钮的 onclick 事件处理程序上。其作用十分简单，即打开新窗口。

本例不打算在页面的任意位置打开新窗口，而是使用内置的 screen 对象确定用户显示器的分辨率，将新窗口放在屏幕中央。screen 对象是 window 对象的一个属性，有很多只读属性，这里只对 width 和 height 属性感兴趣。变量 winTop 初始化为弹出窗口的上边界在屏幕上的垂直位置。变量 winLeft 设置为弹出窗口的左边界在屏幕上的水平位置。本例将弹出窗口设置为水平和垂直居中。

```
function btnOpenWin_onclick()
{
    var winTop = (screen.height / 2) - 125;
    var winLeft = (screen.width / 2) - 125;
```

然后给窗口特性建立一个字符串，并存储在 windowFeatures 变量中。把窗口的宽度和高度设置为 250，再使用刚才填充的 winLeft 和 winTop 变量创建窗口的初始位置。

```
var windowFeatures = "width=250,height=250,";
windowFeatures = windowFeatures + "left=" + winLeft + ",";
windowFeatures = windowFeatures + "top=" + winTop;
```

最后打开新窗口，确保将 window.open()方法的返回值保存在全局变量 newWindow 中，以便可以在以后操作它。

```
    newWindow = window.open("newWindow.htm","myWindow",windowFeatures);
}
```

下一个函数用于从 newWindow 的表单上获取文本框中的文本。

这个函数使用 if 语句来检查两个条件。首先检查是否定义了 newWindow，其次检查是否打开了该窗口。这是因为我们不希望访问不存在的窗口，例如没有打开任何窗口，或者用户已关闭了窗口。typeof 运算符将返回变量的类型信息，如 number、string、Boolean、object 和 undefined。如果没有为变量赋值，typeof 运算符将返回 undefined。因为如果没有打开任何新窗口，newWindow 就不存在。

确认已经打开窗口后，还需要检查它是否仍然处于打开状态，为此可以使用 window 对象的 closed 属性。如果它返回 true，则窗口已关闭；如果返回 false，则窗口仍然处于打开状态(不要将这个 closed 属性与前面介绍的 close()方法混淆)。

在 if 语句中，先检查是否已定义 newWindow，这样做才不会出错。如果未定义 newWindow，则 newWindow.closed 将导致错误，因为 newWindow 中不包含任何数据。但是，如果 if 语句的结果总是由某个条件来决定是 true 还是 false，而不管其他条件的值如何，则忽略其他条件。

```
function butGetText_onclick()
{
    if (typeof(newWindow) == "undefined" || newWindow.closed == true)
    {
        alert("No window is open");
    }
```

如果 newWindow 存在并且已经打开，就会执行 else 语句的代码。注意 newWindow 包

含对已打开窗口的 window 对象的引用。这表示可以使用 newWindow 对应的 window 对象中的 document 对象访问 newWindow 中的表单，就好像访问运行脚本的页面上的表单一样。

```
        else
        {
            document.form1.text1.value = newWindow.document.form1.text1.value;
        }
    }
```

第三个函数是 window_onunload()，它连接到这个页面的 onunload 事件上，当关闭浏览器窗口或者用户导航到另一个页面时，将触发 onunload 事件。在 window_onunload()函数中，用相同的办法检查 newWindow 窗口是否有效、是否打开。必须先检查是否定义了 newWindow 变量。对于&&运算符，JavaScript 仅在第一部分为 true 时，才检查该操作的第二部分。如果定义了 newWindow，并保存了一个 window 对象(即使它是一个关闭的窗口)，就可以检查 window 对象的 closed 属性。但是，如果未定义 newWindow，则不检查该窗口的 closed 属性，也不会出错。如果先检查 closed 属性，而 newWindow 未定义，就会出错，因为未定义的变量没有 closed 属性。

```
function window_onunload()
{
    if (typeof(newWindow)!= "undefined" && newWindow.closed == false)
    {
        newWindow.close();
    }
}
```

如果 newWindow 已定义并且已经打开，就关闭它。这就禁止在关闭源窗口后单击 newWindow 窗口的 Get Text 按钮(因为关闭源窗口时，将触发这个函数)。

下面分析加载到 newWindow 中的 ch08_examp5_popup.htm 页面的代码。这个页面包含一个函数 btnGetText_onclick()，它连接到该页面上 Get Text 按钮的 onclick 事件处理程序上。该函数接收源窗口的文本框中的文本。

```
function btnGetText_onclick()
{
    document.form1.text1.value = window.opener.document.form1.text1.value;
}
```

在这个函数中，使用 window.opener 属性获得对打开弹出窗口的源窗口的 window 对象的引用，再使用该引用从源窗口中获取表单中的文本框值。这个值放在当前页面的文本框内。

8.2.3 移动窗口并重置其大小

除了打开和关闭窗口之外，还可以移动窗口，重置窗口的大小。

打开窗口后，可以使用 window 对象的 moveTo()和 resizeTo()方法，改变其屏幕位置和

大小，这两个方法都带两个参数，其单位是像素。

考虑下面打开新窗口的代码：

```
var newWindow = window.open(myURL,"myWindow","width=125,height=150,resizable");
```

为了将该窗口的大小设置为 350 像素宽、200 像素高，并将该窗口移动到距离屏幕左边界 100 像素，距离上边界 400 像素的位置，需要什么代码？

```
newWindow.resizeTo(350,200);
newWindow.moveTo(100,400);
```

上面的代码使用 resizeTo()将窗口大小设置为 350 像素宽、200 像素高。再使用 moveTo()将该窗口移动到距离屏幕左边界 100 像素、距离上边界 400 像素的位置。

window 对象还有 resizeBy()和 moveBy()方法，它们都接受两个参数，单位是像素。例如：

```
newWindow.resizeBy(100,200);
```

这行代码把 newWindow 的大小水平增加了 100 像素，垂直增加了 200 像素。同样，下面的代码把 newWindow 水平移动 20 像素，垂直移动 50 像素：

```
newWindow.moveBy(20,50);
```

使用这些方法时，必须注意，用户可以手工重置这些窗口的大小。另外，不同用户的客户屏幕的大小也是不同的。

8.3　安全性

浏览器对脚本可以在框架和窗口之间访问的信息有一些限制。

与这些例子一样，如果这些框架和窗口中的所有页面都来自于同一个服务器或者在同一台计算机上，而且把这些页面本地加载到浏览器上，则脚本可以访问的信息和执行的操作有很大的自由度。但存在一些限制。例如，如果不是在脚本打开的浏览器窗口中，而是在用户打开的窗口中加载一个脚本页面，在该页面上使用 window.close()方法，就会显示一个消息框，允许用户根据情况取消 close()方法，使窗口一直打开。

如果一个框架或窗口中的页面来自某个服务器，它试图访问另一个窗口或框架的属性，而另一个窗口或框架包含来自另一个服务器的页面，则同源策略会起作用，此时脚本能执行的操作非常有限。

假如在 URL 为 http://www.myserver.com 的服务器上有一个页面，在该页面中包含如下脚本：

```
var myWindow =
    window.open("http://www.anotherserver.com/anotherpage.htm","myWindow");
```

现在就有了两个窗口，第一个窗口位于 www.myserver.com，而第二个窗口位于另一个服务器 www.anotherserver.com 上。尽管上述代码是有效的，但同源策略将禁止在一个页面

中访问另一个页面的 document 对象。例如，在源页面中，下面的代码将导致一个安全问题，并被浏览器阻止。

```
var myVariable = myWindow.document.form1.text1.value;
```

尽管可以访问另一个服务器上的页面的 window 对象，但仅能访问其属性和方法的一个有限子集。

框架(常规框架和 iframe)与窗口的同源限制是完全相同的。其理念非常合理：防止黑客将他人的页面加载在他们的页面中，然后利用他们页面中的代码来获取信息。但是，这种限制非常严格，这意味着如果页面来自不同的服务器，就应避免在框架或窗口之间编写脚本。

8.4 小结

出于各种原因，基于框架的网站非常有用。因此，我们应能创建 JavaScript，与框架、框架中的文档和代码进行交互。

- 框架的一个优点是将所有公用的函数放在一个框架中，以便创建网站中所有页面都能使用的 JavaScript 代码模块。
- 用框架编写代码的关键就是获得一个对其他框架的 window 对象的引用。访问层次结构中较高层的框架有两种方法：使用 window 对象的 parent 属性和 top 属性。
- parent 属性返回包含当前 window 对象的 window 对象，该对象是包含创建当前窗口的框架集的页面。top 属性则返回包含所有其他框架的窗口的 window 对象。
- 可以采用三种方法访问框架集中的每个框架。第一是使用框架的名称。第二是使用 frames 集合并指定框架的索引。第三是在框架集合中使用框架的名称，例如 parent.frames.frameName。第三种方法是最安全的，它避免了与全局变量发生冲突。
- 如果要访问的框架在另一个窗口中定义，则需要使用 parent 属性或 top 属性来获得对定义该框架的 window 对象的引用，然后指定该框架的名称或在框架集合的位置。

接着讨论了如何使用脚本打开新的浏览器窗口。

- 使用 window 对象的 open()方法可以打开新窗口。该方法的第一个参数是要打开的页面的 URL；第二个参数是新窗口的名称；第三个可选参数用于定义新窗口拥有什么特性。
- window.open()方法返回一个对新窗口的 window 对象的引用。通过该引用，可以访问新窗口的文档、脚本和方法，这与框架十分类似。如有必要，还需要确保将该引用保存到变量中。
- 要关闭一个窗口，只需使用 window.close()方法。要检查窗口是否关闭，可使用 window 对象的 closed 属性，如果窗口关闭了，该属性就返回 true，如果窗口仍然处于打开状态，则该属性返回 false。

- 要使新打开的 window 对象访问打开该窗口的源窗口，可以使用 window.opener 属性。与框架的 window.parent 一样，window.opener 属性是对打开新窗口的 window 对象的引用，允许访问源窗口 window 对象及其属性。
- 打开一个窗口后，可以使用 resizeTo(x,y)和 resizeBy(x,y)重置窗口的大小，使用 moveTo(x,y)和 moveBy(x,y)移动窗口。

本节还简要介绍了不同源的窗口和框架的安全限制。“不同源”是指一个框架中的文档来自一个服务器，另一个框架中的文档来自另一个服务器。此时，有非常严格的限制，限制了在框架或窗口之间编写脚本的程度。

第 9 章介绍高级的字符处理技术。

8.5　习题

本章习题的参考答案在附录 A 中。

习题 1：

在第 7 章的习题中，创建了一个表单，允许用户选配计算机系统。用户可以查看系统的信息及其总价，单击按钮，就会把信息写入一个 textarea。请将该例子改为基于框架的网页：该应用程序不把信息写入 textarea，而是写入另一个框架。

提示：使用 about:blank 作为要写入的框架的 src 属性，使用 document 对象的 close()和 open()方法清空信息框架中以前写入的数据。

习题 2：

本章的第 4 个例子(ch08.examp4.htm)是一个带图书图像的页面，单击某本书的图片，将在一个弹出窗口中显示该图书的信息。修改该例，在弹出窗口中添加一个按钮或链接，单击该按钮或链接时，把该图书添加到购物车中。另外，在主页面中用某种方式打开购物车窗口，其中包含用户购买的所有商品，并允许用户从中删除商品。

第 9 章

字符串操作

第 4 章介绍了 String 对象，它是 JavaScript 的内置对象之一。它包括下列属性和方法：

- length——字符串的长度。
- charAt()和 charCodeAt()——返回字符串中指定位置处的字符或字符编码。
- indexOf()和 lastIndexOf()——在一个字符串中查找另一个子串是否存在，如果存在，则返回子串在字符串中的位置。
- substr()和 substring()——返回字符串的一部分。
- toUpperCase()和 toLowerCase()——将字符串转换为大写或小写形式。

本章将学习 String 对象的 4 个新方法，即 split()、match()、replace()和 search()。后 3 个方法提供了非常强大的文本处理功能。但是，要充分利用这些功能，还需要学习一个稍微复杂的主题。

split()、match()、replace()和 search()方法都可以使用正则表达式。在 JavaScript 中，正则表达式封装为 RegExp 对象。正则表达式允许定义字符的模式，用于查找或替换文本。例如，要将字符串中用以括起文本的所有单引号替换为双引号。这看起来很简单——只需搜索 ' 字符，并替换为 " 字符——但如果字符串是 Bob O'Hara said "Hello"?，则不需要替换 O'Hara 中的单引号。不使用正则表达式也可以进行这个文本替换，但需要的代码比使用正则表达式多两行。

split()、match()、replace()和 search()与正则表达式结合使用，能充分发挥其威力，而这几个方法也可以用于普通文本。下面分析它们如何应用于较简单的文本，以便熟悉这几个方法。

9.1 新的字符串方法

本节介绍 split()、match()、replace()和 search()方法，以及在不使用正则表达式的情况下如何使用它们。

9.1.1　split()方法

String 对象的 split()方法将一个字符串拆分为一个子串数组。在何处拆分字符串由传给该方法的分割参数确定，这个参数可以是一个字符或文本字符串。

例如，要拆分字符串"A,B,C"，得到的数组用逗号之间的字符来填充，代码应如下：

```
var myString = "A,B,C";
var myTextArray = myString.split(',');
```

对于上面的代码，JavaScript 将创建一个具有 3 个元素的数组。第一个元素是从 myString 字符串的开始到第一个逗号之间的字符。第二个元素是第一个逗号与第二个逗号之间的字符。最后，第三个元素是第二个逗号到字符串结束的字符。因此，myTextArray 数组将如表 9-1 所示：

表　9-1

第 1 个元素	第 2 个元素	第 3 个元素
A	B	C

如果字符串是"A,B,C,"，则 JavaScript 把它拆分为 4 个元素，最后一个元素包含从最后一个逗号到字符串结束之间的字符，即最后一个元素是一个空字符串，如表 9-2 所示：

表　9-2

第 1 个元素	第 2 个元素	第 3 个元素	第 4 个元素
A	B	C	

如果不小心，最后一个元素很容易被忽略掉。

试一试　反转文本的顺序

下面使用 split()方法创建一个短示例，反转<textarea>元素中各行的顺序。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Example 1</title>
<script language="JavaScript" type="text/JavaScript">
function splitAndReverseText(textAreaControl)
{
   var textToSplit = textAreaControl.value;
   var textArray = textToSplit.split('\n');
   var numberOfParts = 0;
   numberOfParts = textArray.length;
   var reversedString = "";
   var indexCount;
```

```
    for (indexCount = numberOfParts - 1; indexCount >= 0; indexCount-- )
    {
       reversedString = reversedString + textArray[indexCount];
       if (indexCount > 0)
       {
          reversedString = reversedString + "\n";
       }
    }

    textAreaControl.value = reversedString;
}
</script>
</head>
<body>
<form name="form1">
<textarea rows="20" cols="40" name="textarea1" wrap="soft">Line 1
Line 2
Line 3
Line 4</textarea>
<br />
<input type="button" value="Reverse Line Order" name="buttonSplit"
   onclick="splitAndReverseText(document.form1.textarea1)">
</form>
</body>
</html>
```

保存为 ch9_examp1.htm，并在浏览器中加载它，屏幕应如图 9-1 所示。

单击 Reverse Line Order 按钮，将颠倒行序，如图 9-2 所示。

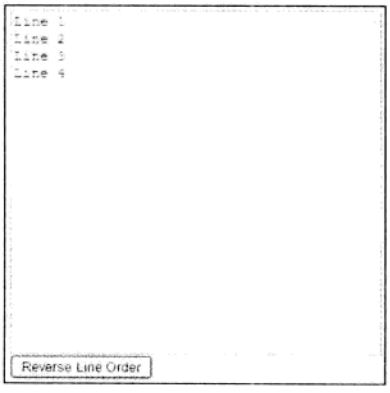

图 9-1

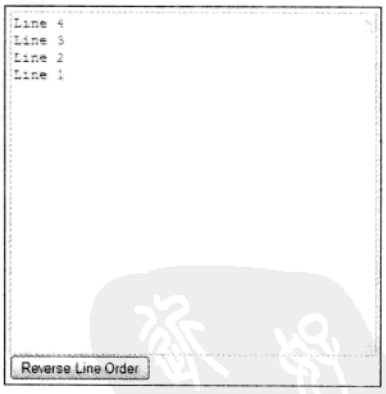

图 9-2

修改文本区域中的各行，再进行测试。

注意：

尽管这个示例在 IE 浏览器中能够正常运行，但会插入一个额外的行。对于 IE，如果出现了这个问题，只需将每个\n 替换为\r\n 就可以修正它。

这段代码起作用的关键是 splitAndReverseText()函数。该函数在页面顶部的脚本块中定义，并连接到页面后面的按钮的 onclick 事件处理程序上。

```
<input type="button" value="Reverse Line Order" name=buttonSplit
   onclick="splitAndReverseText(document.form1.textarea1)">
```

可以看出，这段代码将要反转顺序的文本区域的引用作为参数传递给函数，而不是在函数中直接引用该元素，这样可以使该函数更通用，可以用于任何 textarea 元素。

现在来分析 splitAndReverseText()函数。首先将 textarea 元素的文本赋予变量 textToSplit。然后用 String 对象的 split()方法将字符串拆分为一个文本行数组，把得到的数组放在变量 textArray 中。

```
function splitAndReverseText(textAreaControl)
{
   var textToSplit = textAreaControl.value;
   var textArray = textToSplit.split('\n');
```

那么，应使用什么分隔符作为 split()方法的参数？第 2 章提到，转义字符\n 表示换行，但 IE 需要\r\n 而不是\n 来表示换行，这一点可能引起混淆。

接下来定义并初始化其他 3 个变量：

```
var numberOfParts = 0;
numberOfParts = textArray.length;
var reversedString = "";
var indexCount;
```

有了字符串数组后，就要反转这个数组。为此构造一个新的字符串，并按自下而上的顺序添加数组中的每个字符串。为此使用一个 for 循环，但 for 循环不是像往常那样从 0 开始递增，而是从一个大于 0 的数字开始递减，递减到 0 时循环结束。

```
for (indexCount = numberOfParts - 1; indexCount >= 0; indexCount-- )
{
   reversedString = reversedString + textArray[indexCount];
   if (indexCount > 0)
   {
      reversedString = reversedString + "\n";
   }
}
```

最后，将 textarea 元素中的文本赋为新构造的字符串：

```
   textAreaControl.value = reversedString;
}
```

学习完正则表达式之后，我们还将再次回顾 split()方法。

9.1.2 replace()方法

replace()方法在字符串中搜索子串，当找到匹配的子串时，就将把匹配的子串替换为指定的第三个子串。

下面是一个例子，假如字符串包含子串 May，如下所示：

```
var myString = "The event will be in May, the 21st of June";
```

要将 May 替换为 June，可以使用 replace()方法，如下所示：

```
myCleanedUpString = myString.replace("May","June");
```

myString 的值不会改变。replace()方法会返回 myString 的值，但用 June 替换了 May，接着把返回的字符串赋予变量 myCleanedUpString，其中包含了正确的文本。

```
"The event will be in June, the 21st of June"
```

9.1.3 search()方法

search()方法用于在字符串中查找特定的文本段。如果找到文本，则返回子串在字符串中的位置，否则返回 - 1。该方法仅接收一个参数，即要查找的子串。

用于普通文本时，search()方法并不比其他方法更好用，例如前面学过的 indexOf()方法。但是，一旦与正则表达式相结合，search()方法的强大威力就会显现出来。

下面的例子查找 myString 字符串中是否包含 Java 这个单词：

```
var myString = "Beginning JavaScript, Beginning Java, Professional JavaScript";
alert(myString.search("Java"));
```

执行上面的代码，弹出的警告对话框中将显示 10，这是 Java 第一次出现时 J 的位置。

9.1.4 match()方法

match()方法与 search()方法非常类似，只是 search()返回找到的子串的位置，而 match()方法返回一个数组，该数组的每个元素都包含一个找到的匹配子串。

尽管可以对纯文本使用 match()方法，但没有多少意义。例如，下面的代码：

```
var myString = "1997, 1998, 1999, 2000, 2000, 2001, 2002";
myMatchArray = myString.match("2000");
alert(myMatchArray.length);
```

这段代码返回的 myMatchArray 有一个包含数值 2000 的元素。如果知道搜索字符串是 2000，则 match()方法毫无意义。

但是，match()方法与正则表达式结合使用，就非常有意义。例如，要在上面的字符串中查找所有属于 21 世纪的年份——即以 2 开始的年份，则得到的数组包含 2000、2000、2001 和 2002，这是非常有用的信息。

9.2　正则表达式

在进一步学习 String 对象的 split()、match()、replace()和 search()方法之前，需要了解正则表达式和 RegExp 对象。正则表达式是一种定义字符模式的方法，我们可以拆分、查找或替换字符串中与模式字符匹配的字符。

JavaScript 的正则表达式语法极大地借鉴了另一种脚本语言 Perl 的正则表达式语法。一些较新版本的语言，如 VBScript，已经集成了正则表达式，一些应用程序，如 Microsoft Word 的 Find 特性允许使用正则表达式。Dreamweaver 也是如此。即使在 JavaScript 范围之外，正则表达式也是有用的。

在 JavaScript 中，正则表达式是通过 RegExp 对象使用的，RegExp 对象是 JavaScript 中的一个内置对象，与 String、Array 类似。可以通过两种方式创建新的 RegExp 对象。较简单的办法是采用正则表达式字面值，例如：

```
var myRegExp = /\b'|'\b/;
```

其中，斜杠(/)表示正则表达式的开始和结束。这个特殊的语法告诉 JavaScript，这是一个正则表达式，就像引号表示字符串的开始和结束一样。现在不必考虑正则表达式的语法(\b'|'\b)，稍后会详细介绍。

另外，还可以使用 RegExp 对象的构造函数 RegExp()，如下所示：

```
var myRegExp = new RegExp("\\b'|'\\b");
```

这两种指定正则表达式的方法都可行，但前一种方法对于 JavaScript 更简短有效，因此通常使用它。本章主要使用第一种方法。使用第二种方法的主要原因是它允许在运行期间(在代码执行时而不是在编写代码时)确定正则表达式。例如，正则表达式需要以用户的输入为基础时，第二种方法就非常有用。

熟悉了正则表达式后，再介绍定义它们的第二种方法，即使用 RegExp()构造函数。用第二种方法创建正则表达式的语法略有不同，所以后面将详细介绍。

尽管我们关注的是如何在 String 对象的 split()、match()、replace()和 search()方法中把 RegExp 对象用作参数，但是 RegExp 对象有自己的方法和属性。例如，其 test()方法可以测试传入的参数字符串是否匹配 RegExp 对象中定义的模式。稍后的一个例子将介绍 test()方法。

9.2.1　简单的正则表达式

使用正则表达式的语法定义模式字符可能会非常复杂。本节将介绍正则表达式模式的基础知识。通过实例来学习是最佳的办法。

下面的例子使用 replace()方法和正则表达式实现一个简单的文本替换。假定有如下字符串：

```
var myString = "Paul, Paula, Pauline, paul, Paul";
```

要将所有的"Paul"替换为"Ringo"。

因此，需要查找的文本模式是简单的 Paul。将它表示为正则表达式，得到如下代码：

```
var myRegExp = /Paul/;
```

如前所述，斜杠字符标记了正则表达式的开始和结束。接下来，将该表达式应用于 replace()方法。

```
myString = myString.replace(myRegExp, "Ringo");
```

replace()方法接收两个参数，第一是 RegExp 对象，用于定义要查找和替换的模式，第二是替换文本。

将上面的内容合并到一个例子中，则代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<script language="JavaScript" type="text/JavaScript">
  var myString = "Paul, Paula, Pauline, paul, Paul";
  var myRegExp = /Paul/;
  myString = myString.replace(myRegExp, "Ringo");
  alert(myString);
</script>
</body>
</html>
```

在浏览器中加载这些代码，则屏幕如图 9-3 所示。

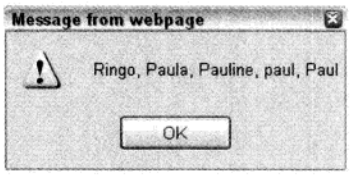

图 9-3

可以看到，字符串中的第一个 Paul 已被替换。但是我们需要替换字符串中的每个 Paul，而字符串末尾的两个 paul 仍然未替换，怎么回事？

RegExp 对象默认仅查找与模式匹配的第一个子串，在本例中即第一个 Paul，然后停止查找。这是 RegExp 对象的一个重要而常见的特性。正则表达式从字符串的一端开始查找，当找到第一个匹配的子串时，就停止查找。

这里需要的是全局查找，即查找所有可能的匹配并进行替换。为此，RegExp 对象有三个可以定义的属性，如表 9-3 所示：

表 9-3

属性字符	说明
g	全局匹配，查找所有与模式匹配的子串，而不是在找到第一个匹配的子串后就停止
i	模式不区分大小写。例如，将 Paul 与 paul 视为相同的字符模式
m	多行标记，只能用于 IE 5.5+和 NN 6+，该标记指定，特殊字符^和$可以匹配多行文本和字符串的开始和结束，本章后面将介绍这些字符

如果修改代码中的 RegExp 对象，如下所示，则执行全局查找，并忽略大小写：

```
var myRegExp = /Paul/gi;
```

现在运行代码，结果如图 9-4 所示。

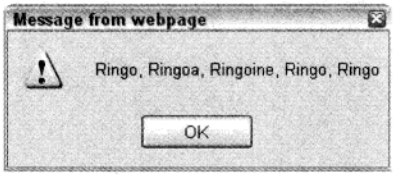

图　9-4

这似乎更糟糕了。正则表达式匹配了在字符串开始和结束位置的 Paul 子串，以及倒数第二个 paul 子串，这与预想的一样，但是，在 Pauline 和 Paula 中的 Paul 子串也被替换了！

RegExp 对象正确完成了其工作。我们要求替换所有匹配 Paul 的子串，结果正是如此。实际上，我们只想替换作为一个单词的 Paul，而不想替换包含在其他单词中的 Paul，例如 Paula。要实现这样的替换，关键在于正确定义字符模式，可以只匹配该模式，不匹配其他子串。下面就定义该字符模式：

(1) 要替换 paul 或 Paul。

(2) 不替换包含在其他单词(例如 Pauline)中的 paul 或 Paul。

如何指定第二个条件？如何知道 Paul 何时会连接到其他字符上，而不是仅连接到空格或标点符号上，或者连接到字符串的开始或结束位置？

要使用正则表达式得到期望的结果，需要借助正则表达式的特殊字符。下一节将介绍这些特殊字符，之后就能解决上面的问题。

试一试　正则表达式测试器

使正则表达式的语法正确需要一定的思考和时间，所以本例将创建一个简单的正则表达式测试器，以简化编码。

在文本编辑器中输入以下代码，保存为 ch9_examp2.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Regular Expression Tester</title>
<style type="text/css">

body,td,th {
    font-family: Arial, Helvetica, sans-serif;
}

</style>

<script type="text/javascript">
```

```
function getRegExpFlags()
{
        var regExpFlags = '';
        if ( document.form1.chkGlobal.checked )
        {
                regExpFlags = 'g';
        }

        if ( document.form1.chkCaseInsensitive.checked )
        {
                regExpFlags += 'i';
        }

        if ( document.form1.chkMultiLine.checked )
        {
                regExpFlags += 'm';
        }

        return regExpFlags;
}

function doTest()
{
        var testRegExp = new RegExp(document.form1.txtRegularExpression.value,
 getRegExpFlags());
        if ( testRegExp.test(document.form1.txtTestString.value) )
        {
                document.form1.txtTestResult.value = "Match Found! ";
        }
        else
        {
                document.form1.txtTestResult.value = "Match NOT Found!";
        }
}

function findMatches()
{
        var testRegExp = new RegExp(document.form1.txtRegularExpression.value,
getRegExpFlags());
        var myTestString = new String(document.form1.txtTestString.value)
        var matchArray = myTestString.match(testRegExp);

        document.form1.txtTestResult.value = matchArray.join('\n');
}

</script>

</head>

<body>

<form id="form1" name="form1" method="post" action=" ">
  <p>
    Regular Expression:<br />
<label>
      <input name="txtRegularExpression" type="text" id="txtRegularExpression"
```

```
size="100" value=" "/>
      <br />
      Global
      <input name="chkGlobal" type="checkbox" id="chkGlobal" value="true"/>
</label>

  Case Insensitive
  <label>
    <input name="chkCaseInsensitive" type="checkbox" id="chkCaseInsensitive"
value="true" />
  </label>

  Multi Line
  <label>
    <input name="chkMultiLine" type="checkbox" id="chkMultiLine" value="true" />
  </label>
  </p>
  <p>
    <label>
      Test Text:<br />
      <textarea name=" txtTestString" id="txtTestString" cols="100"
rows="8" ></textarea>
    </label>
  </p>
  <p>Result:<br />
    <textarea name="txtTestResult" id="txtTestResult" cols="100"
rows="8"></textarea>
  </p>
  <p>
    <label>
      <input type="button" name="cmdTest" id="cmdTest" value="TEST"
onclick="doTest();"/>
    </label>
    <label>
      <input type=" button" name="cmdMatch" id="cmdMatch" value="MATCH"
onclick="findMatches();" />
    </label>
    <label>
      <input type="reset" name="cmdClearForm" id="cmdClearForm" value="Reset Form"
 />
    </label>
  </p>
  <p> </p>
</form>
</body>
</html>
```

把页面加载到浏览器中，屏幕应如图 9-5 所示。

图　9-5

在顶部的框中输入正则表达式。可以单击复选框，设置全局和区分大小写等属性。在 Test Text 框中输入用于测试正则表达式的文本，结

果显示在 Result 框中。

作为测试，输入正则表达式\d{3}，如稍后所述，该正则表达式匹配 3 个数字。再单击 Global 框，查找所有的匹配。最后，输入测试文本 ABC123DEF456GHI789。

如果单击 Test 按钮，代码就会测试是否有匹配的子串(即测试文本是否包含 3 个数字)，结果如图 9-6 所示，找到一个匹配的子串。

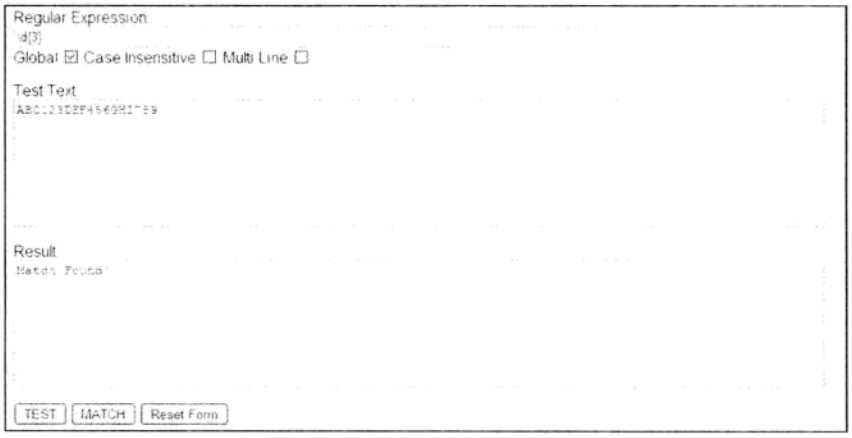

图 9-6

现在要查找所有匹配的子串，应单击 Match 按钮，这次的结果如图 9-7 所示。

图 9-7

在 Test Text 框中找到的、与正则表达式匹配的每个子串都放在 Results 框的单独一行上。

按钮 cmdTest 和 cmdMatch 的单击事件链接到 doTest()和 findMatches()函数上。下面首先分析 doTest()函数。

首先创建正则表达式对象：

```
var testRegExp = new RegExp(document.form1.txtRegularExpression.value,
                            getRegExpFlags());
```

对象构造函数的第一个参数是包含在 txtRegularExpression 文本框中的正则表达式，这很容易得到。但第二个参数包含正则表达式标记，它们是通过表单中的复选框获得的。为了把复选框转换为正确的标记，创建了 getRegExpFlags()函数，该函数的返回值为正则表达式构造函数提供了标记值。getRegExpFlags()函数由 doTest()和 getMatches()函数使用。getRegExpFlags()函数相当简单，它首先声明 regExpFlags，并把它设置为一个空串。

```
var regExpFlags = '';
```

接着检查每个复选框，确定是否选中。如果已经选中，就把对应的标记添加到 regExpFlags 中，全局标记如下所示：

```
if ( document.form1.chkGlobal.checked )
{
      regExpFlags = 'g';
}
```

这个规则也应用于区分大小写标记和多行标记。

现在返回到 doTest()函数上。创建好正则表达式对象，设置了其标记后，就测试正则表达式是否匹配 Test Text 框中的内容。

```
if ( testRegExp.test(document.form1.txtTestString.value) )
{
      document.form1.txtTestResult.value = "Match Found!";
}
else
{
      document.form1.txtTestResult.value = "Match NOT Found!";
}
```

如果找到了一个匹配，就把“Match Found!”写入 Results 框中，否则写入“Match NOT Found!”。

正则表达式对象的 test()方法用于测试正则表达式是否匹配方法的唯一参数提供的测试字符串。若找到一个匹配，它就返回 true，否则返回 false。全局标记与 test()方法无关，因为 test()仅查找第一个匹配，如果找到匹配，就返回 true。

下面来分析 findMatches()函数，它在单击 cmdMatches 按钮时运行。与 doTest()函数一样，第一行也使用在表单的 Regular Expression 文本框中输入的正则表达式，和通过 getRegExpFlags()函数设置的标记，创建一个新的正则表达式对象。

```
var testRegExp = new RegExp(document.form1.txtRegularExpression.value,
                            getRegExpFlags());
```

接着创建一个新的 String 对象，再使用 String 对象的 match()方法查找匹配。

```
var myTestString = new String(document.form1.txtTestString.value)
var matchArray = myTestString.match(testRegExp);
```

match()方法返回一个数组，以及在数组每个元素中找到的所有匹配。变量 matchArray 用于存储该数组。

最后，匹配结果显示在表单的 Results 框中。

```
document.form1.txtTestResult.value = matchArray.join('\n');
```

String 对象的 join()方法把数组中的所有元素连接起来，把它们返回为一个字符串。每个元素都用传送给 join()方法的唯一参数的值分隔。这里传送了换行字符\n，这表示字符串显示在 Results 框中时，每个匹配会单独占一行。

9.2.2 正则表达式：特殊字符

本节将介绍三种特殊字符。

1. 文本、数字和标点符号

第一组特殊字符包含字符类(character class)的特殊字符。字符类指数字、字母和空白字符。这些特殊字符如表 9-4 所示：

表 9-4

字 符 类	匹配的字符	示 例
\d	0～9 的任何数字	\d\d 将匹配 72，但不匹配 aa 或者 7a
\D	任何非数字字符	\D\D\D 匹配 abc，但不匹配 123 或者 8ef
\w	任何单词字符，即 A～Z、a～z、0～9，以及下划线(_)字符	\w\w\w\w 匹配 Ab_2，但不匹配£$%*或者 Ab_@
\W	任何非单词字符	\W 匹配@，但不匹配 a
\s	任何空白字符	\s 与制表符、回车符、换页符和竖杠匹配
\S	任何非空白字符	\S 匹配 A，但不匹配制表符
.	除换行符(\n)之外的任意单个字符	.匹配 a 或者 4，或者@
[...]	匹配位于方括号之内的任何一个字符，[a-z]将匹配在 a～z 范围内的任何字符	[abc]匹配 a、b 或 c，但不匹配任何其他字符
[^...]	匹配除方括号内的字符之外的任何字符	[^abc]将匹配除了 a、b 或 c 之外的任何字符 [^a-z]将匹配除了 a～z 范围之外的任何字符

注意，大小写字符的含义大相径庭，因此在使用正则表达式时要特别注意。

下面看一个例子。要匹配格式为 1-800-889-5474 的电话号码，可使用如下正则表达式：

```
\d-\d\d\d-\d\d\d-\d\d\d\d
```

可以看到，其中有很多重复的字符，这使正则表达式比较冗长。为了进行简化，正则表达式可以定义重复次数，详见本章后面的内容，现在分析另一个例子。

试一试　**检查密码短语中的字母和数字字符**

这个完整示例将应用前面学到的正则表达式知识，检查密码短语是否只包含字母和数字字符，不包含标点符号或@、%等符号。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Example 3</title>

<script type="text/JavaScript">
function regExpIs_valid(text)
{
   var myRegExp = /[^a-z\d ]/i;
   return !(myRegExp.test(text));
}
function butCheckValid_onclick()
{
   if (regExpIs_valid(document.form1.txtPhrase.value) == true)
   {
      alert("Your passphrase contains only valid characters");
   }
   else
   {
      alert("Your passphrase contains one or more invalid characters");
   }
}
</script>

</head>
<body>

<form name="form1">
Enter your passphrase:
<br />
<input type="text" name="txtPhrase">
<br />
<input type="button" value="Check Character Validity" name="butCheckValid"
   onclick="butCheckValid_onclick()">
</form>

</body>
</html>
```

将该页面保存为 ch9_examp3.htm，并在浏览器中加载它。在文本框中仅输入字母、数字和空格，并单击 Check Character Validity 按钮，则代码将提示该密码短语包含有效的字符。试着输入标点符号或特殊字符如@、^、$等，则程序将提示密码短语无效。

下面分析在页面顶部的脚本块开头定义的 regExpIs_valid()函数，其作用是使用正则表达式检查密码短语的有效性。

```
function regExpIs_valid(text)
{
   var myRegExp = /[^a-z\d ]/i;
   return !(myRegExp.test(text));
}
```

该函数只接收一个参数，即要检查有效性的文本。该函数首先声明一个变量 myRegExp，并把它设置为一个新的正则表达式，这将隐式地创建一个新的 RegExp 对象。

正则表达式本身非常简单，下面先考虑一下要查找的模式。我们要确定密码短语字符串是否包含不属于字母 A～Z、字母 a～z、数字 0～9 或者空格的字符。这是如何转换为正则表达式的？

(1) 使用一个带^符号的方括号：

```
[^]
```

这表示要匹配未在方括号中指定的字符。

(2) 添加 a-z，指定从 a 到 z 的任意字符。

```
[^a-z]
```

这样，该正则表达式匹配任何不在 a～z 之间的字符。注意，因为在正则表达式定义的末尾添加了 i，所以该模式不区分大小写。因此，实际上该正则表达式将匹配任何非 A～Z 或 a～z 的字符。

(3) 添加\d，以表示任何数字字符，即 0～9 之间的字符。

```
[^a-z\d]
```

(4) 该正则表达式将匹配任何不在 a～z、A～Z 或者 0～9 之间的字符。我们认为空格是有效的，因此在方括号中添加它：

```
[^a-z\d ]
```

综上所述，该正则表达式将匹配任何非字母、数字或空格的字符。

(5) 在函数的第二行和最后一行，使用 RegExp 对象的 test()方法返回一个值：

```
return !(myRegExp.test(text));
```

RegExp 对象的 test()方法检测所传入的参数字符串，确定正则表达式语法指定的字符是否与字符串中的内容。如果存在匹配项，则返回 true，否则返回 false。如果该正则表达式找到第一个无效的字符，则返回 true 值，表示这是一个无效的密码短语。但是 Is_valid()函数在结果无效时返回 true 是不合乎逻辑的。因此，添加 NOT 运算符(!)，将返回的结果取反。

这个仅有 2 行代码的有效性检查函数使用了正则表达式。如果不使用正则表达式，需要增加多少行代码？下面的函数实现了与 regExpIs_valid()相同的功能，但没有使用正则表达式：

```
function is_valid(text)
```

```
{
   var isValid = true;
   var validChars = "abcdefghijklmnopqrstuvwxyz1234567890 ";
   var charIndex;
   for (charIndex = 0; charIndex < text.length;charIndex++)
   {
      if ( validChars.indexOf(text.charAt(charIndex).toLowerCase()) < 0)
      {
         isValid = false;
         break;
      }
   }
   return isValid;
}
```

这也许是最简练的非正则表达式版本了，但仍有 15 行，是正则表达式版本长度的 6 倍。

这个函数的工作原理与正则表达式版本类似。该函数有一个变量 validChars，包含了所有合法字符。然后，在 for 循环中使用 charAt()方法获取密码短语字符串的每个字符，检查相应字符是否存在于 validChars 字符串中。如果不存在，则说明该字符串包含无效字符。

在上例中，函数的非正则表达式版本有 15 行代码。对于更复杂的问题，可能需要 20 行甚至 30 行代码才能解决问题，而正则表达式只需几行代码就能完成任务。

回到例子中的代码，页面顶部定义的另一个函数是 butCheckValid_onclick()。顾名思义，单击页面体上定义的 butCheckValid 按钮时，将调用该函数。

这个函数在 if 语句中调用 regExpis_valid()函数，检查用户在 txtPhrase 文本框中输入的密码短语是否有效。如果有效，则使用一个警告框通知用户。

```
function butCheckValid_onclick()
{
   if (regExpIs_valid(document.form1.txtPhrase.value) == true)
   {
      alert("Your passphrase contains valid characters");
   }
```

否则使用另一个警告框告知用户输入的文本无效。

```
   else
   {
      alert("Your passphrase contains one or more invalid characters");
   }
}
```

2. 重复字符

正则表达式包含重复字符，用于指定要匹配多少个前一项或字符。这是非常有用的，例如，指定电话号码中的字符需要重复特定的次数。表 9-5 列出了最常用的重复字符及其含义。

表 9-5

特殊字符	含义	示例
{n}	前一项出现 n 次	x{2}与 xx 匹配
{n,}	前一项出现 n 次，或者 n 次以上	x{2,}匹配 xx、xxx、xxxx、xxxxx 等
{n,m}	前一项至少出现 n 次，最多出现 m 次	x{2,4}匹配 xx、xxx 和 xxxx
?	前一项出现 0 次或 1 次	x? 匹配空串或者 x
+	前一项出现 1 次或多次	x+匹配 x、xx、xxx、xxxx、xxxxx 等
*	前一项出现 0 次或多次	x*匹配空串、或者 x、xx、xxx、xxxx 等

前面提到过，要匹配格式为 1-800-889-5474 的电话号码，正则表达式是\d-\d\d\d-\d\d\d-\d\d\d\d。下面看看如何使用重复字符简化它。

这个模式以一个数字开始，后跟一个短横线，因此正则表达式应为：

```
\d-
```

接下来是 3 个数字，后跟一个短横线。现在可以使用重复特殊字符了——\d{3}精确匹配 3 个\d，即 3 个数字字符。

```
\d-\d{3}-
```

接下来也是 3 个数字，后跟一个短横线。因此，正则表达式如下所示：

```
\d-\d{3}-\d{3}-
```

最后是 4 个数字，即\d{4}。

```
\d-\d{3}-\d{3}-\d{4}
```

这个正则表达式声明如下：

```
var myRegExp = /\d-\d{3}-\d{3}-\d{4}/
```

第一个 / 和最后一个 / 告诉 JavaScript，位于这两个斜杠之间的是正则表达式。JavaScript 将根据这个正则表达式，创建一个 RegExp 对象。

另一个例子是，如果想将字符串 Paul Paula Pauline 中的 Paul 和 Paula 替换成 George，该怎么办？这需要定义一个匹配 Paul 和 Paula 的正则表达式。

先将任务分解一下，要得到 Paul，正则表达式应是：

```
Paul
```

现在还需要匹配 Paula，但如果将表达式定义为 Paula，就无法匹配 Paul。此时应使用特殊字符?，它可将其前的字符定义为可选——即可以出现 1 次或者 0 次。因此，答案是：

```
Paula?
```

于是，应声明如下正则表达式：

```
var myRegExp = /Paula?/
```

3. 位置字符

第三组特殊字符可以指定匹配从哪里开始或结束，或者哪些字符位于字符模式的某一端。例如，有时模式应位于字符串或文本行的开始位置，有时模式应位于两个单词之间。表 9-6 列出了最常见的位置字符及其含义。

表　9-6

位置字符	描　述
^	模式必须位于字符串的开头，如果是多行字符串，模式就位于一行的开头。对于多行文本(即包含回车符的字符串)，在定义正则表达式时，需要使用/myreg ex/m 设置多行标记。注意，该字符仅能用于 IE 5.5 和 NN 6 及其更新版本
$	模式必须位于字符串的结束位置，如果是多行字符串，模式就位于一行的结束位置。对于多行文本(即包含回车符的字符串)，在定义正则表达式时，需要使用/myreg ex/m 设置多行标记。注意，该字符仅能用于 IE 5.5 和 NN 6 及其更新版本
\b	匹配单词分界位置，即单词字符与非单词字符之间的位置
\B	匹配非单词分界位置

例如，如果要确保模式位于一行的开头，可使用如下代码：

```
^myPattern
```

这将与一行开头的 myPattern 匹配。
要在一行的结尾匹配该模式，可使用如下代码：

```
myPattern$
```

单词分界字符\b 和\B 容易引起混淆，因为它们不匹配字符，而是匹配字符之间的位置。假定在代码中定义字符串"Hello world!, let's look at boundaries said 007."：

```
var myString = "Hello world!, let's look at boundaries said 007.";
```

为了标识出单词的分界(即单词之间的分隔位置)，可以将分界替换为 | 字符：

```
var myRegExp = /\b/g;
myString = myString.replace(myRegExp, "|");
alert(myString);
```

上面的代码将所有的单词分界\b 替换为 | ，消息框如图 9-8 所示。

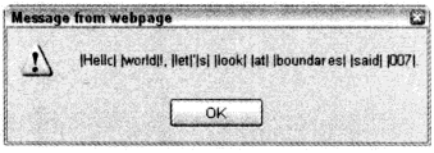

图　9-8

可以看到，任何单词字符(即字母、数字或下划线字符)之间的位置以及任何非单词字符都是单词分界。另外注意，字符串的开始或结束位置与单词字符之间的边界也是单词分

界。这个字符串的结尾是一个句号，该句号与字符串的结尾之间是一个非单词分界，因此没有插入 | 字符。

修改这个例子中的正则表达式，以替换所有的非单词分界：

```
var myRegExp = /\B/g;
```

结果如图 9-9 所示。

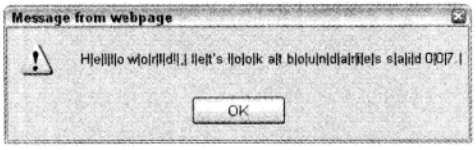

图 9-9

现在，任何字母、数字或下划线与另一个字母、数字或下划线之间的位置都是非单词分界，并替换为 | 字符。另外，两个非单词字符之间的位置，如感叹号和逗号之间的位置，也是一个非单词分界。这可能使人有点疑惑，但仔细思考确实如此。可是，创建正则表达式时，这一点容易被忽略。

开始学习正则表达式时，有一个例子：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<script language="JavaScript" type="text/JavaScript">

  var myString = "Paul, Paula, Pauline, paul, Paul";
  var myRegExp = /Paul/gi;
  myString = myString.replace(myRegExp, "Ringo");
  alert(myString);

</script>
</body>
</html>
```

该代码将 Paul 或者 paul 的所有实例都替换为 Ringo。

但是，该代码实际上把所有的 Paul 都替换为 Ringo，包括包含在其他单词中的 Paul。

解决这个问题的一个办法是仅替换后跟一个非单词字符的 Paul 子串。表示非单词字符的特殊字符是\W，因此，需要将正则表达式修改为：

```
var myRegExp = /Paul\W/gi;
```

结果如图 9-10 所示。

比以前好多了，但仍不是希望获得的结果。注意，第 2 个和第 3 个 Paul 子串之后的逗号也被替换了，因为它与\W 字符匹配。另外，字符串结尾的 Paul 子串仍没有替换，这是因为在最后一个 Paul 的字母 l 后没有任何字符。那么，在最后一个 Paul 的 l 后有什么东西？只是单词字符与非单词字符的分界。其实这正是答案，即正则表达式是 Paul 后跟一个

单词分界。输入下面的代码，修改正则表达式：

```
var myRegExp = /Paul\b/gi;
```

于是得到了需要的结果，如图 9-11 所示。

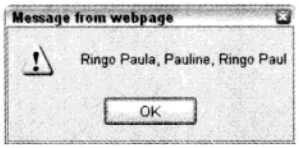

图　9-10

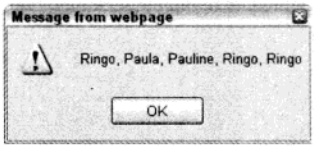

图　9-11

最后获得了正确的答案，完成了这个例子。

9.2.3　包含所有的可能性

正则表达式最难的一点是确保它包含了所有可能的情形。对于上面的例子，正则表达式正确处理了字符串，但是它会处理下面的代码吗？

```
var myString = "Paul, Paula, Pauline, paul, Paul, JeanPaul";
```

这里把 JeanPaul 中的 Paul 子串改为 Ringo。而我们希望转换的 Paul 子串是一个独立的单词，它的两边都是单词分界。将正则表达式修改为：

```
var myRegExp = /\bPaul\b/gi;
```

就得到了最终想要的答案，它能确保仅匹配作为独立单词出现的 Paul 或 paul 子串。

9.2.4　正则表达式的分组

在介绍使用 match()、replace()和 search()方法的例子之前，正则表达式的最后一个主题是如何对表达式分组。实际上这非常简单。如果要把多个表达式处理为一个组，只需将它们放在圆括号中，如/(\d\d)/。正则表达式中的圆括号是把字符模式组合在一起的特殊字符，圆括号本身不是要匹配的字符。

为什么要对正则表达式分组？将字符组合到模式中后，可以对整个字符组应用重复字符，而不是对单个字符应用重复字符。

例如，在 myString 中定义如下字符串：

```
var myString = "JavaScript, VBScript and Perl";
```

如何使用同一个正则表达式匹配 JavaScript 和 VBScript？这两个子串的共同之处是它们都是一个完整的单词，且都以 Script 结尾。一个简单的办法是使用圆括号组合模式 Java 和 VB，然后对每一个字符组应用?特殊字符，使模式匹配 Java 或 VB 子串出现 0 次或 1 次，且以 Script 子串结尾的单词。

```
var myRegExp = /\b(VB)?(Java)?Script\b/gi;
```

分解该表达式，它需要的模式为：

(1) 一个单词分界：\b

(2) 0 个或 1 个 VB 实例：(VB)?

(3) 0 个或 1 个 Java 实例：(Java)?

(4) 字符串：Script

(5) 一个单词分界：\b

综上所述，得到：

```
var myString = "JavaScript, VBScript and Perl";
var myRegExp = /\b(VB)?(Java)?Script\b/gi;
myString = myString.replace(myRegExp, "xxxx");
alert(myString);
```

该代码的输出结果如图 9-12 所示。

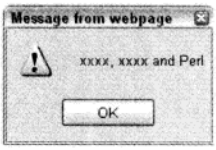

图 9-12

查看前面的重复字符表，注意重复字符应用于其前的项。该项可以是一个字符，也可以是用圆括号括起来的一组字符。

但是，上面定义的正则表达式存在一个潜在问题。该正则表达式不仅匹配 VBScript 和 JavaScript，也匹配 VBJavaScript，显然这并不是我们希望看到的。

为此，需要联合使用分组和特殊字符 | 。| 是“二选一”字符，与 if 语句中的||具有相似的含义，它匹配该字符两边的项中的一个。

重新思考上面的问题。模式需要匹配 VBScript 或 JavaScript。显然它们都有 Script。因此，要定义的模式是以 Java 或 VB 开头，且以 Script 结束。

首先，单词必须以字符分界开头：

```
\b
```

接下来，单词应以 VB 或 Java 开头，正则表达式中的 | 提供了“或”的功能，因此，根据正则表达式的语法，应写为：

```
\b(VB|Java)
```

它匹配模式 VB 或 Java。现在可以加上 Script 部分：

```
\b(VB|Java)Script\b
```

最终的代码如下所示：

```
var myString = "JavaScript, VBScript and Perl";
var myRegExp = /\b(VB|Java)Script\b/gi;
myString = myString.replace(myRegExp, "xxxx");
alert(myString);
```

9.2.5　重用字符组

可以在正则表达式中重用通过一组字符指定的模式。要引用前面的字符组，只需输入\和表示分组顺序的数字即可。例如，第 1 个分组可引用为\1，第 2 个分组引用为\2，依此类推。

下面看一个示例。假如字符串由一列数字组成，每个数字用逗号来分隔。出于某种原因，不允许在一行内出现两个相同的数字。因此，下面的字符串是有效的：

```
009,007,001,002,004,003
```

但下面的字符串是无效的：

```
007,007,001,002,002,003
```

因为重复出现了 007 和 002。

如何找到重复数字，并替换为单词 ERROR？需要在正则表达式中引用分组。

首先定义字符串，如下所示：

```
var myString = "007,007,001,002,002,003,002,004";
```

现在需要搜索一系列 1 个或多个数字字符。在正则表达式中，\d 表示任意数字字符，+表示前面的字符出现一次或多次。因此，正则表达式应是：

```
\d+
```

我们需要匹配一个数字系列后跟一个逗号，因此，需要添加一个逗号：

```
\d+,
```

该表达式匹配一个数字系列后跟一个逗号。但是如何搜索某个数字系列，后跟一个逗号，之后又是这个数字系列？数字可以是任何一个数字，因此不能在正则表达式中直接添加，例如：

```
\d+,007
```

这个表达式不能处理重复出现的 002。我们需要将第一个数字系列放在一组中，然后可以指定再次匹配这组数字，即使用\1，它表示：匹配使用圆括号定义的第一组中的字符。综上所述，正则表达式应定义为：

```
(\d+),\1
```

这定义了一个分组，其字符模式是一个或多个数字字符。该分组必须后跟一个逗号，然后是与第一组相同的模式。将这些放在 JavaScript 代码中，如下所示：

```
var myString = "007,007,001,002,002,003,002,004";
var myRegExp = /(\d+),\1/g;
myString = myString.replace(myRegExp,"ERROR");
alert(myString);
```

警告框将显示如下信息：

```
ERROR,1,ERROR,003,002,004
```

这就结束了对正则表达式语法的简要介绍。由于正则表达式比较复杂，最好先从简单的开始，逐步建立复杂的正则表达式，如前面的例子所示。实际上，大多数正则表达式很难一次写成功。

如果正则表达式看起来仍有点古怪，有点混乱，不必担心。下一节将介绍 String 对象的 split()、replace()、search()和 match()方法，并给出大量的正则表达式实例。

9.3 String 对象——split()、replace()、search()和 match()方法

使用正则表达式的主要函数是 String 对象的 split()、replace()、search()和 match()方法。前面学习了它们的语法，现在将主要讨论它们与正则表达式的联合使用，同时学习更多的正则表达式的语法和用途。

9.3.1 split()方法

split()方法可以根据指定为参数的字符拆分字符串。这个方法的结果是一个数组，其中的每个元素都包含一个拆分后的子串。例如下面的字符串：

```
var myListString = "apple, banana, peach, orange"
```

可以拆分为一个数组，其中的每个元素都包含一种不同的水果，例如：

```
var myFruitArray = myListString.split(", ");
```

但是如何拆分下面的字符串？

```
var myListString = "apple, 0.99, banana, 0.50, peach, 0.25, orange, 0.75";
```

该字符串包含了水果的名称和价格。如何拆分该字符串，只获取水果的名称，而不包含价格？虽然不使用正则表达式也可以完成这个拆分的，但那样做需要编写很多行代码。如果使用正则表达式，则可以使用相同的代码，但需要修改 split()方法的参数。

试一试　拆分水果字符串

下面创建一个例子，解决上述问题——拆分字符串，结果只包含水果名称，而不包含价格。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
```

```
<script type="text/JavaScript">
var myListString = "apple, 0.99, banana, 0.50, peach, 0.25, orange, 0.75";
var theRegExp = /[^a-z]+/i;
var myFruitArray = myListString.split(theRegExp);
document.write(myFruitArray.join("<br />"));

</script>
</body>
</html>
```

将该文件保存为 ch9_examp4.htm，在浏览器中加载它。页面从字符串中输出了 4 种水果的名称，每种水果占一行。

在脚本块中，首先定义一个包含水果名称和价格的字符串：

```
var myListString = "apple, 0.99, banana, 0.50, peach, 0.25, orange, 0.75";
```

如何拆分它，使结果只包含水果的名称？首先我们可能会想到把逗号作为 split()方法的参数，但结果将包含价格。必须考虑的问题是：在所需的项之间有什么？换句话说，在水果名称之间可用于定义拆分操作的是什么？在水果名称之间有各种字符，如逗号、空格、数字、句号、多个数字字符，最后是另一个逗号。这些字符有什么共同点，可以把它们与我们需要的水果名称区分开？它们的共同点是这些字符都不在 a～z 之间。如果使用不在 a～z 之间的一个字符组作为分隔符拆分字符串，就可以得到期望的结果。现在我们知道创建正则表达式所需的信息了。

我们需要的是不在 a～z 范围内的字符，因此，可以使用如下代码开头：

```
[^a-z]
```

^表示：匹配任何未在方括号内指定的字符。这里指定了不匹配的字符范围——a～z 的所有字符。如前所述，该正则表达式只匹配一个字符。但是我们需要的是不在 a～z 范围内的一个或多个字符所组成的一组字符。为此，需要添加重复字符+，表示：将前面指定的字符或字符组匹配一次或多次。

```
[^a-z]+
```

最终结果如下：

```
var theRegExp = /[^a-z]+/i
```

/ 和 / 字符是正则表达式的开始和结束标记，其 RegExp 对象在变量 theRegExp 中保存为引用。在尾部添加 i，表示匹配忽略大小写。

创建正则表达式的过程可能会遇到挫折，不太直观，不要灰心。开始时可能需要多次尝试，才能创建正确的正则表达式，但是随着经验的积累，创建正则表达式会变得相当容易。在某些情况下，不使用正则表达式，完成任务会非常困难，甚至不可能完成。

下面的脚本将 RegExp 对象传递给 split()方法，用于确定在何处拆分字符串。

```
var myFruitArray = myListString.split(theRegExp);
```

拆分之后，变量 myFruitArray 将包含一个数组，它的每个元素都包含一种水果的名称，

如表 9-7 所示：

表 9-7

数组元素的索引	0	1	2	3
元素的值	apple	banana	peach	orange

然后使用 Array 对象的 join()方法将字符串连接起来，参见第 4 章。

```
document.write(myFruitArray.join("<BR>"))
```

9.3.2 replace()方法

前面学习过 replace()方法的语法和用途。replace()方法的独特之处在于它可以根据与正则表达式匹配的分组来替换文本。为此，要使用$符号和分组的序号。正则表达式中的每个分组都给定了一个 1～99 之间的数字，大于 99 的分组是不可访问的。注意在早期的浏览器中，分组的序号只能是 1～9(例如 IE5、Netscape 4 和更早的版本)。要引用一个分组，可以使用$符号，后跟分组的序号。例如对于如下代码：

```
var myRegExp = /(\d)(\W)/g;
```

$1 引用分组(\d)，$2 引用分组(\W)。还设置了全局标志 g，以保证替换所有匹配的模式——而不仅仅替换第一个匹配的模式。

下面的例子更清晰地说明了这一点。假定有如下字符串：

```
var myString = "1999, 2000, 2001";
```

如果想通过正则表达式把它改为"the year 1999, the year 2000, the year 2001"，该怎么办？首先，需要写出作为正则表达式的模式，在本例中就是 4 个数字，即：

```
var myRegExp = /\d{4}/g;
```

但是，每一次匹配的年份都是不同的，如何把对应年份值放在替换字符串中？修改正则表达式，把它放在一个分组中，即：

```
var myRegExp = /(\d{4})/g;
```

现在，就可以在替换字符串中使用序号为 1 的分组了：

```
myString = myString.replace(myRegExp, "the year $1");
```

变量 myString 包含所需的字符串"the year 1999, the year 2000, the year 2001"。

下面的另一个例子要把文本中的单引号替换为双引号。测试字符串如下所示：

```
'Hello World' said Mr. O'Connerly.
He then said 'My Name is O'Connerly, yes that's right, O'Connerly'.
```

测试字符串要澄清的一个问题是，只用双引号替换对话内容两边的成对的单引号，而

不替换用作撇号的单引号，如单词 that's 中的撇号，或人名中间的单引号，如 O'Connerly。

下面开始定义这个正则表达式。首先，它必须包含一个单引号，即：

```
var myRegExp = /'/;
```

但是，这将替换所有的单引号，不是我们想要的结果。

观察该文本，会发现需要替换的单引号总是位于单词的开头或结尾，即分界。粗略地看，很容易认为是单词分界。但是，不要忘了 ' 是一个非单词字符，因此分界将位于它与另一个非单词字符(如空格)之间。因此，该分界是一个非单词分界，即\B。

因此，需要的字符模式为：一个非单词分界后跟一个单引号，或者是一个单引号后跟一个非单词分界。关键是“或者”，为此，在正则表达式中使用 | 。于是正则表达式如下：

```
var myRegExp = /\B'|'\B/g;
```

它匹配 | 左边的模式，或者 | 右边的模式。为了将所有匹配的单引号替换为双引号，在结尾添加了 g，表示需要进行全局匹配。

试一试　将单引号替换为双引号

下面的例子使用了前面定义的正则表达式。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>example</title>
<script type="text/JavaScript">
function replaceQuote(textAreaControl)
{
   var myText = textAreaControl.value;
   var myRegExp = /\B'|'\B/g;
   myText = myText.replace(myRegExp,' " ');
   textAreaControl.value = myText;
}
</script>
</head>
<body>
<form name="form1">
<textarea rows="20" cols="40" name="textarea1">
'Hello World' said Mr O'Connerly.
He then said 'My Name is O'Connerly, yes that's right, O'Connerly'.
</textarea>
<br>
<input type="button" VALUE="Replace Single Quotes" name="buttonSplit"
   onclick="replaceQuote(document.form1.textarea1)">
</form>

</body>
</html>
```

将页面保存为 ch9_examp5.htm。在浏览器中加载该页面，结果如图 9-13 所示。

单击 Replace Single Quotes 按钮，就替换了文本区域中的单引号，如图 9-14 所示。

图 9-13

图 9-14

试着在文本区域中输入其他带单引号的文本，并检查结果。

可以看到，使用正则表达式，只使用几行简单的代码就完成了任务。如果不使用正则表达式，则可能需要多达四五倍的代码。

首先看看位于页面顶部、执行所有操作的 replaceQuote()函数。

```
function replaceQuote(textAreaControl)
{
   var myText = textAreaControl.value;
   var myRegExp = /\B'|'\B/g;
   myText = myText.replace(myRegExp, ' " ');
   textAreaControl.value = myText;
}
```

这个函数的参数是页面后面定义的 textarea 对象，该 textarea 中的单引号将被替换。textarea 对象在按钮的标记定义中传送。

```
<input type="button" value="Replace Single Quotes" name="buttonSplit"
   onclick="replaceQuote(document.form1.textarea1)">
```

在 onclick 事件处理程序中，调用了 replaceQuote()，并把 document.form1.textarea1(即 textarea 对象)作为参数传送。

回到函数中，第一行代码获取 textarea 的值，并放在变量 myText 中。接着定义了正则表达式(如前所述)，它匹配一个非单词边界后跟一个单引号，或者一个单引号后跟一个非单词分界。例如，它匹配 'H 或 H' ，但不匹配 O'R，因为 O'R 中的单引号位于两个单词分界之间。请记住，单词分界是在单词开始或结束与一个非单词字符(如空格或者标点符号)之间的位置。

该函数的最后两行首先使用 replace()方法进行字符模式搜索和替换，再将 textarea 对象的值设置为改过的字符串。

9.3.3　search()方法

search()方法可以在字符串中搜索字符模式。如果找到了该模式，则返回找到该模式的字符位置，否则返回 - 1。该方法只接收一个参数，即已创建的 RegExp 对象。

尽管 indexOf()方法足以完成基本的搜索工作，但是要进行更复杂的搜索，如搜索数字模式，或者查找某个边界中的单词，则 search()是一种更强大、更灵活但有时较为复杂的方式。

下面的例子要确定字符串是否包含单词 Java。但是，我们只查找作为独立单词出现的 Java，而不查找包含在另一个单词(如 JavaScript)中的 Java：

```
var myString = "Beginning JavaScript, Beginning Java 2, Professional JavaScript";
var myRegExp = /\bJava\b/i;
alert(myString.search(myRegExp));
```

首先定义字符串，然后创建正则表达式。要匹配的字符模式 Java 应位于两个单词分界之间。在正则表达式之后加上 i ，可以在查找时忽略大小写。注意，对于 search()方法，全局匹配标记 g 与此无关，不起作用。

最后一行代码输出定位模式的位置，在本例中是 32。

9.3.4　match()方法

match()方法与 search()方法非常类似，但 search()方法返回找到匹配的位置，而 match()方法返回一个数组，该数组的每个元素包含一个匹配的文本。

例如，如果字符串如下：

```
var myString = "The years were 1999, 2000 and 2001";
```

要从这个字符串中提取出年份，只需使用 match()方法。要匹配每个年份，应查找位于两个单词分界之间的 4 个数字，它应转换为如下正则表达式：

```
var myRegExp = /\b\d{4}\b/g;
```

我们要匹配所有的年份，因此在最后添加 g，进行全局搜索。

要匹配并存储结果，可以使用 match()方法进行查找，再将它返回的 Array 对象保存在变量中：

```
var resultsArray = myString.match(myRegExp);
```

为了验证上面的代码，可以编写一些代码输出数组的每个元素。代码添加一个 if 语句，复核结果是否真正包含一个数组。如果未找到匹配串，则返回的数组将为 null——如果变量包含一个不为 null 的值，则 if (resultsArray)返回 true。

```
if (resultsArray)
{
  var indexCounter;
  for (indexCounter = 0; indexCounter < resultsArray.length; indexCounter++)
```

```
    {
        alert(resultsArray[indexCounter]);
    }
}
```

上面的代码将弹出三个警告框，分别包含数字 1999、2000、2001。

试一试 拆分 HTML

下面的例子把一个 HTML 字符串拆分为多个组分。例如，将 HTML 代码<P>Hello </P>拆分为一个数组，其元素的内容如表 9-8 所示：

表 9-8

元 素 0	元 素 1	元 素 2
<P>	Hello	</P>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>example 6</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/JavaScript">
function button1_onclick()
{
   var myString = "<table align=center><tr><td>";
   myString = myString + "Hello World</td></tr></table>";
   myString = myString + "<br><h2>Heading</h2>";
   var myRegExp = /<[^>\r\n]+>|[^<>\r\n]+/g;
   var resultsArray = myString.match(myRegExp);
   document.form1.textarea1.value = "";
   document.form1.textarea1.value = resultsArray.join ("\r\n");
}
</script>
</head>
<body>
<form name="form1">
   <textarea rows="20" cols="40" name="textarea1"></textarea>
   <input type="button" value="Split HTML" name="button1"
      onclick="return button1_onclick();">
</form>

</body>
</html>
```

将这个文件保存为 ch9_examp6.htm。在浏览器中加载该页面，并单击 Split HTML 按钮，会拆分 HTML 字符串，每个标记都放在文本区域的单独一行上，如图 9-15 所示：

图　9-15

单击 Split HTML 按钮时，会触发页面顶部定义的 button1_onclick()函数。该函数开始的几行代码定义了要拆分的 HTML 字符串：

```
function button1_onclick()
{
   var myString = "<table align=center><tr><td>";
   myString = myString + "Hello World</td></tr></table>";
   myString = myString +"<br><h2>Heading</h2>";
```

接下来创建 RegExp 对象，并初始化为正则表达式：

```
var myRegExp = /<[^>\r\n]+>|[^<>\r\n]+/g;
```

下面进行分析，看看要匹配的模式。首先请注意，该模式被一个选择符号“｜”分为两个部分。这意味着要么匹配该符号左边的模式，要么匹配该符号右边的模式。下面分别分析这两个模式。左边的模式为：

- 该模式必须以<开始。
- [^>\r\n]+指定除>、\r(回车)、\n(换行)之外的一个或多个字符。
- >指定该模式必须以>结束。

右边的模式为：

- [^< >\r\n]+指定，该模式是一个或多个除<、>、\r 或者\n 字符之外的字符，这将匹配纯文本。

在正则表达式定义的后面有一个 g，表示这是全局匹配。

因此，正则表达式<[^>\r\n]+>将匹配 HTML 的任何开始或结束标记，如 或者 。另一个模式[^< >\r\n]+将匹配非开始或结束标记的任何字符模式。

下面的代码将 match()方法返回的 Array 对象赋予变量 resultsArray：

```
var resultsArray = myString.match(myRegExp);
```

其余的代码用拆分后的 HTML 填充文本区域。我们使用 Array 对象的 join()方法，将数组的所有元素以\r\n 为分隔符，连接成一个字符串。这样，每个 HTML 标记或文本段就放在单独一行上，如下所示：

```
    document.form1.textarea1.value = "";
    document.form1.textarea1.value = resultsArray.join("\r\n");
}
```

9.4 使用 RegExp 对象的构造函数

前面在创建 RegExp 对象时，一直用斜杠 / 和 / 来定义正则表达式的开始和结束，如下面的示例所示：

```
var myRegExp = /[a-z]/;
```

尽管这通常是首选方式，但前面还提到，也可以使用构造函数 RegExp()来创建 RegExp 对象。我们大多采用第一种方法。但是在某些情况下，例如要根据用户的输入来构造正则表达式时，则必须采用第二种方法来创建 RegExp 对象，如稍后的“试一试”所述。

采用第二种方法，可将上面的正则表达式定义为：

```
var myRegExp = new RegExp("[a-z]");
```

这里将正则表达式作为字符串参数传递给构造函数 RegExp()。

使用这种方法的一个非常重要的区别是如何使用特殊的正则表达式字符，如\b，这些字符之前都有一个反斜杠。问题是在 JavaScript 字符串中，反斜杠表示一个转义字符。例如，\b 表示退格(backspace)。为了区分\b 是字符串中的转义字符，还是正则表达式中的特殊字符，必须在正则表达式特殊字符之前再加一个反斜杠。因此\b 就变成了\\b，表示正则表达式中匹配单词分界的特殊字符，而不是退格字符。

例如，假定使用下面的代码定义 RegExp 对象：

```
var myRegExp = /\b/;
```

要使用构造函数 RegExp()声明它，则需要使用下面的代码：

```
var myRegExp = new RegExp("\\b");
```

而不能使用如下代码：

```
var myRegExp = new RegExp("\b");
```

对于所有正则表达式的特殊字符，如\w、\b、\d 等，当使用构造函数 RegExp()创建它们时，都必须在之前再添加一个反斜杠\。

使用/和/方法定义正则表达式时，可以在最后一个 / 之后添加特定的标记 m、g 和 i，分别表示模式的匹配应是多行的、全局的或者忽略大小写的。而使用构造函数 RegExp()时，如何设置这些标记？

很简单，构造函数 RegExp()的第二个可选参数设置相应的标记，以指定全局匹配或者忽略大小写。例如，下面的代码将进行忽略大小写的全局模式匹配：

```
var myRegExp = new RegExp("hello\\b","gi");
```

也可以只指定一个标记，例如下面的代码：

```
var myRegExp = new RegExp("hello\\b","i");
```

或者：

```
var myRegExp = new RegExp("hello\\b","g");
```

试一试　**表单验证模块**

这个示例将创建一组有效的 JavaScript 函数，使用正则表达式验证如下内容：

- 电话号码
- 邮政编码
- 电子邮件地址

本例的验证仅检查格式，例如，它不能检查电话号码是否实际存在，而只检查它是否有效。

首先是包含输入验证代码的.js 代码文件，注意下面的代码块太宽，本书的篇幅放不下——应确保每个正则表达式都放在一行上。

```
function isValidTelephoneNumber( telephoneNumber )
{
                var telRegExp = /^(\+\d{1,3} ?)?(\(\d{1,5}\)|\d{1,5}) ?\d{3}
 ?\d{0,7}( (x|xtn|ext|extn|pax|pbx|extension)?\.? ?\d{2-5})?$/i
                return telRegExp.test( telephoneNumber );
}

function isValidPostalCode( postalCode )
{
                var pcodeRegExp = /^(\d{5}(-\d{4})?|([a-z][a-z]?\d\d?|
                                                                [a-z{2}\d[a-z])
    ?\d[a-z][a-z])$/i
        return pcodeRegExp.test( postalCode );
}

function isValidEmail( emailAddress )
{
        var emailRegExp = /^(([^<>()\[\]\\.,;:@"\x00-\x20\x7F]|\\.)+|( "" "
([^\x0A\x0D"\\]|\\\\)+" ""))@(([a-z]|#\d+?)([a-z0-9-]|#\d+?)*
([a-z0-9]|#\d+?)\.)+([a-z]{2,4})$/i
        return emailRegExp.test( emailAddress );
}
```

保存为 ch9_examp7_module.js.。

为了测试代码，需要一个简单的页面，其中包含一个文本框和 3 个按钮，分别用于验证电话号码、邮政编码和电子邮件地址。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<title>example 7</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="ch9_examp7_module.js"></script>
</head>
<body>
<form name="form1">
  <p>
    <label>
       <input type="text" name="txtString" id="txtString" />
    </label>
</p>
<p>
  <label>
    <input type="button" name="cmdIsValidTelephoneNumber"
  id="cmdIsValidTelephoneNumber"
  value="Is Valid Telephone Number?"
onclick="alert('Is valid is ' +
                    isValidTelephoneNumber(document.form1.txtString.value))"
     />

     <input type="button" name="cmdIsValidPostalCode"
            id="cmdIsValidPostalCode"
            value="Is Valid Postal Code? "
            onclick="alert('Is valid is '
+ isValidPostalCode( document.form1.txtString.value )) " />
     <input type="button" name="cmdIsEmailValid" id="cmdIsEmailValid"
value="Is Valid Email? "
onclick="alert( 'Is valid is '
+ isValidEmail( document.form1.txtString.value )) " />
    </label>
  </p>
</form>

</body>
</html>
```

保存为 ch9_examp7.htm，并加载到浏览器中。浏览器会显示包含一个文本框和 3 个按钮的页面。输入一个有效的电话号码(本例使用+1 (123) 123 4567)，单击 Is Valid Telephone Number 按钮，屏幕将如图 9-16 所示。

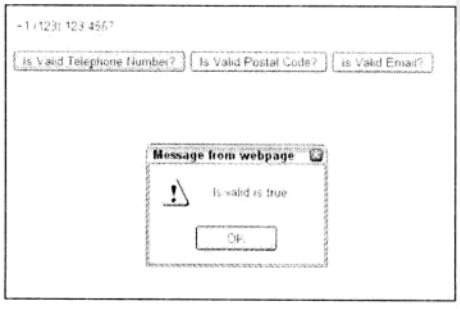

图 9-16

如果输入一个无效的电话号码，结果将是 Is Valid is false。这个例子非常简单，但足以测试代码。

实际的代码非常简单，但正则表达式较难创建，下面就详细讨论它们，从电话号码的验证开始。

9.4.1　验证电话号码

电话号码验证起来比较困难。其问题是：

- 每个国家的电话号码都不同。
- 输入有效的电话号码有不同的方式(例如是否添加国家代码或国际代码)。

对于这个正则表达式，不仅需要指定有效的字符，还需要指定数据的格式。例如，下面的电话号码都是有效的：

+1 (123) 123 4567

+1123123 456

+44 (123) 123 4567

+44 (123) 123 4567 ext 123

+44 20 7893 4567

正则表达式需要进行的验证如表 9-9 所示(用空格隔开是可选的)

表　9-9

国际号码	“+”后跟 1 到 3 个数字(可选)
本地区号	2～5 个数字，有时带括号(必选)
实际的用户号码	3～10 个数字，有时带空格(必选)
分机号码	2～5 个数字，其前面是 x、xtn、extn、pax、pbx 或 extension，有时带括号

显然，在一些国家中，这是无效的，需要根据客户和合作伙伴所处的位置进行处理。下面的正则表达式相当复杂，也很长，必须放在两行上，但用户应把它输入在一行上：

```
^(\+\d{1,3} ?)?(\(\d{1,5}\)|\d{1,5}) ?\d{3} ?\d{0,7}
( (x|xtn|ext|extn|pax|pbx|extension)?\.? ?\d{2-5})?$
```

需要给它设置不区分大小写的标记和显式捕获选项。尽管这似乎很复杂，但如果分解开来，这个正则表达式就相当简单。

首先分析匹配国际号码的模式：

```
(\+\d{1,3} ?)?
```

前面匹配过一个加号(\+)后跟 1～3 个数字(\d{1,3})和一个可选空格(?)的模式。由于+字符是一个特殊字符，所以在它的前面加上\字符，指定这是一个实际的+字符。放在括号中的字符表示一组字符。该组字符的右括号后面的?字符表示，允许有一个空格，且整组字符匹配 0 次或 1 次。

接下来是匹配区号的模式：

```
(\(\d{1,5}\)|\d{1,5})
```

这个模式包含在括号中，表示这是一组字符，匹配括号中的 1 到 5 个数字((\d{1,5}))或者仅匹配 1 到 5 个数字(\d{1,5})。由于在正则表达式的语法中，括号字符是特殊字符，而这里需要匹配实际的括号，所以需要在括号字符的前面加上\字符。还要注意使用了竖杠符号(|)，它表示“或”，即匹配这两个模式中的任意一个。

之后是匹配用户号码：

```
?\d{3,4} ?\d{0,7}
```

注意：

在第一个?符号前有一个空格：这个空格和问号表示“匹配 0 个或 1 个空格”。其后是 3 或 4 个数字(\d{3,4})——在美国，始终是 3 个数字，而在英国，则常常是 4 个数字。接着是另一个“0 个或 1 个空格”，最后是 0 到 7 个数字(\d{0,7})。

最后添加的部分处理一个可选的分机号码：

```
( (x|xtn|ext|extn|extension)?\.? ?\d{2-5})?
```

这个组是可选的，因为其括号的后面是一个问号。该组本身会检查一个空格，其后是 x、ext、xtn、extn 或 extension，再往后是 0 个或 1 个句点(注意字符，在正则表达式语法中，.字符是一个特殊字符)、0 个或 1 个空格、2 到 5 个数字。把这 4 个模式放在一起，再加上开头和结尾的语法，就可以构建出整个正则表达式。这个正则表达式从^开始，以$结束。^字符指定模式必须从字符串的开头处匹配，$字符表示模式必须匹配到字符串的结尾。这说明，字符串必须与模式完整匹配，在匹配的字符串前后都不能包含其他任何字符。

解释了正则表达式后，现在就可以把它添加到 JavaScript 模块 ch9_examp7_module.js 中，如下所示：

```
function isValidTelephoneNumber( telephoneNumber )
{
                var telRegExp = /^(\+\d{1,3} ?)?
                (\(\d{1,5}\)|\d{1,5}) ?\d{3} ?\d{0,7}
               ( (x|xtn|ext|extn|pax|pbx|extension)?
                \.? ?\d{2-5})?$/i
                return telRegExp.test( telephoneNumber );
}
```

注意在这个示例中，一定要在表达式定义的结尾添加 i，以设置不区分大小写标记，否则，该正则表达式就不能匹配 ext 部分。还要注意，正则表达式在代码中必须占一行——这里把正则表达式放在 4 行上，是因为本书的宽度有限。

9.4.2 验证邮政编码

刚才验证了世界范围内的电话号码，但验证世界范围内的邮政编码是非常困难的。下面创建的函数仅验证美国和英国的邮政编码。如果需要验证其他国家的邮政编码，就需要修改代码。在一个正则表达式中验证多于一个或两个邮政编码会很难管理，而用一个正则

表达式验证一个国家的邮政编码就容易多了。但这里把验证英国和美国的邮政编码的正则表达式合并起来：

```
^(\d{5}(-\d{4})?|[a-z][a-z]?\d\d? ?\d[a-z][a-z])$
```

这个正则表达式实际包含两部分：第一部分验证美国的邮政编码，第二部分验证英国的邮政编码。首先看看第一个部分：

邮政编码可以表示为两种格式：5 个数字(12345)或者 5 个数字后跟一个短划线和 4 个数字(12345-1234)。匹配这个邮政编码的正则表达式如下：

```
\d{5}(-\d{4})?
```

这个正则表达式匹配 5 个数字，后跟一个可选的非捕获组，该组匹配一个短划线后跟 4 个数字。

对于匹配英国邮政编码的正则表达式，也要先考虑其格式。英国邮政编码的格式是一个或两个字母后跟一个或两个数字、一个可选的空格、一个数字和两个字母。另外，伦敦的一些中心邮政编码类似于 SE2V 3ER，在第一部分的最后有一个字母。目前，只有一些邮政编码以 SE、WC 和 W 开头，但这可能有变化。有效的英国邮政编码示例有 CH3 9DR、PR29 1XX、M27 1AE、WC1V 2ER 和 C27 3AH。

因此，需要的模式如下：

```
([a-z][a-z]?\d\d?|[a-z]{2}\d[a-z]) ?\d[a-z][a-z]
```

这两个模式用“|”字符合并起来，表示匹配其中一个模式，并用括号把它们组合起来。接着在模式开头添加“^”字符，在结尾添加“$”字符，确保字符串中的信息只能是邮政编码。尽管邮政编码应是大写，但小写的邮政编码也有效，所以在使用该正则表达式时，也要设置不区分大小写的选项，如下所示：

```
^(\d{5}(-\d{4})?|([a-z][a-z]?\d\d?|[a-z{2}\d[a-z]) ?\d[a-z][a-z])$
```

验证模块需要的函数与上一个示例基本相同，如下所示：

```
function isValidPostalCode( postalCode )
{
        var pcodeRegExp = /^(\d{5}(-\d{4})?|
([a-z][a-z]?\d\d?|[a-z{2}\d[a-z]) ?\d[a-z][a-z])$/i
        return pcodeRegExp.test( postalCode );
}
```

再次注意，应把正则表达式放在一行上。

9.4.3　验证电子邮件地址

在使用正则表达式匹配电子邮件地址之前，需要先了解有效的电子邮件地址的类型。例如：

- someone@mailserver.com

- someone@mailserver.info
- someone.something@mailserver.com
- someone.something@subdomain.mailserver.com
- someone@mailserver.co.uk
- someone@subdomain.mailserver.co.uk
- someone.something@mailserver.co.uk
- someone@mailserver.org.uk
- some.one@subdomain.mailserver.org.uk

另外，如果使用 SMTP RFC (http://www.ietf.org/rfc/rfc0821.txt)，电子邮件地址可以是：

- someone@123.113.209.32
- "" "Paul Wilton" ""@somedomain.com

这是一个很长的列表，包含许多需要处理的变体。所以最好先分解它。首先，上面的两个电子邮件地址要注意两个地方。后两个版本的变形比较大，在下面创建的正则表达式中不予考虑。

需要把电子邮件地址分解为几个部分，先看看@符号后面的部分。

9.4.4 验证域名

由于允许使用 Unicode 域名，所以域名变得越来越复杂。但是电子邮件 RFC 仍不允许使用 Unicode 域名，所以这里仅考虑使用 ASCII 描述一个域的传统定义。域名包含以句点分隔的单词列表，最后一个单词的长度是 2 到 4 个字符。如果国家使用包含 2 个字母的单词来表示，则域名的前面常常至少有两个部分：分组的域(.co, .ac 等)和指定的域名。但是，随着.tv 名称的出现，情况发生了变化。可以把这种域名看作特例，为允许使用的顶级域(TLD)提供该特例，但这会使正则表达式非常大，而仅执行 DNS 查找的效率会比较高。

域名的每个部分都有一些必须遵循的规则。域名可以包含任何字母、数字或者连字符，但必须以字母开头。但有一个例外：在域名的任何地方，都可以使用#后跟一个数字，表示该字母的 ASCII 码，或者 16 位的 Unicode 值。知道了这些，下面就开始建立这个正则表达式，首先是域名部分，假定在代码的后面设置了不区分大小写的标记：

```
([a-z]|#\d+)([a-z0-9-]|#\d+)*([a-z0-9]|#\d+)
```

它把域分成 3 部分。RFC 没有指定这里可以包含多少个数字，所以我们也没有指定。第一部分只能包含一个 ASCII 字母，第二部分必须包含 0 个或多个字母、数字或连字符，第三部分必须包含字母或数字。顶级域限制的更多，如下所示：

```
[a-z]{2,4}
```

这表示顶级域可以有 2、3 或 4 个字母。将这些放在一起，最后添加一个句点：

```
^(([a-z]|#\d+?)([a-z0-9-]|#\d+?)*([a-z0-9]|#\d+?)\.)+([a-z]{2,4})$
```

域名应从字符串的开头开始，到字符串的结尾结束。首先要添加一个额外的组，以允许包含一个或多个 name.部分，再在它自己的组中用 2 到 4 个字母的域名结尾。我们还使

大多数通配符进行懒惰匹配。由于模式的许多部分是类似的，所以这么做是合理的，否则就需要非常多的回溯。而第二组还使用了一个“贪婪”通配符，它会匹配尽可能多的字符，直到遇到不匹配的字符为止。接着回溯一个位置，尝试第三组的匹配。在利用资源方面，这比懒惰匹配的效率高，因为它可以一直向前尝试匹配。每个名字回溯一次是可以接受的额外处理量。

1. 验证用户的地址

现在可以尝试验证@符号前面的部分。RFC 指定该部分可以包含其代码在 33～126 之间的任意 ASCII 字符。假定仅匹配 ASCII，所以可以假定引擎仅需要匹配 128 个字符。本例采用这种假定，排除了如下的情况：

```
[^<>()\[\],;:@ "\x00-\x20\x7F]+
```

有了这个假定，就表示只要字符没有包含在方括号中，这个部分允许使用任意多个字符。[、]和\字符必须转义。但 RFC 允许进行其他类型的匹配。

2. 验证完整的地址

阅读了前面的各节后，就可以为验证完整的电子邮件地址建立正则表达式了。首先，把前面的正则表达式合并起来，并包含@符号：

```
^([^<>()\[\],;:@ "\x00-\x20\x7F]|\\.)+@
```

这很简单。下面是域名部分：

```
^([^<>()\[\],;:@ "\x00-\x20\x7F]|\\.)+@(([a-z]|#\d+?)([a-z0-9-]
|#\d+?)*([a-z0-9]|#\d+?)\.)+([a-z]{2,4})$
```

由于本书的宽度有限，只能把该正则表达式放在两行上，但在实际的代码中，它必须单独占一行。

最后为 JavaScript 模块创建函数：

```
function isValidEmail( emailAddress )
{
        var emailRegExp = /^([^<>()\[\],;:@"\x00-\x20\x7F]|\\.)+
@(([a-z]|#\d+?)([a-z0-9-]|
#\d+?)*([a-z0-9]|#\d+?)\.)+([a-z]{2,4})$/i
        return emailRegExp.test( emailAddress );
}
```

注意正则表达式必须放在一行上。

分析完了模块的建立后，就该分析测试该模块的代码了。

首先，链接到测试页的模块如下所示：

```
<script type="text/javascript" src="ch9_examp7_module.js"></script>
```

接着，3 个测试按钮把各自的单击事件链接到模块中的验证函数上，如下所示：

```
<input type="button" name="cmdIsValidTelephoneNumber"
        id="cmdIsValidTelephoneNumber"
 value="Is Valid Telephone Number? "
onclick="alert('Is valid is ' +
isValidTelephoneNumber( document.form1.txtString.value )) " />
      <input type="button" name="cmdIsValidPostalCode" d="cmdIsValidPostalCode"
 value="Is Valid Postal Code? "
onclick="alert('Is valid is ' +
isValidPostalCode( document.form1.txtString.value )) " />
      <input type="button" name="cmdIsEmailValid" id="cmdIsEmailValid"
value="Is Valid Email? "
onclick="alert('Is valid is'+isValidEmail( document.form1.txtString.value ))" />
```

对于电话号码验证测试按钮，添加一个 onclick 事件处理程序：

```
onclick="alert('Is valid is ' +
isValidTelephoneNumber( document.form1.txtString.value )) "
```

这会显示一个警告框，返回验证模块中 isValidTelephoneNumber()函数的 true 或 false 值。在非测试情况下，可以显示更友好的消息。其他两个测试按钮的工作方式相同，但调用不同的验证函数。

9.5 小结

本章介绍了 String 对象的一些高级方法，以及如何使用正则表达式优化它们的用法。本章的主要内容如下：

- split()方法将一个字符串拆分为一个字符串数组。给该方法传送某个字符串或者正则表达式，以决定在何处拆分。
- replace()方法将一个字符模式替换为第二个参数指定的另一个模式。
- search()方法返回与参数匹配的第一个模式的字符位置。
- match()方法匹配模式，把匹配的文本返回到一个数组中。
- 正则表达式允许定义要匹配的字符模式。根据该模式，可以对字符串进行拆分、查找、文本替换或者匹配。
- 在 JavaScript 中，正则表达式使用 RegExp 对象的形式。可以使用 myRegExp = /myRegularExpression/或 myRegExp = new RegExp("myRegularExpression")创建 RegExp 对象，第二种形式要求，原来前面只有一个\的特殊字符现在需要使用\\。
- 正则表达式末尾的 g 和 i 字符(例如，myRegExp =/Pattern/gi)可确保进行全局匹配，并忽略大小写。
- 除了定义实际的字符之外，正则表达式还有一些特殊字符，可用于匹配特定的字符组，如数字、单词或非单词字符。
- 可以使用特殊字符来设置模式或字符的重复。另外，还可以指定模式边界必须是什么，例如，在字符串的开始或结束位置进行匹配，或者与单词分界、非单词分界进行匹配。

- 最后，可以定义字符组，该分组可在之后的正则表达式中使用，或者在包含 replace() 方法的表达式中使用。

第 10 章将介绍在 JavaScript 中如何使用、处理日期和时间，如何在不同的时区之间转换时间；以及如何创建一个计时器，以便在加载页面后，每隔一段时间就执行一次代码。

9.6　习题

本意习题的参考答案在附录 A 中。

习题 1：

下面的代码解决了什么问题？

```
var myString = "This sentence has has a fault and and we need to fix it."
var myRegExp = /(\b\w+\b) \1/g;
myString = myString.replace(myRegExp,"$1");
```

现在假定修改代码，以创建如下的 RegExp 对象：

```
var myRegExp = new RegExp("(\b\w+\b) \1");
```

为什么这行代码不起作用？该如何修正存在的问题？

习题 2：

请编写一个正则表达式，以查找下述语句中的所有单词 a，并替换为单词 the：

"a dog walked in off a street and ordered a finest beer"

替换之后的语句应为：

"the dog walked in off the street and ordered the finest beer"

习题 3：

假如有一个带有留言板的网站，请编写一个正则表达式，以删除禁用的词汇(可以自己设置禁用的词汇)。

第10章

日期、时间和计时器

第 5 章通过 Date 对象讨论了 JavaScript 中的日期和时间，学习了 Date 对象的一些属性和方法，包括：

- getDate()、getDay()、getMonth()和 getFullYear()方法可以从 Date 对象中获取相应的日期值。
- setDate()、setMonth()和 setFullYear()方法可以给已有的 Date 对象设置相应的日期值。
- getHours()、getMinutes()、getSeconds()和 getMilliseconds()方法可以从 Date 对象中获取相应的时间值。
- setHours()、setMinutes()、setSeconds()和 setMilliseconds()方法可以给已有的 Date 对象设置相应的时间值。

时间取决于世界上不同的地理位置，第 5 章未提及这方面的内容。本章将学习日期、时间与世界时(world time)的关系，以修正前面忽略的问题。

例如，假定网站上有一个聊天室，现在希望在某个日期和时间组织一次聊天会。如果网站吸引了其他国家的访客，则简单地将聊天时间定为 15:30 开始就不太合适。因为 15:30 可以是东部标准时间(EST)、太平洋标准时间(PST)、英国时间、甚或吉隆坡时间。当然，可以说明是 15:30 EST，让访客计算他所在地区的时间，但这并非万全之策，因为澳大利亚和美国都有 EST。如果能自动将该时间转换为访客所在地区的时间，岂不是更好？本章将介绍如何实现此类转换。

除了介绍世界时外，本章还将介绍如何在网页中创建计时器(timer)。使用计时器，可以每隔一定时间就触发一次代码，或者仅执行一次代码(例如在页面加载后的 5 秒)。本章将介绍如何使用计时器在网页上创建一个实时时钟，以及如何在状态栏中创建滚动的文本。在 Web 应用程序中，计时器还可以用来创建动画或者特效。最后，小测验的用户可以使用计时器来设置回答问题的时限。

10.1　世界时

“现在”这个概念的含义是世界上每个地方在此刻的同一时间。但是，此刻的时间用数字来表示，在不同的地方，该数字是不同的。因此需要一个标准数字来表示不同地点的同一时间。这就是协调世界时(Coordinated Universal Time，UTC)，该标准于 1964 年执行，是国际性的民用和科学记时的基础。它的前身是格林尼治标准时间(Greenwich Mean Time，GMT)，实际上，UTC 时间 0:00 是伦敦格林尼治的午夜。

表 10-1 显示了 UTC 时间 0:00 时世界各地的本地时间：

表　10-1

圣弗朗西斯科	纽约(EST)	伦敦格林尼治	德国柏林	日本东京
下午 4:00	下午 7:00	0:00 (午夜)	上午 1:00	上午 9:00

注意：

上表给出的是各地的冬季时间——夏令时未考虑在内。

JavaScript 中的 Date 对象提供了许多方法以支持 UTC 时间，它们类似于前面介绍的方法。对于每个 set 或 get 类型的日期时间方法，都有一个相应的 UTC 方法。例如，setHours()设置 Date 对象本地时间的小时值，setUTCHours()方法设置 UTC 时间的小时值。下一节将详细介绍这些方法。

另外，Date 对象还有 3 个方法也涉及到世界时。

toUTCString()方法和 toLocaleString()方法根据 UTC 或本地时间，把 Date 对象中保存的日期和时间返回为一个字符串。大部分现代浏览器都支持 toLocaleTimeString()、toTimeString()、toLocaleDateString()和 toDateString()等方法。

如果只想计算当前本地时间与 UTC 时间的差值，可以使用 getTimezoneOffset()方法。如果时区在 UTC 之后，如美国，该方法就返回一个正数。如果当前时区在 UTC 之前，如澳大利亚或日本，该方法就返回一个负数。

试一试　Date 对象的世界时方法

下面的代码使用了 toLocaleString()、toUTCString()、getTimezoneOffset()、toLocale Time String()、toTimeString()、toLocaleDateString()和 toDateString()方法，并将它们的值输出在页面上。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>example 1</title>

</head>

<body>
```

```
<div id="DisplayResultsDiv"></div>

<script type="text/javascript">
  var localTime = new Date();
  var resultsHTML = '<p>UTC Time is ' + localTime.toUTCString() + '</p>';
  resultsHTML += 'Local Time is ' + localTime.toLocaleString() + '</p>';

  resultsHTML += '<p>Time Zone Offset is ' + localTime.getTimezoneOffset() +
    '</p>';
  resultsHTML += '<p>Using toLocalTimeString() gives: '
                  + localTime.toLocaleTimeString() + '</p>';
  resultsHTML += '<p>Using toTimeString() gives: '
                  + localTime.toTimeString() + '</p>';
  resultsHTML += '<p>Using toLocaleDateString() gives: '
                  + localTime.toLocaleDateString() + '</p>';
  resultsHTML += '<p>Using toDateString() gives: : '
                  + localTime.toDateString() + '</p>';
  document.getElementById('DisplayResultsDiv').innerHTML = resultsHTML;

</script>

</body>
</html>
```

保存为 timetest.htm，并加载到浏览器中。当然，结果取决于计算机所设置的时区，但浏览器中的页面应如图 10-1 所示。

UTC Time is Mon, 18 May 2009 09:13:32 UTC
Local Time is 18 May 2009 05:13:32
Time Zone Offset is 240
Using toLocalTimeString() gives: 05:13:32
Using toTimeString() gives: 05:13:32 EDT
Using toLocaleDateString() gives: 18 May 2009
Using toDateString() gives: : Mon May 18 2009

图 10-1

当前计算机的时间是美国东部标准时间(例如，纽约)2009 年 5 月 18 日上午 05:13:32。代码是怎样运行的？在页面顶部的脚本块中，代码如下：

```
var localTime = new Date();
```

该代码创建了一个新的 Date 对象，并根据客户端计算机的时钟初始化为当前日期和时间。注意，Date 对象只保存了自 UTC 时间 1970 年 1 月 1 日午夜到客户端计算机时钟的日期时间的毫秒数。

在脚本块的其余代码中，得到了各个时间日期函数的结果。结果存储在 resultsHTML 变量中，并使用最后一行代码和 innerHTML 属性显示在页面上。

下面的代码将 toUTCString()方法返回的字符串存储在 resultsHTML 变量中：

```
var resultsHTML = '<p>UTC Time is ' + localTime.toUTCString() + '</p>';
```

它把 Date 对象 localTime 中的日期和时间转换为对应的 UTC 日期和时间。

接着，下面的代码在字符串中存储了本地日期和时间值：

```
resultsHTML += 'Local Time is ' + localTime.toLocaleString() + '</p>';
```

这个时间仅基于用户计算机的时钟，该方法返回的字符串可能是夏令时(如果计算机时钟调整为夏令时)。

接下来的代码在字符串中存储了本地时间与 UTC 时间之间的差值：

```
resultsHTML += '<p>Time Zone Offset is ' + localTime.getTimezoneOffset() +
  '</p>';
```

在图 10-1 中可以看到，纽约时间与 UTC 时间之差是 240 分钟，即 4 小时。但是在前面的表格中，纽约时间比 UTC 时间晚 5 小时，这是怎么回事呢？

纽约在 5 月 18 日使用了夏令时。当 UTC 时间为 0:00 时，夏令时的纽约时间为下午 8:00；冬令时的纽约时间为下午 7:00。因此，夏天，getTimezoneOffset()方法返回 240 分钟，而冬天，getTimezoneOffset()方法返回 300 分钟。

请对比图 10-1 和图 10-2，在图 10-2 中，计算机时钟的日期提前到 12 月，此时是冬天，没有使用夏令时。

```
UTC Time is Fri, 18 Dec 2009 10:23:08 UTC
Local Time is 18 December 2009 05:23:08
Time Zone Offset is 300
Using toLocalTimeString() gives: 05:23:08
Using toTimeString() gives: 05:23:08 EST
Using toLocaleDateString() gives: 18 December 2009
Using toDateString() gives: : Fri Dec 18 2009
```

图　10-2

下面两个方法是 toLocaleTimeString()方法和 toTimeString()方法，如下所示：

```
resultsHTML += '<p>Using toLocalTimeString() gives: ' +
                  localTime.toLocaleTimeString() + '</p>';
resultsHTML += '<p>Using toTimeString() gives: ' +
                  localTime.toTimeString() + '</p>';
```

这些方法仅显示 Date 对象中日期/时间的时间部分。toLocaleTimeString()方法显示用户在其计算机上指定的时间。第二个方法显示了时间部分和对应的时区(在本例中，EST 表示美国东部标准时间)。

最后两个方法显示日期/时间的日期部分。toLocaleDateString()方法按用户在其计算机上设定的格式显示日期。在 Windows 操作系统中，可以在计算机控制面板的“区域和语言选项”中设置。但是，由于该方法依赖用户计算机的设置，因此日期的格式因计算机而异。toDateString()方法使用标准格式显示用户计算机上的当前日期。

当然，这个例子取决于用户计算机是否正确设置了时钟，很多用户的本地时区设置可能是错误的，因此并不能保证结果完全正确。

设置和获取 Date 对象的 UTC 日期/时间

在创建新的 Date 对象时，可以将其初始化为一个值，或者让 JavaScript 将它设置为当前的日期和时间。无论采用哪种方法，JavaScript 都假定我们设置的是本地的时间值。如果要指定 UTC 时间，则需要使用 setUTC 类型的方法，例如 setUTCHours()。

下面的 7 个方法用于设置 UTC 日期和时间：

- setUTCDate()
- setUTCFullYear()
- setUTCHours()
- setUTCMilliseconds()
- setUTCMinutes()
- setUTCMonth()
- setUTCSeconds()

这些函数的名称说明了其功能。下面是一个设置 UTC 时间的简单例子。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>example 2</title>

</head>
<body>
<div id="DisplayResultsDiv"></div>

<script type="text/javascript">
        var myDate = new Date();
        myDate.setUTCHours(12);
        myDate.setUTCMinutes(0);
        myDate.setUTCSeconds(0);
        var resultsHTML = '<p>' + myDate.toUTCString() + '</p>';
        resultsHTML += '<p>' + myDate.toLocaleString() + '</p>';

        document.getElementById('DisplayResultsDiv').innerHTML = resultsHTML;

</script>

</body>
</html>
```

保存为 settimetest.htm 文件。在浏览器中加载它，网页应如图 10-3 所示，其中显示的日期依赖于当前的日期和计算机所在的时区。

Mon, 18 May 2009 12:00:00 UTC
18 May 2009 08:00:00

图 10-3

修改计算机上的时区和年份，以测试在不同时区和使用夏时制时页面的变化。例如英国的用户将计算机上的设置修改为美国东部标准时间。在 Windows 中，可以打开控制面板，双击“日期/时间”图标进行设置。

那么，这个例子是如何工作的？首先声明了一个变量 myDate，并将其设置为一个新的 Date 对象。由于未将该 Date 对象设置为任何值，因此它包含当前日期和时间。

接着，使用 setUTC 方法，设置小时、分钟和秒，使时间是 12:00:00 UTC(注意是中午，而不是午夜)。

接下来，将 myDate 的值输出为一个 UTC 字符串，会得到 12:00:00 和当天的日期。而将 Date 对象的值输出为本地字符串时，将得到当天的日期和一个将 UTC 时间 12:00:00 转换为本地时间的值。当然，本地时间的值与计算机时区相关。例如由于夏令时的缘故，在夏季，UTC 时间 12:00:00 是纽约时间 08:00:00，而在冬季是纽约时间 07:00:00。在英国，UTC 时间 12:00:00 的冬季时间是 12:00:00，而夏季时间是 13:00:00。

与设置 UTC 的日期/时间类似，获取 UTC 的日期/时间具有相应的函数，但使用 getUTC 类型的方法，例如，使用 getUTCHours()，而不是 setUTCHours()。

- getUTCDate()
- getUTCDay()
- getUTCFullYear()
- getUTCHours()
- getUTCMilliseconds()
- getUTCMinutes()
- getUTCMonth()
- getUTCSeconds()

注意，其中有一个额外的方法 getUTCDay()。它的工作方式与 getDay()方法相同，它返回一个表示星期几的数字，其中，0 表示星期日，6 表示星期六。由于星期几通过日期的年、月、日来确定，因此没有 setUTCDay()方法。

在介绍计时器之前，先应用上面学到的 Date 对象和世界时知识，创建一个世界时转换器。本章后面学习了如何使用计时器后，就更新这个示例，生成世界时的时钟。

试一试　世界时转换器(第一部分)

世界时转换器可以计算不同国家的时间。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>example 3</title>

<script type="text/javascript">
var timeDiff;
var selectedCity;
var daylightSavingAdjust = 0;
```

```
function updateTimeZone()
{
   var lstCity = document.form1.lstCity;
   timeDiff = lstCity.options[lstCity.selectedIndex].value;
   selectedCity = lstCity.options[lstCity.selectedIndex].text;
   updateTime();
}
function getTimeString(dateObject)
{
   var timeString;
   var hours = dateObject.getHours();
   if (hours < 10)
   hours = "0" + hours;
   var minutes = dateObject.getMinutes();
   if (minutes < 10)
      minutes = "0" + minutes;
   var seconds = dateObject.getSeconds()
   if (seconds < 10)
      seconds = "0" + seconds;
timeString = hours + ":" + minutes + ":" + seconds;
   return timeString;
}
function updateTime()
{
   var nowTime = new Date();
   var resultsText = '<p>Local Time is ' + getTimeString(nowTime) + '</p>';

   nowTime.setMinutes(nowTime.getMinutes() + nowTime.getTimezoneOffset() +
      parseInt(timeDiff) + daylightSavingAdjust);

   resultsText += '<p>' + selectedCity + ' time is ' +
                     getTimeString(nowTime) +  '</p>' ;

   document.getElementById('ConversionResultsDIV').innerHTML = resultsText;

}
function chkDaylightSaving_onclick()
{
   if (document.form1.chkDaylightSaving.checked)
   {
      daylightSavingAdjust = 60;
   }
   else
   {
      daylightSavingAdjust = 0;
   }
   updateTime();
}
</script>
</head>
<body onload="updateTimeZone()">
```

```
<div id="ConversionResultsDIV" ></div>

<form name="form1">
<select size="5" name="lstCity" onchange="updateTimeZone();">
<option value="60" selected>Berlin
<option value="330">Bombay
<option value="0">London
<option value="180">Moscow
<option value="-300">New York (EST)
<option value="60">Paris
<option value="-480">San Francisco (PST)
<option value="600">Sydney
</select>
<p>
It's summertime in the selected city
and its country adjusts for summertime daylight saving
<input type="checkbox" name="chkDaylightSaving"
   onclick="return chkDaylightSaving_onclick()">
</p>
</form>


</body>
</html>
```

把这个页面保存为 WorldTimeConverter.htm，然后加载到浏览器中。

表单布局如图 10-4 所示。只要用户单击了列表中的一个城市，就会显示该用户的本地时间和所选城市的对应时间。在图 10-4 显示的示例中，本地区域设置为美国东部标准时间(对应于纽约市)，选中的城市是柏林，并选中了夏令时复选框。

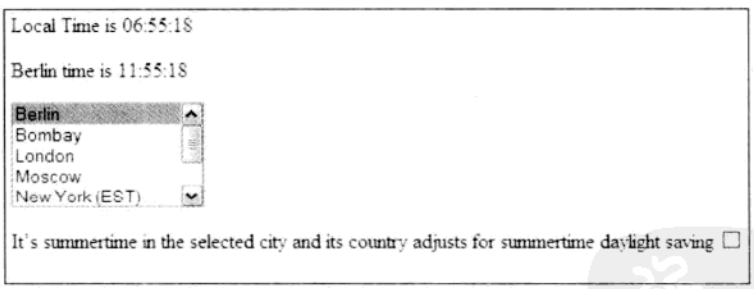

图　10-4

需要指出的是，这仅是一个例子，并非一个无懈可击的程序，原因在于夏令时的问题。某些国家没有使用夏令时，一些国家则在一年中的某些固定时期使用夏令时，还有一些国家在一年的不同时期使用夏令时。所以很难准确地预设某个国家使用夏令时的时期。因此本例尝试解决这个问题，在页面中加入了一个复选框，如果用户从列表中选择的城市使用了夏令时，就可以选中这个复选框(假定夏令时是提前 1 小时)。

另外，不要忘了，一些用户的区域设置可能有误——这个问题没有什么简单的方法来解决。

在 World Time Converter 页面正文中有一个表单，在表单中使用元素定义了一个<select>列表框。

```
<select size="5" name="lstCity" onchange="updateTimeZone();">
<option value="60" selected>Berlin
<option value="330">Bombay
<option value="0">London
<option value="180">Moscow
<option value="-300">New York (EST)
<option value="60">Paris
<option value="-480">San Francisco (PST)
<option value="600">Sydney
</select>
```

每个选项在列表框中显示一个城市名称，其值设置为该城市所在时区(冬季时间)与 UTC 时间之间相差的分钟数。因此，对于使用 UTC 的伦敦，该选项的值为 0。巴黎比 UTC 早 1 小时，因此巴黎选项的值为 60(即 60 分钟)。纽约比 UTC 晚 5 小时，因此纽约选项的值为 - 300。

代码捕获了<select>元素的 change 事件，并将它连接到在页面顶部的脚本块中定义的 updateTimeZone()函数上。这个函数使用了在脚本块顶部定义的三个全局变量。

```
var timeDiff;
var selectedCity;
var daylightSavingAdjust = 0;
```

updateTimeZone()函数更新了其中两个全局变量的值，即将变量 timeDiff 设置为列表框选中项的值(即所选城市时间与 UTC 时间之差)，将变量 selectedCity 设置为列表框选中项的文本(即所选城市名)。

```
function updateTimeZone()
{
   var lstCity = document.form1.lstCity;
   timeDiff = lstCity.options[lstCity.selectedIndex].value;
   selectedCity = lstCity.options[lstCity.selectedIndex].text;
```

updateTimeZone()函数的最后部分调用 updateTime()函数，如下所示：

```
   updateTime();
}
```

在介绍 updateTime()函数之前，先看看页面表单的最后部分。这是一个复选框，如果用户所选城市所属的国家使用夏令时，而且当前处于夏季，就可以选中该复选框。

```
<input type="checkbox" name="chkDaylightSaving"
   onclick="return chkDaylightSaving_onclick()">
```

可以看到，复选框的 click 事件连接到另一个函数 chkDaylightSaving_onclick()上。

```
function chkDaylightSaving_onclick()
{
```

```
    if (document.form1.chkDaylightSaving.checked)
    {
       daylightSavingAdjust = 60;
    }
    else
    {
       daylightSavingAdjust = 0;
    }
```

在 if 语句中，代码访问复选框的 checked 属性，如果复选框被选中，则该属性返回 true，否则返回 false。如果复选框被选中，则将 daylightSavingAdjust 全局变量设置为 60，以表示夏令时，否则设置为 0。

```
    updateTime();
}
```

这个函数的结尾处(与前面的 updateTimeZone()函数末尾一样)调用了 updateTime()函数。下面分析 updateTime()函数。

在 updateTime()函数中，将当前时间和所选城市的相应时间写入在框架集页面中定义的、ID 为 ConversionResultsDIV 的结果 DIV 中。

该函数首先创建一个新的 Date 对象，并保存在变量 nowTime 中。该 Date 对象初始化为当前本地时间。

```
function updateTime()
{
   var nowTime = new Date();
```

接下来，为使代码更简洁易读，定义了一个变量 resultsText，在把转换结果写入页面包含的 DIV 对象/ConversionResultsDIV 中之前，使用该变量存储这些结果。

首先，根据刚才创建的新 Data 对象在变量 resultsText 中保存本地时间。但是，时间应格式化为“小时:分钟:秒”，因此编写另一个函数 getTimeString()进行这个格式化。稍后讨论它。

```
var resultsText = '<p>Local Time is ' + getTimeString(nowTime) + '</p>';
```

将当前的时间保存在 resultsText 变量中后，需要计算所选城市的对应时间，然后保存在 resultsText 变量中。

第 5 章说过，如果将 Date 对象各部分的值(如小时、分钟或秒)设置为超过其正常范围的值，JavaScript 就假定要调整相应的日期、小时或者分钟。例如，如果把小时数设置为 36，JavaScript 就把小时数设置为 12，再把 Date 对象保存的日期加上 1 天。下面的代码利用了这个特性：

```
nowTime.setMinutes(nowTime.getMinutes() + nowTime.getTimezoneOffset() +
   parseInt(timeDiff) + daylightSavingAdjust);
```

分析上面的代码，分析其工作方式。假定纽约当地夏季时间是 5:11，那么柏林时间现在是几点，这该如何计算呢？

首先，获取当前本地时间的分钟数，现在是 5:11，因此，nowTime.getMinutes()返回 11。

第二步，使用 nowTime.getTimezoneOffset()计算用户本地时间与 UTC 相差的分钟数。例如纽约夏季时间与 UTC 时间相差 4 小时，即相差 240 分钟。

第三步，计算所选城市的标准冬季时间与 UTC 时间差值(取整数)，并保存在变量 timeDiff 中。这里使用了 parseInt()函数，因为这是 JavaScript 会混淆的几种情况之一，JavaScript 假定我们要把两个字符串连接在一起，而不是把它们的值当作数字，对它们求和。注意，timeDiff 保存了 HTML 元素的值，而 HTML 元素的值都是字符串，即使它保存的是数字，该值也是一个字符串。我们需要柏林时间，而柏林时间与 UTC 时间相差 60 分钟，因此，timeDiff 的值为 60。

最后加上 daylightSavingsAdjust 的值。在函数 chkdaylightsaving_onclick()中设置该变量，稍后讨论这个函数。因为在夏季，柏林使用夏令时，所以这个值为 60。

结果如下：

```
11 + 240 + 60 + 60 = 371
```

这样，nowTime.setMinutes()把分钟数设置为 371。显然，小时后的分钟数不可能为 371，因此 JavaScript 假定这表示 5:00 后的 6 小时 11 分，即 11:11——我们需要的柏林时间。

最后，updateTime()函数更新 resultsText 变量，再把结果写入 ConversionResultsDIV 中。

```
   resultsText += '<p>' + selectedCity + ' time is ' +
                     getTimeString(nowTime) + '</p>' ;

   document.getElementById('ConversionResultsDIV').innerHTML = resultsText;
}
```

在 updateTime()函数中，使用 getTimeString()函数格式化时间字符串。现在讨论这个函数。该函数接收一个 Date 对象作为参数，并使用它创建一个形如“小时:分钟:秒”的字符串。

```
function getTimeString(dateObject)
{
   var timeString;
   var hours = dateObject.getHours();
   if (hours < 10)
      hours = "0" + hours;
   var minutes = dateObject.getMinutes();
   if (minutes < 10)
      minutes = "0" + minutes;
   var seconds = dateObject.getSeconds()
   if (seconds < 10)
      seconds = "0" + seconds;
   timeString = hours + ":" + minutes + ":" + seconds;
   return timeString;
}
```

为什么需要这个函数？不能使用如下字符串：

```
getHours() + ":" + getMinutes() + ":" + getSeconds()
```

这个字符串没有考虑上述函数的 3 个结果小于 10 的情况。例如，12 点过 1 分显示为 12:1:00，而不是 12:01:00。

因此，这个函数获取小时、分钟和秒的值，并检查是否它们小于 10。如果小于 10，则在其前添加一个 0。检查了这些值后，把它们连接到一个变量 timeString 中，返回给调用函数。

下一节将介绍如何添加一个计时器，使显示的时间每秒更新一次，就像时钟一样。

10.2　网页中的计时器

可以创建两类计时器，即一次性计时器和定期触发的计时器。一次性计时器仅在指定的时间后触发一次，而第二种计时器每隔一定的时间就触发一次。下面两节将详细讨论这两种计时器。

只要在合理的限制内，就可以设置任意多个计时器，并在代码的任何地方启动计时器，例如在窗口的 onload 事件中或单击按钮时启动计时器。计时器常用于定期变化的横幅广告图片，或者在网页中显示变化的时间。另外，使用 DHTML 创建的各种动画，都需要 setTimeout()或 setInterval()方法——本书后面将介绍 DHTML。

10.2.1　一次性计时器

设置一次性计时器非常简单：只需使用 window 对象的 setTimeout()方法。

```
window.setTimeout("your JavaScript code", milliseconds_delay)
```

setTimeout()方法接收两个参数，第一个是要执行的 JavaScript 代码，第二个是执行代码之前延迟的毫秒(即千分之一秒)数。

该方法返回一个整数，即该计时器的唯一 ID。如果后来想要停止计时器的启动，就可以使用该 ID 告诉 JavaScript 指的是哪个计时器。

例如，要设置一个在页面加载 3 秒后触发的计时器，就可以使用如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script type="text/javascript">
var timerID;
function window_onload()
{
   timerID = setTimeout("alert('Times Up!') ",3000);
   alert('Timer Set');
}
</script>
```

```
</head>
<body onload="window_onload()">
</body>
</html>
```

将这个文件保存为 timertest.htm，并在浏览器中加载。在窗口的 onload 事件触发的 3000 毫秒(即 3 秒)后，将弹出一个消息框。

setTimeout()方法还可以直接引用函数，而不是引用 JavaScript 字符串。例如如果有一个函数 myFunction，就可以采用如下方式调用 setTimeout()：

```
window.setTimeout(myFunction, milliseconds_delay)
```

尽管 setTimeout()是 window 对象的方法，但由于 window 对象位于层次结构的顶端，所以在引用其属性或者方法时，不需要使用其名称。因此，可以将 window.setTimeout()方法简写为 setTimeout()方法。

值得注意的是，设置计时器，并不会停止脚本的继续执行。计时器将在后台运行，并在到时间后触发。同时，页面像往常一样运行，在启动计时器的倒计时后，脚本将立即运行。因此，上面的例子会在执行完设置计时器的代码后，立即显示一个说明计时器已设置的警告框。

但是，如何在计时器触发前停止计时器的执行？

要清除计时器，可以使用 window 对象的 clearTimeout()方法。该方法仅接收一个参数，即 setTimeout()方法返回的唯一的计时器 ID。

修改前面的例子，以提供一个按钮，当单击按钮时将停止计时器的执行。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script type="text/javascript">
var timerID;
function window_onload()
{
   timerID = setTimeout("alert('Times Up!') ",3000);
   alert('Timer Set');
}

function butStopTimer_onclick()
{
   clearTimeout(timerID);
   alert("Timer has been cleared");
}

</script>
</head>
<body onload="window_onload()">

<form name="form1">
<input type="button" value="Stop Timer" name="butStopTimer"
```

```
    onclick="return butStopTimer_onclick()" />
</form>

</body>
</html>
```

保存为 timertest2.htm，并在浏览器中加载该文件。现在，如果在 3 秒钟的延迟来临之前，单击 Stop Timer 按钮，就会清除该计时器。这是因为该按钮连接到 butStopTimer_onclick()函数上，该函数使用 window 对象的 clearTimeout()方法，清除了 ID 为 timerID 的计时器。

试一试　更新横幅广告

下面是使用 setTimeout()方法的一个较大的例子。这个示例将创建一个网页，其中的横幅广告每隔几秒就更新一次。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script language=JavaScript type="text/javascript">
var currentImgNumber = 1;
var numberOfImages = 3;
function window_onload()
{
   setTimeout("switchImage()",3000);
}
function switchImage()
{
   currentImgNumber++;
   document.imgAdvert.src = 'AdvertImage' + currentImgNumber + '.jpg';
   if (currentImgNumber < numberOfImages)
   {
      setTimeout("switchImage()",3000); }
   }
</script>
</head>
<body onload="window_onload()">
<img src="AdvertImage1.jpg" name="imgAdvert" />

</body>
</html>
```

输入上面的代码后，就把该页面保存为 adverts.htm。另外还需要创建三个图像 AdvertImage1.jpg、AdvertImage2.jpg 和 AdvertImage3.jpg 文件(也可以从本书的下载代码中获得这三个文件)。

加载页面时，会先显示 AdvertImage1.jpg，如图 10-5 所示。

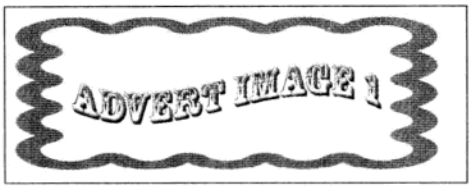

图 10-5

3 秒后显示第二幅图片，如图 10-6 所示。

图 10-6

最后，再过 3 秒将显示第三幅图片，也就是最后一幅图片，如图 10-7 所示。

图 10-7

页面加载时，<img>标记的 src 属性设置为第一幅图片。

```
<img src="AdvertImage1.jpg" name="imgAdvert">
```

在<body>标记中，将 window 对象的 onload 事件处理程序连接到 window_onload()函数上：

```
function window_onload()
{
   setTimeout("switchImage()",3000)
}
```

在这个函数中，使用 setTimeout()方法启动计时器的运行，该计时器在 3 秒后调用函数 switchImage()。因为不必清除这个计时器，所以不必保存 setTimeout()方法返回的计时器 ID。

switchImage()函数更改与页面中的<img>标记对应的 img 对象的 src 属性值：

```
function switchImage()
{
   currentImgNumber++;
   document.imgAdvert.src = 'AdvertImage' + currentImgNumber + '.jpg';
```

广告图片名中的编号从 1 到 3，即 AdvertImage1.jpg、AdvertImage2.jpg 和 AdvertImage3.jpg。把页面中当前加载的广告图片的编号保存在全局变量 currentImgNumber 中，该变量定义于脚本块的顶部，并初始化为 1。要获得下一幅图片，只需给变量 currentImgNumber 递增 1，然后使用变量 currentImgNumber 来构造完整的图片文件名，设置 img 对象的 src 属性，以更新所加载的图片。

```
    if (currentImgNumber < numberOfImages)
    {
        setTimeout('switchImage()',3000);
    }
}
```

要显示的广告图片有三幅。在 if 语句中，将检查当前图片的编号 currentImgNumber 是否小于 3。如果小于 3，则表示还有图片要显示，因此把另一个计时器设置为在 window 对象的 onload 事件处理程序中设置的计时器，这个计时器会在 3 秒后再次调用 switchImage()函数。

在早期的浏览器中，只能使用这个办法来创建定期触发的计时器。但在大多数现代浏览器(例如 IE6+和 Firefox)中，还可以使用下面介绍的简单方法。

10.2.2　创建定期触发的计时器

现代浏览器给 window 对象添加了设置计时器的新方法，即 setInterval()和 clearInterval()方法。这两个方法的工作方式与 setTimeout()和 clearTimeout()方法非常类似，但它们会定期触发计时器，而不是仅触发一次。

setInterval()方法的参数与 setTimeout()方法相同，但第二个参数不是计时器触发前的时间，而是计时器的触发间隔，该间隔以毫秒为单位。

例如要设置一个每隔 5 秒就触发一次 myFunction()函数的计时器，可以使用如下代码：

```
var myTimerID = setInterval("myFunction()",5000);
```

与 setTimeout()类似，setInterval()方法也返回唯一的计时器 ID，如果要清除该计时器，可以使用该 ID 调用 clearInterval()，clearInterval()与 clearTimeout()的工作方式相同。因此，要停止前面代码中启动的计时器，可以使用如下代码：

```
clearInterval(myTimerID);
```

试一试　**世界时转换器(第二部分)**

下面修改前面例子中的世界时转换器，以把本地时间和所选城市的时间显示为持续更新的时钟。

要修改 WorldTimeConverter.htm，需要在文本编辑器中打开它，在已定义的函数前，添加如下函数：

```
var daylightSavingAdjust = 0;
```

```
function window_onload()
{
   updateTimeZone();
   window.setInterval("updateTime()",1000);
}
function updateTimeZone()
{
```

接着编辑<body>标记，如下所示：

```
<body onload="return window_onload()">
```

再次保存该文件，并在浏览器中加载 WorldTimeConverter.htm 页面。该页面应与时间转换器的上一个版本相同，但时间会每秒更新一次。

这里所做的修改很少、很简单。在 window_onload()函数中添加了一个计时器，该计时器每隔 1000 毫秒(即 1 秒)调用一次 updateTime()函数，直到用户退出页面为止。前面仅当用户选中了列表框中的另一个城市，或者单击了夏令时复选框时，才会调用 updateTime()函数。

在<body>标记中，window_onload()函数连接到 window 对象的 onload 事件上，因此页面加载后，时钟就开始运行。

这就完成了本例和计时器和介绍。

10.3 小结

本章开头介绍了国际标准时间，即协调世界时(Coordinated Universal Time，UTC)。然后讨论了在网页中如何创建计时器。本章主要内容如下：

- 可以使用 Date 对象设置和获取 UTC 时间，其方式与设置或获取本地时间类似，例如 setUTCHours()方法和 getUTCHours()方法。使用相应的 UTC 类型的方法，可以设置或获取 UTC 时间的年、月、分钟或秒等。
- 在国际时间转换中，getTimezoneOffset()方法是非常有用的，它返回用户本地时间与 UTC 相差的分钟数。该方法的一个缺陷是假定了用户在计算机上正确设置了用户所在的时区。否则，getTimezoneOffset()就会失效。如果用户的时钟设置不正确，则其他日期和时间的方法也是无效的。
- setTimeout()方法可以启动计时器，该计时器仅在指定的毫秒数过后触发一次。setTimeout()方法接受两个参数，第一个是要执行的代码，第二个是代码执行前的延迟时间。该方法返回计时器的唯一 ID，如果以后要引用这个计时器，就可以使用这个 ID。例如，要在计时器触发之前将其停止，可以使用该 ID 调用 clearTimeout()方法。
- 要创建定期触发的计时器，可以使用 setInterval()方法，它的工作方式与 setTimeout()相同，只是计时器总是会定期触发，但除非用户退出了页面，或者调用了 clearInterval()方法。

第 11 章将讨论如何使用 cookie 在用户的计算机上存储信息。尽管 cookie 并不足以保存用户的所有访问历史记录，但足以跟踪用户对网站的访问和查看过的页面了。有了这些信息，就可以为用户提供更个性化的体验了。

10.4　习题

本章习题的参考答案在附录 A 中。

习题 1：

创建一个顶部带广告图片的网页。当页面加载时，给广告随机选择一幅图片。每隔 4 秒将广告更换为另一幅图片，并确保选择不同的图片，直到显示了所有广告图片为止。

习题 2：

创建一个表单，以获取用户的生日信息。然后根据用户的生日，计算出他生日那天是星期几。

第11章

使用 Cookie 存储信息

Web 站点程序员的一个目标是使用户访问网站的过程尽可能轻松愉悦。显然，精心设计的页面和易于导航的布局是该目标的核心，但这还远远不够。必须进一步了解用户，使用从用户处获得的信息，使网站实现个性化。

例如，用户第一次访问网站时已经注册了用户名，当他再次访问网站时，就可以用他的名字问候他，欢迎他再次访问网站。还有其他更好的网站实例，如 Amazon(亚马逊)网站的“一点即购(one-click purchasing)”系统。已知用户的购买信息后，如信用卡号和投递地址，就允许用户浏览书籍，然后单击即可购买，从而极大地增加了用户的购买可能性。另外，还可以根据用户以前的购买信息和浏览模式，为用户购书提供建议。

这种个性化需要在用户两次访问网站之间，将用户的信息保存起来。前面的章节提到，出于浏览器的安全性限制，从 Web 应用程序中访问用户本地文件系统是有许多限制的。但是，Web 站点的开发人员可以使用 Cookie，将少量信息保存在用户本地硬盘的指定空间中。该技术叫做 Cookie。也许有某种逻辑性的原因，但这往往闹出一些与食物有关的笑话来！

11.1 烘焙第一个 Cookie

Cookie 的关键在于 document 对象的 Cookie 属性。使用这个属性，可以在 JavaScript 代码中创建 cookie，访问 cookie 中的数据。

要设置 cookie，可以将 document.cookie 设置为 Cookie 字符串。本章随后将详细介绍 Cookie 字符串的构成，现在先创建一个简单的 Cookie 例子，看看信息保存在用户计算机的什么地方。

11.1.1 新鲜出炉的 Cookie

下面的代码设置了一个 Cookie，其 UserName 设置为 Paul，到期时间为 2020 年 12 月 28 日。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script language="JavaScript" type="text/javascript">
   document.cookie = 'UserName=Paul;expires=Tue, 28 Dec 2020 00:00:00;';
</script>
</head>
<body>
<p>This page just created a cookie</p>
</body>

</html>
```

将该页面保存为 FreshBakedCookie.htm。在学习 Cookie 字符串时会讨论上面代码的工作方式，现在先讨论创建 Cookie 时会发生什么。

不使用代码时，查看 Cookie 的方式随所使用的浏览器而异。下面介绍如何在 IE 和 Firefox 中查看 Cookie。

1. 在 IE 中查看 cookie

本节介绍如何查看 IE 保存在计算机中的 cookie。之后，加载用上述代码创建的 cookie 创建页面，看看效果如何。

(1) 首先打开 IE。本章的例子使用 IE 8，如果使用的是 IE 的早期版本，则屏幕图和菜单将略有差异。

(2) 在查看 Cookie 前，先清除浏览器的 Internet 临时文件夹，以便查看浏览器保存的 Cookie。选择 IE 中的 Tools | Internet Options，如图 11-1 所示。

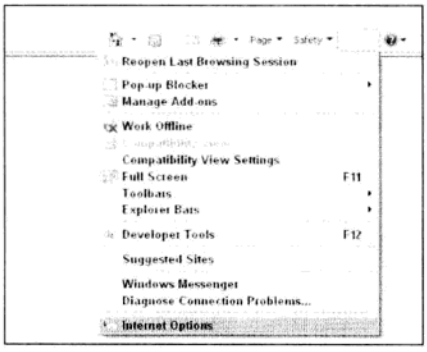

图　11-1

选择了这个选项后，将打开 Internet Options 对话框，如图 11-2 所示。

(3) 单击 Browsing history 下的 Delete 按钮，打开另一个对话框，如图 11-3 所示。

(4) 确保仅选中 Temporary Internet Files 旁边的复选框，再单击 Delete 按钮，现在就有了一个非常干净的缓存，以便于查看新创建的 cookie。

(5) 关闭该对话框，返回到 Internet Options 主对话框。

现在看看当前保存在计算机上的 Cookie。

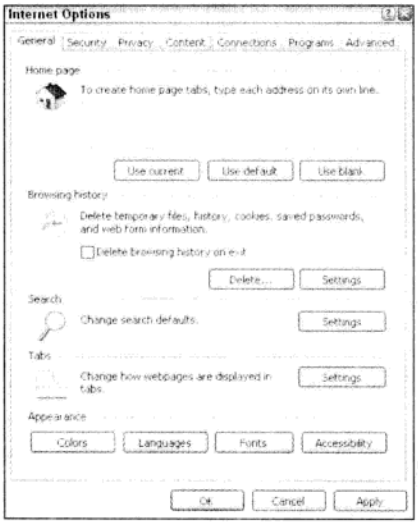

图 11-2

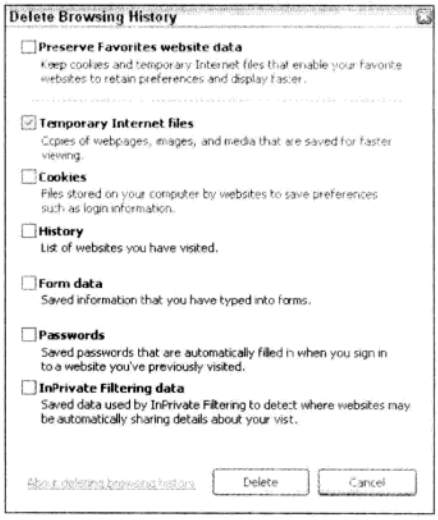

图 11-3

(6) 在 Internet Options 对话框中，单击 Browsing history 分组中 Delete 按钮旁边的 Settings 按钮，将打开如图 11-4 所示的对话框。

(7) 单击 View files 按钮，就会列出计算机上所有的临时页面和 Cookie 文件。如果按前面的步骤删除了所有的 Internet 临时文件，就不显示临时页面，如图 11-5 所示。

实际的 Cookie、其名称和值将因计算机操作系统的而异。

双击 Cookie 文件就可以查看其中的内容。注意系统将警告打开文本文件可能存在潜在的安全风险，但打开 Cookie 文件是非常安全的，因为它们是简单的文本文件。图 11-6 中显示了由 Google 搜索引擎设置的名为 google 的 Cookie 文件所包含的内容。

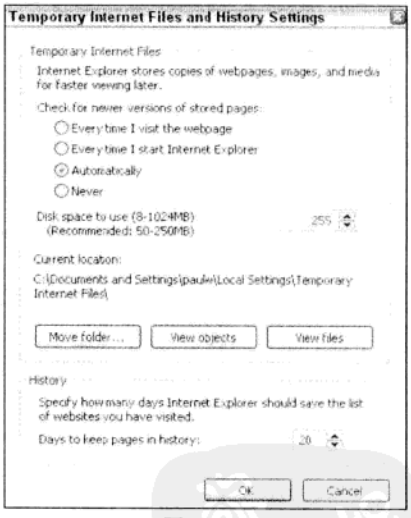

图 11-4

可以看出，Cookie 只是一个普通的文本文件。每个网站或域名都有一个自己的文本文件，以保存该站点的所有 Cookie。本例仅给 google.co.uk 保存了一个 Cookie。类似 amazon.com 这样的域肯定设置了许多 Cookie。

在图 11-6 中，可以看到 Cookie 的详细信息。例如，该 Cookie 的名称为 PREF，它的值为一个字符序列，但该字符序列很难理解，只对 Google 网站有意义。它由域 google.co.uk 设置，且与根目录/相关。Cookie 的内容看起来是一些混乱的字符，但是无须担心，在学习如何编写 Cookie 时，会发现无需采用这种格式设置 Cookie 的信息。

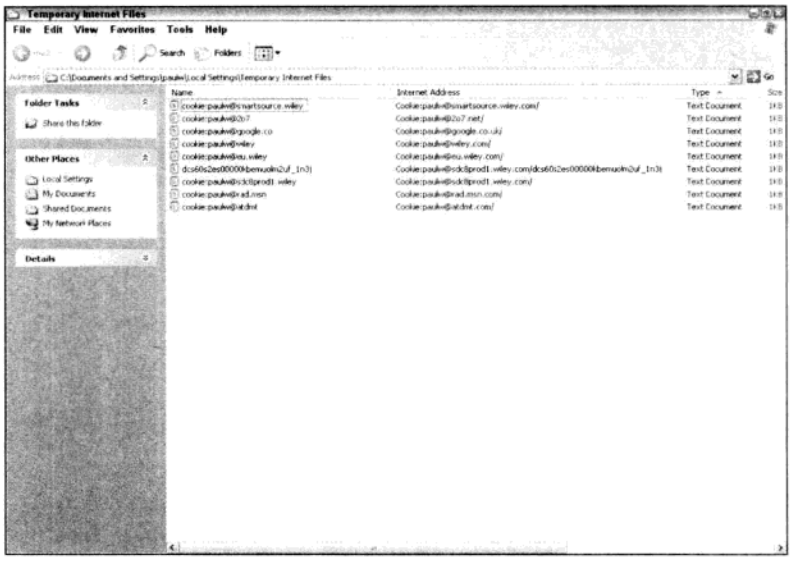

图　11-5

图　11-6

之后，关闭 Cookie，并单击对话框的 OK，返回浏览器。

现在，在 IE 浏览器中加载 FreshBakedCookie.htm 页面。该页面将设置一个 Cookie。下面看看这有什么变化：

(1) 选择 Tools | Internet Options，返回 Internet Options 对话框。

(2) 单击 Settings 按钮。

(3) 单击 View Files，计算机将显示如图 11-7 所示的信息。

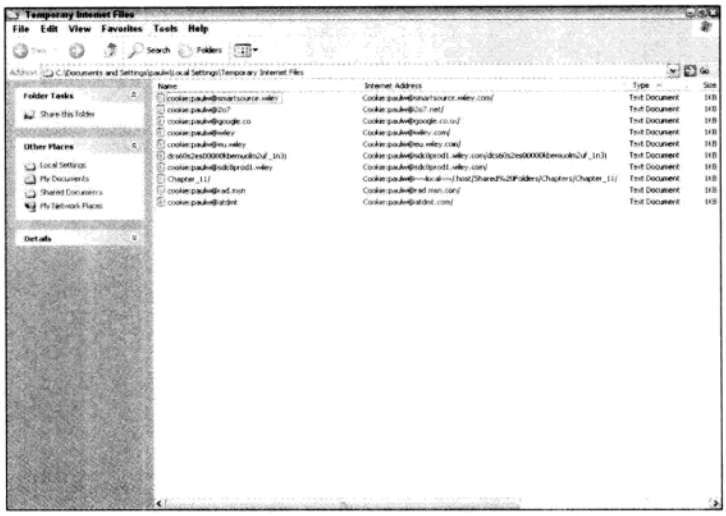

图 11-7

由于 Cookie 在本地硬盘的一个网页上创建，而不是在服务器上创建，因此其域名设置为网页所在的目录名。显然，这仅仅是出于演示的目的。实际上，用户将通过 Internet 访问网站中的页面，而不是从本地硬盘上加载页面。Internet 地址将基于 FreshBakedCookie.htm 文件所在的目录。另外，该 Cookie 在 2020 年 12 月 28 日过期，这是创建 Cookie 时设置的。双击该 Cookie，查看内容，其内容将如图 11-8 所示。

图 11-8

在图 11-8 的左上角可以看到该 Cookie 的名称 UserName、值 Paul、目录信息，还有过期时间，但其形式不易辨认。注意，有时在查看 Cookie 文件之前，需要先关闭浏览器，再重新打开。

2. 在 Firefox 中查看 Cookie

Cookie 不在浏览器之间共享。因此，在使用 IE 访问某个网站时保存的 Cookie 不能用于 Firefox，反之亦然。

Firefox 保存 Cookie 的地方与 IE 完全不同，查看 Cookie 内容的方法也与 IE 不同。要查看 Firefox 中的 Cookie，需要执行以下步骤：

(1) 选择 Tools | Internet Options.

(2) 选择 Privacy 选项.

(3) 单击 Show Cookies 按钮，打开如图 11-9 所示的对话框。

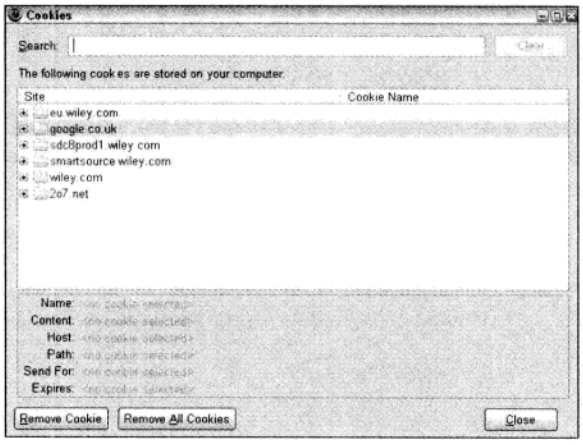

图　11-9

(4) 单击 Close 返回浏览器，并加载 FreshBakedCookie.htm。

(5) 重复上面的步骤，打开 Cookie 管理器，会发现名为 UserName 的 Cookie 已添加进来。由于该 Cookie 是从 PC 的文件中加载的，而不是通过 Internet 访问的，因此它的 Web 地址为空。展开的 Cookie 信息如图 11-10 所示。

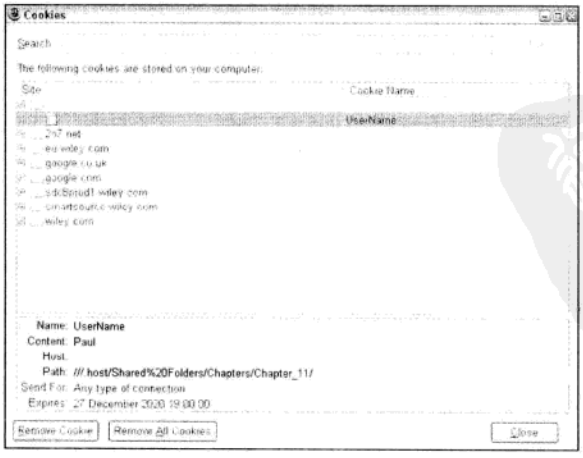

图　11-10

注意，Cookie 管理器底部的按钮可以删除选中的 Cookie 或保存的所有 Cookie。

知道如何手工查看 Cookie 之后，下面分析如何使用代码创建和读取 Cookie。首先分析构成 Cookie 字符串的各个部分。

11.1.2 Cookie 字符串

创建 Cookie 时，可以设置 6 个部分：name(名称)、value(值)、expires(过期时间)、path(路径)、domain(域)和 secure(安全性)，后 4 个是可选的。下面逐一介绍它们。

1. 名称和值

Cookie 字符串的第一部分包含 Cookie 的名称和值。名称用来引用 Cookie，值是 Cookie 的信息部分。

Cookie 字符串的这个名称/值对是必选的。如果不存储名称或值，则 Cookie 是无效的，因为保存信息是使用 Cookie 的目的。应确保这个部分在 Cookie 字符串的开头。

Cookie 的值是一个基本字符串，但如果要保存的是数值数据，该字符串也可以保存数字字符。如果保存文本，则某些字符(如分号)，在 Cookie 字符串中用作不同部分的分隔符)就不能存储在该值中，除非使用了特殊的编码，稍后介绍。

下面的代码设置了一个 Cookie，其名称为 UserName，值为 Paul：

```
document.cookie = "UserName=Paul;";
```

这个 Cookie 的生存期很短，生存期是 Cookie 中的信息持续存在的时间。如果没有设置过期时间，Cookie 就在用户关闭浏览器后过期。下次用户打开浏览器时，该 Cookie 便不复存在了。如果信息只需在用户会话(用户单次访问网站的持续时间)期间保存，这就很合适。但是，如果要确保 Cookie 能用于更长的时间，就必须设置其过期时间，参见下面的内容。

2. 过期时间

如果希望 Cookie 存在的时间比单个用户会话更长，则需要使用 Cookie 字符串的第二部分 expires 设置过期时间，如下所示：

```
document.cookie = "UserName=Paul;expires=Tue, 28 Dec 2020 00:00:00 GMT; ";
```

上述代码设置的 Cookie 在 2020 年 12 月 28 日之前是可用的。

注意：

过期时间的格式是非常重要的，对于 IE 浏览器尤其重要。它应与 toGMTString()方法给出的 cookie 的格式相同。这个方法类似于第 10 章介绍的 toUTCString()方法。

实际上，可以使用 Date 对象获取当前日期，再将 Cookie 设置为该日期后的 3 或 6 个月过期。否则，就需要在 2020 年 12 月 28 日重写页面。

例如，可以编写下面的代码：

```
var expireDate = new Date();
expireDate.setMonth(expireDate.getMonth() + 6);
document.cookie = "UserName=Paul;expires="+ expireDate.toGMTString() + ";";
```

上面的代码创建了一个新 Cookie，其名称为 UserName，值为 Paul，在当前日期后的 6 个月过期。注意，其他因素可能导致cookie在过期日期之前就失效，例如用户删除了 Cookie，或者达到 Cookie 数量的上限。

3. 路径

在 99%的情况下，都仅需要设置 Cookie 的 name、value 和 expires 部分。但是，有时需要设置其余三个部分，如本节介绍的 path 部分。最后两个部分 domain 和 secure 用于高级用户，超出了本书的范围，但是为了完整起见，将简要介绍它们。

我们习惯使用硬盘上的目录。我们将硬盘划分为这些目录，而不是将所有东西放在一起。例如，可以将字处理文件放在 My Documents 目录中，把图片文件放在 My Images 目录中，等等。目录还可以细分为子目录，所以在 My Images 目录下可能有 My Family 和 My Holiday 子目录。

Web 服务器也采用与此相同的原理。Web 服务器常常将整个网站合理地划分为不同的目录，而不是放在一个 Web 目录中。例如，如果访问 Wrox 网站 www.wrox.com，然后单击一个书籍分类，则所浏览的页面的路径是 www.wrox.com/Books/。

这很有意思，但这与 Cookie 有什么关系？

问题在于，Cookie 不仅特定于某个 Web 域，如 www.wrox.com，还特定于该域中的具体路径。例如，如果 www.wrox.com/Books/中的页面设置了一个 Cookie，那么只有该目录或其子目录中的页面才能读取和修改这个 Cookie。如果 www.wrox.com/academic/下的页面试图读取这个 Cookie，则会失败。为什么要对 Cookie 进行这样的限制？

下面以免费 Web 空间为例。Web 上的很多公司允许申请免费的 Web 空间。通常，每个申请 Web 空间的人的站点都在同一个域中。例如，Bob 的网站是 www.freespace.com/members/bob/，Belinda 的网站是 www.freespace.com/members/belinda。如果无论 cookie 的路径是什么，都可以检索和修改 Cookie，那么由 Bob 网站设置的任何 Cookie 都可以由 Belinda 查看，反之亦然。显然，任何用户都不希望如此。这不仅是一个安全问题，而且如果用户并不相互了解，这两个站点都有一个名为 MyHotCookie 的 Cookie，那么它们设置或读取同一个 Cookie 时，就会产生问题。免费 Web 空间提供者常常有许多用户，所以可能造成混乱。

既然 Cookie 是特定于某一路径的，如果想通过服务器上的两个不同的路径访问某一 Cookie，该怎么办？例如 www.mywebsite.com/mystore/有一个联机商店，该商店还细分为两个子目录/Books 和/Games。假如结账页面位于 www.mywebsite.com/mystore/Checkout 目录中，那么/Books 和/Games 目录中设置的 Cookie 无法互访，也不能由/Checkout 目录中的页面访问。要解决这个问题，可以只在/mystore 目录中设置 Cookie，这样 Cookie 就能被/mystore 目录及其所有的子目录读取。或者，可以使用 Cookie 字符串的 path 属性，指定 Cookie 的路径是/mystore，无论 Cookie 是在/Books、Games 还是在/Checkout 子目录中设置，都是如此。

例如，使用下面的代码：

```
document.cookie = "UserName=Paul;expires=Tue, 28 Dec 2020 00:00:00"+
";path=/mystore;";
```

即使 Cookie 由/Books 目录中的页面设置，仍能由/mystore 目录及其子目录中的页面访问，如/Checkout 和/Games。

如果要指定 Cookie 可用于设置它的域的所有子目录，则可以使用/字符指定根目录的路径：

```
document.cookie = "UserName=Paul;expires=Tue, 28 Dec 2020 00:00:00;path=/;";
```

现在，该 cookie 可供设置它的域上的所有目录访问。如果该域包含许多站点，则最好不要这样设置，否则其中的网站就可以访问其他网站的 Cookie 信息。

注意，尽管 Windows 计算机不区分目录名的大小写，但是其他操作系统会区分。例如，如果网站在基于 Unix 或 Linux 的服务器上，则 path 属性是区分大小写的。

4. 域

Cookie 字符串的第 4 部分是域，域的例子有 wrox.com、pawilton.com 等。与 cookie 字符串的路径部分类似，域部分的设置也是可选的，而且域似乎并不常用。

默认情况下，Cookie 仅用于设置它的域上的页面。例如，假设第一个网站运行在域为 MyPersonalWebSite.MyDomain.Com 的服务器上，第二个网站运行在 MyBusinessWeb Site. MyDomain.Com 上，则一个网站上设置的 Cookie 就不能由另一个域名下的页面访问，反之亦然。在大多数情况下，这正是我们所需要的。但如果不是这样，就可以使用 Cookie 字符串的 domain 部分指定，Cookie 可用于指定域的所有子目录。例如，下述代码设置的 Cookie 可由两个子域共享：

```
document.cookie = "UserName=Paul;expires=Tue, 28 Dec 2020 00:00:00;path=/"+
";domain=MyDomain.Com;";
```

注意，域必须相同：不能共享 www.MyDomain.Com 与 www.SomeoneElsesDomain.com。

5. 安全

Cookie 字符串的最后一部分是 secure。这只是一个布尔值，如果它设置为 true，则 Cookie 仅发送给尝试使用安全通道检索它的 Web 服务器。默认值 false 表示，总是发送 Cookie，而不考虑安全性。这仅应用于用 SSL(Secure Sockets Layer，安全套接字层)建立服务器的情况。

11.2 创建 Cookie

为简单起见，下面编写一个函数，以便更加轻松地创建新的 Cookie 并设置某些属性。这是第一个有效的函数，后面将创建许多这样有用的函数，并添加到一个独立的.js 文件中，

以便更容易在将来的项目中重用其中的代码。下面先看看函数的代码，再使用它创建一个示例。首先创建一个文件 CookieFunctions.js，添加如下代码：

```
function setCookie(cookieName, cookieValue, cookiePath, cookieExpires)
{
  cookieValue = escape(cookieValue);
  if (cookieExpires == "")
  {
     var nowDate = new Date();
     nowDate.setMonth(nowDate.getMonth() + 6);
     cookieExpires = nowDate.toGMTString();
  }
  if (cookiePath != "")
  {
     cookiePath = ";Path="+ cookiePath;
  }
  document.cookie = cookieName + "="+ cookieValue +
     ";expires="+ cookieExpires + cookiePath;
}
```

Cookie 字符串的 domain 和 secure 部分很少用到，因此该函数仅设置 Cookie 的 name、value、expires 和 path 部分。如果不需要设置路径或过期日期，只需给这两个参数传送空字符串。如果未指定路径，则使用当前目录及其子目录。如果未设置过期日期，则假定过期时间为自创建之日起 6 个月。

函数的第一行引入了一个新的函数 escape()。

```
cookieValue = escape(cookieValue);
```

前面设置 cookie 的值时，提到过不能直接使用某些字符，如分号(这也适用于 Cookie 的名称)。要解决这个问题，可以使用内置的 escape()函数和 unescape()函数。escape()函数将非字母或数字字符转换为 Latin-1 字符集中对应的十六进制编码，并加上%字符前缀。

例如，空格的十六进制编码为 20，分号的十六进制编码为 3B。因此，下述代码的输出结果如图 11-11 所示。

图　11-11

```
alert(escape("2001 a space odyssey;"));
```

可以看到，每个空格都转换为%20，其中的%指出这是一个转义或特殊字符，而不是实际的字符，20 表示该字符的 ASCII 值。分号转换为%3B。

如后面所述，读取 Cookie 的值时，可以使用 unescape()函数将转义的字符转换为普通文本。

回到 setCookie()函数上，接下来是一个 if 语句。

```
if (cookieExpires == "")
{
   var nowDate = new Date();
   nowDate.setMonth(nowDate.getMonth() + 6);
   cookieExpires = nowDate.toGMTString();
}
```

这些代码用于处理传递给函数的 cookieExpires 参数值为空字符串("")的情形。在很多情况下，Cookie 的存在时间都应长于创建它的会话，因此 expires 的默认值设置为当前日期后的 6 个月。

接下来，如果传递给 cookiePath 参数的值不是空字符串""，则需要在创建 Cookie 时添加这个值。只需在 cookiePath 参数值前添加"path="。

```
if (cookiePath != "")
{
   cookiePath = ";Path="+ cookiePath;
}
```

最后一行代码将 cookieName、cookieValue、cookieExpires 和 cookiePath 部分连接起来，创建 Cookie。

```
document.cookie = cookieName + "=" + cookieValue +
   ";expires=" + cookieExpires + cookiePath;
```

每当需要创建新的 Cookie 时，就可以使用 setCookie()函数。因为它更容易设置 cookie，而无须记住要设置的各个部分。更重要的是，它还可以将过期时间默认设置为当前日期后的 6 个月。

例如，要使用该函数设置一个 expires 和 path 为默认值的 Cookie，只需使用如下代码：

```
setCookie("cookieName","cookieValue","","")
```

试一试 使用 setCookie()函数

下面的简单例子使用 setCookie()函数设置三个名为 Name、Age 和 FirstVisitcookie 的 Cookie。然后显示 document.cookie 属性的内容，看看其效果。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script language="JavaScript" type="text/JavaScript">
function setCookie (cookieName, cookieValue, cookiePath, cookieExpires)
{
  cookieValue = escape(cookieValue);
   if (cookieExpires == "")
   {
      var nowDate = new Date();
```

```
        nowDate.setMonth(nowDate.getMonth() + 6);
        cookieExpires = nowDate.toGMTString();
    }
    if (cookiePath != "")
    {
        cookiePath = ";Path=" + cookiePath;
    }
    document.cookie = cookieName + "=" + cookieValue +
        ";expires=" + cookieExpires + cookiePath;
}
setCookie("Name","Bob","","");
setCookie("Age","101","","");
setCookie("FirstVisit","10 May 2007","","");
alert(document.cookie);
</script>
</head>
<body>
</body>
</html>
```

保存为 CreateCookie.htm，并在浏览器中加载。

所显示的警告框如图 11-12 所示。注意，三个 Cookie 都显示为名称/值对，并用分号来分隔，但是没有显示过期日期。路径参数即使设置了，也不会显示出来。还显示了前面示例中的名为 UserName 的 Cookie。

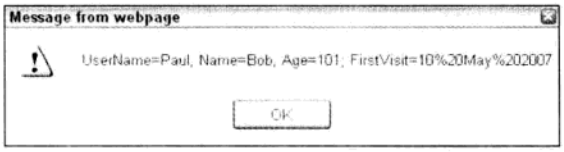

图　11-12

前面介绍了 setCookie()函数的工作原理，因此只需分析使用函数创建 3 个新 cookie 的三行代码。

```
setCookie("Name","Bob","","");
setCookie("Age","101","","");
setCookie("FirstVisit","10 May 2007","","");
```

这非常简单。第一个参数是 Cookie 的名字(稍后介绍如何根据 Cookie 的名字来获取其值)。注意，Cookie 的名字应只使用字母和数字字符，没有空格、标点符号或者特殊字符。尽管 Cookie 的名字可以使用这些字符，但使用它们会比较复杂，最好避免使用。第二个参数是 Cookie 的值。第三个参数是路径，第四个参数是 Cookie 的过期时间。

例如，在使用 setCookie()函数的第一行代码中，所设置的 Cookie 的名字是 Name，其值是 Bob。我们没有设置 path 和 expires 部分，因此将它们设置为空字符串("")。注意，必须传入空字符串，不能省略它们。

余下的两行代码分别创建了名为 Age 和 FirstVisit 的 Cookie，其值分别设置为 101 和 10 May 2007。

如果要设置路径和过期时间，该如何修改代码？

假如希望路径是/MyStore，过期时间为 1 年后，则可使用下面的 setCookie()函数：

```
var expireDate = new Date();
expireDate.setMonth(expireDate.getMonth() + 12);
setCookie("Name", "Bob","/MyStore",expireDate.toGMTString());
```

首先创建一个新的 Date 对象，但没有给构造函数传送参数，将该对象初始化为当前日期。下一行代码给该日期加上 12 个月。接下来，使用 setCookie()函数设置 Cookie 时，将"/MyStore"传送为路径，expireDate.toGMTString()传送为过期时间。

创建了一个名为 Name，值为 Bob 的 Cookie 后，如何修改该 Cookie 的值？要此，只需再次设置 cookie，但使用新值。例如，要将名为 Name 的 Cookie 的值从 Bob 改为 Bobby，可以使用如下代码：

```
setCookie("Name","Bobby","","");
```

如何删除已有的 Cookie？很简单，只需修改 Cookie 的值，并将其过期时间设置为过去的时间，使该 Cookie 过期即可。例如下面的代码：

```
setCookie("Name","","","Mon, 1 Jan 1990 00:00:00");
```

11.3 获取 Cookie 的值

前面的例子使用了 document.cookie 属性来获取一个字符串，其中包含已设置的 cookie 的信息。但这个字符串有两个局限性。

- 用名称/值对检索 cookie，并用分号作为分隔符。无法获得 Cookie 的 expires、path、domain 和 secure 部分，也不能检索它们。
- cookie 属性仅允许获取为某一路径和 Web 服务器(它们驻留在 Web 服务器上时)设置的所有 Cookie。例如，无法直接获取名为 Age 的 Cookie 的值。为此，必须使用前面章节学过的字符串处理技术，从返回的字符串中截取需要的信息。

可以通过多种方法获取单个 Cookie 的值，下面使用的方法适用于所有支持 Cookie 的浏览器。需要将下面的函数添加到 CookieFunctions.js 文件中：

```
function getCookieValue(cookieName)
{
   var cookieValue = document.cookie;
   var cookieStartsAt = cookieValue.indexOf(" " + cookieName + "=");
   if (cookieStartsAt == -1)
   {
      cookieStartsAt = cookieValue.indexOf(cookieName + "=");
   }
   if (cookieStartsAt == -1)
   {
      cookieValue = null;
```

```
    }
    else
    {
       cookieStartsAt = cookieValue.indexOf("=", cookieStartsAt) + 1;
       var cookieEndsAt = cookieValue.indexOf(";", cookieStartsAt);
       if (cookieEndsAt == -1)
       {
          cookieEndsAt = cookieValue.length;
       }
       cookieValue = unescape(cookieValue.substring(cookieStartsAt,
          cookieEndsAt));
    }
    return cookieValue;
}
```

该函数的第一个任务是获取 document.cookie 字符串，并保存在变量 cookieValue 中。

```
var cookieValue = document.cookie;
```

接下来，在 cookieValue 字符串中查找其名称作为参数传入的 Cookie 的位置。这里使用 String 对象的 indexOf()方法查找此信息，代码如下所示：

```
var cookieStartsAt = cookieValue.indexOf(" " + cookieName + "=");
```

该方法返回找到的 Cookie 的字符位置，如果未找到 cookie，则返回－1，表示不存在这个 cookie。我们查找""+ cookieName + "="，以免在无意间查找包含所需名称的 Cookie 名称或值。例如，如果 Cookie 的名称是 xFoo、Foo 和 yFoo，则查找前面不包含空格的 Foo，会先匹配 xFoo，但这并不是我们需要的 Cookie。

如果 cookieStartsAt 是－1，则表示该 cookie 不存在，或者该 Cookie 位于 Cookie 字符串的开头，这时该 Cookie 名称前也不包含空格。为确定是哪一种情况，需要再进行一次查找，这次的查找不包含空格：

```
if (cookieStartsAt == -1)
{
   cookieStartsAt = cookieValue.indexOf(cookieName + "=");
}
```

在接下来的 if 语句中，检查是否找到 Cookie，如果没有找到，则将 cookieValue 变量设置为 null。

```
if (cookieStartsAt == -1)
{
   cookieValue = null;
}
```

如果找到 Cookie，则在 else 语句中，从 document.cookie 字符串中截取该 cookie 的值。为此，需要找到该 Cookie 的值部分的开头和结尾。其开头是 Cookie 名称及等号(=)后的下一个字符位置。因此，在下面的代码中，indexOf()方法从 Cookie 名称/值对的起始字符开始搜索，查找字符串中跟在 Cookie 名称后面的等号。

```
else
{
   cookieStartsAt = cookieValue.indexOf("=", cookieStartsAt) + 1;
```

然后，给这个值加 1，以获取该 Cookie 值的开始位置。

该 Cookie 值的结束位置要么到下一个分号，要么到字符串的结尾。下面的代码从 cookieStartsAt 索引开始搜索分号。

```
var cookieEndsAt = cookieValue.indexOf(";", cookieStartsAt);
```

如果这是字符串中的最后一个 cookie，就搜索不到分号，变量 cookieEndsAt 将为 - 1。此时，该 Cookie 值的结尾一定是字符串的结尾，因此可将 cookieEndsAt 变量设置为字符串的长度。

```
if (cookieEndsAt == -1)
{
   cookieEndsAt = cookieValue.length;
}
```

接着使用 substring()方法从主字符串中截取 cookie 的值。由于使用 escape()函数编码了字符串，因此需要使用 unescape()函数进行解码，以获得实际的值。

```
   cookieValue = unescape(cookieValue.substring(cookieStartsAt,
      cookieEndsAt));
}
```

最后将该 Cookie 的值返回给调用函数。

```
return cookieValue;
```

试一试　有新内容吗？

知道如何创建和获取 Cookie 后，就在一个示例中使用这些知识，检查自用户上次访问网站之后，网站是否有变化。

该例子将创建两个页面。第一个是该网站的主页面，第二个页面包含了网站更新和新增内容。只有以前用户访问过第二个页面(即存在一个 Cookie)，但自从该页面更新以来用户还没有访问过，才在第一个页面上显示对第二个页面的链接。

下面创建第一个页面。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Cookie Example</title>
<script language="JavaScript" type="text/JavaScript"
src="CookieFunctions.js"></script>
<script language="JavaScript" type="text/javascript">
var lastUpdated = new Date("Tue, 28 Dec 2020");
```

```
</script>
</head>
<body>
<h1 align=center>Welcome to my website</h1>

<div align="center" id="WhatsNewDiv"></div>
<script>
var lastVisit = getCookieValue("LastVisit");
if (lastVisit != null)
{
    lastVisit = new Date(lastVisit);
    if (lastVisit < lastUpdated)
    {
                document.getElementById('WhatsNewDiv').innerHTML = '<a
href="WhatsNew.htm"><img src="WhatsNew.jpg" border=0></a>'
    }
}
var nowDate = new Date();
setCookie("LastVisit", nowDate.toGMTString(),"","")
</script>

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

这个页面保存为 MainPage.htm。注意它包含前面创建的两个函数 setCookie()和 getCookieValue()。另外，该页面引用了一个图片 WhatsNew.jpg，可以自己创建这个图片，或者从本书的代码下载中获得它。

接下来，创建一个简单的页面，作为 What's New 链接指向的页面。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Untitled Document</title>
</head>
<body>
<h2 align=center>Here's what's new on this website</h2>
</body>
</html>
```

将该页面保存为 WhatsNew.htm。

在浏览器中加载 MainPage.htm。第一次访问主页时，页面上仅有一个标题 Welcome to my website。显然，实际的网站会有更多的内容，但是就本例来说，这已经足够了。但是，刷新该页面后会看到如图 11-13 所示的页面。

图　11-13

如果单击图片，则会进入 WhatsNew.htm 页面，其中包含了自上次访问站点后，添加到网站的所有内容。显然，在加载和刷新该页面之间，站点内容并没有什么改变。为了进行测试，将变量 lastUpdated 中站点上次更新的日期设置为将来的某个日期(即 2020 年 12 月 28 日)。

WhatsNew.htm 页面只是一个没有脚本的简单 HTML 页面。因此只需看看 MainPage.htm 页面。在该页面顶部的第一个脚本块中，声明了一个变量 lastUpdated。

```
var lastUpdated = new Date("Tue, 28 Dec 2020");
```

只要站点发生了改变，这个变量就需要更新。目前将它设置为 2020 年 12 月 28 日，只是为了确保在刷新页面时，显示 What's New 图片。实际页面的另一个更好的办法是使用 document.lastModified 属性，它返回上次修改页面的日期。

第一个脚本块的其余部分包含了前面介绍的两个函数 getCookieValue()和 setCookie()，这两个函数没有变化，因此这里不再详细讨论。

最有意义的是页面正文中的第二个脚本块。首先，使用 getCookieValue()函数从名为 LastVisit 的 Cookie 中获取上次用户访问站点的日期。

```
var lastVisit = getCookieValue("LastVisit");
```

如果 lastVisit 是 null，则表示用户从未访问过站点，或者自上一次访问该站点以来，已超过了 6 个月，Cookie 已经过期。对于这两种情况，都不显示 What's New 图片。因为用户是第一次访问站点，或者最近 6 个月，站点的变化太大，不能用 What's New 页面来显示。

如果 lastVisit 不是 null，就需要检查自站点上次更新以来，该用户是否访问过网站。如果未访问过，则把该用户导航到显示新内容的页面上，如下面的 if 语句所示。

```
if (lastVisit != null)
{
   lastVisit = new Date(lastVisit);
   if (lastVisit < lastUpdated)
   {
               document.getElementById('WhatsNewDiv').innerHTML = '<a
href="WhatsNew.htm">' +
'<img src="WhatsNew.jpg" border=0></a>'
    }
}
```

首先根据 lastVisit 的值创建一个新的 Date 对象，并保存回 lastVisit 变量。接下来，在内层 if 语句的条件中，比较用户上次访问网站的日期与网站上次更新的日期。如果自上次用户访问网站以来，站点已经发生了改变，则将 What's New 图片输出到页面上，这样用户便可单击它，查看更新的信息。

最后，在脚本块的尾部，使用 setCookie()函数将名为 LastVisit 的 Cookie 重置为本次访问网站的日期和时间。

```
var nowDate = new Date();
```

```
setCookie("LastVisit", nowDate.toGMTString(),"","")
```

11.4　Cookie 的局限性

使用 Cookie 时，应注意以下一些局限性。

11.4.1　用户可能禁用 Cookie

第一个局限性是，尽管所有的现代浏览器都支持 cookie，但用户可能禁用了它们。在 Firefox 中，可以选择 Options 子菜单，再选择 Privacy 选项卡和 cookies 选项卡，来禁用 Cookie。在 IE 中，可选择 Tools 菜单下的 Internet Options，选择 Privacy 选项卡，然后拖动滚动条，来设置使用 Cookie 的级别。大多数用户都默认启用会话 Cookie，会话 Cookie 是仅在用户浏览网站期间存在的 Cookie，用户关闭浏览器后，该 Cookie 就被清除。通常还启用了更持久的 Cookie。但是，第三方 Cookie(来自第三方站点)通常是禁用的，它们用于跟踪用户对不同网站的访问，因此人们对这种 Cookie 侵犯隐私的关注比较多。

禁用 Cookie 时，前面用于创建 Cookie 和获取 Cookie 值的函数不会出错，但是所获取 Cookie 的值将为 null，因此需要确保代码能够处理这种情况。

可以在禁用 Cookie 时设置一个默认动作。例如在前面的例子中，如果禁用了 Cookie，就不显示 What's New 图片。

另外，也可以在页面上放置一条消息，告诉用户，该网站需要 Cookie 才能运行。

另一个策略是主动检查是否启用了 Cookie。如果未启用，则采取相应的应对措施，例如将用户重定向到一个不需要使用 Cookie 的简化页面。那么，如何检查是否启用了 cookie 呢？

下面的脚本设置了一个测试 Cookie，再获取它的值。如果其值为 null，则说明禁用了 Cookie。

```
setCookie("TestCookie","Yes","","");
if (getCookieValue("TestCookie") == null)
   {
      alert("This website requires cookies to function");
   }
```

11.4.2　数字和信息的限制

第二个局限性是，能够在用户计算机上为某个网站设置的 Cookie 的数量是有限的，每个 Cookie 能够保存的信息也是有限的。在早期的浏览器中，最多只能为每个域保存 20 个 Cookie，每个 Cookie 的名称/值对不得超过 4096 字节。另一个要点是，所有的浏览器都设置了能保存的 Cookie 总数上限。到达这个限制时，无论是否过期，都会删除一些旧的 Cookie。现代浏览器(例如 IE7+和 Firefox)支持的 Cookie 总数为 50 个，但这个数字在不同的浏览器上不尽相同。

为了突破 Cookie 总数的限制，可以在一个 cookie 中保存多条信息。下面的例子使用了多个 Cookie。

```
setCookie("Name","Karen","","")
setCookie("Age","44","","")
setCookie("LastVisit","10 Jan 2001","","")
```

可以将这些信息合并到一个 Cookie 中，各条信息用分号分隔开。

```
setCookie("UserDetails","Karen;44;10 Jan 2001","","")
```

由于 setCookie()函数转义了 Cookie 的值，因此 Cookie 值中分隔数据的分号不会与分隔 Cookie 中各部分的分号混淆。使用 getCookieValue()函数取回该 Cookie 的值时，只需将它拆分为各组成部分，但是，需要记住各个部分的存储顺序。

```
var cookieValues = getCookieValue("UserDetails");
cookieValues = cookieValues.split(";")
alert("Name = " + cookieValues[0]);
alert("Age = " + cookieValues[1]);
alert("Last Visit = " + cookieValues[2]);
```

这样，就可以获得三条信息，还有 19 个 Cookie 可供使用。

11.5 Cookie 的安全性和 IE 6+

根据 World Wide Web Consortium(W3C)的 P3P 建议，IE 6 给 Cookie 引入了新的安全策略。W3C 是一个 Web 标准化组织，它不仅制定 Cookie 的标准，还制定了 HTML、XML 和各种其他浏览器标准(第 13 章将介绍 W3C，其网址为 www.w3.org 和 http:/www.w3.org/P3P/，包含了大量信息，但是不容易看懂)。P3P 的基本目标是确保 Cookie 不用来收集用户浏览习惯的私人信息。在 IE 6+中，可以选择 Tools | Internet Options，再选择 Privacy 选项卡，查看与 Cookie 有关的隐私级别设置(如图 11-14 所示)。必须在高级别与低级别之间找到一个平衡点，使网站不至于无法正常工作，也不至于用户的浏览习惯和隐私数据被跟踪记录。

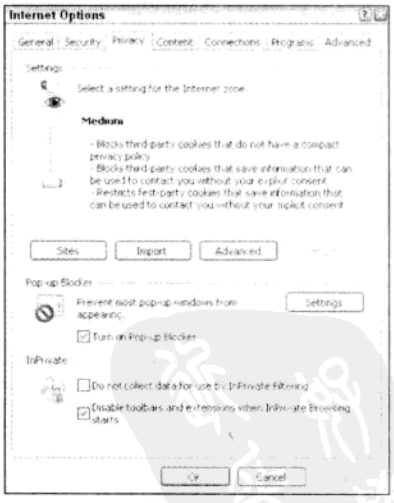

图 11-14

通常情况下，会话 Cookie——即仅在用户浏览站点期间存在的 Cookie——是默认可用的。一旦用户关闭了浏览器，会话便告结束。但是，如果我们希望 Cookie 存在的时间比用户访问网站的时间更长，就需要创建一个符合 P3P 建议的隐私策略。这似乎很复杂，策略的细节也的确很复杂。但是，IBM 开发的一个软件可以很容易创建该策略的 XML。这个软件并不便

宜，但有一个 90 天的免费试用版本，该版本可以从 www.alphaworks.ibm.com/tech/p3peditor 上下载。

还有很多易用的策略创建软件。P3PEdit 就非常便宜，可从 http://policyeditor.com/上获得。

11.6　小结

本章介绍了如何将信息保存在用户的计算机上，并利用这些信息实现网站的个性化。本章的主要内容如下：

- Cookie 的关键是 document 对象的 cookie 属性。
- 要创建 Cookie，只需设置 document.cookie 属性。Cookie 有 6 个可以设置的部分：名称、值、过期时间、可以使用的路径、可以使用的域以及是否仅通过安全连接来传送。
- 尽管设置新的 Cookie 非常简单，但是获取它的值时，实际上得到的是给当前域和路径设置的所有 Cookie，还需要使用 String 对象的方法，拆分 Cookie 的名称/值对，才能获取指定的 Cookie。
- Cookie 有很多局限性。首先，用户可以将浏览器设置为禁用 Cookie，其次，在 IE7+ 和 Firefox 上，每个域最多能保存 50 个 Cookie，每个 Cookie 名称/值对最多有 4096 个字符。

11.7　习题

本章习题的参考答案在附录 A 中。

习题 1：

创建一个页面，来跟踪用户上个月访问该页面的次数。

习题 2：

使用 Cookie，在用户每次访问页面时加载不同的广告。

第12章

动态 HTML 和 W3C 文档对象模型

JavaScript 在 Web 开发中的主要作用是与用户交互，在网页上添加某种行为，前面的章节已经介绍了这些内容，尤其在第 7、8 章给表单、框架和窗口编写脚本时做了重点描述。但用户交互操作并不仅是这些。实际上，JavaScript 允许在网页加载到浏览器中后，完全改变网页的所有方面，这个技术称为动态 HTML(DHTML)，JavaScript 在网页上提供这个功能的基础是文档对象模型(Document Object Model，DOM)，这是网页的树状表示方法。

DOM 是 W3C 提出的最被人误解的标准集合之一。W3C 的一组开发人员提出了浏览器制造商和 Web 开发人员应遵循的标准。DOM 为开发人员提供了一种表示网页中的所有元素的方式，以便在 JavaScript 中通过一组通用的属性和方法来访问。这里的“所有元素”是网页中的任意元素。通过 JavaScript 修改相应的 DOM 属性，就能修改页面中的图形、表格、表单甚至文本自身。

DOM 模型不应与第 6 章介绍的浏览器对象模型(Browser Object Model，BOM)混淆。稍后将详细介绍二者的区别。目前可以将 BOM 视为一个代表浏览器各种特性的独立于浏览器的对象，包括浏览器的按钮、URL 地址栏、标题栏、浏览器窗口控件以及网页的各部分。而 DOM 仅处理浏览器窗口或网页(即 HTML 文档)中的内容，于是，任何浏览器都可以使用完全相同的代码来访问和操作文档的内容。简而言之，BOM 可以访问浏览器以及文档的某些部分，而 DOM 可以访问文档中的任何元素，但仅限于文档本身。

DOM 的伟大之处在于它与浏览器和平台无关。这意味着开发人员终于可以使用 JavaScript 代码动态更新页面，这些代码在任何兼容 DOM 的浏览器上都能正常运行，无须做任何修改。无须为不同的浏览器编写不同的代码，也无须在编写代码时担心兼容问题。

DOM 将页面的内容表示为通用的树结构，实现了这种无关性。在 BOM 中访问某个对象时，需要查找与该浏览器部分相关的属性。而 DOM 只需在页面的树形结构中导航到与浏览器无关的节点或属性上。稍后将讨论这个结构。

但为了使用 DOM 标准，开发人员需要完全实现该标准的浏览器，但没有任何浏览器能 100%地实现该标准。更糟的是，任何一种浏览器都没有实现其他浏览器支持的所有特性，但不要被这个吓跑。所有现代浏览器都支持 DOM 标准提出的许多特性。

为了对 DOM 标准有一个真实的了解，需要简要了解一下 DOM 标准与其他现存的 Web 标准之间的关系。我们还将介绍 DOM 标准为什么会有多个版本，为什么标准也存在

一定的差异(特别是 Microsoft 给 W3C DOM 增加了大量的扩展)。了解了这些关系后，就可以使用 JavaScript 导航 DOM，在多个浏览器上动态修改网页的内容，这种方式在纯粹的 DHTML 中是无法实现的。相关内容如下：

- (X)HTML、ECMAScript 和 XML Web 标准
- DOM 标准
- 操作 DOM
- 编写跨浏览器的 DHTML

注意：

本章的示例将围绕 DOM 编写(但有几处例外)，因此只能用于 IE 8+、Firefox 1+、Opera、Safari 3+和 Chrome。

12.1　Web 标准

Tim Berners-Lee 在 1991 年创建 HTML 时，只是想利用这个技术，通过一组标签为他的全局超文本项目(称为 World Wide Web)标记科学文献。没想到，20 世纪 90 年代中期，这个技术成为两大商业软件巨头厮杀的战场。HTML 是元语言 SGML(Standard Generalized Markup Language，标准的通用标记语言)的一种简单的派生形式。SGML 在学术机构应用了数十年，其目的是把文档和创建文档的结构一起保存起来。HTML 依赖于超文本传输协议(HyperText Transfer Protocol，HTTP)，在资源和查看者之间来回传递文档(例如，在服务器和客户端计算机之间)。这两种技术构成了 Web 的基础，到了 20 世纪 90 年代初期，人们很快发现，需要建立某种规范，以确保 HTML 和 HTTP 有通用的实现方式，这样才能在世界范围内进行通信。

在 1994 年，Tim 创立了万维网联盟(World Wide Web Consortium，W3C)，该组织致力于 Web 技术演变的管理。它有 3 个主要目标：

- 提供通用的访问技术，让任何人都能使用 Web
- 开发相应的软件环境，允许用户利用 Web
- 指导 Web 的发展，考虑由 Web 引发的法律、社会和经济问题

Web 技术规范的每个新版本成为标准之前，必须经过 W3C 严格的审核。HTML 和 HTTP 规范都接受了这个审核，对这些规范的每一组更新都会产生标准的一个新版本。在成为完全可操作的标准之前，每个标准都必须经过草案、备选建议、推荐标准等阶段。在该过程的每个阶段，W3C 联盟的成员都将投票表决哪些方面需要改进，甚至决定是否完全取消该标准，从头再来。

这种创建标准格式的方法似乎非常痛苦、费力，并不像人们想象的是做尖端技术革命的先锋。实际上，20 世纪 90 年代中期，很多软件公司感到这一过程太过漫长，因此决定自己实现一些革新，再提交给标准化组织进行审批。Netscape 率先在其浏览器中引入了新的元素，如<font/>元素，以增强网页内容的表现力。这些元素大受欢迎，因此 Netscape 添加了大量元素，允许用户修改网页的显示外观和样式。实际上，JavaScript 本身就是 Netscape 的一个革新。

Microsoft 加入这场战争时，通过 IE 的最初两个版本来追赶 Netscape。但是，当 1996 年发布 Internet Explorer 3 时，为了与 Netscape 抗衡，Microsoft 建立起一套大致相当的特性，可以添加自己的浏览器专用元素。Web 浏览器迅速分化为 IE 和 Netscape 两极，在一种浏览器中能够显示的页面在另一种浏览器中经常无法显示。问题在于，Microsoft 凭借在软件市场上无与伦比的强大地位，免费发放 IE，而 Netscape 仍需要销售其浏览器，因为它负担不起免费发布其旗舰产品。为了保持自己的竞争地位，Netscape 需要提供新的特性，使用户宁愿花钱购买它的浏览器，也不使用免费的 Microsoft 浏览器。

这两个公司发布 4.0 版本的浏览器时，竞争达到了白热化程度，这个版本的浏览器引入了动态页面的功能。遗憾的是，Netscape 通过<layer />元素来实现该功能，而 Microsoft 选择通过脚本语言的属性和方法来实现。W3C 需要坚持其立场，因为它的三个主要目标之一已经做出了让步：通用的访问技术。如果用户必须使用指定厂商的浏览器才能访问特定的页面，访问技术如何通用化？W3C 决定使用已有的标准 HTML 元素和层叠样式表，Microsoft 的解决方案部分采用了这两项技术。结果，Microsoft 在浏览器大战中取得了统治地位，并且从未拱手让出这个统治地位。Microsoft 对 IE 的不断升级，使 Netscape Navigator 浏览器毫无还手之力，其继任者 Firefox 的用户群也发展缓慢。其他浏览器，例如 Opera、Safari 和 Chrome，与 Firefox 一起，继续动摇着 Microsoft 在市场上的统治地位，但 Microsoft 的 IE 仍是目前使用最广泛的浏览器。

HTML 4.01 是一个相对稳定的 HTML 标准，其中包含大量特性，任何浏览器厂商都需要很长时间才能完全实现该标准，于是人们转而关注 Web 的其他领域。20 世纪 90 年代后期，引入了一批新标准来控制 HTML 的呈现方式(样式表)和 HTML 在脚本中的表示方式(DOM)。还建立了其他标准，如可扩展标记语言(Extensible Markup Language，XML)提供了一种以保留其结构的方式来表示数据的通用格式。

W3C 网站(www.w3.org)提供了大量处于创建各个阶段的标准。并不是每个标准都与我们有关，并不是每个我们关心的标准都能在该网站上找到，但是，绝大多数我们关心的标准都能在该网站中找到。

下面将简要介绍与 JavaScript 有关的标准和技术，了解它们的相关背景。某些技术我们可能并不熟悉，但是至少应该知道存在这些技术。

12.1.1 HTML

HTML 标准由 W3C 维护。这个标准看起来相当简单，每一版本都似乎只引入了几个新元素。但实际上，由于浏览器厂商的竞争，使标准内容的制定变得异常复杂。HTML 1.0 和 2.0 很简单，其定义文档很小。但当 W3C 讨论 HTML 3.0 版时，所讨论的许多新功能已经被新扩展取代，例如<applet />和<style />元素取代了 3.0 版本浏览器的 appletstyle。因此，HTML 3.0 版本被取消，一个新版本 3.2 成为了标准。

但是，应浏览器厂商的要求引入 HTML 3.2 的大量新特性偏离了 HTML 的本质，因为 HTML 本质上仅用来定义结构。源自<font />元素的新特性就偏离这个宗旨，在 HTML 中添加了不必要的外观特性。随着样式表技术的引入，这些外观特性就是多余的。因此，在 3.0 版本的浏览器中，有 3 种不同的方法定义某项文本的样式。哪种方法是正确的？如果

同时使用这三种方法，文本最终使用哪个样式？HTML 4.0 标准处理了这些问题，指出了将在下一版本的标准中废弃(删除)的大量元素。HTML 4.0 是目前为止最庞大的版本，包含了链接到样式表和 DOM 的特性，还增加了用于有视觉障碍的人员及受忽视的弱势群体的特性，提供了访问的便利性。当前 HTML 标准的版本为 HTML 4.01。

12.1.2　XML

XML 是一个创建标记语言(如 HTML)的标准。XML 本身看上去与 HTML 非常类似，但二者存在本质上的区别。

HTML 实际上是元语言 SGML 的一个应用，SGML 也是生成标记语言的一个标准，它用来创建很多标记语言，但 HTML 是唯一一个被全世界熟悉并广泛应用的标记语言。另一方面，XML 是 SGML 的一个直接子集。通常情况下，SGML 过于复杂，人们无法在计算机上正确地应用 SGML。因此，XML 是 SGML 的简化子集，也比 SGML 更加易读。

XML 主要用来创建结构和外观与 HTML 非常类似的定制标记语言。XML 的一个主要用途是表示数据。普通的数据库可以保存信息，但是不允许其存储的项包含结构信息。XML 则可以使用标记语言的元素结构来表示任何类型的数据，例如数学、化学符号乃至莎士比亚的全部著作等，结构所包含的信息也不会丢失。例如，XML 文档可以用来记录莎士比亚的剧作《凯撒大帝》中 Mark Antony 直到第二场第一幕才出场的信息，而关系数据库必须使用很多额外字段才能保存。如下面的示例：

```
<play>
    <act1>
        <scene1>
            ...
        </scene1>
        <scene2>
            <mark_anthony>
                Caesar, my lord?
            </mark_anthony>
        </scene2>
        <scene3>
            ...
        </scene3>
    </act1>
    <act2>
        ...
    </act2>
    <act3>
        ...
    </act3>
    <act4>
        ...
    </act4>
    <act5>
        ...
```

```
    </act5>
</play>
```

XML 也是完全跨平台的，因为它只保存文本。这意味着 Windows 上的应用程序可以以这种格式打包数据，而 Unix 平台上另一个完全不同的应用程序应可以拆包，并读取相应的数据。

12.1.3 XHTML

XHTML 1.0 是 XML 与 HTML 标准的结合。XHTML 仅是按照 XML 应用的要求重新修改了 HTML 4.01 标准的规范。其优点是允许 XHTML 避免由于浏览器对 HTML 的独特解释而造成的问题。更重要的是，XHTML 提供了一种规范，允许除浏览器之外的其他客户端使用 Web，如掌上电脑、移动电话或者任何可能连接到 Internet 的软件设备(甚或冰箱)。

XHTML 还提供了一种通用的方法来定义自己的元素，而不能随意添加元素。可以使用 XML DTD 和 XML 名称空间(namespace，唯一地标识一组元素，同其他元素区分开来)，通过通用的方法指定新元素(名称空间)。这特别适用于新的标记语言，如无线标记语言(Wireless Markup Language，WML)，它专用于移动技术，要求在较小的界面上显示一组不同的元素。

因此，任何熟悉 HTML 的人都应能查看 XHTML 页面，理解其含义。XHTML 与 HTML 也存在一些不同，但 XHTML 并没有增加新的元素和属性。

下面列出了 XHTML 与 HTML 的主要区别：

- XHTML 建议在文件顶部的第一行放置一个 XML 声明，其格式如下：

  ```
  <?xml version='1.0'?>。
  ```

- 在文件顶部还必须提供一个 DTD 声明，指出所使用的 DTD 标准的版本。
- 在 HTML 元素中必须添加对 XML 名称空间的引用。
- 由于 XML 区分大小写，因此所有的 XHTML 元素名都必须是小写。
- <head/>和<body/>元素必须总是包含在 XHTML 文档中。
- 必须正确地关闭和嵌套标记。只需要一个标记时，如换行标记，必须用一个斜线关闭该标记，例如
。
- 属性的值必须用引号括起来。

这组规则使元素具有严格的层次结构，从而确保 DOM 正常工作，同时，也可以在所有设备类型上对标记语言进行标准化。因此，下一版本的 WML(移动设备的标记语言)也将符合 XHTML 标准。现在创建的 HTML 文档应该遵循上述规则。这样，将非常容易使用 JavaScript 通过 DOM 来操纵页面，使页面按照期望的那样工作。

12.1.4 ECMAScript

JavaScript 的发展历程类似于 HTML。它最初在 Netscape Navigator 中使用，后来添加到 IE 中。JavaScript 的 IE 版本命名为 Jscript，与 Netscape Navigator 中的 JavaScript 非常相似。但是很快，这两种实现方式之间就产生了差异，为这两种浏览器编写脚本时必须关注

这些差异。

奇怪的是，欧洲计算机制造商协会(European Computer Manufacturers Association，ECMA)为 JavaScript 提出了一个标准规范。直到 JavaScript 发布了几个版本后，才开始制定 JavaScript 的标准。HTML 一开始就是由 W3C 联盟制定的，而 JavaScript 是一个私有产品。这是 JavaScript 标准由另一个标准化组织管理的原因。Microsoft 和 Netscape 都同意使用 ECMA 作为标准传播媒介/论坛，因为 ECMA 能够快速地追踪相关标准，而且保持中立。选择 ECMAScript 这个名称是为了不偏向任何厂商，而且 JavaScript 中的 Java 是 Sun 公司许可 Netscape 使用的商标。ECMA-262 标准所制订的规范大致等价于 JavaScript 1.1 规范。

但是，ECMAScript 标准仅包含 JavaScript 的核心特性，如数值、字符串和布尔等基本数据类型，Date、Array、Math 等内置对象，for、while 循环语句，以及 if、else 条件语句。该标准没有包含客户端对象或集合，如 window、document、forms、links 和 images。因此，JavaScript 和 Jscript 都遵循该标准时，这个标准有助于使核心编程任务具备兼容性，但无助于使客户端对象的脚本编程在主流浏览器之间具备兼容性。仍存在一些不兼容的问题。

现在所有的 JavaScript 实现方案都遵循当前的 ECMAScript 标准，即 1999 年 12 月发布的 ECMAScript 3。2006 年 11 月，ECMAScript 4 已经开始制定。

尽管 Microsoft 与 Netscape 在 JavaScript 的实现方案上存在不少差异，但是它们非常接近，可以视为同一种语言。Opera 和 Safari 浏览器也提供了对该标准的相同支持。显然，标准在浏览器的实现方案上提供了统一的语言，尽管 JavaScript 仍然存在类似 HTML 的特性竞争，但是程度大大降低了。

下面介绍 DOM。

12.2　文档对象模型

如前所述，文档对象模型(DOM)是一种独立于浏览器类型来表示文档的方法。它允许开发人员通过一组通用的对象、属性、方法和事件来访问文档，并通过脚本动态修改网页内容。

可以使用几种脚本语言，如 JavaScript 和 VBScript。每种语言的语法都不同，因此编程方法也不同。即使使用像 JavaScript 这样对所有浏览器都通用的语言，也要注意浏览器厂商通常会做少量的改变。因此，为了确保开发人员不与某一实现方案冲突，W3C 以 DOM 标准的形式提供了一组可用于所有脚本语言的通用对象、属性和方法。

12.2.1　DOM 标准

前面一直没有讨论 DOM 标准，有一个特殊的原因：它不是最容易理解的标准。支持一组通用的属性和方法是一项非常复杂的任务，而 DOM 标准分为不同的级别和部分，以处理不同的领域。标准的不同级别都是完成该标准的不同阶段。

1. 0 级

0 级有点名不符实，因为并没有真正的 0 级标准。这个术语实际上表示“旧方式”——

DOM 标准出台之前，浏览器厂商实现的方法。0 级属性是指访问属性和方法的一种较线性的符号。例如，一般使用如下代码访问表单中的一项：

```
document.forms[0].elements[1].value = "button1";
```

本章不打算介绍这类属性和方法，因为它们已经被新方法取代。

2. 1 级

1 级是标准的第一个版本，它分为两部分：一部分定义为核心(包含适用于 XML 和 HTML 的对象、属性和方法)；另一部分定义为 HTML(HTML 专用的对象、属性和方法)。第一部分处理如何导航和操纵文档的结构，这部分的对象、属性和方法都是非常抽象的。第二部分只处理 HTML，提供了对应于所有 HTML 元素的一组对象。本章主要讨论 1 级标准的第二部分。

2000 年，1 级标准进行了修定和补充，尽管这使 1 级标准还处于草案阶段，并没有形成完整的 W3C 推荐阶段。

3. 2 级

2 级标准已经完成，许多属性、方法和事件都已由现代浏览器实现。其中的一些部分为核心和 HTML 专用属性及事件增加了事件和样式表规范(还提供了视图和遍历范围的部分，但本书不包含这些内容，更多信息请访问 www.w3.org/TR/2000/PR-DOM-Level-2-Views-20000927/和 www.w3.org/TR/2000/PR-DOM-Level-2-Traversal-Range-20000927/)。本章将使用 DOM 的这级标准中一些事件和样式表的特性，因为最新版本的浏览器都实现了这些特性。

4. 3 级

3 级标准于 2004 年达到推荐状态，它试图解决在 2 级标准的事件模型中依然存在的许多复杂问题，并添加了对 XML 特性的支持，例如内容模型，以及允许将 DOM 保存为 XML 文档。只有几个浏览器支持 3 级标准的一些特性。

5. 浏览器对标准的支持

虽然任何浏览器都不会百分之百地遵循某个标准，但是某些浏览器，如 Firefox、Opera 和 Safari/Chrome，已经相当接近 DOM 标准。尽管一部分 1 级和 2 级标准中的对象、属性和方法可用于所有的浏览器，但是不能保证 DOM 标准中所有的对象、属性和方法可用于给定版本的浏览器。

DOM 标准中的许多内容直到最近才阐明，只有最新版本的浏览器才添加了对大量 DOM 特性的支持。因此，本章的例子仅能在最新版本的 IE、Firefox、Opera、Safari 和 Chrome 中运行。尽管切合实际的目标是跨浏览器的脚本编程，但向后兼容支持是必不可少的。

尽管仍然没有完全实现标准，但是阐明了应如何实现某个属性或方法，为所有的浏览器厂商提供了一个指导，使它们在浏览器的以后版本中朝着统一的规范前进。DOM 并没有引入任何新的 HTML 元素或样式表属性，来达到其目标。DOM 的思想是使用现有的技

术，并且常常是主流浏览器中的已有属性和方法。

12.2.2　DOM 与 BOM 的区别

如前所述，文档对象模型(DOM)和浏览器对象模型(BOM)之间存在两个主要的区别。但是，有时 BOM 会放在 DOM 名称的下面，使这个问题变得更复杂了。在查阅这个主题的资料时应注意这一点。

- 首先，DOM 仅包含 Web 页面的文档，而 BOM 提供了浏览器各个领域的脚本编程访问，包括按钮、标题栏以及页面的某些部分。
- 其次，BOM 专用于某个浏览器。浏览器是不能标准化的，因为它们必须提供有竞争力的特性。因此，需要另外一组属性、方法甚至对象，才能使用 JavaScript 操作它们。

12.2.3　将 HTML 文档表示树形结构

由于 HTML 是标准化的，因此网页只能包含 HTML 语言支持的特性，如表单、表格和图片等，且需要一种通用的方法，来访问这些特性。此时就可以使用 DOM，它提供了 HTML 文档的统一表达方式，即将整个 HTML 文档/网页表示为树形结构。

实际上，可以将任何 HTML 文档(或任何 XML 文档)表示为树形结构。唯一的前提条件是 HTML 文档应是结构良好的。不同的浏览器或多或少地允许包含一些错误，如未关闭的标记，或者 HTML 表单控件未包含在<form />元素中；但是，要精确地表达 HTML 文档的结构，就必须总是能预计文档的结构。结构方面的陋习，如未关闭的标记，将阻止把结构表达为真正的层次结构，因此是不允许的。通过 DOM 访问元素的能力，取决于将页面表示为层次结构的能力。

1. 树形结构

如果不熟悉树形结构的概念，不用担心。树形结构只是表示层次结构的一种图表方式。

考虑一个包含几章的图书例子。只需略微搜索一下，就可以找到第 543 页的第 3 行。如果这本书的新版本包含额外的章节，再按相同的指令搜索，就可能无法找到相同的内容。但是，如果把指令改为“查找静物画一章、水彩画一节、定位光源一段的内容”，即使新版图书添加了额外的章节，仍能找到指定的内容，只是需要耗费的精力比第一次查找略多。

图书的变化不是特别快，但是网页的信息可能每天、甚至每小时都在变化。此时，第二组指令比第一组更常用。同样的原则也适用于 DOM。以层次方式导航 DOM 比使用完全线性的方式更有意义。将 DOM 视为一棵树，就很容易以这种方式导航页面。在 Windows 中使用 Windows 资源管理器定位文件时，Windows 资源管理器创建了文件夹的树形结构，通过这个树形结构，就可以进入特定的文件夹，定位文件，而不是按照字母顺序来查找。

创建树形结构的规则非常简单。树的顶层是文档和包含页面中所有其他元素的元素，文档就是根节点。节点只是树中的一个点，表示某个元素、元素的属性或特性，甚或元素包含的文本。根节点包含了所有的其他节点，如 DTD 声明、XML 声明(如果有)以及根元

素(包含所有其他元素的 HTML 或 XML 元素)。在 HTML 文档中，根元素应总是<html/>元素。在根元素之下，是根元素包含的 HTML 元素。HTML 页面一般在<html/>元素中包含<head/>和<body/>元素。这些元素表示为根元素节点下的节点，而根元素节点本身在树顶部的根节点之下，如图 12-1 所示。

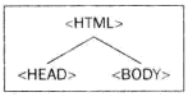

图 12-1

代表<head/>和<body/>元素的两个节点是子节点，它们上面的<html/>元素节点是父节点。由于<head/>和<body/>元素都是<html/>元素的子节点，所以它们位于父节点<html/>元素下的同一层。<head/>和<body/>元素又包含其他子节点/HTML 元素，这些子节点位于<head/>和<body/>节点的下一层。所以，子节点也可以是父节点。另一个元素中的一组 HTML 元素构成了树形结构中同一层上的另一个子节点。清晰说明这些结构的最简单方式是列举一个例子。

2. HTML 页面示例

考虑如下所示的基本 HTML 页面：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
</head>
<body>
    <h1>My Heading</h1>
    <p>This is some text in a paragraph.</p>
</body>
</html>
```

<html/>元素包含<head/>和<body/>两个元素，只有<body/>元素包含实际内容，它包含一个<h1/>元素和一个<p/>元素。<h1/>元素又包含文本 My Heading。到达不包含其他内容的一项时，如文本、图片或者元素，树结构就在该节点终止。这种节点称为叶节点。<p/>节点包含一段文本，文本也是文档中的一个节点。这个页面可以表达为如图 12-2 所示的树形结构。

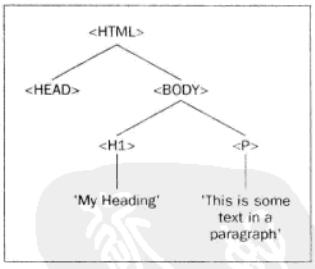

图 12-2

非常简单。这个例子太简单，下面是一个略微复杂的例子，它包含了一个表格。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>This is a test page</title>
</head>
```

```
<body>
    <span>Below is a table...</span>
    <table border="1">
        <tr>
           <td>Row 1 Cell 1</td>
           <td>Row 1 Cell 2</td>
        </tr>
        <tr>
           <td>Row 2 Cell 1</td>
           <td>Row 2 Cell 2</td>
        </tr>
    </table>
</body>
</html>
```

这个例子没有不寻常的内容；文档包含一个表格，该表格有两行，每行有两个单元格。可以用树形结构表示该页面的层次结构(例如，<html/>元素包含<head/>和<body/>元素，<head/>元素又包含了一个<title/>元素)，如图 12-3 所示。

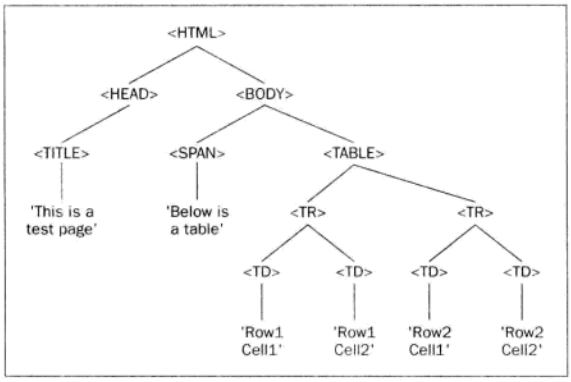

图　12-3

树的顶层非常简单，<html/>元素包含<head/>和<body/>元素。<head/>元素又包含一个<title/>元素，该<title/>元素又包含一些文本。该文本节点是一个结束了分支的子节点(叶节点)。下面看看另一个节点，即<body/>元素节点，沿着这个分支向下浏览。<body/>元素包含两个元素和<table/>。元素仅包含文本，是叶节点。而<table/>元素包含两个<tr/>表行元素，每个<tr/>表行元素又包含两个<td/>表单元格元素，每个<td/>单元格都包含文本，这是树的底层。现在这棵树是 HTML 代码的完整表达。

12.2.4　DOM 核心对象

前面的介绍都是理论知识，下面进入实践阶段。

DOM 提供了一组具体的对象、属性和方法，通过 JavaScript 可以访问它们，以便浏览 DOM 的树形结构。下面先介绍 DOM 中的对象集，它们表示树中的节点(如元素、属性或文本)。

1. 基本的 DOM 对象

表 12-1 中的三个对象称为基本的 DOM 对象。

表 12-1

对　象	说　明
Node	文档中的每个节点都有自己的 Node 对象
NodeList	这是 Node 对象的列表
NamedNodeMap	允许按名称(而不是按索引)访问所有 Node 对象

这是DOM与BOM的不同之处。BOM对象的名称与浏览器的特定部分相关，如window对象、forms 和 images 集合。如前所述，要将网页视为一棵树并在其中导航，就必须使之抽象化。无须事先知道页面的结构，所有的东西最终都只是一个节点。从一个 HTML 元素移到另一个 HTML 元素，或者从元素移到元素属性上，就是从一个节点移动到另一个节点。这也意味着，可以添加、替换或移除网页的各个部分，而不会影响整个页面的结构，因为我们只是改变了节点。因此有 3 个晦涩难懂的对象表示树形结构。

如前所述，树形结构的顶部是根节点，它包含了 XML 声明、DTD 和根元素。因此要表示文档，仅有这三个对象是不够的。实际上，有不同的对象表示树中不同类型的节点。

2. 高级 DOM 对象

DOM 中的所有东西都是节点，所以节点肯定有各种类型。节点是元素、属性还是纯文本？Node 对象有不同的子对象，来表示每种类型的节点。表 12-2 完整列出了可以通过 DOM 访问的所有不同的节点类型对象。本书没有介绍其中的许多节点对象，因为它们更适用于 XML 文档，而不是 HTML 文档。但应注意三种主要的节点类型，即元素、属性和文本。

表 12-2

对　象	说　明
Document	文档的根节点
DocumentType	XML 文档的 DTD 或模式类型
DocumentFragment	文档部分的临时存储空间
EntityReference	XML 文档中的实体引用
Element	文档中的一个元素
Attr	文档中元素的一个属性
ProcessingInstruction	处理指令
Comment	XML 文档或 HTML 文档中的注释
Text	构成元素子节点的纯文本
CDATASection	XML 文档中的 CDATA 部分
Entity	DTD 中未解析的实体
Notation	DTD 中声明的记号

本章不介绍其中的大多数对象，但如果需要导航 XML 文档的 DOM，就要使用它们。

这些对象都继承了 Node 对象的所有属性和方法，还包含自己的一些方法和属性。下一节将介绍一些例子。

12.2.5　DOM 对象及其属性和方法

如果要讨论 DOM 中所有对象的属性和方法，将需要半本书的篇幅。本书只关注三个对象 Node、Element 和 Document。通过这三个对象，就可以创建、修改和导航树形结构。另外，本书也不研究这些对象的每个属性和方法，只讨论最常用的属性和方法，使用它们实现指定的功能。

注意：

附录较为全面地介绍了 DOM 及其对象和属性。

1. Document 对象及其方法

Document 引用类型提供了各种属性和方法，非常有助于编写 DOM 的脚本。其方法允许查找单个元素或元素组，创建新元素、属性和文本节点。任何 DOM 编写脚本人员都应了解这些方法和属性，因为它们非常常用。

Document 对象的方法可能是最重要的。尽管我们有许多工具，但只有 Document 对象的方法可以在页面上查找、创建和删除元素。

查找一个或多个元素

假定有一个 HTML 网页，如何在页面的脚本中找到特定的元素？Document 引用类型提供了表 12-3 中所示的方法来完成这个任务：

表　12-3

document 对象的方法	说　　明
getElementById(idValue)	根据所提供的元素的 id 值，返回对该元素的引用(节点)
getElementsByTagName(tagName)	根据参数中提供的标记，返回对一组元素的引用(节点列表)

第一个方法 getElementById()要求确保，要在页面中快速访问的每个元素都使用 id 属性，否则该方法返回 null(表示缺失或未知的值)。在第一个例子中给元素中添加 id 属性。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"><head>
    <title>example</title>
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
</body>
</html>
```

现在，可以使用 getElementById()方法，返回一个对页面上带 id 属性的任意 HTML 元素的引用。例如，如果在突出显示的部分添加如下代码，就可以找到并引用<h1/>元素。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
   <title>example</title>
</head>
<body>
    <h1 id="Heading1">My Heading</h1>
    <p  id="Paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
       alert(document.getElementById("Heading1"));
    </script>
</body>
</html>
```

图 12-4 显示了在 Firefox 中运行这段代码的结果。

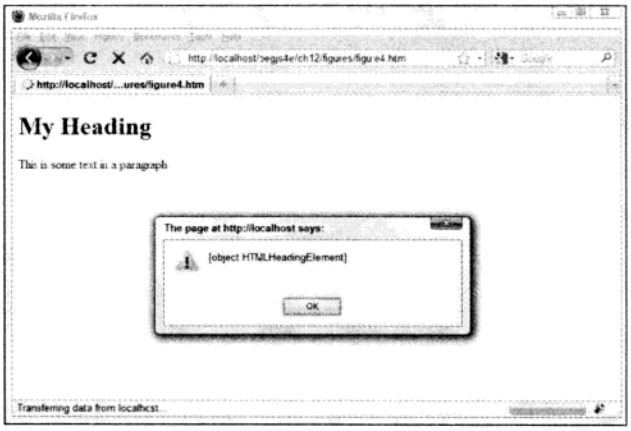

图 12-4

注意：

HTMLHeadingElement 是 HTML DOM 的一个对象。所有 HTML 元素在 DOM 中都有一个对应的引用类型。HTML DOM 的更多对象参见附录 C。

除了<h1/>或<h1 id="heading1">之外，也许读者认为代码将返回某个内容，但实际上只返回了对<h1/>元素的一个引用。这个<h1/>元素引用是非常有用的，可以用于修改<h1/>元素的属性，如通过 style 对象修改字号或颜色。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>example</title>
```

```
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
        var h1Element = document.getElementById("heading1");
        h1Element.style.color = "red";
    </script>
</body>
</html>
```

如果在浏览器中进行显示，就会发现，可以直接在脚本中修改<h1/>元素的属性，这里将文本改为红色。

提示：

Style 对象指向元素的样式属性，它允许修改赋予元素的 CSS 样式。参见本章后面介绍的样式对象。

第二个方法 getElementsByTagName()的工作方式相同，但顾名思义，它可以返回多个元素。在包含表格的 HTML 文档示例中，代码使用这个方法返回表单元格(<td/>)，就将得到一个包含全部 4 个单元格的节点列表。该方法仍只返回一个对象，但是该对象是一个元素集合。注意集合是类似于数组的结构，所以应指定集合中特定元素的索引号。可以使用方括号来指定，另一种方法是使用 NodeList 对象的 item()方法：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>This is a test page</title>
</head>
<body>
    <span>Below is a table... </span>
    <table border="1">
        <tr>
            <td>Row 1 Cell 1</td>
            <td>Row 1 Cell 2</td>
        </tr>
        <tr>
            <td>Row 2 Cell 1</td>
            <td>Row 2 Cell 2</td>
        </tr>
    </table>
    <script type="text/javascript">
        var tdElement = document.getElementsByTagName("td").item(0);
        tdElement.style.color = "red";
    </script>
</body>
</html>
```

如果运行这个例子，再次使用 style 对象，就会修改表格中第一个单元格的内容的样式。如果要以这种方式修改所有单元格的颜色，就应遍历节点列表，如下所示：

```
<script type="text/javascript">
    var tdElements = document.getElementsByTagName("td");
    var length = tdElements.length;

    for (var i = 0; i < length; i++)
    {
        tdElements[i].style.color = "red";
    }
</script>
```

getElementsByTagName()方法的一个注意事项是它的参数是放在引号中的元素名，没有包含通常围绕着标记的尖括号。

创建元素和文本

Document 对象还提供了一些创建元素和文本的方法，如表 12-4 所示。

表 12-4

Document 对象的方法	说　明
createElement(elementName)	使用指定的标记名创建一个元素节点，返回所创建的元素
createTextNode(text)	创建并返回包含所提供文本的文本节点

下面的代码演示了这些方法的用法：

```
var pElement = document.createElement("p");
var text = document.createTextNode("This is some text.");
```

这段代码创建了一个<p/>元素，并把其引用保存在 pElement 变量中。接着创建一个包含文本“This is some text.”的文本节点，并将其引用保存在 text 变量中。

但仅创建节点是不够的，还必须把它们添加到文档中。下面就讨论这个过程。

Document 对象的属性：获取文档的根元素

现在我们知道如何引用页面上的某个元素，但是如何访问整个树形结构呢？树形结构包含页面上所有的元素和节点，并将它们放在一个层次结构中。如果要引用这个结构，需要使用 document 对象的一个特殊属性，该属性返回文档的最外层元素。在 HTML 中，它应总是<html/>元素。返回这个元素的属性是 documentElement，如表 12-5 所示。

表 12-5

Document 对象的属性	说　明
documentElement	返回对文档最外层元素的引用(即根元素，如<html/>)

可以按下例中的方式使用 documentElement。对于简单的 HTML 页面，可以把整个 DOM 传送给一个变量，如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>example</title>
</head>
<body>
   <h1 id="Heading1">My Heading</h1>
   <p id="Paragraph1">This is some text in a paragraph</p>
   <script type="text/javascript">
       var container = document.documentElement;
   </script>
</body>
</html>
```

变量 container 现在包含根元素<html/>。documentElement 属性以对象的形式返回对这个元素的引用，确切地讲是 Element 对象。Element 对象有自己的属性和方法。如果想使用它们，可以通过变量名后跟属性名或方法名进行引用：

```
container.elementObjectProperty
```

幸运的是，Element 对象仅有一个属性。

2. Element 对象

Element 对象非常简单，与 Node 对象相比尤其如此(稍后介绍)，它只有几个成员(属性和方法)，如表 12-6 所示。

表　12-6

成　员　名	说　　明
tagName	返回元素的标记名称
getAttribute()	获取属性的值
setAttribute()	用指定的值设置属性
removeAttribute()	从元素中删除指定的属性及其值

获取元素的标记名称：tagName 属性

Element 对象的唯一属性 tagName 返回对元素的标记名称的引用。

在前面的例子中，变量 container 包含元素<html/>。添加以下突出显示的代码，以使用 tagName 属性。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>example</title>
</head>
```

```
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
        var container = document.documentElement;
        alert(container.tagName);
    </script>
</body>
</html>
```

这段代码证明，变量 container 保存了最外层的元素，其他元素都包含在该元素中，如图 12-5 所示。

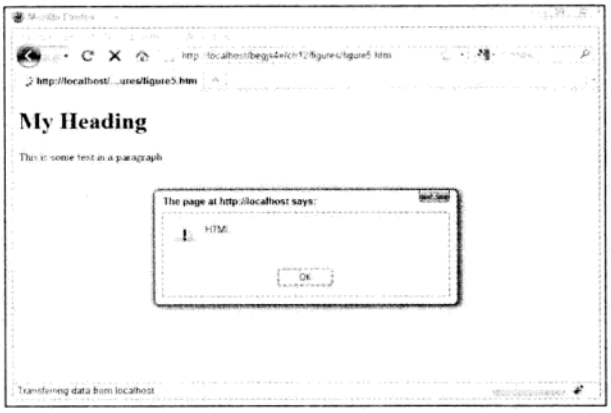

图 12-5

Element 对象的方法：获取和设置属性

如果要设置元素中除 style 之外的其他属性，应使用 Element 对象专用于 DOM 的方法。

可以用于返回和修改 HTML 元素属性的内容的 3 个方法是 getAttribute()、setAttribute() 和 removeAttribute()，如表 12-7 所示。

表 12-7

Element 对象的方法	说　明
getAttribute(attributeName)	返回所提供的属性的值。如果该属性不存在，就返回 null 或空字符串
setAttribute(attributeName, value)	设置属性的值
removeAttribute(attributeName)	删除属性的值，代之以默认值

下面简要介绍这些方法的工作方式。

试一试　处理属性

打开文本编辑器，输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
    <title>Chapter 12: Example 1</title>
</head>
<body>
    <p id="paragraph1">This is some text.</p>
    <script type="text/javascript">
        var pElement = document.getElementById("paragraph1");
        pElement.setAttribute("align", "center");

        alert(pElement.getAttribute("align"));

        pElement.removeAttribute("align");
    </script>
</body>
</html>
```

保存为 ch12_examp1.htm，并在浏览器中打开。屏幕中央会显示<p/>元素的文本，警告框则显示文本 center，如图 12-6 所示。

图　12-6

单击 OK 按钮，文本就会左对齐，如图 12-7 所示。

图　12-7

这个 HTML 页面包含 id 值为 paragraph1 的一个<p/>元素。在 JavaScript 代码中使用 getElementById()方法通过这个值查找元素节点，把其引用存储在 pElement 变量中。

```
var pElement = document.getElementById("paragraph1");
```

有了元素的引用后，就使用 setAttribute()方法把 align 属性设置为 center。

```
pElement.setAttribute("align", "center");
```

这行代码的结果是把文本移到浏览器窗口的中心位置。

接着使用 getAttribute()方法获取 align 属性的值，并在警告框中显示它。

```
alert(pElement.getAttribute("align"));
```

这行代码在警告框中显示值 center。

最后，使用 removeAttribute()方法删除 align 属性，令文本左对齐。

注意：

严格来讲，align 属性在 HTML 4.0 中被废弃，但可以使用它，因为它是有效的，而且非常容易在网页上演示可见效果。

3. Node 对象

从网页中获取了元素后，如果要在页面中一个元素一个元素或一个属性一个属性地访问，该怎么办？这需要返回到更低层次。要遍历元素、属性和文本，需要沿着树形结构的节点移动。至于节点包含什么内容，甚至节点是什么类型，都是无关紧要的。因此需要使用 DOM 核心规范的一个对象 Node，整个树形结构都是由这些基本的 Node 对象构成的。

Node 对象：导航 DOM

表 12-8 列出了 Node 对象的常用属性，这些属性提供了节点的信息，即当前节点是元素、属性还是文本，并可以从一个节点移到另一节点。

表 12-8

Node 对象的属性	对属性的说明
firstChild	返回元素的第一个子节点
lastChild	返回元素的最后一个子节点
previousSibling	在同级子节点中，返回当前子节点的前一个兄弟节点
nextSibling	在同级子节点中，返回当前子节点的后一个兄弟节点
ownerDocument	返回包含节点的文档的根节点(该属性不能用于 IE 5 或 IE 5.5)
parentNode	返回树形结构中包含当前节点的元素
nodeName	返回节点的名称
nodeType	返回一个数字，表示节点的类型
nodeValue	以纯文本格式获取或设置节点的值

下面简要介绍这些属性的用法。考虑下面的例子：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>example</title>
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
        var h1Element = document.getElementById("heading1");
        h1Element.style.color = "red";
    </script>
</body>
</html>
```

现在，可以使用 h1Element 浏览树形结构，进行必要的更改。下面的代码把 h1Element 作为起点，查找<p/>元素，并修改其文本颜色。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>example</title>
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
        var h1Element = document.getElementById("heading1");
        h1Element.style.color = "red";

        var pElement;
        if (h1Element.nextSibling.nodeType == 1)
        {
           pElement = h1Element.nextSibling;
        }
        else
        {
           pElement = h1Element.nextSibling.nextSibling;
        }
        pElement.style.color = "red";
    </script>
</body>
</html>
```

这段代码演示了 IE 的 DOM 和其他浏览器中的 DOM 之间的一个根本区别。Firefox、

Safari、Chrome 和 Opera 的 DOM 把 DOM 树中的所有东西都看作节点，包括元素之间的空白。而 IE 会去除不必要的空白。上一个示例中的<p/>元素与<h1/>元素同级，所以要定位<p/>元素，需要检查下一个同级元素的 nodeType 属性。元素的节点类型是 1，文本节点的类型是 3。如果 nextSibling 的 nodeType 是 1，就把这个同级节点的引用赋予 pElement。否则，就获取 h1Element 同级节点(空白文本节点)的下一个同级节点(<p/>元素)。

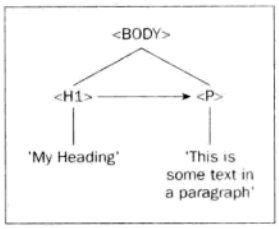

图 12-8

实际上，属性结构的导航如图 12-8 所示。

相同的原则也适用于反向导航。修改代码，从<p/>元素导航到的<h1/>元素。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>example</title>
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>
    <script type="text/javascript">
        var pElement = document.getElementById("paragraph1");
        pElement.style.color = "red";

        var h1Element;
        if (pElement.previousSibling.nodeType == 1)
        {
            h1Element = pElement.previousSibling;
        }
        else
        {
            h1Element = pElement.previousSibling.previousSibling;
        }
        h1Element.style.color = "red";
    </script>
</body>
</html>
```

这里的过程正好相反：首先把<p/>元素的 id 属性传送给 getElementById()方法，找到<p/>元素，再把返回的元素引用保存到 pElement 变量中。然后查找正确的上一个同级节点，这样代码可以在所有浏览器上工作，并把文本颜色改成红色。

试一试　使用 DOM 导航 HTML 文档

前面一直没有真正地导航 HTML 文档，只是使用 document.getElementById()方法返回一个元素，再从该元素导航到其他节点。现在，使用 document 对象的 documentElement 属

性，真正地导航 HTML 文档。从树的顶部开始，向下导航到各个子节点，获取相应的元素，接着使用前面的方法访问这些元素的子节点，修改属性。

在文本编辑器中输入下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 2</title>
</head>
<body>
    <h1 id="heading1">My Heading</h1>
    <p id="paragraph1">This is some text in a paragraph</p>

    <script type="text/javascript">
    var htmlElement; // htmlElement stores reference to <html>
    var headElement; // headingElement stores reference to <head>
    var bodyElement; // bodyElement stores reference to <body>
    var h1Element; // h1Element stores reference to <h1>
    var pElement; // pElement stores reference to <p>

    htmlElement = document.documentElement;
    headElement = htmlElement.firstChild;

    alert(headElement.tagName);

    if (headElement.nextSibling.nodeType == 3)
    {
        bodyElement = headElement.nextSibling.nextSibling;
    }
    else
    {
        bodyElement = headElement.nextSibling;
    }

    alert(bodyElement.tagName);

    if (bodyElement.firstChild.nodeType == 3)
    {
        h1Element = bodyElement.firstChild.nextSibling;
    }
    else
    {
        h1Element = bodyElement.firstChild;
    }

    alert(h1Element.tagName);
    h1Element.style.fontFamily = "Arial";

    if (h1Element.nextSibling.nodeType == 3)
    {
        pElement = h1Element.nextSibling.nextSibling;
```

```
        }
        else
        {
            pElement = h1Element.nextSibling;
        }

        alert(pElement.tagName);
        pElement.style.fontFamily = "Arial";

        if (pElement.previousSibling.nodeType==3)
        {
            h1Element = pElement.previousSibling.previousSibling
        }
        else
        {
            h1Element = pElement.previousSibling
        }
        h1Element.style.fontFamily = "Courier"
        </script>
    </body>
    </html>
```

保存为 ch12_examp2.htm 文件。在浏览器中打开该页面，在弹出的每个消息框中单击 OK 按钮，最后的页面如图 12-9 所示。但 IE 在打开、关闭了所有的警告框后才显示样式的变化。

图 12-9

上面的代码添加了几个警告框，来说明当前位于树形结构的哪一部分，使这个例子的执行过程非常清晰。另外，将变量命名为各自的元素名，以便清楚地说明每个变量保存的内容(也可以简单地将变量命名为 a、b、c、d 和 e，因此不要以为必须遵循这个命名约定)。

脚本块的顶部用 documentElement 属性来获取整个文档。

```
var htmlElement = document.documentElement;
```

根元素是<html/>，因此第一个变量命名为 htmlElement。在树中可以看到，<html/>元素有两个子节点：一个包含<head/>元素；另一个包含<body/>元素。首先移动到<head/>元素。为此使用 Node 对象的 firstChild 属性，它包含<html/>元素。使用第一个警告框证明，这是正确的：

```
alert(headingElement.tagName);
```

<body/>元素是<head/>元素的下一个同级元素，所以创建一个变量，它保存从<head/>元素导航的下一个同级元素。

```
if (headingElement.nextSibling.nodeType == 3)
{
    bodyElement = headingElement.nextSibling.nextSibling;
}
else
{
    bodyElement = headingElement.nextSibling;
}

alert(bodyElement.tagName);
```

上面的代码检查 headingElement.nextSibling 节点的 nodeType 属性。如果它返回 3(文本节点的 nodeType 是 3)，则将 bodyElement 设置为 headingElement.nextSibling.nextSibling；否则，设置为 headingElement.nextSibling。

使用警告框证明，目前位于<body/>元素上。

```
alert(bodyElement.tagName);
```

本页面中的<body/>元素也有两个子元素<h1/>和<p/>。使用 firstChild 属性导航到<h1/>元素。这里为非 IE 浏览器检查该子节点是否是空白的。接着用警告框证明，目前位于<h1/>。

```
if (bodyElement.firstChild.nodeType == 3)
{
    h1Element = bodyElement.firstChild.nextSibling;
}
else
{
    h1Element = bodyElement.firstChild;
}

alert(h1Element.tagName);
```

第三个警告框显示后，修改第一个元素的样式，把字体设置为 Arial。

```
h1Element.style.fontFamily = "Arial";
```

接着使用 nextSibling 属性导航到<p/>元素，再次检查空白。

```
if (h1Element.nextSibling.nodeType == 3)
{
    pElement = h1Element.nextSibling.nextSibling;
}
else
{
    pElement = h1Element.nextSibling;
}

alert(pElement.tagName);
```

将<p/>元素的字体也设置为 Arial。

```
pElement.style.fontFamily = "Arial";
```

最后使用 previousSibling 属性返回树形结构的<h1/>元素，这次将字体设置为 Courier。

```
if (pElement.previousSibling.nodeType==3)
{
    h1Element = pElement.previousSibling.previousSibling
}
else
{
    h1Element = pElement.previousSibling
}

h1Element.style.fontFamily = "Courier";
```

这是一个非常容易的例子，因为使用了前面创建的树形结构，本例还说明了 DOM 如何高效地创建了这个层次结构，并可以使用脚本在其中移动。

6. Node 对象的方法

Node 对象的属性可以导航 DOM，而其方法提供了完全不同的功能：在 DOM 中同添加和删除节点，从根本上改变了 HTML 文档的结构。表 12-9 列出了这些方法。

表 12-9

Node 对象的方法	说　明
appendChild(newNode)	将一个新 node 对象添加到子节点列表的末尾。该方法返回追加的节点
cloneNode(cloneChildren)	返回当前节点的一个副本。该方法的参数是一个布尔值，如果该值为 true，则克隆当前节点及其所有的子节点。如果该值为 false，则仅克隆当前节点，而不包含其子节点
hasChildNodes()	如果节点有子节点，则返回 true，否则返回 false
insertBefore(newNode, referenceNode)	在 referenceNode 指定的节点前，插入一个 node 对象，返回新插入的节点
removeChild(childNode)	从 node 对象的子节点列表中，删除一个子节点，并返回删除的节点

试一试　**使用 DOM 方法创建 HTML 元素和文本**

下面将创建一个仅包含段落<p/>和标题<h1/>元素的网页，但不是使用 HTML 创建，而是使用 DOM 属性和方法把这些元素添加到网页中。打开文本编辑器，输入以下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 3</title>
</head>
<body>
    <script type="text/javascript">
        var newText = document.createTextNode("My Heading");
        var newElem = document.createElement("h1");

        newElem.appendChild(newText);
        document.body.appendChild(newElem);

        newText = document.createTextNode("This is some text in a paragraph");
        newElem = document.createElement("p");

        newElem.appendChild(newText);
        document.body.appendChild(newElem);
    </script>
</body>
</html>
```

将这个页面保存为 ch12_examp3.htm 页面，并在浏览器中打开它，如图 12-10 所示。

图　12-10

结构看起来很乏味。其实使用 HTML 可以更简单地完成这个页面。但这不重要。本例说明，可以在 JavaScript 中使用 DOM 的属性和方法，插入这些功能。脚本块的前两行代码定义了变量，并初始化为要插入页面的文本和 HTML 元素。

```
var newText = document.createTextNode("My Heading");
var newElem = document.createElement("h1");
```

从树的底层开始，通过 createTextNode()方法创建一个文本节点，然后使用 create Element()方法创建一个 HTML 标题。

到目前为止，这两个变量是完全相互独立的，文本节点和<h1>元素是毫无关联的。下一行代码把文本节点关联到 HTML 元素上。通过变量名 newElem 引用前面创建的 HTML 元素，然后使用节点的 appendChild()方法，将前面创建的变量 newText 的内容作为参数传入。

```
newElem.appendChild(newText);
```

下面总结一下。首先创建一个文本节点，并保存在变量 newText 中。然后创建一个<h1>元素，并保存在变量 newElem 中。接着把文本节点追加为<h1>元素的子节点。现在仍然存在一个问题：我们创建了一个包含值的元素，但是该元素仍不在文档中。还需要将前面创建的所有内容关联到文档正文上。同样，只需使用 document.body 的 appendChild()方法，但这次将 newElem(也是一个 Node 对象)作为参数传入。

```
document.body.appendChild(newElem);
```

代码的第一部分就完成了。接下来，只需按同样的办法处理<p/>元素。

```
newText = document.createTextNode("This is some text in a paragraph");
newElem = document.createElement("p");

newElem.appendChild(newText);
document.body.appendChild(newElem);
```

先创建一个文本节点，再创建一个元素，并将文本关联到元素上，最后，将元素和文本关联到文档正文上。

注意创建节点顺序并不重要。这个示例先创建文本节点，再创建元素节点，也可以酌情先创建元素节点，再创建文本节点。

但追加节点的顺序对性能非常重要。更新 DOM 是一个昂贵的过程，如果对 DOM 的改动次数比较多，性能就会降低。例如，本例把完成的元素追加到文档正文中，仅更新了 DOM 两次。如果把元素追加到文档正文中，再把文本降低追加到元素中，就需要更新 4 次 DOM。一般尽可能仅将完成的元素节点(即元素及其属性和文本)追加到文档中。

现在可以导航和修改 DOM 了。下面分析如何操作 DOM 节点。

12.3 操作 DOM

如本章开头所述，DHTML 是浏览器加载 HTML 页面后，操纵该页面的技术。前面学

习了基本 DOM 对象的属性和方法，了解到如何通过 JavaScript 遍历 DOM。

上一节列举了操作 DOM 的一些示例，主要演示了如何更改包含在元素中的文本颜色和字体。本节将扩展这些知识。

12.3.1　访问元素

如上一节所述，DOM 包含了查找和访问 HTML 元素所需的工具；getElementById()方法使用得很频繁，并通过示例说明了如何在页面中查找特定的元素。

给 DOM 编写脚本时，常常需要确定要操作什么元素。找到这些元素的最简单方法是使用 id 属性和 getElementById()方法。不必担忧给 HTML 元素赋予 id 属性，这是在页面中查找元素最简捷高效的方式。

12.3.2　改变元素的外观

DOM 最常见的操作是改变元素的外观。这种改变可以使网站的访客获得交互式体验，甚至可以向用户提示重要的信息或者需要执行某种操作。改变页面元素的外观，主要指的是改变 HTML 元素的 CSS 属性。在 JavaScript 中，这有两种途径：

- 使用 style 属性改变各个 CSS 属性
- 修改元素的 class 属性值

1. 使用 style 属性

要修改特定的 CSS 属性，必须使用 style 属性。所有现代浏览器都实现了这个对象，该对象直接映射到元素的 style 属性。style 对象包含 CSS 属性，使用该对象，可以修改浏览器支持的任何 CSS 属性。前面介绍过 style 属性的用法。这里简单复习一下：

```
element.style.cssProperty = value;
```

CSS 属性名通常匹配 CSS 样式表使用的名称；因此，要改变元素的文本颜色，可使用 color 属性，如下面的代码所示：

```
var divAdvert = document.getElementById("divAdvert"); //Get the desired element

divAdvert.style.color = "blue"; //Change the text color to blue
```

但是，有时属性名与其在 CSS 文件中的名称略有不同。包含连字符(-)的 CSS 属性就是这个例外的好例子。对于这些属性，需要去掉连字符，并把连字符后的第一个字母改成大写。下面的代码演示了其错误和正确做法：

```
divAdvert.style.background-color = "gray"; //Wrong

divAdvert.style.backgroundColor = "gray"; //Correct
```

也可以使用 style 对象来获取以前声明的样式。但是，如果要获取的 style 属性没有用样式属性(即内联样式) 或 style 对象设置，则不能获取属性的值。下面的 HTML 包含一个样式表和 元素：

```
<style type="text/css">
#divAdvert
{
    background-color: gray;
}
</style>

<div id="divAdvert" style="color: green">I am an advertisement.</div>
```

浏览器呈现这个元素时，会显示灰底绿字。如果使用 style 对象获取 background-color 和 color 属性的值，将获得以下混合效果：

```
var divAdvert = document.getElementById("divAdvert"); // Get the desired element
alert(divAdvert.style.backgroundColor); // Alerts an empty string
alert(divAdvert.style.color); // Alerts green
```

之所以得到这些结果，是由于 style 对象直接映射到元素的 style 属性上。如果在 <style/>块中设置样式声明，则不能通过 style 对象来获取该属性的值。

试一试　使用 style 对象

下面的简单例子使用 style 对象来更改一些文本的外观。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 4</title>
    <style type="text/css">
    #divAdvert
    {
        font: 12pt arial;
    }
    </style>
    <script type="text/javascript">
    function divAdvert_onMouseOver()
    {
        var divAdvert = document.getElementById("divAdvert");
        divAdvert.style.fontStyle = "italic";
        divAdvert.style.textDecoration = "underline";
    }

    function divAdvert_onMouseOut()
    {
        var divAdvert = document.getElementById("divAdvert");
        divAdvert.style.fontStyle = "normal";
        divAdvert.style.textDecoration = "none";
    }
    </script>
</head>
<body>
```

```
    <div id="divAdvert" onmouseover="divAdvert_onMouseOver()"
        onmouseout="divAdvert_onMouseOut()">
        Here is an advertisement.
    </div>
</body>
</html>
```

保存为 ch12_examp4.htm 文件。在浏览器中运行它时，页面将显示一行文本，如图 12-11 所示。

图　12-11

将鼠标移到文本上，该文本将变成带下划线的斜体，如图 12-12 所示。

图　12-12

把鼠标移出文本时，该文本将变回普通的样式。

在页面正文中创建了一个 id 为 divAdvert 的 元素，将 mouseover 和 mouseout 事件分别连接到 divAdvert_onMouseOver()和 divAdvert_onMouseOut()函数上，这两个函数定义于页面开头的<script/>块中。

鼠标指针移入 元素时，会调用 divAdvert_onMouseOver()函数。

```
function divAdvert_onMouseOver()
{
    var divAdvert = document.getElementById("divAdvert");
    divAdvert.style.fontStyle = "italic";
    divAdvert.style.textDecoration = "underline";
}
```

在处理 元素前，必须先获取它。为此，只需使用 getElementById()方法。有了这个元素后，就可以修改它的样式。首先，用 fontStyle 属性把文本改为斜体，接着将 textDecoration 属性修改为 underline，给文本加上下划线：

我们一般不希望文本一直以斜体和下划线方式显示。所以 mouseout 事件允许把文本恢复为最初的样式。触发这个事件时，将调用 divAdvert_onMouseOut()函数。

```
function divAdvert_onMouseOut()
{
    var divAdvert = document.getElementById("divAdvert");
    divAdvert.style.fontStyle = "normal";
    divAdvert.style.textDecoration = "none";
}
```

这个函数的代码与 divAdvert_onMouseOver()函数类似。首先获取 divAdvert 元素，然后将 fontStyle 属性设置为 normal，删除斜体样式。再将 textDecoration 属性设置为 none，删除下划线样式。

2. 修改 class 属性

使用元素的 class 属性可以把 CSS 类赋予元素。在 DOM 中，该属性通过 className 属性来使用，可以通过 JavaScript 修改 className 属性，为元素应用另一个样式规则。

```
element.className = sNewClassName;
```

使用 className 属性来修改元素样式具有两个优点。

- 减少了需要编写的 JavaScript 代码量，这是大家都希望的。
- 可将样式信息从 JavaScript 文件中提取出来，放在它所属的 CSS 文件中。这样，对样式规则的任何修改都更容易，因为无需打开多个文件，修改它们。

试一试　使用 className 属性

打开前一节的 ch12_examp4.htm，进行一些修改：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 5</title>
    <style type="text/css">
```

```
        .defaultStyle
        {
            font: normal 12pt arial;
            text-decoration: none;
        }
        .newStyle
        {
            font: italic 12pt arial;
            text-decoration: underline;
        }
    </style>
    <script type="text/javascript">
    function divAdvert_onMouseOver()
    {
        var divAdvert = document.getElementById("divAdvert");
        divAdvert.className = "newStyle";
    }
    function divAdvert_onMouseOut()
    {
        var divAdvert = document.getElementById("divAdvert");
        divAdvert.className = "defaultStyle";
    }
    </script>
</head>
<body>
    <div id="divAdvert" class="defaultStyle" onmouseover="divAdvert_onMouseOver()"
       onmouseout="divAdvert_onMouseOut()">
       Here is an advertisement.
    </div>
</body>
</html>
```

保存为 ch12_examp5.htm 文件。该页面的执行方式与 ch12_examp4.htm 完全相同。把鼠标指针移到文本上时，文本将变成带下划线的斜体。把指针移出文本时，文本恢复为普通样式。

这个 HTML 页面与使用 style 对象创建的页面有几个重要的差别。首先，删除了 #divAdvert 样式规则，代之以两个 CSS 类：

```
.defaultStyle
{
    font: normal 12pt arial;
    text-decoration: none;
}

.newStyle {
    font: italic 12pt arial;
    text-decoration: underline;
}
```

第一个类 defaultStyle 是首先应用于 元素的规则。它声明了 normal 12pt Arial 字体，且不带下划线。第二个类 newStyle 包含的样式声明指定了 12pt italic Arial 字体，且带

下划线。进行了这些修改， 元素的定义就改为使用 CSS 类 defaultStyle。

```
<div id="divAdvert" class="defaultStyle" onmouseover="divAdvert_onMouseOver()"
    onmouseout="divAdvert_onMouseOut()">
    Here is an advertisement.
</div>
```

注意，id 属性是相同的；仍然需要访问元素，才能修改其 className 属性。onmouseover 和 onmouseout 事件处理程序保持不变，因为本例需要与 style 对象示例相同的功能。

最后一个修改是JavaScript代码本身。触发元素上的mouseover事件时，会调用divAdvert_onMouseOver()函数。该函数包含两行代码，而用于 style 对象的相应函数包含 3 行代码：

```
function divAdvert_onMouseOver()
{
    var divAdvert = document.getElementById("divAdvert");
    divAdvert.className = "newStyle";
}
```

第一个语句使用 getElementById()方法获取 元素。然后将 className 属性改成 newStyle。这样，divAdvert 元素将采用新的样式规则，浏览器就会改变其显示方式。

把鼠标指针移出文本时，将触发 mouseout 事件，执行 divAdvert_onMouseOut()函数。该函数与 divAdvert_onMouseOver()函数几乎相同，只是将 className 设置回初始值：

```
function divAdvert_onMouseOut()
{
    var divAdvert = document.getElementById("divAdvert");
    divAdvert.className = "defaultStyle";
}
```

将 className 设置回 defaultStyle，浏览器将按原来的方式显示 元素，即移除了斜体和下划线样式。

注意：

尽管这里没有演示，但 HTML 的 class 属性和 className 属性都可以包含多个 CSS 类名。第 15 章将更详细地讨论类名。

12.3.3 定位和移动内容

改变元素的外观是 DHTML 的一个重要模式，在很多 DHTML 脚本中都能看到这种模式。DHTML 不仅能改变元素在页面上的显示方式，还可以通过 JavaScript 改变元素的位置。

使用 JavaScript 移动内容与使用 style 对象一样简单。使用 position 属性来改变所需位置的类型(即相对或绝对位置)，使用 left 和 top 属性可以定位元素。

```
var divAdvert = document.getElementById("divAdvert");

divAdvert.style.position = "absolute";
```

```
divAdvert.style.left = "100px"; //Set the left position
divAdvert.style.top = "100px"; //Set the right position
```

这段代码首先获取 divAdvert 元素，然后将该元素的 position 设置为绝对位置，把元素移动到距离左边界和上边界各 100 像素的地方。注意，需要在位置值后加上 px。许多浏览器都要求在设置位置值时指定单位，否则，浏览器将无法定位该元素。

注意：

定位元素需要使用绝对或相对位置。

试一试　移动元素

在页面上移动元素与使用 style 对象改变其他样式非常类似。但在页面上移动元素比较常用，本章后面肯定能看到它。因此创建一个页面，允许用户通过表单字段指定元素的位置。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 6</title>
    <style type="text/css">
    #divBox
    {
        position: absolute;
        background-color: silver;
        width: 150px;
        height: 150px;
    }
    input
    {
        width: 100px;
    }
    </style>
    <script type="text/javascript">
    function moveBox() {
        var divBox = document.getElementById("divBox");
        var inputLeft = document.getElementById("inputLeft");
        var inputTop = document.getElementById("inputTop");

        divBox.style.left = parseInt(inputLeft.value) + "px";
        divBox.style.top = parseInt(inputTop.value) + "px";
    }
    </script>
</head>
<body>
    <div id="divBox">
       <form id="formBoxController" onsubmit="moveBox(); return false;"
           action=" ">
           <p>
              Left:
```

```
            <input type="text" id="inputLeft" />
        </p>
        <p>
            Top:
            <input type="text" id="inputTop" />
        </p>
        <p>
            <input type="submit" value="Move The Box" />
        </p>
    </form>
  </div>
</body>
</html>
```

将这个文件保存为 ch12_examp6.htm。在浏览器中加载该页面时，屏幕左上角会显示一个银灰色(silver)的框。框中是包含两个字段和一个按钮的表单，如图 12-13 所示。

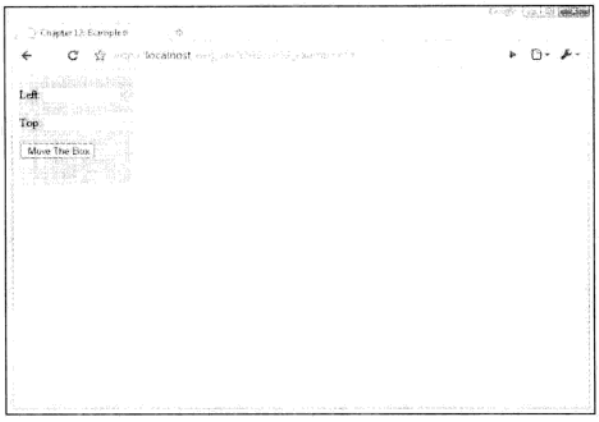

图 12-13

在文本字段中输入数字，并单击 Move The Box 按钮时，该框就移动到指定的坐标位置。图 12-14 显示该框移动到 100,100。

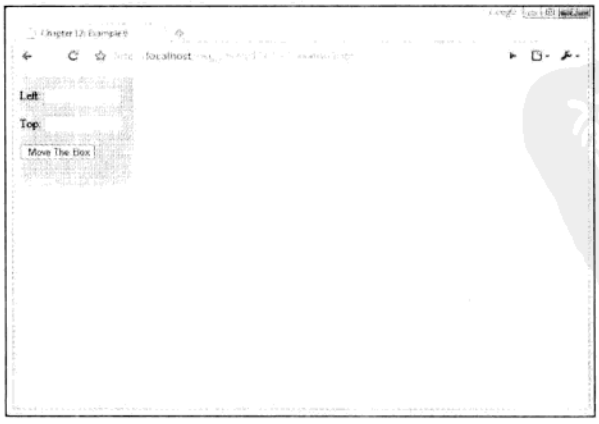

图 12-14

在页面体中定义了一个 id 为 divBox 的 标记。

```
<div id="divBox"></div>
```

在这个元素中是带三个<input/>元素的表单。前两个<input/>是文本框，用户可以在其中输入移动 的 left 和 top 位置，它们的 id 分别为 inputLeft 和 inputTop。第三个<input/>是一个 Submit 按钮。

```
<div id="divBox">
    <form id="formBoxController" onsubmit="moveBox(); return false; " action=" ">
        <p>Left: <input type="text" value="0" id="inputLeft" /></p>
        <p>Top: <input type="text" value="0" id="inputTop" /></p>
        <p><input type="submit" value="Move The Box" /></p>
    </form>
</div>
```

单击 Submit 按钮时，浏览器将触发表单的 submit 事件。单击 Submit 按钮后，浏览器将尝试把数据发送给 Web 服务器。这个尝试将导致浏览器重载页面，通过 DHTML 所做的任何修改都会复位。因此，必须强制浏览器不重载页面，为此，把 submit 事件设置为返回 false。

为了在页面中移动 元素，需要进行定位。本例使用绝对位置定位元素，但也可以使用相对位置定位元素。

```
#divBox
{
    position: absolute;
    background-color: silver;
    width: 150px;
    height: 150px;
}
```

除了位置之外，还指定该框的背景色为银灰色，并将宽度和高度都设置为 150 像素，使之变成正方形。但是，使用这个尺寸，表单中的文本框将超出该框的边界。为了纠正这个问题，为<input/>元素设置一个规则。

```
input
{
    width: 100px;
}
```

将<input/>元素设置为 100 像素宽，就可以将文本框放在方框中。现在就完成了 HTML，设置了其样式。余下的工作是编写 JavaScript，从表单字段中获取值，并将该框移动到表单指定的坐标上。

用来实现该功能的函数是 moveBox()，这也是页面中唯一的函数。

```
function moveBox() {
    var divBox = document.getElementById("divBox"); //Get the box
    var inputLeft = document.getElementById("inputLeft"); //Get one form field
```

```
    var inputTop = document.getElementById("inputTop"); //Get the other one
```

该函数首先获取移动方框所需要的 HTML 元素。先获取 元素本身，再从文本框中获取 left 和 top 位置值，并将这两个位置值分别保存在变量 inputLeft 和 inputTop 中。有了需要的元素后，就可以移动方框了。

```
    divBox.style.left = parseInt(inputLeft.value) + "px";
    divBox.style.top = parseInt(inputTop.value) + "px";
}
```

moveBox()新增的这两行代码移动了方框。第一行代码使用 value 属性来获取 left 文本框中的值，把这个值传送给 parseInt()函数，以保证该值是一个整数。然后给该数字追加 px，以确保所有的浏览器都能正确地定位方框。第二行代码用相同的办法来获取 top 位置值：获取 top 文本框中的值，并传送给 parseInt()，最后加上 px。

可以看出，在页面上移动元素非常简单，是动画效果的一个组成部分。

12.3.4 示例：动态广告

改变元素的外观和位置是 DHTML 中的重要模式，在很多 DHTML 脚本中都包含了这些模式。最有创意的 DHTML 应用是连续改变页面上的内容。使用 DHTML 可以创建各种动画。例如，可以文本元素或图片淡入和淡出页面，创建擦除动画(使元素就像是在页面上被擦掉一样)，或者元素在页面上移动的动画。

动画可以使重要信息更引人注目，易于被浏览者发现，或者让页面显得很酷。使用 DHTML 创建的动画遵循与其他类型的动画同样的原理，即每次都按顺序使元素发生微小的变化，直到动画结束。本质上，任何动画都包含 3 个要素：

(1) 起始状态

(2) 向最终目标运动

(3) 结束状态，停止动画

本节介绍的连续改变一个绝对定位的元素也是如此。首先，使用 CSS 将元素定位在起始位置，然后执行动画，直至到达终点，此时结束动画。

本节将介绍如何创建在起点和终点之间来回移动的动画内容。为此，需要一个重要的信息：内容当前的位置。

1. 确定内容的当前位置

现代浏览器中的 DOM 模型提供了 HTML 元素对象的 offsetTop 和 offsetLeft 属性。这两个属性返回元素相对于父元素的偏移量，其中，offsetTop 表示 top 偏移量，offsetLeft 表示 left 偏移量。这两个属性的返回值都是数值，因此，很容易检查元素在动画中的当前位置。例如：

```
var endPointX = 394;

if (element.offsetLeft < endPointX)
```

```
{
    // Continue animation
}
```

上面的代码指定终点(在本例中是 394)，并将其赋给变量 endPointX。接下来检查元素的 offsetLeft 值当前是否小于终点的相应值。如果是，则继续移动该元素。这个示例将我们带到了移动内容的下一个主题，实现动画效果。

2. 实现动画效果

为了实现动画效果，必须修改 style 对象的 top 和 left 属性，使之快速地递增。在 DHTML 中，可以周期性地执行函数，直到动画结束。为此，需要用到 window 对象的 setTimeout()或 setInterval()方法之一。本例使用 setInterval()方法，以周期性地移动元素。

试一试　创建动画

下面的 HTML 页面把一个元素从页面的右边移动左边。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Moving Content</title>
    <style type="text/css">
    #divAdvert
    {
        position: absolute;
        font: 12px Arial;
        top: 4px;
        left: 0px;
    }
    </style>
    <script type="text/javascript">
    var switchDirection = false;

    function doAnimation() {
        var divAdvert = document.getElementById("divAdvert");
        var currentLeft = divAdvert.offsetLeft;
        var newLocation;

        if (switchDirection == false)
        {
            newLocation = currentLeft + 2;

            if (currentLeft >= 400)
            {
            switchDirection = true;
            }
        }
        else
```

```
        {
            newLocation = currentLeft - 2;

            if (currentLeft <= 0)
            {
            switchDirection = false;
            }
        }

        divAdvert.style.left = newLocation + "px";
    }
    </script>
</head>
<body onload="setInterval(doAnimation, 10)">
    <div id="divAdvert">Here is an advertisement.</div>
</body>
</html>
```

将这个页面保存为 ch12_examp7.htm，在浏览器中加载它。加载页面后，内容应从浏览器窗口的左边开始，向右移动。当该元素移动到距离浏览器左边 400 像素的位置时，将改变方向，开始向左返回。这个动画会连续执行，因此该元素将在两点(0 和 400)之间来回移动。

在页面正文中是一个 元素，其 id 为 divAdvert。所以可以使用 getElementById()方法获取它，这是要连续改变的元素。

```
<div id="divAdvert">Here is an advertisement.</div>
```

这个元素没有 style 属性，因为所有样式信息都在页面顶部的样式表中。在样式表中，定义了该 元素的起点。元素应先从左向右移动，且从浏览器的左边界开始移动。

```
#divAdvert
{
    position: absolute;
    font: 12pt arial;
    top: 4px;
    left: 0px;
}
```

第一个样式声明以绝对方式定位元素。第二个样式声明指定字体为 12pt Arial。接下来的样式声明将元素定位在距离浏览器窗口上边界 4 个像素的位置，这样文本较容易阅读。最后一行使用 left 属性把 divAdvert 元素定位在紧靠着浏览器窗口的左边界。

在脚本块中定义了一个全局变量 switchDirection：

```
var switchDirection = false;
```

这个变量跟踪内容当前移动的方向。如果 switchDirection 为 false，则内容从左向右移动，这是默认值。如果 switchDirection 为 true，则内容从右向左移动。

在脚本块中，接下来是 doAnimation()函数，它实现了动画效果。

```
function doAnimation()
{
    var divAdvert = document.getElementById("divAdvert"); //Get the element
    var currentLeft = divAdvert.offsetLeft; //Get the current left position
    var newLocation; //Will store the new location
```

首先使用 getElementById()方法获取 divAdvert 元素，再获取 offsetLeft 属性，并将其值赋予变量 currentLeft。这个变量用来检查元素的当前位置。接着创建 newLocation 变量，它保存了新的 left 位置，但是在给它赋值之前，需要了解内容的移动方向。

```
if (switchDirection == false)
{
    newLocation = currentLeft + 2;

    if (currentLeft >= 400)
    {
        switchDirection = true;
    }
}
```

首先检查 switchDirection 变量，以判断当前的移动方向。如果该变量是 false，内容将从左向右移动。因此，将 newLocation 设置为内容当前的位置+ 2，即将内容向右移动 2 个像素。

接着检查内容的 left 位置是否达到 400 像素。如果是，则需要切换内容的移动方向。为此，只需将 switchDirection 改为 true。在下次运行 doAnimation()时，内容将开始从右向左移动。

内容从右向左移动的代码与从左向右移动的代码非常类似，但有几个重要区别。

```
else
{
    newLocation = currentLeft - 2;

    if (currentLeft <= 0)
    {
        switchDirection = false;
    }
}
```

第一个区别在于赋给 newLocation 的值，不是给内容的当前位置加上 2，而是减去 2，即把内容向左移动 2 个像素。接着检查 currentLeft 是否小于等于 0。如果是，则从右向左移动已经到达了终点，需要将 switchDirection 设置为 false，再次切换内容移动的方向。

最后，设置内容的新位置：

```
    divAdvert.style.left = newLocation + "px";
}
```

函数的最后一行代码将元素的 left 属性设置为 newLocation 变量的值加上字符串 px。

要运行该动画，需要在<body/>元素中使用 onload 事件处理程序，并使用 window.

setInterval()方法连续执行 doAnimation()。下面的代码每隔 10 毫秒运行一次 doAnimation()。

```
<body onload="setInterval(doAnimation, 10)">
```

以这样的速度，浏览网站的用户更容易看到元素的移动。如果要加快或减慢动画的速度，只需修改 setInterval()的第二个参数，改变 setInterval()函数调用 doAnimation()的频率。

前面介绍了 DOM 结构，它把 HTML 文档表示为树形结构。可以通过 DOM 对象(Node 对象)及其属性，导航 DOM 对象的不同部分，我们还改变了对象的属性，以便改变网页的内容。还有一个重要的 DOM 主题未讨论：事件模型。

12.4 DOM 和事件

20 世纪 90 年代后期的两个主流浏览器是 Internet Explorer 4 和 Netscape 4——第一场浏览器大战。两个浏览器厂商实现了完全不同的 DOM 和事件模型，把 Web 分为两大阵营——仅支持 Netscape 的网站和仅支持 IE 的网站。很少有开发人员选择进行易受挫的跨浏览器开发。

显然，这种森严的壁垒和受挫感催生了一种标准。所以 W3C 引入了 DOM 标准，它演变为 DOM 2 级标准时，包含了一个标准的事件模型。

DOM 事件模型是处理事件的方式，并提供了这些事件的信息，以编写脚本。DOM 事件模型提供了一组规则，以标准的方式确定何种元素生成事件、生成何种类型的事件、以及何时何地触发事件。它引入了一组基本的对象、属性和方法，把某些重要的特性区分开来。

尽管事件模型在尝试标准化，但开发人员仍必须处理多个事件模型。Firefox、Chrome、Safari 和 Opera 等浏览器实现了标准的事件模型，但 Internet Explorer 没有实现，还需要付出更多努力才能建立跨浏览器的、事件驱动的应用程序。但不要着急，尽管有不同的实现方式，但 DOM 和 IE 事件模型共享一些公共属性，许多不共享的属性也很容易转换为其他属性。

本节将学习两个模型。本章的后面将使用这些信息来编写跨浏览器的 DHTML。

12.4.1 DOM 事件处理

DOM 标准描述了一个 Event 对象，该对象提供了触发事件的元素信息，并允许在脚本中获取该元素。如果要在脚本中使用 Event 对象，则必须将它作为参数，传入与事件处理程序关联的函数。

注意：
IE 没有实现 DOM 事件模型，所以本节的代码不能用于 IE。

1. 访问事件对象

第 6 章学习了如何使用 HTML 属性处理事件，但没有学习如何访问 Event 对象，而

Event 对象在许多情况下都非常有用。很容易访问 Event 对象，只需查询触发事件的元素所创建的 Event 对象。例如，在下面的代码中，<p/>元素会引发 dblclick 事件。

```
<p ondblclick="handle(event)">Paragraph</p>

<script type="text/javascript">
function handle(e)
{
    alert(e.type);
}
</script>
```

注意在 ondblclick 属性中，把 event 对象传给函数 handle()。这个 event 变量是特殊的，因为它没有在任何地方定义，只有通过 HTML 属性连接的事件处理程序才能使用这个参数，在触发事件时，它传送了当前事件对象的引用。

如果运行前面的例子，只会显示哪种事件触发了事件处理函数。在上面的例子中，这似乎是不言自明的。但是如果上例包含如下所示的代码，则三个元素都可能触发函数。

```
<p ondblclick="handle(event)">Paragraph</p>
<h1 onclick="handle(event)">Heading 1</h1>
<span onmouseover="handle(event)">Special Text</span>

<script type="text/javascript">
function handle(e)
{
    alert(e.type);
}
</script>
```

这使代码非常有用。通常可以使用较少的事件处理程序来处理任意多个事件，并且可以使用事件属性作为筛选器，以判断会发生哪类事件，哪个 HTML 元素触发了它，以便分别处理每个事件。

在下面的例子中可以看到，可以根据返回的事件类型采取不同的处理过程。

```
<p ondblclick="handle(event)">Paragraph</p>
<h1 onclick="handle(event)">Heading 1</h1>
<span onmouseover="handle(event)">Special Text</span>

<script type="text/javascript">
function handle(e)
{
    if (e.type == "mouseover")
    {
        alert("You moved over the Special Text");
    }
}
</script>
```

这段代码使用 type 属性确定发生了哪类事件。如果用户把鼠标放在元素上，就用一个警告框说明这个情况。

如果使用 HTML 属性指定事件处理程序，访问事件信息就相当简单。而使用 JavaScript 对象的属性指定事件处理程序时，访问事件数据也很简单：在触发事件时，浏览器会自动把事件对象传送给处理函数。考虑下面的代码：

```
<p id="p">Paragraph</p>
<h1 id="h1">Heading 1</h1>
<span id="span">Special Text</span>

<script type="text/javascript">
function handle(e)
{
    if (e.type == "mouseover")
    {
        alert("You moved over the Special Text");
    }
}

document.getElementById("p").ondblclick = handle;
document.getElementById("h1").onclick = handle;
document.getElementById("span").onmouseover = handle;
</script>
```

这段代码略微不同于上一个使用 HTML 属性的示例。元素给定了 id 属性，以便于使用 getElementById()方法在 DOM 中访问它们的对象。为每个元素指定了一个事件处理程序，在触发事件时，就调用 handle()。结果与前面相同，用户会看到一个警告框，说明用户把鼠标指针移到 Special Text 上。

2. 使用事件数据

标准给出了 Event 对象的几个属性，这些属性提供了事件的相关信息：在哪个元素上发生了事件，发生了什么类型的事件，在何时发生等。这些都是 Event 对象提供的数据。表 12-10 列出了规范中指定的属性。

表 12-10

Event 对象的属性	说　明
bubbles	表示是否允许事件冒泡(即控制权从事件目标开始，沿着层次结构从一个元素向上传递给另一个元素)
cancelable	表示是否可以取消事件的默认行为
currentTarget	表示当前处理的事件处理程序的事件目标
eventPhase	表示事件流当前处于哪个阶段
target	该属性表示引发事件的元素；在 DOM 事件模型中，文本节点也可能是事件目标
timestamp	表示事件发生的时间
Type	表示事件的名称

另外，DOM 事件模型还引入了一个 MouseEvent 对象，专门用来处理鼠标引发的事件。这是非常有用的，因为我们可能需要事件的更详细信息，如指针的位置(像素)，或者鼠标来自哪个元素，如表 12-11 所示。

表　12-11

MouseEvent 对象的属性	描　　述
altKey	表示事件发生时，是否按下 Alt 键
button	表示按下鼠标的哪一个按钮
clientX	表示事件发生时，鼠标指针在浏览器窗口中的水平坐标
clientY	表示事件发生时，鼠标指针在浏览器窗口中的垂直坐标
ctrlKey	表示事件发生时，是否按下 Ctrl 键
metaKey	表示事件发生时，是否按下 meta 键
relatedTarget	用于表示第二个事件目标。对于 mouseover 事件，该属性表示鼠标指针退出的元素；对于 mouseout 事件，该属性表示鼠标指针进入的元素
screenX	表示事件发生时，鼠标指针相对于屏幕坐标原点的水平坐标
screenY	表示事件发生时，鼠标指针相对于屏幕坐标原点的垂直坐标
shiftKey	表示事件发生时，是否按下 Shift 键

尽管任何事件都可能创建 Event 对象，但是只有一组选定的事件才能生成 MouseEvent 对象。如果产生了 MouseEvent 对象，就可以访问 Event 和 MouseEvent 对象的属性。对于非鼠标事件，则上表中的 MouseEvent 对象属性均不可用。下列的鼠标事件可以创建 MouseEvent 对象。

- click 事件：当指针位于某个元素或文本上时，单击(按下并释放)鼠标将触发 click 事件。
- mousedown 事件：当指针位于某个元素或文本上时，按下鼠标将触发 mousedown 事件。
- mouseup 事件：当指针位于某个元素或文本上时，释放鼠标键将触发 mouseup 事件。
- mouseover 事件：指针移入某个元素或文本时，将触发 mouseover 事件。
- mousemove 事件：当指针位于某个元素或文本上时，移动指针将触发 mousemove 事件。
- mouseout 事件：指针从某个元素或文本上移出时，将触发 mouseout 事件。

试一试　使用 DOM 事件模型

下面的例子使用了 MouseEvent 对象的一些属性。

打开文本编辑器，输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 8</title>
    <style type="text/css">
    .underline
    {
        color: red;
        text-decoration: underline;
    }
    </style>
    <script type="text/javascript">
    function handleEvent(e)
    {
```

```
        if (e.target.tagName == "P")
        {
            if (e.type == "mouseover")
            {
                e.target.className = "underline";
            }

            if (e.type == "mouseout")
            {
                e.target.className = "";
            }
        }

        if (e.type == "click")
        {
            alert("You clicked the mouse button at the X:"
                + e.clientX + " and Y:" + e.clientY + "coordinates");
        }
    }

    document.onmouseover = handleEvent;
    document.onmouseout = handleEvent;
    document.onclick = handleEvent;
    </script>
</head>
<body>
<p>This is paragraph 1.</p>
<p>This is paragraph 2.</p>
<p>This is paragraph 3.</p>
</body>
</html>
```

保存为 ch12_examp8.htm，并在浏览器中运行。把鼠标移动到一个段落上，段落文本的颜色就会变成红色，并加上了下划线。单击页面的任意位置，会看到如图 12-15 所示的警告框。

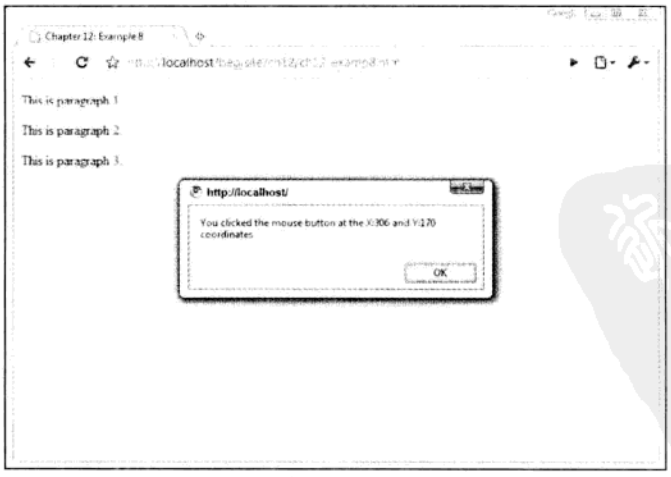

图 12-15

单击 OK 按钮，把指针移动到浏览器中，再次单击，会显示另一个结果。

这个例子与事件处理行为是一致的：浏览器等待事件的到来，每次触发事件时，浏览器将调用相应的函数。之后浏览器继续等待事件的发生，直到用户退出了浏览器或者关闭了该网页。在本例中，为 document 对象上的 mouseover、mouseout 和 click 事件指定了事件处理程序。

```
document.onmouseover = handleEvent;
document.onmouseout = handleEvent;
document.onclick = handleEvent;
One function, handleEvent() handles all three of these events.
```

只要触发了上述事件，都将执行 handleClick()函数，生成一个新的 MouseEvent 对象。记住，MouseEvent 对象允许访问 MouseEvent 对象的属性，以及 Event 对象的属性。本例使用这些属性。

handleClick()函数将接收该 MouseEvent 事件对象，并将它赋给引用 e。

```
function handleEvent(e)
{
    if (e.target.tagName == "P")
    {
```

在该函数中，首先检查事件目标(引发事件的元素)的 tagName 是否是 P。如果该目标是一个段落元素，则需要使用 type 属性查找的下一个信息是事件的类型。

```
        if (e.type == "mouseover")
        {
            e.target.className = "underline";
            }

            if (e.type == "mouseout")
            {
                e.target.className = "";
            }
        }
```

如果事件是 mouseover，就把段落的 CSS 类指定为页面样式表中定义的 underline 类。如果事件类型是 mouseout，则清除元素的 className 属性，把文本恢复为初始样式。只有触发事件的元素是段落元素，才执行这个改变样式的代码。

接着，函数再次检查 type 属性，确定用户是否单击了鼠标。

```
if (e.type == "click")
    {
        alert("You clicked the mouse button at the X: "
            + e.clientX + "and Y: " + e.clientY + "coordinates");
    }
}
```

如果用户在页面的某个地方单击了，就使用 alert()方法在屏幕上显示 mouseevent 对象

的 clientX 和 clientY 属性值。

每次生成一个事件时，都会改写并重建提供给这个函数的 MouseEvent 对象。所以下次单击鼠标或移动指针时，都会创建一个新的 MouseEvent 对象，其中包含鼠标 x 和 y 的位置坐标，和触发事件的元素信息。下面不再讨论 DOM 事件模型，因为并不是所有的浏览器都支持它。例如最流行的浏览器 IE 就不完全支持它。尽管缺乏 DOM 标准的支持，我们仍可以使用 IE 的事件模型获得给定事件的这些有效信息。

12.4.2 IE 中的事件处理

IE 的事件模型自从 1997 年在 IE4 中引入后就没有什么变化，它使用全局的 event 对象(是 window 对象的一个属性)，这个对象存在于每个打开的浏览器窗口中。每次用户引发事件时，浏览器都会更新 event 对象，它提供的信息类似于标准的 DOM Event 对象。

1. 访问 event 对象

由于 event 对象是 window 的一个属性，所以易于访问。

```
<p ondblclick="handle()">Paragraph</p>

<script type="text/javascript">
function handle()
{
    alert(window.event.type);
}
</script>
```

这段代码使用 handle()函数处理<p/>元素的 dblclick 事件。该函数执行时，会获取使 handle()函数执行的事件的类型。由于 event 对象是全局的，所以不需要像 DOM 事件模型那样，把该对象传送给处理函数。还要注意与 window 对象的其他属性一样，不需要在 event 对象的前面加上 window 前缀。

使用对象属性，通过 JavaScript 指定事件处理程序时，这个规则也适用。

```
<p id="p">Paragraph</p>
<h1 id="h1">Heading 1</h1>
<span id="span">Special Text</span>

<script type="text/javascript">
function handle()
{
    if (event.type == "mouseover")
    {
        alert("You moved over the Special Text");
    }
}

document.getElementById("p").ondblclick = handle;
document.getElementById("h1").onclick = handle;
document.getElementById("span").onmouseover = handle;
</script>
```

2. 使用事件数据

可以看出，IE 的 event 对象非常简单，很容易使用，但它提供的属性不同于 DOM 标准的 Event 和 MouseEvent 对象，尽管它们提供的数据类似。

表 12-12 列出了 IE 的 event 对象的一些属性。

表　12-12

event 对象的属性	说　　明
altKey	表示事件发生时，是否按下 Alt 键
button	表示按下鼠标的哪一个按钮
cancelBubble	获取或设置当前事件是否应沿着事件处理程序的层次结构向上冒泡
clientX	表示事件发生时，鼠标指针在浏览器窗口中的水平坐标
clientY	表示事件发生时，鼠标指针在浏览器窗口中的垂直坐标
ctrlKey	表示事件发生时，是否按下 Ctrl 键
fromElement	获取鼠标指针退出的元素对象
keyCode	获取与引发事件的键相关的 Unicode 键码
screenX	表示事件发生时，在浏览器窗口中鼠标指针相对于屏幕坐标原点的水平坐标
screenY	表示事件发生时，在浏览器窗口中鼠标指针相对于屏幕坐标原点的垂直坐标
shiftKey	表示事件发生时，是否按下 Shift 键
srcElement	获取引发事件的元素对象
toElement	获取鼠标指针进入的元素对象
type	获取事件的名称

前面编写页面的示例 8 利用了 DOM 事件模型，下面改写它，使之可以在 IE 中运行。

试一试　**使用 IE 事件模型**

打开文本编辑器，输入如下代码。可以从示例 8 中复制并粘贴页面正文和样式表中的元素。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 9</title>
    <style type="text/css">
    .underline
    {
        color: red;
        text-decoration: underline;
    }
    </style>
```

```
    <script type="text/javascript">
    function handleEvent()
    {
        if (event.srcElement.tagName == "P")
        {
            if (event.type == "mouseover")
            {
                event.srcElement.className = "underline";
            }
            if (event.type == "mouseout")
            {
                event.srcElement.className = "";
            }
        }

        if (event.type == "click")
        {
            alert("You clicked the mouse button at the X: "
                + event.clientX + " and Y:" + event.clientY + " coordinates");
        }
    }

    document.onmouseover = handleEvent;
    document.onmouseout = handleEvent;
    document.onclick = handleEvent;
    </script>
</head>
<body>
<p>This is paragraph 1.</p>
<p>This is paragraph 2.</p>
<p>This is paragraph 3.</p>
</body>
</html>
```

保存为 ch12_examp9.htm，并加载到 IE 中。其外观和操作方式与示例 8 类似：把鼠标指针移动到某段落文本上时，该文本会改为红色，并加上下划线。鼠标指针退出该段落时，文本会恢复原来的状态。单击鼠标，会显示一个警告框，列出单击鼠标时鼠标指针的坐标。

这段代码使用 handleEvent()函数来处理 document 对象上的 mouseover、mouseout 和 click 事件。

```
document.onmouseover = handleEvent;
document.onmouseout = handleEvent;
document.onclick = handleEvent;
```

引发某个事件时，浏览器会更新 event 对象，调用 handleClick()函数。

```
function handleEvent()
{
    if (event.srcElement.tagName == "P")
    {
```

首先检查源元素的 tagName 属性是否为 P，如果是，就使用 type 属性确定事件的类型。

```
    if (event.type == "mouseover")
    {
        event.srcElement.className = "underline";
    }

    if (event.type == "mouseout")
    {
        event.srcElement.className = "";
    }
}
```

对于 mouseover 事件，把段落的 CSS 类改为 underline。如果事件类型是 mouseout，就把元素的 className 属性设置为空字符串——把文本恢复为原来的样式。

如果单击了鼠标按钮，下一段代码就显示鼠标指针的位置。

```
    if (event.type == "click")
    {
        alert("You clicked the mouse button at the X: "
            + event.clientX + " and Y: " + event.clientY + "coordinates");
    }
}
```

比较示例 8 和 9，注意它们的主要区别是事件信息的访问方式，以及获取引发事件的元素的方式。标准 DOM 事件模型和 IE 的事件模型在其他方面是相同的。

下一节学习如何处理两个事件模型之间的根本区别，并编写跨浏览器的 DHTML 代码。

12.5　编写跨浏览器的 DHTML

前面编写了同一个 DHTML 脚本的两个版本，一个用于 IE，一个用于支持标准 DOM 事件模型的浏览器(Firefox、Safari、Chrome 和 Opera)。实际上，创建网站的几个版本并不是最佳实践方式，编写跨浏览器的网页版本要容易得多。本节将使用 DOM、标准 DOM 事件模型和 IE 事件模型编写一个跨浏览器的 DHTML 脚本。这个脚本包含一个选项卡控件，其中有 3 个选项卡。单击任意选项卡会把内容动态添加到网页上。这是一个非常粗略、不完整的选项卡控件。本章的一个习题将要求读者给它添加更多功能。

试一试　**一个粗略的跨浏览器选项卡**

打开文本编辑器，输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Example 10</title>
```

```
<style type="text/css">
.tabStrip
{
    background-color: #E4E2D5;
    padding: 3px;
    height: 22px;
}

.tabStrip div
{
    float: left;
    font: 14px arial;
    cursor: pointer;
}

.tabStrip-tab
{
    padding: 3px;
}

.tabStrip-tab-hover
{
    border: 1px solid #316AC5;
    background-color: #C1D2EE;
    padding: 2px;
}

.tabStrip-tab-click
{
    border: 1px solid #facc5a;
    background-color: #f9e391;
    padding: 2px;
}
</style>
<script type="text/javascript">
function handleEvent(e)
{
    var eSrc;

    if (window.event)
    {
        e = window.event;
        eSrc = e.srcElement;
    }
    else
    {
        eSrc = e.target;
    }

    if (e.type == "mouseover")
    {
        if (eSrc.className == "tabStrip-tab")
```

```
            {
                eSrc.className = "tabStrip-tab-hover";
            }
        }

        if (e.type == "mouseout")
        {
            if (eSrc.className == "tabStrip-tab-hover")
            {
                eSrc.className = "tabStrip-tab";
            }
        }

        if (e.type == "click")
        {
            if (eSrc.className == "tabStrip-tab-hover")
            {
                eSrc.className = "tabStrip-tab-click";
                var num = eSrc.id.substr(eSrc.id.lastIndexOf("-") + 1);

                showDescription(num);
            }
        }

    }

    function showDescription(num)
    {
        var descContainer = document.getElementById("descContainer");

        var div = document.createElement("div");
        var text = document.createTextNode("Description for Tab " + num);

        div.appendChild(text);
        descContainer.appendChild(div);
    }

    document.onclick = handleEvent;
    document.onmouseover = handleEvent;
    document.onmouseout = handleEvent;
    </script>
</head>
<body>
    <div class="tabStrip">
        <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
        <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
        <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
    </div>
    <div id="descContainer"></div>
</body>
</html>
```

把这个文件保存为 ch12_examp10.htm。在多个浏览器中打开它，页面的外观和操作方

式在所有的浏览器中都相同。把鼠标指针移动到一个选项卡上，其样式就改为蓝色背景，且带有深蓝色边框。单击一个选项卡时，其样式就再次改为浅橙色背景，且带有深橙色边框，并把文本添加到页面上。例如，单击选项卡 3，会把文本 Description for Tab 3 添加到页面上。

下面看看页面正文中的 HTML。先看看其样式。

```
<div class="tabStrip">
    <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
    <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
    <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
</div>
<div id="descContainer"></div>
```

第一个 元素的 CCS 类是 tabStrip，它包含的 3 个 元素表示 3 个选项卡。每个选项卡 元素都指定了其 id 属性值，其 CSS 类是 tabStrip-tab。

选项卡控件 元素有一个 id 值为 descContainer 的同级 元素。该同级元素目前不包含任何子元素，也没有关联的 CSS 类。

在本例中，选项卡控件设置为可见，且通过灰色背景与页面的其余内容分隔开。

```
.tabStrip
{
    background-color: #E4E2D5;
    padding: 3px;
    height: 22px;
}
```

高度设置为 28 像素(高度+顶部空隙和底部空隙)。这个高度和空隙使选项卡 元素在选项卡控件中垂直居中。

选项卡用几个 CSS 规则定义了它们在浏览器中的显示方式，因为它们有 3 个状态：正常、悬停和单击。尽管有这 3 个状态，它们仍是选项卡，所以拥有一些共同的可视化特性。第一条规则指定了这些共享属性。

```
.tabStrip div
{
    float: left;
    font: 14px arial;
    cursor: pointer;
}
```

选择器告诉浏览器，把这些属性应用于选项卡控件中的所有 元素。这些元素设置为浮于左边，使它们具备内嵌外观(是块元素，默认显示在新的一行上)。

下一个规则是 tabStrip-tab 类，定义了正常状态。

```
.tabStrip-tab
{
    padding: 3px;
}
```

这个规则给元素的所有边都添加了 3 个像素的空隙。接着是悬停状态，由 tabStrip-tab-hover 类定义：

```
.tabStrip-tab-hover
{
    border: 1px solid #316AC5;
    background-color: #C1D2EE;
    padding: 2px;
}
```

这个规则把空隙减为 2 个像素，增加了一个 1 像素宽的边框，并把背景色改为浅蓝色。边框与空隙类似，会使元素变大。添加边框的同时减小空隙，会使处于悬停状态的元素与正常状态有相同的高度和宽度。

最后一个规则声明了 tabStrip-tab-click 类：

```
.tabStrip-tab-click
{
    border: 1px solid #facc5a;
    background-color: #f9e391;
    padding: 2px;
}
```

这个类类似于 hover 类，唯一的区别是它使用暗橙色的边框颜色和浅橙色的背景色。

下面来分析执行操作的 JavaScript 代码。代码包含一个 handleEvent()函数，该函数指定为 document 对象的 onmouseover、onmouseout 和 onclick 事件处理程序。

```
document.onclick = handleEvent;
document.onmouseover = handleEvent;
document.onmouseout = handleEvent;
```

函数首先声明一个变量 eSrc(表示事件源，只要这个变量名是有意义的，它使用什么名称都不重要)。这个变量包含 IE 和实现了 DOM 事件模型的浏览器的 srcElement 和 target 属性。

```
function handleEvent(e)
{
    var eSrc;

    if (window.event)
    {
        e = window.event;
        eSrc = e.srcElement;
    }
    else
    {
        eSrc = e.target;
    }
```

声明变量后，再检查 window.event 对象，以便确定浏览器实现了哪个事件模型。如果

存在 window.event，浏览器就是 IE，于是把 e 变量设置为包含对 window.event 对象的引用。注意 e 定义为函数的一个参数，因为 IE 不把参数传送给事件处理程序，所以没有定义事件处理程序的参数。把 e 设置为对 window.event 的引用，就可以很方便地使用 IE 事件对象和 DOM 的 Event 和 MouseEvent 对象共享的属性(例如后面使用的 type 属性)。接着把 srcElement 对象赋予 eSrc 变量。如果浏览器不是 IE，就把 DOM Event 对象的 target 属性赋予 eSrc。无论查看这个页面的浏览器是什么，eSrc 变量都包含了引发事件的元素引用。

现在需要确定事件的类型，并对 DOM 进行相应的修改。首先检查 mouseover 事件。

```
if (e.type == "mouseover")
{
    if (eSrc.className == "tabStrip-tab")
    {
        eSrc.className = "tabStrip-tab-hover";
    }
}
```

如果引发事件的元素的类名是 tabStrip-tab，该元素就是处于正常状态下的选项卡，于是把该元素的 className 属性改为 tabStrip-tab-hover。这样，选项卡就处于悬停状态。

如果引发了 mouseout 事件，则还需要修改 DOM。

```
if (e.type == "mouseout")
{
    if (eSrc.className == "tabStrip-tab-hover")
    {
        eSrc.className = "tabStrip-tab";
    }
}
```

只有鼠标指针退出的选项卡处于悬停状态下，这段代码才把选项卡的 className 属性改为 tabStrip-tab(正常状态)。

最后一个需要检查的事件是 click，代码如下：

```
if (e.type == "click")
{
    if (eSrc.className == "tabStrip-tab-hover")
    {
        eSrc.className = "tabStrip-tab-click";
```

这段代码把选项卡元素的 className 改为 tabStrip-tab-click，使之处于单击状态。接着需要在页面上添加选项卡的描述。这个过程首先从 元素的 id 属性中获取选项卡的号码，注意选项卡元素的 id 值采用 tabStrip-tab-1 格式。所以只需获取选项卡的编号。为此需要使用 substr()方法。

```
        var num = eSrc.id.substr(eSrc.id.lastIndexOf("-") + 1);

        showDescription(num);
    }
}
}
```

这段代码在 id 字符串中提取最后一个“-”字符的索引，给它加上 1，再把该值传送给 substr()方法。如果没有给 lastIndexOf()的返回值加上 1，substr()的结果就是-1。有了选项卡编号，就把它传送给 showDescription()函数。

```
function showDescription(num)
{
    var descContainer = document.getElementById("descContainer");
```

选项卡的描述添加到 id 为 descContainer 的 元素上，如这段代码所示，首先使用 getElementById()方法获取该元素。

描述是这个函数动态创建的，所以现在需要创建要添加到 descCounter 元素中的 DOM 节点。首先，使用 document 对象的 createElement()方法创建一个 元素。

```
    var div = document.createElement("div");
```

现在创建一个文本节点，以包含该选项卡的描述。在这个例子中，描述很简单，只包含选项卡的号码。

```
    var text = document.createTextNode("Description for Tab " + num);
```

最后把文本节点添加到新建的 元素中，再把该元素追加到 descContainer 中。使用 Node 对象的 appendChild()方法执行这两个操作。

```
    div.appendChild(text);
    descContainer.appendChild(div);
}
```

JavaScript 不仅用于 HTML，有时还可以用 JavaScript 打开 XML 文件，并读取它。由于 HTML 和 XML 的类似性(它们都是结构化的文档)，所以也可以使用 DOM 加载和读取 XML 文档。

12.6　JavaScript 与 XML

W3C 开发 XML 的目的是描述数据，而不是以特定的格式显示信息(这是 HTML 的作用)。XML 没有什么特别，它只是一些纯文本，加上一些位于尖括号中的 XML 标记。任何可以处理纯文本的软件都可以创建和编辑 XML。

XML 是一种以数据为中心的语言。它不仅包含数据，还使用具有语义的元素名称来描述数据。文档的结构也在描述中起一定的作用。与 HTML 不同的是，XML 并不是一种格式化语言，实际上，具有良好结构的 XML 文档是不包含任何格式化元素的。通常这个概念称为“内容与样式的分离”，这正是 XML 的成功之处，它使 XML 语言既简洁又易用。

例如可以把 XML 用作数据存储，就像数据库那样。实际上，XML 非常适用于处理庞大而复杂的文档，因为其中的数据是结构化的，我们可以使用自己的元素描述元素包含的数据，来设计结构，并实现它。定义 XML 文档中使用的结构和元素，使 XML 成为一种“自描述”语言。也就是说，元素描述它们包含的数据，而结构描述了数据之间的关系。

另一种使用 XML 的方式是从远程服务器上获取数据。这种方式最著名的应用是 Web 内容聚合的 RSS 和 Atom 格式。XML 文档和其他类似的文档，包含了对任何用户都可用的信息。Web 站点或程序可以连接到远程服务器上，下载一份 XML 文档的副本，以需要的方式使用这些信息。

XML 的第三个特别有效的应用是在不兼容的系统之间传递数据。XML 文档是纯文本文档，因此所有的操作系统都可以读写 XML 文档，唯一的要求是以一个能理解 XML 语言及文档结构的应用程序。例如，Microsoft 最近发布了 Microsoft Office Open XML 的详细规范，该文件格式用在 Microsoft Office 2007 中。这类文件实际上是 Zip 文件。因此，任何可以读取 Zip 文件中的 XML 文件的程序都显示其中的数据，无论它们是在 Windows、Mac OS X、任何版本的 Linux 还是其他操作系统下编写。

12.6.1 使用 JavaScript 操作 XML

如前所述，可以使用 DOM 加载、读取和操作 XML 数据，因此本章前面的章节已经介绍了在 Web 浏览器中操作 XML 所需的大多数知识。

第一个任务是读取 XML 文档，大多数跨浏览器问题都出在这里，因为 IE 不遵循 DOM 标准，而 Firefox、Safari、Chrome 和 Opera 则遵循该标准。但是，一旦加载了 XML 文档，浏览器之间的差异就变小了，尽管如此，Microsoft 还是在 XML 实现方案中添加了许多有用(但不是标准)的扩展。

12.6.2 在 IE 中获取 XML 文件

IE 依靠 ActiveXObject()对象和 MSXML 库来获取和打开 XML 文档。在脚本中可以使用大量的 ActiveX 对象。要创建 ActiveX 对象，只需调用 ActiveXObject()构造函数，并传入一个包含 ActiveX 对象版本的字符串：

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.load("myfile.xml");
```

这段代码使用版本字符串 Microsoft.XMLDOM，创建了一个 XML DOM 对象，以加载和操作 XML 文档。在创建了 XML DOM 对象后，就可以使用 load()方法来加载 XML 文档。上面的代码加载了一个假想文件 myfile.xml。

Microsoft MSXML 库存在多个版本，每个新版本都提供了比前一个版本更多的特性和更好的性能。但是，在使用这些版本之前，用户必须在计算机上安装它们，版本选择的代码会变得比较复杂。因此，Microsoft 建议只需检查两个版本的 MSXML，其版本字符串是：

- Msxml2.DOMDocument.6.0
- Msxml2.DOMDocument.3.0

在创建 XML DOM 对象时，我们总是希望使用最新版本的 MSXML 库，此时可以使用下面的函数。

```
function createDocument()
{
```

```
    var xmlDoc;

    if (window.ActiveXObject)
    {
        var versions =
        [
            "Msxml2.DOMDocument.6.0",
            "Msxml2.DOMDocument.3.0"
        ];

        for (var i = 0; i < versions.length; i++)
        {
            try
            {
                xmlDoc = new ActiveXObject(versions[i]);
                return xmlDoc;
            }
            catch (error)
            {
                //do nothing here
            }
        }
    }

    return null;
}
```

这段代码定义了 createDocument()函数。该函数的第一行定义了 xmlDoc 变量。这是创建 XML DOM 时使用的一个临时变量。下一行代码是一个 if 语句，它检查是否存在 window.Active XObject，以判断浏览器是否是 IE。如果条件为 true，则创建一个 versions 数组，并且把两个 MSXML 版本添加为该数组的元素。

```
var versions =
[
    "Msxml2.DOMDocument.6.0",
    "Msxml2.DOMDocument.3.0"
];
```

添加顺序是非常重要的。我们总是希望先检测最新版本，所以把最新版本的字符串放在索引为 0 的元素中。

接下来是一个 for 循环，用于遍历 versions 数组的元素。在该循环中是一个 try...catch 语句。

```
for (var i = 0; i < versions.length; i++)
{
   try
   {
       xmlDoc = new ActiveXObject(versions[i]);
       return xmlDoc;
   }
```

```
    catch (error)
    {
        //do nothing here
    }
}
```

在 try 块中，如果 ActiveXObject 对象的创建失败，则代码的执行流转移到 catch 块。而在 catch 块中不做任何处理：循环迭代 versions 的下一个索引，并尝试使用另一个 MSXML 版本字符串来创建另一个 ActiveXObject 对象。如果每次尝试都失败了，则退出循环，并返回 null。createDocument()函数的用法如下：

```
var xmlDoc = createDocument();
```

使用这个函数，很容易创建最新版本的 MSXML XML DOM 对象。

在实际操作 XML 文档之前，先确保该文档已经完全加载到客户端的浏览器缓存中。否则，每次浏览页面时，只要脚本在 XML 文件下载完成之前执行，都可能抛出 JavaScript 错误。幸运的是，可以检测 XML 文档的当前下载状态。

async 属性表示浏览器继续下载页面其余的部分之前，是否等待指定的 XML 文档下载完。这个属性表示“异步(asynchronous)”，默认设置为 true，表示浏览器无须等待 XML 文档完全下载，即可呈现其后的内容。将这个属性设置为 false，表示浏览器先加载文件，再加载页面的其余部分。

```
var xmlDoc = createDocument();
xmlDoc.async = false; //Download XML file first, then load rest of page.
xmlDoc.load("myfile.xml");
```

async 属性的简单性并不完美。把它设置为 false 时，IE 将暂停页面的执行，直到接收完指定的 XML 文件为止。如果浏览器在连接网络或者下载文件时出了故障，页面将被一直挂起。此时可以使用 onreadystatechange 事件处理程序和 readyState 属性加以处理(只要 async 属性是 true)。

IE 的 XML 对象和很多 HTML 对象都有 readyState 属性，用于返回对象当前的加载状态。表 12-13 列出了 readyState 属性可能的值：

表 12-13

readyState 属性的返回值	说　明
1	对象正在初始化，尚无任何可读的数据，正在载入数据(loading)
2	数据正在载入对象并解析(loaded)
3	部分对象的数据已经读取并解析，因此对象模型是可用的。但是仍无法读取完整的对象数据(interactive)
4	对象已经载入，内容已经解析完毕(completed)

这里关注的是最后一个属性值 4，它表示对象加载完毕。要使用 readyState 属性，只需指定一个函数来处理 readystatechange 事件，每次 readyState 改变时，都会触发该事件。

```
function xmlDoc_readyStateChange()
{
    //Check for the readyState. If it's 4, it's loaded!
    if (xmlDoc.readyState == 4)
    {
        alert("XML file loaded!");
    }
}

var xmlDoc = createDocument();
xmlDoc.onreadystatechange = xmlDoc_readyStateChange;

xmlDoc.load("myfile.xml");
```

这段代码首先创建 xmlDoc_readyStateChange()函数，用于处理 readystatechange 事件。在该函数中，检查 readyState 属性的值是否为 4。如果是，则表示 XML 文档加载完毕，并显示警告文本“XML file loaded!”。接着创建 XML DOM 对象，然后将 xmlDoc_ready StateChange()函数设置为 onreadystatechange 事件处理程序。最后一行代码则加载 myfile.xml. 文件。

12.6.3　在 Firefox 和 Opera 浏览器中获取 XML 文件

在 Firefox 和 Opera 中加载 XML 文档与 IE 略有不同，因为这些浏览器更加强调将标准作为中心。创建 XML DOM 不需要像 IE 那样使用插件，DOM 是浏览器和 JavaScript 实现的一部分。

```
var xmlDoc = document.implementation.createDocument("","",null);
xmlDoc.load("myfile.xml");
```

注意：

Safari 和 Chrome 不支持以本节介绍的方式把 XML 文件加载到 DOM 对象中，而必须使用 XMLHttpRequest 对象，从服务器上请求 XML 文档，这个对象参见第 14 章。

这段代码中，使用 document.implementation 对象的 createDocument()方法创建了一个空的 DOM 对象。之后，使用 load()方法来加载 XML 文档。Firefox 和 Opera 都支持 load()方法。

注意：

Safari 和 Chrome 都支持 CreatDocament()方法，都不支持 load()方法。

与 IE 类似，Firefox 和 Opera 都支持 async 属性，允许异步或同步加载文件。这两个浏览器的同步加载方式与 IE 相同，但异步载入文件时，情况有所不同。

在异步方式下， Firefox 和 Opera 使用与 IE 不同的实现方式检测 XML 文件的加载状态。实际上，这两个浏览器都不能使用 readyState 属性来检测文档的加载状态，而是在文件加载完，且 DOM 对象可用时，执行 onload 事件处理程序。

```
function xmlDoc_load()
```

```
{
    alert("XML is loaded! ");
}

var xmlDoc = document.implementation.createDocument("","",null);
xmlDoc.onload = xmlDoc_load;
xmlDoc.load("myfile.xml");
```

这段代码按异步方式加载假想文件 myfile.xml。加载过程完成时，将触发 load 事件，并调用 xmlDoc_load()，将文本 XML is loaded!显示给用户。

12.6.4 跨浏览器读取 XML 文件

可以看出，不同的浏览器创建 XML DOM 对象的方法是不同的，因此需要找到一种跨浏览器的解决方案。通过对象检测技术，很容易判断当前使用的是何种浏览器。实际上，只需修改 createDocument()函数，包含对 Firefox 和 Opera 的支持即可。代码如下所示：

```
function createDocument()
{
    var xmlDoc;

    if (window.ActiveXObject)
    {
        var versions =
        [
            "Msxml2.DOMDocument.6.0",
            "Msxml2.DOMDocument.3.0"
        ];

        for (var i = 0; i < versions.length; i++)
        {
            try
            {
                xmlDoc = new ActiveXObject(versions[i]);
                return xmlDoc;
            }
            catch (error)
            {
                //do nothing here
            }
        }
    }
    else if (document.implementation && document.implementation.createDocument)
    {
        xmlDoc = document.implementation.createDocument("","",null);
        return xmlDoc;
    }
    return null;
}
```

突出显示的代码就是添加到函数中的新代码，这部分代码首先检查 implementation 对象和 implementation.createDocument()方法是否存在，如果存在，就使用 document.implementation.createDocument()方法为支持 DOM 的浏览器创建 XML DOM 对象，并将该 DOM 对象返回给调用方。这个函数的用法同前，但它现在可用于 IE、Firefox 和 Opera。

```
var xmlDoc = createDocument();
xmlDoc.async = false;
xmlDoc.load("myfile.xml");
```

12.6.5　示例：显示每日信息

知道如何加载 XML 文档后，下面创建第一个支持 XML 的 JavaScript 应用程序，显示每日信息。

首先需要使用如下所示的简单 XML 文件。程序将读取该文件，并使用 DHTML 显示每日信息。该 XML 文件保存为 motd.xml。

```
<?xml version="1.0"?>

<messages>
  <daily>Today is Sunday.</daily>
  <daily>Today is Monday.</daily>
  <daily>Today is Tuesday.</daily>
  <daily>Today is Wednesday.</daily>
  <daily>Today is Thursday.</daily>
  <daily>Today is Friday.</daily>
  <daily>Today is Saturday.</daily>
</messages>
```

可以看出，这个基本的 XML 文件为一周中的每一天填充了不同的消息。

接着是 HTML 页面，其中把 createDocument()函数添加到脚本块中：

```
<html>
<head>
    <title>Message of the Day</title>
    <script type="text/javascript">
    function createDocument()
    {
        var xmlDoc;

        if (window.ActiveXObject)
        {
            var versions =
            [
                "Msxml2.DOMDocument.6.0",
                "Msxml2.DOMDocument.3.0"
            ];

            for (var i = 0; i < versions.length; i++)
            {
```

```
                try
                {
                    xmlDoc = new ActiveXObject(versions[i]);
                    return xmlDoc;
                }
                catch (error)
                {
                    //do nothing here
                }
            }
        }
        else if (document.implementation
                 && document.implementation.createDocument)
        {
            xmlDoc = document.implementation.createDocument("", "",null);
            return xmlDoc;
        }

        return null;
    }

    //More code to come
    </script>
</head>
```

在探讨页面正文之前，还需要在页面顶部添加另一个函数 getDailyMessage()。该函数用来获取并返回当日的提示信息。在 createDocument()函数定义的下面添加它：

```
function getDailyMessage()
{
    var messages = xmlDoc.getElementsByTagName("daily");
    var dateobj = new Date();
    var today = dateobj.getDay();

    return messages[today].firstChild.nodeValue;
}
```

这个函数首先使用 getElementsByTagName()方法来获取<daily/>元素，返回一个包含所有<daily/>元素的节点列表。接下来，确定当前日期是星期几。为此，先创建一个 Date 对象，并使用其 getDay()方法，获得一个 0～6 之间的数字，其中 0 表示星期日，1 表示星期一，以此类推。把这个数字保存在变量 today 中。最后使用该变量作为 messages 节点列表的索引，选择正确的<daily/>元素，并获取其文本。

注意，getDailyMessage()中的 xmlDoc 未在 HTML 文档的头部声明。在 createDocument()中也使用了 xmlDoc，但该变量在 createDocument()内部声明了。实际上，我们在 HTML 页面正文中定义 xmlDoc 全局变量：

```
<body>
<div id="messageContainer"></div>

<script type="text/javascript">
```

```
    var xmlDoc = createDocument();
    xmlDoc.async = false;
    xmlDoc.load("motd.xml");

    document.getElementById("messageContainer").innerHTML = getDailyMessage();
</script>
</body>
</html>
```

页面正文中的第一个 HTML 元素是 id 为 messageContainer 的 元素，用于显示每日信息。

在 元素之后是页面中的最后一个<script/>元素。在这个代码块中，创建了一个 DOM 文档，并保存在全局变量 xmlDoc 中。然后以同步方式加载 motd.xml 文件，并调用 getDailyMessage()，将消息容器的 innerHTML 设置为每日信息。

试一试　使用表格显示宠物狗数据

本例将编写一个应用程序，使用 XML 文档来保存一位作者的宠物狗数据。打开文本编辑器，输入下面的代码：

```
<html>
<head>
    <title>Chapter 12: Example 11</title>
    <script type="text/javascript">
    function createDocument()
    {
        var xmlDoc;

        if (window.ActiveXObject)
        {
            var versions =
            [
                "Msxml2.DOMDocument.6.0",
                "Msxml2.DOMDocument.3.0"
            ];

            for (var i = 0; i < versions.length; i++)
            {
                try
                {
                    xmlDoc = new ActiveXObject(versions[i]);
                    return xmlDoc;
                }
                catch (error)
                {
                    //do nothing here
                }
            }
        }
        else if (document.implementation &&
```

```
            document.implementation.createDocument)
    {
        xmlDoc = document.implementation.createDocument("", "",null);
        return xmlDoc;
    }

    return null;
}

var xmlDocument = createDocument();
xmlDocument.load("ch12_examp11.xml");

function displayDogs()
{
    var dogNodes = xmlDocument.getElementsByTagName("dog");

    var table = document.createElement("table");
    table.setAttribute("cellPadding",5); //Give the table some cell padding.
    table.setAttribute("width", "100%");
    table.setAttribute("border", "1");

    var tableHeader = document.createElement("thead");
    var tableRow = document.createElement("tr");

    for (var i = 0; i < dogNodes[0].childNodes.length; i++)
    {
        var currentNode = dogNodes[0].childNodes[i];

        if (currentNode.nodeType == 1)
        {
            var tableHeaderCell = document.createElement("th");

            var textData = document.createTextNode(currentNode.nodeName);

            tableHeaderCell.appendChild(textData);

            tableRow.appendChild(tableHeaderCell);
        }
    }

    tableHeader.appendChild(tableRow);

    table.appendChild(tableHeader);

    var tableBody = document.createElement("tbody");

    for (var i = 0; i < dogNodes.length; i++)
    {
        var tableRow = document.createElement("tr");

        for (var j = 0; j < dogNodes[i].childNodes.length; j++)
        {
            var currentNode = dogNodes[i].childNodes[j];
            if (currentNode.nodeType == 1)
            {
                var tableDataCell = document.createElement("td");
```

```
                    var textData = document.createTextNode
                    (
                        currentNode.firstChild.nodeValue
                    );

                    tableDataCell.appendChild(textData);

                    tableRow.appendChild(tableDataCell);
                }
            }

            tableBody.appendChild(tableRow);
        }

        table.appendChild(tableBody);

        document.body.appendChild(table);

    }
    </script>
</head>
<body>
    <a href="javascript: displayDogs();">Display Dogs</a>
</body>
</html>
```

保存为 ch12_examp11.htm，现在在另一个文件中输入如下代码，保存为 ch12_examp11.xml。

```
<?xml version="1.0" encoding="iso-8859-1"?>

<myDogs>
    <dog>
        <name>Morgan</name>
        <breed>Labrador Retriever</breed>
        <age>4 years</age>
        <fullBlood>yes</fullBlood>
        <color>chocolate</color>
    </dog>
    <dog>
        <name>Molly</name>
        <breed>Labrador Retriever</breed>
        <age>12 years</age>
        <fullBlood>yes</fullBlood>
        <color>yellow</color>
    </dog>
    <dog>
        <name>Madison</name>
        <breed>Labrador Retriever</breed>
        <age>10 years</age>
        <fullBlood>yes</fullBlood>
        <color>chocolate</color>
    </dog>
</myDogs>
```

在浏览器中打开 HTML 页面时，该页面仅包含一个链接。单击该链接，页面如图 12-16 所示。

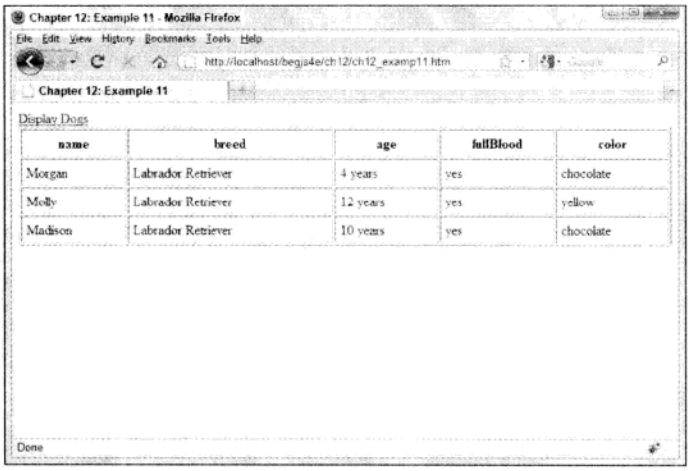

图 12-16

上面的代码首先创建一个 DOM 对象，并加载 XML 文档。

```
var xmlDocument = createDocument();
xmlDocument.load("ch12_examp11.xml");
```

这个页面的主要部分是接下来的 displayDogs()函数，其功能是建立一个表格，用 XML 文件的信息来填充它。

```
function displayDogs()
{
        var dogNodes = xmlDocument.getElementsByTagName("dog");
        var table = document.createElement("table");
        table.setAttribute("cellPadding",5); //Give the table some cell padding.
        table.setAttribute("width", "100%");
        table.setAttribute("border", "1");
```

这个函数首先使用 getElementsByTagName()方法来获取<dog/>元素，并将得到的节点列表赋给 dogNodes。接下来使用 document.createElement()方法创建一个<table/>元素，并使用 setAttribute()设置了其 cellPadding、width 和 border 属性。

接下来创建表格标题和标题单元格。对于每列的标题，使用<dog/>元素的子元素名(name、breed 和 age 等)。

```
var tableHeader = document.createElement("thead");
var tableRow = document.createElement("tr");

for (var i = 0; i < dogNodes[0].childNodes.length; i++)
{
    var currentNode = dogNodes[0].childNodes[i];

    //Loop code here.
}
```

```
tableHeader.appendChild(tableRow);
table.appendChild(tableHeader);
```

上面代码的前几行用于创建<thead/>和<tr/>元素。然后，代码遍历第一个<dog/>元素的子节点(后面将介绍更多相关内容)。完成循环后，将<tr/>元素添加到表格标题，并将标题添加到表格中。下面来分析循环代码：

```
for (var i = 0; i < dogNodes[0].childNodes.length; i++)
{
    var currentNode = dogNodes[0].childNodes[i];
    if (currentNode.nodeType == 1)
    {
        var tableHeaderCell = document.createElement("th");
        var textData = document.createTextNode(currentNode.nodeName);

        tableHeaderCell.appendChild(textData);
        tableRow.appendChild(tableHeaderCell);
    }
}
```

这里的目标是使用元素的名称作为列标题。但是，遍历<dog/>元素的所有子节点时，支持 DOM 的浏览器把元素之间的空白视为一个子节点。为了解决这个问题，只需使用 nodeType 属性检查当前节点的类型。如果该属性等于 1，则子节点是一个元素。接着创建一个<th/>元素，以及一个包含当前节点的 nodeName 的文本节点，并把文本节点追加到标题单元格中。最后把<th/>元素追加到表格行中。

displayDogs()的第二部分建立表格正文，并填充数据，其过程和作用与标题生成代码相同：

```
var tableBody = document.createElement("tbody");

for (var i = 0; i < dogNodes.length; i++)
{
    var tableRow = document.createElement("tr");

    //Inner loop code here

    tableBody.appendChild(tableRow);
}

table.appendChild(tableBody);
```

上面的代码首先创建<tbody/>元素。然后遍历 dogNodes 节点列表，迭代<dog/>元素。在这个循环中创建一个<tr/>元素，并添加到表格正文中。退出循环后，把<tbody/>元素添加到表格中。现在分析内层循环，它给表格行添加数据单元格：

```
for (var j = 0; j < dogNodes[i].childNodes.length; j++)
{
    var currentNode = dogNodes[i].childNodes[j];

    if (currentNode.nodeType == 1)
    {
```

```
            var tableDataCell = document.createElement("td");
            var textData = document.createTextNode(
                currentNode.firstChild.nodeValue
            );

            tableDataCell.appendChild(textData);

            tableRow.appendChild(tableDataCell);
        }

        tableBody.appendChild(tableRow);

    }
```

这个内层循环遍历<dog/>的子节点。首先使用变量 currentNode 来引用当前节点，以便访问这个节点(输入量较少)。接着检查该节点的类型，基于 DOM 的浏览器将空白视为一个子节点，所以需要确保当前节点是一个元素，然后创建一个<td/>元素和一个包含 currentNode 文本的文本节点，并将该文本节点追加到数据单元格上，然后将数据单元格追加到由外层 for 循环创建的表格行上。

到此为止，表格就完成了，所以将该表格添加到 HTML 页面中。为此，使用如下代码：

```
    document.body.appendChild(table);
}
```

现在，只需调用 displayDogs()函数。为此，在页面正文中放置一个链接，单击该链接时，调用 displayDogs()函数：

```
<a href="javascript: displayDogs();">Display Dogs</a>
```

12.7 小结

本章似乎偏离了 JavaScript 的主题，但对了解和掌握 DOM 在 JavaScript 中的地位和重要性是必不可少的。

本章包含以下内容：

- 首先简要介绍 4 个重要标准——HTML、ECMAScript、XML 和 XHTML，并讨论了它们的关系。这些标准的共同目标是为编写 HTML 网页提供指导。这些规则又促进了 DOM 的发展，如果网页的编写遵循这些标准，就可以使用脚本来访问和操作网页上的任何元素。
- 深入介绍了 DOM，DOM 提供了一种独立于浏览器和语言的方式，以访问页面上的元素，解决了旧浏览器中存在的一些问题。DOM 将 HTML 文档表示为树形结构，可以沿着该树形结构导航到不同的元素上，并使用所提供的方法和属性访问页面中的不同部分。

- 虽然遵循标准为操纵网页的内容提供了最好的办法，但是任何主流浏览器都没有完全实现这些标准。本章介绍了一个最新的示例，演示了如何使用 DOM 为创建动态、交互式的网页提供一个坚实的基础。
- 尽管浏览器的发展迅速，但仍存在一些分歧。本章介绍了如何利用代码分支技术把两个不同的 API 合二为一，以处理两种不同的事件对象。
- DHTML 允许在页面加载到浏览器中后修改页面。可以利用各种用户界面技巧，给页面添加一些亮点。
- 本章介绍了如何使用 style 和 className 属性来改变标记的样式。
- 还介绍了一个基本的 DHTML 动画，使文本在两点之间来回移动。
- 最后学习了如何加载 XML 文件，再使用 JavaScript 操作该文档。

12.8　习题

附录 A 中给出了这些练习题的答案。

习题 1：

下面的 HTML 代码创建了一个表格。请使用 JavaScript 和核心 DOM 对象重建该表格，并生成 HTML。在所有可用的浏览器中测试代码，确保代码正常工作。提示：为所编写的每行代码添加注释，以说明当前元素在树形结构中的位置。再为页面中的每个元素创建一个变量(例如，为 9 个 TD 单元格创建 9 个变量，而不是只创建一个变量)。

```
<table>
    <thead>
        <tr>
            <td>Car</td>
            <td>Top Speed</td>
            <td>Price</td>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Chevrolet</td>
            <td>120mph</td>
            <td>$10,000</td>
        </tr>
        <tr>
            <td>Pontiac</td>
            <td>140mph</td>
            <td>$20,000</td>
        </tr>
    </tbody>
</table>
```

习题 2：

前面提到，示例 10 是选项卡控件的一个不完整的 DHTML 脚本，通过进行如下修改对其进行一定程度的完善：

- 一次只能有一个选项卡是激活的。
- 只有激活的选项卡才显示其描述信息。

第13章

在JavaScript中使用ActiveX和插件

现在的浏览器提供了大量的内置功能，但仍有一些工作无法完成，如播放音频或视频。这类功能在Internet上非常普遍，插件及其扩展浏览器功能的能力丰富了用户的Web体验。

插件是可下载的应用程序，可以“插入”到浏览器中。现在有很多不同的插件，常用的有Adobe Flash Player、Microsoft的Silverlight和Apple的QuickTime播放器。

实际上，插件是封装了完成某项工作(例如播放音频文件)所需的所有功能的对象，对网站作者隐藏了复杂的细节。通常用C++或者Java等语言编写插件。

插件通常(但并不总是)有某种用户界面。例如，QuickTime 的用户界面可以显示播放/暂停音频或视频文件的按钮，在播放过程中准确到达某个位置的搜索栏，以及音量控件，如图13-1所示。

图 13-1

某些插件提供了对象的各种方法和属性，用户可通过JavaScript访问，其方式与访问window对象的方法和属性或文档对象模型一样。例如，QuickTime播放器插件提供的Play()方法可以用于播放音频或视频剪辑。

插件已经出现很长时间了，实际上，早在Netscape 3版本中就支持它们。但是Microsoft与其他浏览器厂家不同，IE不支持插件，但是Windows上运行的IE 4.0+支持ActiveX控件，ActiveX控件提供了与插件相同的功能。

幸运的是，只要进行几处修改，就可以通过几乎相同的代码使用ActiveX控件，其方式类似于在其他浏览器上使用插件。主要区别实际上在于首先要确保ActiveX控件或插件可用，且可以运行在用户的浏览器中。在讨论如何使用插件和ActiveX控件之前，先详细介绍Firefox和IE如何解决这个问题。

13.1 在非IE浏览器中检查并嵌入插件

通过创建脚本在网页上使用某个插件的体验令人难忘，但是，除非网页访客在其计算机上安装了这个插件，否则访问该页面时会得到一连串问题和错误消息。因此，不仅在页

面中正确添加使用插件所需的 HTML，还要使用 JavaScript 检测用户的浏览器是否安装了页面要使用的插件。本节将详细讨论这两个问题。

注意：

本节主要讨论 Firefox，其原理同样适用于 Safari、Opera 和 Chrome。

13.1.1 在页面中添加插件

要使用安装在用户浏览器中的插件，必须使用 HTML 代码告诉浏览器，页面将在何时、何处使用它。这一过程称为“嵌入(embedding)”插件。

在 Firefox 中，嵌入插件的关键是非标准的<embed/>元素，该元素会在页面的指定位置嵌入插件的可视化界面。<embed/>元素支持很多可用于所有插件的通用属性，如 height、width、pluginspage、src 和 type。这里重点介绍最后两个属性 src 和 type。下一节介绍 pluginspage 属性。

绝大部分插件都显示保存在 Web 服务器上的内容。例如，处理音频的插件，如 QuickTime 播放器插件，可以播放带有各种扩展名的音乐文件，例如.mp3 和.aac 文件。Flash 插件可以播放 Flash 动画(以.swf 为扩展名的文件)。<embed/>元素的 src 属性可以指定插件要加载和播放的初始文件，这是一个指向文件的 URL，该文件通常与 HTML 页面在同一个 Web 服务器上。利用这个文件，浏览器会确定需要哪种插件。例如，如果 src 是 http://www.myserver.com/myflashmovie.swf，浏览器通过检测文件的类型，就可以确定需要使用 Flash 播放器插件。

但是，并非所有的插件都需要通过 src 属性值来使用外部源中的数据。在这种情况下，浏览器如何确定应加载什么插件？这时，应使用<embed/>元素的 type 属性。type 属性的值专用于某个插件。要找到这个信息，可以在地址栏中输入 about:plugins，插件信息就会加载到浏览器中，如图 13-2 所示。

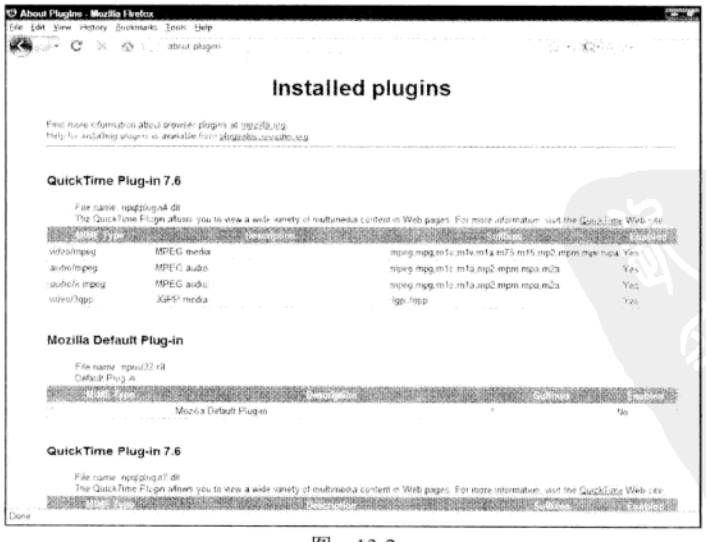

图 13-2

图中列出了浏览器安装的所有插件。type 属性需要的值位于 MIME (Multipurpose Internet Mail Extensions)类型列，该列指定了内容类型，例如网页、图片或 Flash 文件。Flash 的 MIME 类型为 application/x-shockwave-flash。

除了可用于所有插件的许多通用属性之外，还可以使用<embed/>元素指定某个插件的特有属性。例如，Flash 插件支持 quality 属性，用于确定 Flash 动画的画面质量。为了在<embed/>元素中设置这个属性，只需将它添加到属性集列表中，例如下面的代码：

```
<embed id="FlashPlugIn1"
   src="topmenu.swf"
   border=0
   height=100
   width=500
   quality=high
   type="application/x-shockwave-flash" />
```

Firefox 不但支持<embed/>元素，还支持使用 HTML 的标准<object/>元素在页面中嵌入插件，其方式类似于 IE，参见后面的内容。

13.1.2　检查并安装插件

确定要嵌入页面的插件类型后，如果浏览器发现用户的计算机没有安装该插件，该怎么办？

要解决该问题，可以将<embed/>的 pluginspage 属性设置为指向插件创建者的页面的 URL。如果用户的计算机没有安装该插件，网页就显示 pluginspage 属性指定的 URL 链接。用户可以单击该链接，加载插件，使网页正常工作。

例如，Flash 的 pluginspage 属性值应为：

```
http://www.adobe.com/shockwave/download/index.cgi?P1_Prod_Version=Shock
waveFlash
```

但是，如果用户并未安装插件，也可以将用户重定向到一个不依赖该插件的网站版本上。那么，如何确定用户是否安装了某一插件？

第 6 章介绍的 navigator 对象有一个 plugins 属性，它是一个 Plugin 对象的集合，每个 Plugin 对象都表示浏览器安装的一个插件。要访问 plugins 数组中的 Plugin 对象，可以使用在用户浏览器上安装的所有插件的索引的索引值，或者使用插件应用程序的名称。

注意：

IE 有一个 navigator.plugins 集合，但其总是为空。

每个 Plugin 对象都有 4 个属性：description、filename、length 和 name。查看前面的插件信息页面，就可以看到这些属性的值。

下面以 Flash 为例。在地址栏中输入 about:plugins 并按 Enter 键。图 13-3 显示了 Chrome 中已安装的插件页面，这个页面对于所有的非 IE 浏览器而言大致相同。Flash 的 name 属性是 Shockwave Flash，filename 和 description 属性的含义非常明确，length 属性表

示插件支持的 MIME 类型数量。

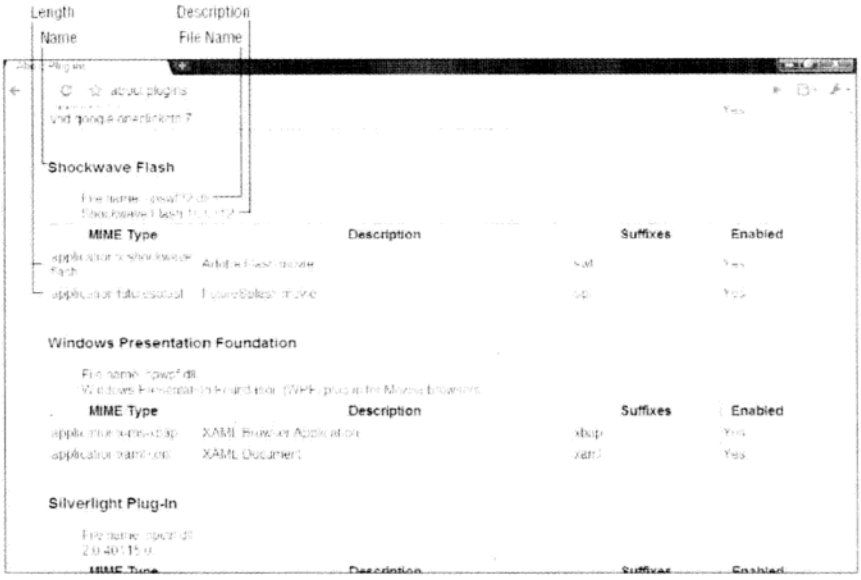

图 13-3

如前所述，name 属性可用于引用 plugins 数组中的 Plugin 对象。因此，如果安装了 Flash，则下面的代码把变量 shockWavePlugin 设置为 Flash 的 Plugin 对象：

```
var shockWavePlugIn = navigator.plugins["Shockwave Flash"];
```

如果未安装 Flash，则 navigator.plugins["Shockwave Flash"]返回 undefined。

使用下面的代码可以重定向浏览器未安装所需插件的用户：

```
if (navigator.plugins["Shockwave Flash"])
{
   window.location.replace("my_flash_enabled_page.htm");
}
else
{
   window.location.replace("my_non_flash_page.htm");
}
```

如果未安装 Flash 插件，navigator.plugins["Shockwave Flash"]就是 undefined，JavaScript 把 undefined 视为 false，从而执行 else 语句。如果安装了 Flash，navigator.plugins["Shockwave Flash"]将返回 Flash 的 Plugin 对象，JavaScript 把它视为 true，从而执行主 if 语句。

这种检测方法的问题是，同一插件在不同操作系统中的名称可能不同。例如，Windows XP 版本的插件名称可能与 Mac 版本不同，也不同于 Linux 版本。一些插件，如 RealPlayer，使用这种检测方法根本不能可靠地工作，因为其名称不是简单的 RealPlayer，而是一个包含 RealPlayer 的字符串。

另一个确定是否安装了某一插件的办法是遍历 plugins[]数组，检查每个 name 是否包含指定的关键字。如果包含，就可以假定该插件已安装。例如，检查是否安装了 QuickTime，可使用如下代码：

```
var pluginsLength = navigator.plugins.length;
for (var i = 0; i < pluginslength; plugInCounter++)
{
    var name = navigator.plugins[i].name.toLowerCase();
    if (name.indexOf("quicktime") > -1)
    {
       alert("QuickTime is installed");
       break;
    }
}
```

上面的 for 循环从索引 0 开始遍历 navigator.plugins 集合，直到最后一个元素。我们检查集合中每个插件的 name 属性是否包含 quicktime，如果包含，就表示已安装 QuickTime，于是中断循环，否则，就表示未安装 QuickTime。

除了使用 navigator 对象的 plugins[]集合之外，还可以使用 navigator 对象的 mimeTypes[]，它是一个 mimeType 对象集合，表示浏览器支持的 MIME 类型。使用这个集合可以检查浏览器是否支持某种媒体。

前面介绍过 MIME 类型——<embed/>元素的 type 属性可用以指定 MIME 类型，以告诉浏览器要嵌入哪个插件。使用已安装插件页面也可以获取某一插件的 MIME 类型。实际上，一个插件可以支持多个 MIME 类型。检查某一 MIME 类型时，就是在检查浏览器是否支持某一类型的文件格式，而不是某个插件。

例如，可以使用 mimeTypes 数组检查 Flash 插件，如下所示：

```
if (navigator.mimeTypes["application/x-shockwave-flash"] &&
    navigator.mimeTypes["application/x-shockwave-flash"].enabledPlugin)
{
   window.location.replace("my_flash_enabled_page.htm");
}
else
{
   window.location.replace("my_non_flash_page.htm");
}
```

if 语句的条件包含了两个用 AND 运算符&&分隔的部分。

第一个部分尝试访问 mimeTypes 集合中的某个 mimeType 对象，检查是否支持指定的 MIME 类型。如果存在这样的对象，就返回 undefined，它表示 false。

条件的第二个部分检查是否启用了处理这个 MIME 类型的插件。浏览器可能支持或识别某种 MIME 类型，但没有安装相应的插件，尽管这种情况不常见却也存在。例如，如果用户安装了 Microsoft Word，则 MIME 类型 application/msword 是有效的，但这并不意味着浏览器上有一个显示它的插件。实际上，mimeType 对象的 enabledPlugin 属性返回一个 Plugin 对象，除非该对象不存在。

13.2 在 IE 中检查和嵌入 ActiveX 控件

尽管 IE 在一定程度上支持插件，但它对 ActiveX 控件的支持更加全面。ActiveX 控件与插件的主要区别在于它们嵌入页面的方式和安装它们的方式。一旦嵌入并安装了 ActiveX 控件，使用 ActiveX 控件和插件的脚本代码就是非常类似的。

ActiveX 控件有点像迷你型程序(mini-programs)，通常用 C++或 Visual Basic 创建。ActiveX 控件并不像 Notepad 或 Microsoft Word 之类的普通程序，它不能独立运行，而必须放到某个容器程序中。并不是所有的程序都能作为 ActiveX 控件的容器，必须使用专门设计的程序，如 Microsoft Access 和 IE。ActiveX 控件的创建者编译代码时，会为 ActiveX 控件分配一个唯一标识的字符串，程序员可以使用该字符串精确地指定要在 IE 中嵌入哪个 ActiveX 控件。

13.2.1 在页面中添加 ActiveX 控件

为在 IE 浏览器的页面中添加 ActiveX 控件，需要使用<object/>元素。<object/>元素的两个重要属性 classid 和 codebase 可用于所有 ActiveX 控件。classid 属性是 ActiveX 控件的创建者在编译时为控件分配的唯一 ID。codebase 属性是用于找到 ActiveX 控件的 URL——下一节将详细讨论这个属性。

如何找到 classid？一个办法是检查 ActiveX 控件附带的文档说明，或者到 ActiveX 控件创建者的网站上查找。如果安装了 ActiveX 控件，也可以通过 IE 来查找，IE 会说明计算机上安装了哪些可用于 IE 的 ActiveX 控件。另外，尽管 IE 会提供 ActiveX 控件的相关信息，如 classid，但不会提供安装在操作系统中的控件信息。例如，虽然 Flash 3 已安装到 Windows 98 系统中，但是 IE 不会提供该控件的信息。

要获得这类信息，只需打开 IE，从 Tools 菜单下选择 Internet Options，如图 13-4 所示。

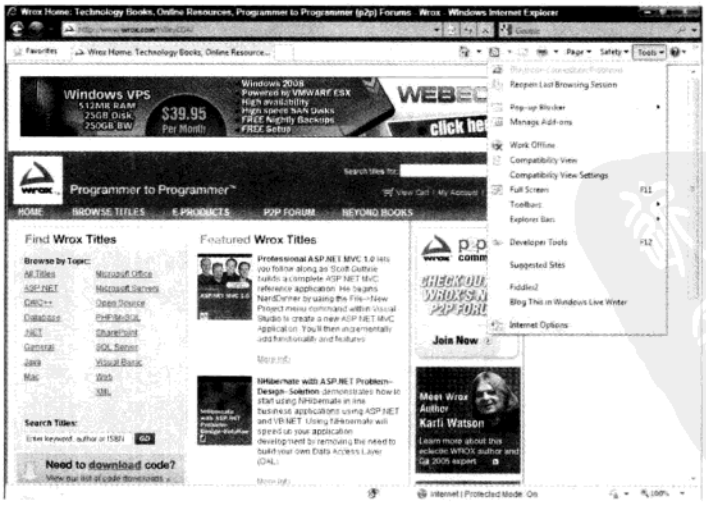

图 13-4

这将打开如图 13-5 所示的窗口。在 Browsing history 区域中，单击 Settings 按钮。在打开的窗口中，单击 View objects 按钮，如图 13-6 所示。

图　13-5

图　13-6

这将显示 IE 从 Internet 上安装的所有 ActiveX 控件。图 13-7 中的列表很可能与用户计算机显示的列表不同。

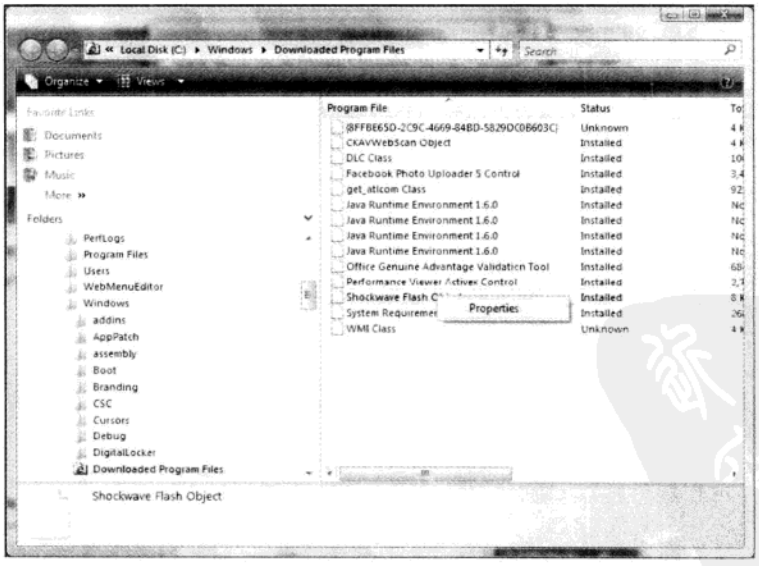

图　13-7

在列表中，可以看到每个控件的许多信息，例如该控件的创建时间及版本号。要找到 classid，可以右击我们感兴趣的 ActiveX 控件，从弹出菜单中选择 Properties。

此时会显示如图 13-8 所示的信息，在不同用户的系统中，所显示的信息可能略有不同。

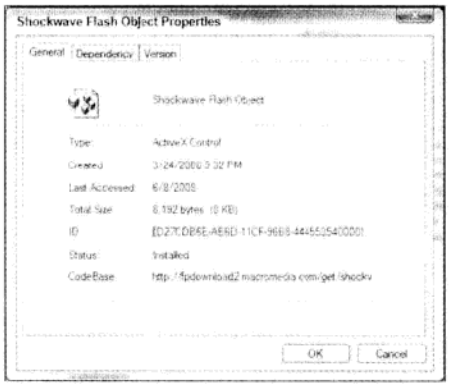

图 13-8

可以看到，classid 属性显示为 ID，codebase 属性显示为 CodeBase。但对于 codebase，还需要选中该行，使用方向键滚动，以查看完整的信息。

因此，要在页面中插入 Flash ActiveX 控件，可以使用下面的<object/>元素：

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="flashPlayer1"
    width="500"
    height="100" />
```

还可以为控件设置属性或参数值。例如，对于 Flash，需要设置 src 属性，指向要加载的.swf 文件，再设置 quality 属性，确定 Flash 动画的画面质量。但是，要设置 ActiveX 控件的参数(与设置<object/>元素的属性不同)，需要在<object>开始标记和</object>结束标记之间插入<param/>元素。

在每个<param/>元素中，需要指定要设置的参数名和参数值。例如，如果要将 src 属性设置为 myFlashMovie.swf 文件，可添加如下所示的<param/>元素：

```
<param name="src" value="myFlashMovie.swf">
```

把这些代码添加到完整的<object/>定义中，同时定义 quality 属性：

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
    id="flashPlayer1"
    width="500"
    height="100">
        <param name="src" value="myFlashMovie.swf">
        <param name="quality" value="high">
</object>
```

13.2.2 安装 ActiveX 控件

前面介绍了如何在页面中插入 ActiveX 控件，但是，如果用户没有在计算机中安装该控件，该怎么办？

这时应使用<object/>元素的 codebase 属性。如果浏览器发现用户的计算机并未安装 ActiveX 控件，就试图从 codebase 属性指定的 URL 中下载和安装该控件。

ActiveX 控件的创建者通常提供了可用作 codebase 属性值的 URL。通过 Tools 菜单的 Internet Options 选项提供的信息，可以了解安装在计算机上的控件的 codebase。但这个 URL 不一定是最佳选择，如果该 URL 未指向控件创建者的链接，就更不合适了。

对于 Flash，其 codebase 是 http://fpdownload.macromedia.com/get/shockwave/cabs/flash/swflash.cab，因此，<object>标记如下所示：

```
<object classid="clsid:D27CDB6E-AE6D-11CF-96B8-444553540000"
codebase="http://fpdownload.macromedia.com/get/shockwave/cabs/flash/swflash.cab"
    id="flashPlayer1"
    width="500"
    height="100">
        <param name="src" value="myFlashMovie.swf">
        <param name="quality" value="high">
</object>
```

如果有许可证，就可以把安装控件的.cab 文件下载到自己的服务器中，并使 codebase 属性指向该文件。

可惜，并没有一种简单的办法来检查用户的计算机安装了哪些 ActiveX 控件。但是，<object/>元素的 object 对象有一个 readyState 属性，它返回 0、1、2、3 或 4，表示对象的操作状态，其含义为：

- 0——控件未初始化，还不能供使用。
- 1——控件正在加载。
- 2——控件已加载完数据。
- 3——尽管还未完全加载控件，但用户可以与控件进行交互。
- 4——控件已加载，可以使用。

在检查 readyState 属性之前，控件必须有一定的时间来加载。因此，最好在窗口的 onload 事件处理程序或 document 对象的 onreadystatechange 事件处理程序触发后进行检查。

要将用户重定向到另一个不需要该控件的页面，可使用如下代码：

```
function window_onload() {
    var flashPlayer1;

    // code to retrieve ActiveX plug-in

    if (flashPlayer1.readyState == 0)
    {
        window.location.replace("NoControlPage.htm");
    }
}

onload = window_onload;
```

这段代码检查添加 ActiveX 控件的 readyState 属性是否为 0。由于这段代码在浏览器加载页面之后执行，所以可以确信 ActiveX 控件没有安装在计算机中。

13.3 使用插件和 ActiveX 控件

将插件或者 ActiveX 控件嵌入页面后，它们的实际用法就几乎是一致的。为了简单起见，大部分插件和 ActiveX 控件的开发者使每个插件和 ActiveX 控件支持类似的属性、方法和事件。但是，一定要阅读说明文档，因为可能有一些不一致的特性。

在<embed/>或者<object/>元素中，为插件或控件设置唯一的 id 值。接着就能访问相应对象的方法、属性和事件，就像访问其他元素一样。尽管插件和控件支持的具体属性、方法和事件是专用的，但是下面介绍一个较常见的控件 Apple 的 QuickTime 播放器，它在非 IE 浏览器中使用的是插件，在 IE 中使用的是 ActiveX 控件。要了解此控件的详细信息，可以访问如下 URL：

```
http://developer.apple.com/documentation/QuickTime/Conceptual/QTScripting_
JavaScript/bQTScripting_JavaScri_Document/QuickTimeandJavaScri.html
```

要运行本章中的示例，需要从 http://www.apple.com/quickTime/上下载免费的 Quick Time 播放器。

注意：
你已可以购买一个具有更多功能的版本，但本书并不要求这么做。

首先，将该控件嵌入网页。在文本编辑器中输入以下的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Using JavaScript to Interface with Quicktime</title>
</head>
<body>
    <object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
        codebase="http://www.apple.com/qtactivex/qtplugin.cab"
        id="audioPlayer" width="0" height="0">
            <param name="src" value="sound1.mp3" />
            <embed height="0" width="0" type="audio/mpeg" src="sound1.mp3"
                pluginspage="www.apple.com/quicktime/download"
                enablejavascript="true" name="audioPlayer" />
    </object>
    <form id="form1" name="form1" action=" ">
        <input type="button" value="Play" id="buttonPlay" name="buttonPlay"
            onclick="buttonPlay_onclick()" />
        <input type="button" value="Stop" id="buttonStop" name="buttonStop"
            onclick="buttonStop_onclick()" />
    </form>
</body>
</html>
```

将上面的代码保存为 quicktime.htm。

首先要注意，用于非 IE 浏览器的<embed/>元素是<object/>元素的一个子元素。Firefox、Opera、Safari 和 Chrome 将忽略<object/>元素，仅显示由<embed/>元素定义的插件。而 IE 将忽略<object/>元素中的<embed/>元素，尽管 IE 支持<embed/>也是如此。如果把<embed/>元素放在<object/>的外面，IE 会识别<object/>和<embed/>，但会混淆 id 值，因为这两个元素的 id 值都是 audioPlayer。

在<object/>元素的下面是带两个按钮的表单，第一个按钮有一个 id 值，其 name 值是 buttonPlay，用户单击它时，会执行 buttonPlay_onclick()函数。第二个按钮是 buttonStop，单击它时会执行 buttonStop_onclick()函数。

13.3.1　确定插件/ActiveX 控件的可用性

我们希望确保没有 QuickTime 的用户在尝试使用带有脚本代码的控件时，看不到错误消息。在这个练习中，禁用了用于控制音频回放的按钮，在 HTML 页面的顶部添加了如下<script/>元素，该脚本定义了一个禁用按钮的函数，并把该函数关联到 window 对象的 onload 事件处理程序上。

```
<script type="text/javascript">
    function window_onload() {
        var plugInInstalled = false;
        if (!window.ActiveXObject) {
            var pluginsLength = navigator.plugins.length;
            for (var i = 0; i < pluginsLength; i++) {
                var pluginName = navigator.plugins[i].name.toLowerCase();
                if (pluginName.indexOf("quicktime") > -1) {
                    plugInInstalled = true;
                    break;
                }
            }
        } else {
            if (document.audioPlayer.readyState == 4) {
                plugInInstalled = true;
            }
        }

        if (!plugInInstalled) {
            document.forms[0].buttonPlay.disabled = true;
            document.forms[0].buttonStop.disabled = true;
            alert("You need Quicktime to play the audio file!");
        }
    }

    onload = window_onload;
</script>
```

在 window_onload()函数中，首先定义了一个变量 plugInInstalled，并将它初始化为 false。接下来，由于要检查插件或控件是否依赖浏览器，于是检查这是否是 Microsoft 浏览器。只要检查 window 对象的 ActiveXObject 属性就足够了。

如果浏览器不是 IE，则使用 for 循环来遍历 navigator 对象的 plugins 集合，检查每个已安装插件的 name 属性是否包含 QuickTime(注意检查该单词的小写版本，这会提供更精确的搜索)。如果找到了 QuickTime，则将变量 plugInInstalled 设置为 true, 再退出 for 循环。

如果这是 Microsoft 的浏览器，则使用<object/>元素的 object 对象的 readyState 属性，检查 ActiveX 控件是否已加载，并成功初始化，已经可以使用了。如果该属性的值为 4，则表示所有系统已经可以使用。因此，将变量 plugInInstalled 设置为 true。

函数中最后的 if 语句检查 plugInInstalled 是 true 还是 false。如果是 false，就禁用按钮，用一个警告框告诉用户，他们需要安装 QuickTime 才能播放音频文件。

13.3.2 完成

创建这个音频播放器的最后一步是给按钮添加功能。如前所述，这些按钮可以启动和停止音频文件的回放。给脚本元素添加如下函数：

```
function buttonPlay_onclick() {
    document.audioPlayer.Play();
}

function buttonStop_onclick() {
    document.audioPlayer.Stop();
}
```

QuickTime 插件或控件提供了 Play()和 Stop()方法，分别播放和暂停回放。所以播放按钮应调用 Play()，停止按钮应调用 Stop()。

有了这些代码，就完成了这个页面。HTML 和 JavaScript 现在应如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Using JavaScript to Interface with Quicktime</title>

    <script type="text/javascript">
        function buttonPlay_onclick() {
            document.audioPlayer.Play();
        }

        function buttonStop_onclick() {
            document.audioPlayer.Stop();
        }

        function window_onload() {
            var plugInInstalled = false;
            if (!window.ActiveXObject) {
                var pluginsLength = navigator.plugins.length;
                for (var i = 0; i < pluginsLength; i++) {
                    var pluginName = navigator.plugins[i].name.toLowerCase();
                    if (pluginName.indexOf("quicktime") > -1) {
                        plugInInstalled = true;
                        break;
```

```
                    }
                }
            } else {
                if (document.audioPlayer.readyState == 4) {
                    plugInInstalled = true;
                }
            }

            if (!plugInInstalled) {
                document.forms[0].buttonPlay.disabled = true;
                document.forms[0].buttonStop.disabled = true;
                alert("You need Quicktime to play the audio file!");
            }

        }

        onload = window_onload;
    </script>
</head>
<body>
    <object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
        codebase="http://www.apple.com/qtactivex/qtplugin.cab"
        id="audioPlayer" width="0" height="0">
            <param name="src" value="sound1.mp3" />
            <embed height="0" width="0" type="audio/mpeg" src="sound1.mp3"
                pluginspage="www.apple.com/quicktime/download"
                enablejavascript="true" name="audioPlayer" />
    </object>
    <form id="form1" name="form1" action=" ">
        <input type="button" value="Play" id="buttonPlay" name="buttonPlay"
            onclick="buttonPlay_onclick()" />
        <input type="button" value="Stop" id="buttonStop" name="buttonStop"
            onclick="buttonStop_onclick()" />
    </form>
</body>
</html>
```

在浏览器中加载 quicktime.htm。只要浏览器支持插件或 ActiveX 控件，并安装了 Quick Time 插件，结果如图 13-9 所示。

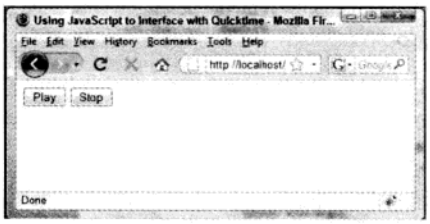

图　13-9

浏览器加载页面时，会自动开始播放所提供的.mp3 文件。使用播放和停止按钮，检查音频播放器的功能。

这是如何工作的？

页面正文中的表单包含两个标准按钮，它们通过 onclick 事件处理程序连接到 JavaScript

函数上。在这些函数中，使用加上前缀 document 的名称访问了嵌入页面的 QuickTime 插件和插件控件。播放函数调用插件和插件控件的 Play()方法播放声音，停止函数调用 Stop()暂停声音的播放。这个脚本在所有主要的浏览器中都会运行，但 IE 访问的是<object/>元素中定义的 ActiveX 控件，而非 IE 浏览器访问的是<embed/>元素中定义的插件。

13.3.3 测试表单的禁用

如果打算使用 ActiveX 控件或插件，就很可能要确保它们已安装在计算机中。问题是，如果已安装了控件，就很容易测试页面是否正常工作；但很难为没有安装该控件的用户测试脚本。这可以采用以下方法：

(1) 使用第 2 台计算机，并重新安装操作系统及浏览器，然后在该计算机中加载页面，这是检查页面的唯一可靠方式。

(2) 卸载插件。根据安装插件或控件的方式，可能有相应的卸载程序，Windows 用户可以使用控件面板中的“Add/Remove programs 删除程序”选项来卸载插件。

(3) 对于非 IE 浏览器，可以安装另一个版本的浏览器。例如，如果安装了 Firefox 3，就尝试安装一个更低版本的 Firefox，如 Firefox 2，或者测试版的 Firefox。当前安装的插件通常不能用于以后安装的浏览器，但也有可能出现例外。

(4) 对于 IE，一次只能安装一个版本的浏览器。但是，很容易在 IE 中移除 ActiveX 控件。在 IE 5+中，选择 Tools | Internet Options。然后单击 Temporary Internet Files(在 IE 7 中是 Browsing History)中的 Settings 按钮，再单击 View Objects 按钮，之后右击要删除的控件名，在弹出的菜单中选择 Remove，就可以删除它。

幸好，这种情况很容易测试。在 window_onload()函数的末尾测试了 pluginInstalled 变量，以确定是否应禁用表单。只要翻转这个检查，就可以查看没有在计算机上安装 QuickTime 的用户在查看页面时会发生什么。所以只需修改 window_onload()的最后一部分代码，如下所示：

```
if (plugInInstalled) {
    document.forms[0].buttonPlay.disabled = true;
    document.forms[0].buttonStop.disabled = true;
    alert("You need Quicktime to play the audio file!");
}
```

重新保存 HTML 文档，在浏览器中刷新页面。图 13-10 显示了 Chrome 的情况。

图 13-10

13.3.4　潜在的问题

插件和 ActiveX 控件提供了一种扩展浏览器功能的绝佳方式，但其代价是由此带来的兼容性问题。下面讨论其中一些问题。

1. 相似但并不相同——浏览器之间的差异

尽管非 IE 浏览器中的插件与 IE 中对应的 ActiveX 控件所支持的属性和方法可能非常接近，但它们常常存在一些重要的、有时很微妙的差异。

例如，RealPlayer 的插件版本和 ActiveX 控件版本都支持 SetSource()方法。下面的代码可以在 IE 上工作：

```
document.real1.SetSource("D:\\MyDir\\MyFile.ra")
```

但在其他浏览器中会产生问题。要在 Firefox 等类似浏览器中工作，应指定加载文件所使用的协议。如果该文件是一个 URL，就指定 http://，对于用户本地硬盘上的文件，则使用 file:///。

为了使代码可跨平台兼容，必须使用如下代码：

```
document.real1.SetSource("file:///D:\MyDir\MyFile.ra")
```

2. 插件的脚本编程有区别

在非 IE 浏览器中编写 QuickTime 插件的脚本时，可嵌入如下代码：

```
<embed height="0" width="0" type="audio/mpeg" src="sound1.mp3"
    pluginspage="www.apple.com/quicktime/download"
    enablejavascript="true" name="audioPlayer" />
```

然后，输入如下代码，通过脚本来访问它：

```
document.audioPlayer.Play()
```

但是，如果为 Flash 播放器编写脚本，则必须在 HTML 中给<embed/>定义添加如下属性：

```
swliveconnect="true"
```

否则，对该插件的任何访问都会造成错误。

```
<embed name="map" swLiveConnect="true" src="topmenu.swf"
 width="300" height="200"
 pluginspage="http://http://www.macromedia.com/go/getflashplayer">
```

必须仔细阅读插件附带的说明文档，以便检查是否存在类似的微妙差别。

3. 操作系统之间的差异

在不同的操作系统之间，对 ActiveX 控件的支持存在较大的差异。Mac 系统中的 IE 支持的 ActiveX 控件，在 Win32 操作系统中不一定支持，例如 Windows 2000、Windows XP

和 WindowsVista。另外，为 Win32 编写的 ActiveX 控件，在 Mac 中也不能正常运行；必须下载 Mac 版本专用的控件。

Mac 中的 IE 既支持 ActiveX 控件，也支持插件。例如 Flash 在 Mac 中是一个插件，而在 Win32 中是一个 ActiveX 控件。显然，如果要同时支持 Mac 和 Windows 用户，还需要编写更复杂的代码。

必须检查用户使用的是哪种操作系统(例如，使用第 6 章末尾列出的脚本)，并处理可能出现的问题。

4. 同一插件或 ActiveX 控件的不同版本之间的差异

插件或 ActiveX 控件的开发者常常定期发布带有新功能的新版本。如果使用这些新功能，则不但需要确保用户加载了正确的插件或 ActiveX 控件，还需要确保其版本是正确的。

ActiveX 控件

对于 ActiveX 控件，可以在<object/>元素的 codebase 属性中添加版本信息。

```
<object classid=clsid:AAA03-8BE4-11CF-B84B-0020AFBBCCFA
    id="myControl"
    codebase="http://myserver/mycontrol.cab#version=3,0,0,0">
</object>
```

现在，浏览器不仅检查该控件是否安装在用户的系统中，还检查安装的版本是否是 3.0 以上。

假定要检查某个控件的版本，如果该版本低于页面需要的版本，则将用户重定向到另一个页面，该怎么办呢？

对于 ActiveX 控件，没有一个简单的办法来使用 JavaScript 代码检查 ActiveX 控件的版本。一个办法是找到一个新控件支持、而旧版本不支持的属性，检查它是否为 null。例如，假定某个控件的最新版本引入了属性 BgColor，要检查已安装的版本是否是所需的版本，可使用如下代码：

```
if (document.myControl.BgColor == null)
{
    alert("This is an old version");
}
```

ActiveX 控件的创建者还可能为其控件添加了某种 version 属性，以供检查控件的版本，但是这因控件而异。

插件

对于插件，需要使用 navigator 对象的 plugins[]数组属性包含的 Plugin 对象，每个 Plugin 对象都有 name、filename 和 description 属性。其中，description 属性可能包含了版本信息。但是，各个插件也可能有所不同。

例如，对于 Win32 上的 Flash Player 10，使用下面的代码获取 description 属性是 Shockwave Flash 10.0 r12：

```
navigator.plugins["Shockwave Flash"].description
```

使用第 9 章介绍的正则表达式，可以从这个字符串中提取版本号。

```
var myRegExp = /\d{1,}.\d{1,}/;
var flashVersion = navigator.plugins["Shockwave Flash"].description;
flashVersion = parseFloat(flashVersion.match(myRegExp)[0]);
```

第一行代码定义了一个正则表达式，用来匹配一个或多个数字，后跟一个句点和一个或多个数字。接着将该 Flash 插件的 description 属性保存在变量 flashVersion 中。最后，在字符串中查找与正则表达式匹配的子串，并返回一个由所有匹配组成的数组。然后在数组的第一个元素，即索引为 0 的元素上使用 parseFloat()函数。

5. IE 6 Service Pack 1b 和 ActiveX 控件的变化

主要是出于法律方面的原因，Microsoft 改变了 ActiveX 控件在 IE 中的运行方式。只要用户浏览了一个包含 ActiveX 控件的页面，就会显示该控件的一个警告，且默认禁止运行 ActiveX 控件，除非用户选择运行该控件。可以通过两种方法来避免这样的问题：

(1) 不访问任何外部数据，也不在定义中包含任何<param/>元素，例如下面的例子：

```
<object classid="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6"></object>
```

(2) 使用新的 noexternaldata 属性，指定不使用任何外部数据。

```
<object noexternaldata="true" classid="CLSid:6BF52A52-394A-11d3-B153-
                                                        00C04F79FAA6">
  <param name="URL"
  value="http://msdn.microsoft.com/workshop/samples/author/dhtml/media/d
                                                                rums.wav"/>
</object>
```

这将忽略 URL 参数，不访问 URL 指定的外部数据(本例中是.wav 文件)。

13.4　小结

本章介绍了如何使用插件和 ActiveX 控件来扩展浏览器的功能，主要内容包括：

- 在 Windows 操作系统上，IE 支持 ActiveX 控件以及部分扩展插件。非 IE 浏览器为插件提供了良好支持，但不支持 ActiveX 控件。
- 绝大部分插件的创建者也提供了等价的 ActiveX 控件。在安装插件还是 ActiveX 控件的问题上，IE 和其他浏览器是不兼容的。
- 使用<embed/>元素可以在网页中嵌入插件。使用<embed/>元素的 src 属性和 type 属性，指定源文件或 MIME 类型，就可以告诉非 IE 浏览器要嵌入哪个插件。如果定义了<embed/>元素的 pluginspage 属性的值，则未安装该插件的用户可以单击一个链接，以安装插件。

- 在地址栏中输入 about:plugins，就可以找到非 IE 浏览器中安装的插件信息，包括插件的描述和类型。
- 要使用脚本检查用户是否有某个插件，可以使用 navigator 对象的 plugins 集合。对于每个已安装的插件，都在这个集合中定义了一个 Plugin 对象。每个 Plugin 对象都有 name、description、filename 和 length 属性，可用于判断用户的计算机上是否存在某插件。也可以使用 navigator 对象的 mimeTypes 集合属性，来检查用户浏览器是否支持某种类型的文件。
- IE 支持 ActiveX 控件，作为插件的替代。ActiveX 控件通过<object/>元素嵌入到网页中。通过 classid 属性指定需要的 ActiveX 控件。如果要为未安装某个控件的用户提供自动安装控件，需要指定 codebase 属性。
- ActiveX 控件的参数通过放在开闭<object>标记之间的<param/>元素指定。
- 通过 Object 对象的 readyState 属性，可以检查是否成功加载控件。它返回一个数值：0 表示控件未安装；1 表示控件在加载；2 表示控件已经加载；3 表示可以与控件交互；4 表示控件已经加载完毕并可供使用。
- 几乎每个不同类型的插件和 ActiveX 控件都提供了自己的界面，其文档提供了详细的说明。本章简要介绍了 Apple 的 QuickTime 控件。
- 插件和 ActiveX 控件都可用于扩展浏览器的功能，但它们也存在潜在问题，包括给插件与 ActiveX 控件编写脚本的方式不同、操作系统的差异、同一插件或者控件的版本差异。

第 14 章将讨论一个“新的”JavaScript 技术——它重新点燃了 Web 应用程序开发人员的激情。

13.5 习题

附录 A 给出了练习答案。

习题 1：

使用 Quicktime 插件或 ActiveX 控件，创建一个包含三个链接的页面。当用户单击任一个链接时，就播放声音。使用警告框告诉没有安装 QuickTime 的用户，单击链接时，必须安装 Quicktime。

在 IE、Firefox、Safari、Chrome 和 Opera 中，该页面都可以正常运行。告诉 QuickTime 播放哪个文件的方法是 SetURL()。

第14章

Ajax

Internet 自问世以来，就在使用具有事务特征(transaction-like)的通信模型：浏览器向服务器发送请求，服务器向客户端返回响应，浏览器重新加载页面。这是典型的 HTTP 通信，HTTP 协议设计为按照这种方式工作。但是这种模型对开发人员而言相当不便，因为它需要 Web 应用程序包含数个页面，这种不连续的页面加载方式导致用户的体验变得杂乱，缺乏连贯性。

21 世纪早期，人们开始寻求和开发新技术，来增强用户的体验，并使 Web 应用程序的操作更类似于传统应用程序。这些新技术的性能和可用性往往与传统的桌面应用程序类似，很快，开发人员开始细化这些过程，为用户提供更丰富的功能。

这个浪潮的核心是一种语言：JavaScript，以及使 HTTP 请求对用户而言是透明的能力。

14.1 Ajax 的含义

本质上，Ajax 允许客户端 JavaScript 向服务器请求和接收数据，而无须刷新 Web 页面。这种技术允许开发人员创建不中断的应用程序，用新数据重载页面的某些部分。

术语 Ajax 来源于 Jesse James Garrett 在 2005 年写的一篇文章“Ajax: A New Approach to Web Applications(Ajax：Web 应用开发的崭新道路)”(www.adaptivepath.com/publications/essays/archives/000385.php)。在这篇文章中，Garrett 认为：Web 应用程序与桌面应用程序之间在交互性上的差距会变得越来越小，并用 Google Maps 和 Google Suggest 等应用程序作为佐证。这个术语最初表示 Asynchronous JavaScript + XML(XML 是浏览器和服务器彼此通信的格式)，目前，Ajax 表示使用 JavaScript 收发来自 Web 服务器的数据，且无须重载整个页面的模式。

尽管 Ajax 这一术语是在 2005 年创造出来的，但是其底层的方法已使用多年。早期的 Ajax 技术包括使用隐藏的框架/内嵌框架，在文档中动态添加<script/>元素，使用 JavaScript 给服务器发送 HTML 请求。近几年使用 JavaScript 发送请求的技术比较流行。这些新技术仅刷新页面的局部，不但减小了发送给浏览器的数据量，还使网页的操作方式更像传统的应用程序。

14.1.1 Ajax 的作用

Ajax 打开了通向高级 Web 应用程序的大门——在形式和功能上模拟桌面应用程序。大量的商业网站都喜欢使用 Ajax，这些网站的外观和操作方式更像桌面应用程序，而不是网站。最著名的支持 Ajax 的 Web 应用程序来自搜索引擎巨头 Google：Google Maps 和 Google Suggest。

1. Google Maps

为了和现存的商业地图站点竞争(并使用 Google Earth 的图像)，Google Maps (http:// maps.google.com)使用 Ajax 将地图图像动态添加到网页中。输入一个位置后，不会重新加载主页面，只是把地图图像动态地加载到地图区域中。Google Maps 还允许用户将地图拖动到新位置，这时地图图像会再次动态添加到地图区域中，如图 14-1 所示。

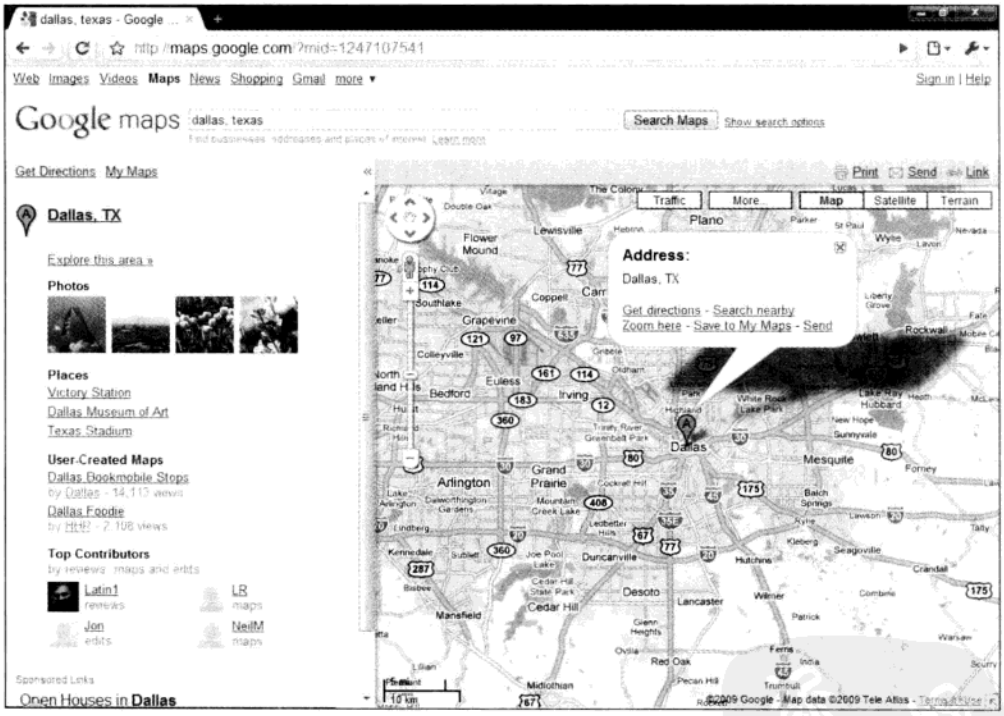

图 14-1

2. Google Suggest

Google Suggest(http://labs.google.com/suggest/)是另一个使用 Ajax 的 Google 创新成果。初看上去，它像是一个普通的 Google 搜索页面。但是用户开始输入数据时，一个下拉框会显示用户可能感兴趣的、与搜索关键字相关的建议。在建议关键字或者短语的右边是搜索该条目返回的结果数，如图 14-2 所示。

图　14-2

14.1.2　浏览器支持

Ajax 只能用于运行 Web 应用程序的浏览器，与本书讨论的所有高级 JavaScript 概念一样，Ajax 功能也因浏览器而异。幸运的是，Ajax 最常见的形式可以用于以下浏览器：

- Internet Explorer 5+
- Firefox 1+
- Opera 9+
- Safari 2+
- Chrome 1+

隐藏的框架技术是一种常见的 Ajax 方法，在这些浏览器中使用该技术时，代码几乎没有差别，由于每个 BOM 都采用相同的方式来处理框架(本章稍后将介绍基于框架的 Ajax)。但是，使用其他形式的 Ajax 时，如 XMLHttpRequest，代码的差异将明显地表现出来。

14.2　使用 XMLHttpRequest 对象

如前所述，可以通过许多方法来创建支持 Ajax 的应用程序。但是，最流行的 Ajax 技术合并了所有主流浏览器都有的 JavaScript 对象 XMLHttpRequest。

注意：

尽管 XMLHttpRequest 对象的名称中包含了 XML，但可以使用它获取其他类型的数据，如纯文本。

XMLHttpRequest 对象起源于一个 Microsoft 组件 XmlHttp，该组件所在的 MSXML 库最初随 IE 5 一起分发。XmlHttp 组件为开发者提供了一种打开 HTTP 连接并检索 XML 数据的简单方法。Microsoft 在 MSXML 的每个新版本中都改进了这个组件，使之运行得更快更高效。

随着 Microsoft 的 XMLHttpRequest 对象越来越流行，Mozilla 决定在 Firefox 中加入自己的 XMLHttpRequest 对象。该对象的 Mozilla 版本保持了与 Microsoft 的 ActiveX 组件相同的属性和方法，使跨浏览器的 Ajax 应用成为可能。很快，Opera Software 和 Apple 沿袭了 Mozilla 的实现方式，使该对象可以在所有现代浏览器中使用。

14.2.1 跨浏览器问题

XMLHttpRequest 对象与浏览器支持的其他 Web 标准没有区别，浏览器对该对象的支持可以分为两类：ActiveX(用于 IE 5 和 IE6)和内置支持(用于其他浏览器)。幸好，这两类浏览器仅在创建 XMLHttpRequest 对象时有区别。在创建该对象后，其余的代码则兼容每种浏览器。

1. 使用 ActiveX

由于 XMLHttpRequest 对象源自 MSXML 库——一个 ActiveX XML 解析器，因此在这些浏览器中实例化 XMLHttpRequest 对象，需要创建一个 ActiveX 对象。第 12 章曾经创建 ActiveX 对象，以遍历 XML DOM。创建 XMLHttp 对象的方法没有太大的区别：

```
var  oHttp = new ActiveXObject("Microsoft.XMLHttp");
```

这行代码创建了 Microsoft 的 XMLHttpRequest 对象的第一个版本。Microsoft 的 XmlHttp 对象有其他许多版本，但是 Microsoft 建议采用下列版本：

- MSXML2.XmlHttp.6.0
- MSXML2.XmlHttp.3.0

创建 XmlHttpRequest 对象时，我们希望使用最新版本，因为最新版本包含了错误的修正，改进了性能，其缺点是，并不是每个人都在计算机上安装了同一版本。但可以编写一个函数，使用用户计算机上安装的最新版本的 XmlHttp 对象。

利用上面的版本信息，编写一个函数 createXmlHttpRequest()，通过用户计算机支持的最高版本创建 XMLHttpRequest 对象。

```
function createXmlHttpRequest()
{
    var versions =
    [
        "MSXML2.XmlHttp.6.0",
        "MSXML2.XmlHttp.3.0"
```

```
    ];
    //more code here
}
```

上面的代码定义了 createXmlHttpRequest()函数。在该函数中，versions 数组包含了 Microsoft 推荐的两个不同版本名。注意，最新版本的名称放在最前面。这是因为我们总是想先检查最新版本，再检查次新版本，直到找到计算机安装的那个版本为止。

为了确定使用哪个版本，可使用 for 循环来遍历数组中的元素，再尝试创建 XMLHttp Request 对象。

```
function createXmlHttpRequest()
{
    var versions =
    [
        "MSXML2.XmlHttp.6.0",
        "MSXML2.XmlHttp.3.0"
    ];

    for (var i = 0; i < versions.length; i++)
    {
        try
        {
           var oHttp = new ActiveXObject(versions[i]);
           return oHttp;
        }
        catch (error)
        {
          //do nothing here
        }
    }
    //more code here
}
```

如果用户的计算机未安装某个版本，就抛出一个错误。因此在循环中使用了一个 try…catch 块来捕获错误。这是判断计算机上是否安装了某版本的唯一方式。如果不存在该版本，就执行 catch 块。由于在这个块中什么也没有发生，所以循环进入数组的下一个元素。如果未在计算机上找到任何版本，函数就返回 null，如下所示：

```
function createXmlHttpRequest()
{
    var versions =
    [
        "MSXML2.XmlHttp.6.0",
        "MSXML2.XmlHttp.3.0"
    ];

    for (var i = 0; i < versions.length; i++)
    {
        try
        {
```

```
            var oHttp = new ActiveXObject(versions[i]);
            return oHttp;
        }
        catch (error)
        {
          //do nothing here
        }
    }

    return null;
}
```

现在就不必考虑使用什么 ActiveX 对象创建 XMLHttp 对象了。只需调用该函数，就会自动完成创建工作。

```
var  oHttp = createXmlHttpRequest();
```

2. 调用内置的构造函数：其他浏览器

IE 7+、Firefox、Opera、Safari 和 Chrome 提供了 XMLHttpRequest 对象的内置(native)实现；该对象位于 window 对象中。所以创建 XMLHttpRequest 对象只需调用其构造函数。

```
var  oHttp = new XMLHttpRequest();
```

这行代码创建了一个 XMLHttpRequest 对象，可以用来连接服务器，并请求和接收服务器上的数据。与上一节的 ActiveX 对象不同，这里的 XMLHttpRequest 没有不同的版本，只需调用构造函数，就能创建和使用 XMLHttpRequest 对象。

3. 合二为一：用一个函数创建适用于所有浏览器的对象

与其他的跨浏览器问题类似，可以找到一个创建可用于所有浏览器的 XMLHttpRequest 对象的方法。前面编写了 createXmlHttpRequest()函数，只需扩展它，提供跨浏览器功能即可。

```
function createXmlHttpRequest()
{
    if (window.XMLHttpRequest)
    {
        var oHttp = new XMLHttpRequest();
        return oHttp;
    }
    else if (window.ActiveXObject)
    {
        var versions =
        [
            "MSXML2.XmlHttp.6.0",
            "MSXML2.XmlHttp.3.0"
        ];

        for (var i = 0; i < versions.length; i++)
        {
```

```
            try
            {
                var oHttp = new ActiveXObject(versions[i]);
                return oHttp;
            }
            catch (error)
            {
               //do nothing here
            }
        }
    }

    return null;
}
```

上面的新代码首先检查是否存在 window.XMLHttpRequest。如果存在，函数就使用 XMLHttpRequest 构造函数来创建 XMLHttpRequest 对象。否则，代码就检测用于 IE 5 和 IE 6 的 window.ActiveXObject，并尝试用最新的 XMLHttp 版本创建对象。如果不能创建适用于任何浏览器的 XMLHttpRequest 对象，函数就返回 null。

注意：

测试浏览器的顺序是非常重要的。先测试 window.XMLHttpRequest 对象，是因为 IE 7+ 支持 window.XMLHttpRequest 和 window.ActiveXObject。

只要用户的浏览器支持 XMLHttpRequest，这个修订过的函数就会创建 XMLHttp Request 对象。

14.2.2　使用 XMLHttpRequest 对象

一旦创建了 XMLHttpRequest 对象，就可以准备使用它请求数据。这一过程的第一个步骤是调用 open()方法，以初始化该 XMLHttpRequest 对象。

```
oHttp.open(requestType, url, async);
```

这个方法接收 3 个参数。第一个参数 requestType 是一个表示请求类型的字符串，其值可以是 GET 或者 POST。第二个参数是作为请求发送目标的 URL。第三个参数是 true 或 false，表示请求应以异步或同步模式发出。异步和同步模式参见第 12 章，这里再强调一下，以同步模式发出的请求会暂停所有 JavaScript 代码的执行，直到从服务器获得响应为止。

注意：

实际的应用程序首选异步模式。

下一步是使用 send()方法发送请求。这个方法接收一个参数，它是一个字符串，包含随请求一起发送的请求体。GET 请求不包含任何信息，所以把 null 传送为参数。

```
var oHttp = createXmlHttpRequest();
oHttp.open("GET", "http://localhost/myTextFile.txt", false);
```

```
oHttp.send(null);
```

这段代码发出一个 GET 请求，以同步模式获取 myTextFile.txt 文件。调用 send()方法会把该请求发送给服务器。

提示：

send()方法必须接收一个参数，甚至可以是 null。

每个 XMLHttpRequest 对象都有 status 属性，该属性包含了与服务器的响应一起发送的 HTTP 状态码。服务器返回状态码 200 表示请求成功，返回 404 表示找不到请求的文件。请分析下面的示例：

```
var oHttp = createXmlHttpRequest();
oHttp.open("GET", "http://localhost/myTextFile.txt", false);
oHttp.send(null);

if (oHttp.status == 200)
{
    alert("The text file was found!");
}
else if (oHttp.status == 404)
{
    alert("The text file could not be found!");
}
else
{
    alert("The server returned a status code of " + oHttp.status);
}
```

这段代码检查 status 属性，确定给用户显示什么消息。如果请求成功(状态码为 200)，就用警告框告诉用户，存在请求的文件。如果该文件不存在(状态码为 404)，用户就会看到一个消息，消息指出服务器找不到该文件。最后，如果状态码不是 200 或 404，就用一个警告框将该状态码告知用户。

有许多不同的 HTTP 状态码，因此不可能检查每个状态码。在大多数情况下，只需关心请求是否成功。因此，上面的代码可以精简为：

```
var oHttp = createXmlHttpRequest();
oHttp.open("GET", "http://localhost/myTextFile.txt", false);

oHttp.send(null);

if (oHttp.status == 200)
{
    alert("The text file was found!");
}
else
{
    alert("The server returned a status code of " + oHttp.status);
}
```

这段代码实现了同样的功能，但仅检查状态码 200，对于其他状态码，则给用户报告一条常见的信息。

14.2.3　异步请求

前面的代码示例演示了同步请求的简洁性，而异步请求会给代码添加一些复杂性，因为必须处理 readystatechange 事件。在异步请求中，XMLHttpRequest 对象提供了 readyState 属性，该属性包含一个数值，每个值都代表请求生存期中的特定状态，如下：

- 0——已创建对象，但未调用 open()方法。
- 1——已调用 open()方法，但未发送请求。
- 2——请求已发送，标题和状态已接收到，并可用。
- 3——接收到来自服务器的响应。
- 4——接收完请求数据。

每当 readyState 属性发生变化时，都会触发 readystatechange 事件，调用 onreadystatechange 事件处理程序。最后一个状态是最重要的，它表示已经完成请求。

注意：

即使请求成功，也可能得不到需要的信息。在请求的服务器端可能发生了错误(404、500 或其他错误)。因此，仍需要检查请求的状态码。

处理 readystatechange 事件的代码如下：

```
var oHttp = createXmlHttpRequest();

function oHttp_readyStateChange()
{
    if (oHttp.readyState == 4)
    {
        if (oHttp.status == 200)
        {
        alert(oHttp.responseText);
        }
        else
        {
            alert("The server returned a status code of " + oHttp.status);
        }
    }
}

oHttp.open("GET", "http://localhost/myTextFile.txt", true);
oHttp.onreadystatechange = oHttp_readyStateChange;

oHttp.send(null);
```

这段代码首先定义处理 readystatechange 事件的 oHttp_readyStateChange()函数。该函数首先检查 readyState 属性是否为 4，以便确定请求是否完成。接下来检查请求状态，以确保服务器返回所请求的数据。一旦满足这两个条件，代码就显示 readyState 属性的值(实际请

求的数据使用纯文本格式)。这使 XMLHttpRequest 对象异步请求数据。

使用异步通信的好处高于处理 readystatechange 事件添加的复杂性，因为在请求对象收发数据的同时，浏览器可以继续加载页面，执行其他 JavaScript 代码。封装了 XMLHttpRequest 对象的用户自定义模块可以使异步请求更易于使用和管理。

注意：

XMLHttpRequest 对象还有一个 responseXML 属性，用来将接收到的数据加载到 XML DOM 对象中(而 responseText 返回纯文本)。这是 Safari 2 将 XML 数据加载到 DOM 中的唯一方法。

14.3 创建简单的 Ajax 模块

在程序设计中，代码重用的概念非常重要，它是使用函数来执行特定的、重复性的常见任务的原因。第 5 章介绍了代码重用的面向对象结构——引用类型。这些结构包含带有数据的属性和/或对这些数据执行操作的方法。

本节将编写自定义的 Ajax 模块 HttpRequest 使异步请求更易于使用和管理。在编写这个模块之前，先讨论 HttpRequest 引用类型拥有的属性和方法。

14.3.1 规划 HttpRequest 模块

只须跟踪一个信息：底层的 XMLHttpRequest 对象。因此，这个模块只有一个属性 request，它包含底层的 XMLHttpRequest 对象。

方法也非常容易定义：

- createXmlHttpRequest()——创建可用于所有支持浏览器的 XMLHttpRequest 对象。它基本上是本章前面编写的同名函数的副本。
- send()方法——向服务器发送请求。

定义了属性和方法后，下面就开始编写这个模块。

14.3.2 HttpRequest 构造函数

引用类型的构造函数定义其属性，并执行该类型正常工作所需要的所有逻辑。

```
function HttpRequest(sUrl, fpCallback)
{
    this.request = this.createXmlHttpRequest();

    //more code here
}
```

构造函数接收两个参数。第一个参数 sUrl 是 XMLHttpRequest 对象请求的 URL。第二个参数 fpCallback 是一个回调函数，当接收到服务器的响应时(即请求的 readyState 为 4，status 是 200)调用它。构造函数的第一行代码初始化 request 属性，并给它赋予一个 XML-

HttpRequest 对象。

request 属性创建好并可供使用后，就该准备要发送的请求了。

```
function HttpRequest(sUrl, fpCallback)
{
    this.request = this.createXmlHttpRequest();
    this.request.open("GET", sUrl, true);

    function request_readystatechange()
    {
        //more code here
    }

    this.request.onreadystatechange = request_readystatechange;
}
```

上述新代码的第一行使用 XMLHttpRequest 对象的 open()方法初始化请求对象。该方法将请求类型设置为 GET，使用 sUrl 参数指定要请求的 URL，并把请求对象设置为使用异步方式。接下来的代码行定义了 request_readystatechange()函数。在一个函数中定义另一个函数似乎有点怪异，但这是完全合法的，这种高级技术称为闭包(closure)。闭包(如 request_readystatechange()函数)不能在容器函数(这里是构造函数)之外访问，但内部函数可以访问其容器函数的变量和参数。这个闭包函数处理请求对象的 readystatechange 事件，绑定闭包函数的方式是把它赋予 onreadystatechange 事件处理程序。

```
function HttpRequest(sUrl, fpCallback)
{
    this.request = this.createXmlHttpRequest();
    this.request.open("GET", sUrl, true);

    var tempRequest = this.request;
    function request_readystatechange()
    {
        if (tempRequest.readyState == 4)
        {
            if (tempRequest.status == 200)
            {
                fpCallback(tempRequest.responseText);
            }
            else
            {
                alert("An error occurred trying to contact the server.");
            }
        }
    }
    this.request.onreadystatechange = request_readystatechange;
}
```

这些新代码看起来有点奇怪。第一行新代码创建 tempRequest 变量，这个变量是指向当前对象的 request 属性的指针，在 request_readystatechange()函数中使用。这个技术可解

决作用域的问题。理想情况下，可以在 request_readystatechange()函数中使用 this.request，但 this 关键字指向 request_readystatechange()函数，而不是 XMLHttpRequest 对象，所以代码不能正常工作。所以看到 tempRequest，就应把它看作 this.request。

在 request_readystatechange()函数中，包含如下代码行：

```
fpCallback(tempRequest.responseText);
```

这行代码调用了构造函数的 fpCallback 参数指定的回调函数，并将 responseText 属性传递给该回调函数。这样回调函数就可以使用从服务器中接收的信息。

14.3.3 创建方法

这个引用类型有两个方法：一个在构造函数中使用；另一个向服务器发送请求。

1. 创建跨浏览器的 XMLHttpRequest 对象

第一个方法是 createXmlHttpRequest()，本章前面讨论了创建跨浏览器的对象的内部工作原理，所以下面仅列出该方法的定义：

```
HttpRequest.prototype.createXmlHttpRequest = function ()
{
    if (window.XMLHttpRequest)
    {
        var oHttp = new XMLHttpRequest();
        return oHttp;
    }
    else if (window.ActiveXObject)
    {
        var versions =
        [
        "MSXML2.XmlHttp.6.0",
        "MSXML2.XmlHttp.3.0"
        ];

        for (var i = 0; i < versions.length; i++)
        {
            try
            {
                var oHttp = new ActiveXObject(versions[i]);
                return oHttp;
            }
            catch (error)
            {
                //do nothing here
            }
        }
    }

    alert("Your browser doesn't support XMLHttp");
}
```

第 5 章学习了如何通过 prototype 对象指定用户自定义的引用类型的方法。在编写 createXmlHttpRequest()方法和下面的 send()方法时，遵循了这个原则。

2. 发送请求

向服务器发送请求，需要调用 XMLHttpRequest 对象的 send()方法。这个 send()方法与此类似，但不接收参数。

```
HttpRequest.prototype.send = function ()
{
    this.request.send(null);
}
```

这个 send()版本的方法非常简单，只需调用 XMLHttpRequest 对象的 send()方法，并给它传递 null。

14.3.4　完整的代码

讨论完代码后，打开文本编辑器中，输入如下内容：

```
function HttpRequest(sUrl, fpCallback)
{
    this.request = this.createXmlHttpRequest();
    this.request.open("GET", sUrl, true);

    var tempRequest = this.request;
    function request_readystatechange()
    {
        if (tempRequest.readyState == 4)
        {
            if (tempRequest.status == 200)
            {
                fpCallback(tempRequest.responseText);
            }
            else
            {
                alert("An error occurred trying to contact the server.");
            }
        }
    }

    this.request.onreadystatechange = request_readystatechange;
}

HttpRequest.prototype.createXmlHttpRequest = function ()
{
    if (window.XMLHttpRequest)
    {
        var oHttp = new XMLHttpRequest();
        return oHttp;

    }
```

```
        else if (window.ActiveXObject)
        {
            var versions =
            [
                "MSXML2.XmlHttp.6.0",
                "MSXML2.XmlHttp.3.0"
            ];
            for (var i = 0; i < versions.length; i++)
            {
                try
                {
                    var oHttp = new ActiveXObject(versions[i]);
                    return oHttp;
                }
                catch (error)
                {
                  //do nothing here
                }
            }
        }

        return null;
    }

    HttpRequest.prototype.send = function ()
    {
        this.request.send(null);
    }
```

将这个文件保存为 httprequest.js。本章后面将用到它。

这个模块的作用是使异步请求更易于使用，下面用一个只包含代码的简短示例来验证是否已达到该目标。

首先需要定义一个函数，用来处理请求所接收到的数据。该函数传递给 HttpRequest 构造函数。

```
function handleData(sResponseText)
{
    alert(sResponseText);
}
```

这段代码定义了一个函数 handleData()，它仅接收一个参数 sResponseText。在执行时，该函数仅显示传入的数据。下面创建一个 HttpRequest 对象，并发送请求。

```
var request = new HttpRequest("http://localhost/myTextFile.txt", handleData);
request.send();
```

上面的代码将文本文件位置和函数 handleData()的指针传递给构造函数 HttpRequest()，并使用 send()方法发送请求。在请求成功时调用函数 handleData()。

这个模块封装了与异步 XMLHttpRequest 请求相关的代码，不需要创建请求对象，处理 readystatechange 事件，或者检查请求的状态，HttpRequest 模块会完成所有这些工作。

14.4　使用 Ajax 验证表单字段

用户可能多次遇到过这样的情况：在网站的论坛上注册为新用户，或者申请基于 Web 的电子邮件时，需要填写整个表单，提交它。但在查看重载了新数据的页面时，发现自己申请的用户名已被其他人占用(还丢失了已输入的一些数据)。表单验证是很容易受挫的，幸好，Ajax 可以减轻这种令人不快的体验，并在提交表单之前，把数据发送给服务器——允许服务器验证数据，并告诉用户验证结果，而无需重载页面。

本节将创建一个使用 Ajax 技术验证表单字段的表单。该表单可以用各种方式建立，最简单的办法就是提供一个链接，给服务器应用程序发送一个 HTTP 请求，以便检查用户提供的信息是否可用。

要建立的表单类似于日常使用的普通表单，它包含以下字段：

- Username(需要校验)——用户在该字段中输入希望申请的用户名。
- Email(需要校验)——用户在该字段中输入 e-mail。
- Password(无须校验)——用户在该字段中输入密码。
- Verify Password(无须校验)——用户在该字段中验证密码。

注意，本例仅显示 Password 和 Verify Password 字段。密码校验肯定由服务器应用程序来完成，但是，使用 JavaScript 进行这个验证会更高效。这么做会增加本例的复杂性，而本例应尽可能简单，以帮助读者掌握 Ajax 的用法。

Username 和 Email 字段的旁边是一个超链接，用于调用一个 JavaScript 函数，使用本章前面创建的 HttpRequest 模块查询服务器。

如前所述，Ajax 是浏览器和服务器之间的通信。所以本例需要一个简单的服务器应用程序，来验证表单字段。虽然 PHP 编程超出了本书的讨论范围，但我们将分析如何从 PHP 应用程序中请求数据，并论述该应用程序给 JavaScript 发回的响应。

14.4.1　请求信息

PHP 应用程序在查询字符串中查找两个参数 username 参数或 email。

要检查用户名是否可用，可使用 username 参数。执行该检查的 URL 如下所示：

```
http://localhost/formvalidator.php?username=[usernameToSearchFor]
```

搜索用户名时，只需将[usernameToSearchFor]替换为实际的用户名。

查询 e-mail 的方法与此类似。查询 e-mail 的 URL 如下所示：

```
http://localhost/formvalidator.php?email=[emailToSearchFor]
```

14.4.2　接收到的数据

成功的请求会得到以下两个值之一：

- available——表示用户名或 e-mail 有效。
- not available——表示用户名或 e-mail 已被使用，因此无效。

这些值以纯文本的方式发送给客户端。只需通过一个简单的比较，即可告诉用户，其用户名或 e-mail 是否已被使用。

14.4.3 准备工作

由于这是一个联机编码(live-code)的 Ajax 示例，因此要运行该示例，计算机必须满足几项要求。

1. Web 服务器

首先，需要一个 Web 服务器。如果使用 Windows 2000(服务器版或专业版)、Windows XP Professional、Windows Server 2003、Windows Vista Business 或更高版本、Windows Server 2008，就可以免费使用 Microsoft 的 Web 服务器软件即 Internet Information Services(IIS)。要在 Windows XP 上安装 IIS，只需在控制面板中打开 Add/Remove Programs，单击 Add/Remove Windows Components 即可。图 14-3 展示了 Windows XP Professional 系统中的 Windows Component Wizard。

只需选中 IIS 旁边的复选框，然后单击 Next 按钮即可进行安装。在 Windows Vista 中，只需在控制面板中打开 Programs and Features 小程序，单击侧面板上的 Turn Windows Features On or Off 链接。然后展开 Internet Information Services，选择要安装的特性，必须选择 World Wide Web Services，如图 14-4 所示。安装时可能需要使用操作系统的安装光盘，以完成安装。

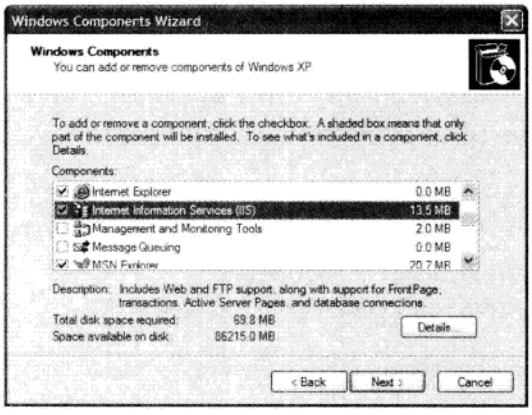

图 14-3

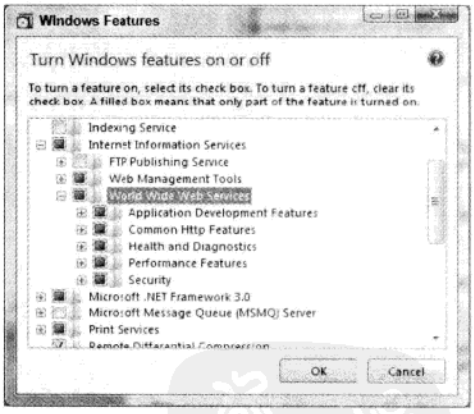

图 14-4

如果使用的不是以上几种操作系统，或者希望使用另一个 Web 服务器应用程序，就可以安装 Apache HTTP Server(www.apache.org)。这是一个开源的 Web 服务器，可以运行在多种操作系统上，如 Linux、Unix 和 Windows 等。

2. PHP

PHP 是一种流行的开源服务器端脚本语言。如果想运行 PHP 脚本，必须在计算机上安装 PHP。从 www.php.net 上可以下载到各种形式的 PHP(如二进制、Windows 安装向导和源

代码)。本例的 PHP 代码是用 PHP 5 编写的。

试一试 XMLHttpRequest 智能表单

打开文本编辑器，输入下列代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Form Field Validation</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
    <script type="text/javascript" src="httprequest.js"></script>
    <script type="text/javascript">
        function checkUsername()
        {
            var userValue = document.getElementById("username").value;

            if (userValue == "")
            {
                alert("Please enter a user name to check!");
                return;
            }

            var url = "formvalidator.php?username=" + userValue;

            var request = new HttpRequest(url, checkUsername_callBack);
            request.send();
        }

        function checkUsername_callBack(sResponseText)
        {
            var userValue = document.getElementById("username").value;

            if (sResponseText == "available")
            {
                alert("The username " + userValue + " is available!");
            }
            else
            {
                alert("We're sorry, but " + userValue + " is not available.");
            }
        }

        function checkEmail()
        {
```

```
            var emailValue = document.getElementById("email").value;

            if (emailValue == "")
            {
                alert("Please enter an email address to check!");
                return;
            }

            var url = "formvalidator.php?email=" + emailValue;

            var request = new HttpRequest(url, checkEmail_callBack);
            request.send();
        }

        function checkEmail_callBack(sResponseText)
        {
            var emailValue = document.getElementById("email").value;

            if (sResponseText == "available")
            {
                alert("The email " + emailValue + " is currently not in use!");
            }
            else
            {
                alert("I'm sorry, but " + emailValue + " is in use by another user.");
            }
        }
    </script>
</head>
<body>
    <form>
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td>
                    <a href="javascript: checkUsername()">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td>
                    <a href="javascript: checkEmail()">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
```

```
                Password:
            </td>
            <td>
                <input type="text" id="password" />
            </td>
            <td />
        </tr>
        <tr>
            <td class="fieldname">
                Verify Password:
            </td>
            <td>
                <input type="text" id="password2" />
            </td>
            <td />
        </tr>
        <tr>
            <td colspan="2" class="submit">
                <input type="submit" value="Submit" />
            </td>
            <td />
        </tr>
    </table>
  </form>
</body>
</html>
```

将这个文件保存在 Web 服务器的根目录中。如果使用 IIS 作为 Web 服务器，则将它保存为 c:\inetpub\wwwroot\validate_form.htm。如果使用的是 Apache，则将它保存在 htdocs 文件夹中：pathTohtdocs\htdocs\validate_form.htm 文件。

还需要将 httprequest.js 文件（HttpRequest 模块）和 formvalidator.php 文件（从代码下载中获得）放在与 validate_form.htm 文件相同的目录中。

现在，打开浏览器，导航到 http://localhost/formvalidator.php。如果 Web 服务器正常工作，就会显示文本“PHP is working correctly. Congratulations!”，如图 14-5 所示。

图　14-5

接下来，把浏览器导航到 http://localhost/validate_form.htm，结果如图 14-6 所示的页面。

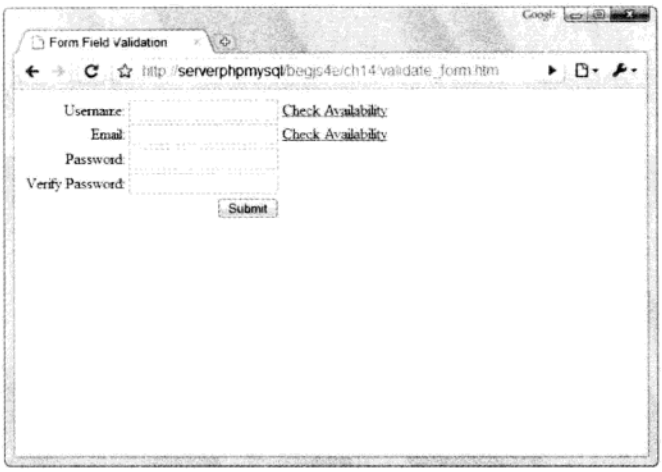

图 14-6

在 Username 字段中输入 jmcpeak，并单击旁边的 Check Availability 链接，弹出的警告框如图 14-7 所示。

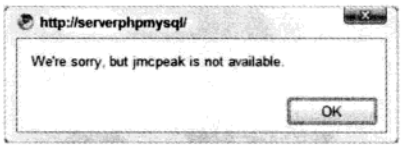

图 14-7

接下来，在 Email 字段中输入 someone@xyz.com，并单击旁边的 Check Availability 链接。这将再次弹出一个警告框，提示该 E-mail 已被使用。在这些字段中输入自己的用户名和 e-mail 地址，并单击相应的链接。警告框将可能提示，这些用户名和/或 e-mail 是有效的(注意：应用程序仅使用了用户名 jmcpeak 和 pwilton，电子邮箱 someone@xyz.com 和 someone @zyx.com)。

该 HTML 页面正文是一个简单的表单，其中的字段包含在一个表格中。每个表单字段占用一个表格行。前两行包含我们最感兴趣的字段 Username 和 Email。

```
<form>
    <table>
        <tr>
            <td class="fieldname">
                Username:
            </td>
            <td>
                <input type="text" id="username" />
            </td>
            <td>
                <a href="javascript: checkUsername()">Check Availability</a>
            </td>
        </tr>
        <tr>
            <td class="fieldname">
                Email:
```

```
            </td>
            <td>
                <input type="text" id="email" />
            </td>
            <td>
                <a href="javascript: checkEmail()">Check Availability</a>
            </td>
    </tr>
    <!-- HTML to be continued later -- >
```

表格的第一列包含了字段的文本标识符。第二列包含了<input/>元素本身。这些标记都有 id 属性，username 用于 Username 字段，email 用于 Email 字段。这将便于找到<input/>元素，并将文本输入其中。第三列包含了一个<a/>元素，该超链接使用 JavaScript:协议来调用 JavaScript 代码。本例在用户单击链接时，调用 checkUsername()和 checkEmail()函数。稍后讨论这两个函数。

表格的其余三行包含了两个密码字段和一个 Submit 按钮(智能表单目前不使用这些字段)。

```
            <!-- HTML continued from earlier -- >
            <tr>
               <td class="fieldname">
                   Password:
               </td>
               <td>
                   <input type="text" id="password" />
               </td>
               <td />
            </tr>
            <tr>
               <td class="fieldname">
                   Verify Password:
               </td>
               <td>
                   <input type="text" id="password2" />
               </td>
               <td />
            </tr>
            <tr>
               <td colspan="2" class="submit">
                   <input type="submit" value="Submit" />
               </td>
               <td />
            </tr>
        </table>
    </form>
```

该 HTML 页面中的 CSS 仅包含了两个 CSS 规则：

```
.fieldname
```

```
{
    text-align: right;
}

.submit
{
    text-align: right;
}
```

这些规则用于对齐字段，使表单看上去齐整有序。

如前所述，超链接是实现 Ajax 功能的关键，因为单击它们会调用 JavaScript 函数。第一个函数 checkUsername()获取用户在 Username 字段中输入的用户名，并使用该信息执行一个 HTTP 请求。

```
function checkUsername()
{
    var userValue = document.getElementById("username").value;

    if (userValue == "")
    {
        alert("Please enter a user name to check! ");
        return;
    }

    var url = "formvalidator.php?username=" + userValue;

    var request = new HttpRequest(url, checkUsername_callBack);
    request.send();
}
```

上面的代码使用 document.getElementById()方法找到<input/>元素，使用 value 属性以获取输入到文本框中的文本。接下来比较变量 userValue 与空字符串(" ")，确定用户是否输入了文本。如果文本框为空，则函数提示用户输入用户名，并禁止函数继续处理。如果代码不进行检查，应用程序就可能向服务器发送不必要的请求。

接下来构造向 PHP 应用程序发送请求的 URL，并把它赋给 url 变量。这个函数的最后步骤是创建一个 HttpRequest 对象，将 URL 和回调函数传递给 HttpRequest()构造函数，并发送请求。

当 HttpRequest 对象接收到服务器的完整响应时，执行函数 checkUsername_callBack()。该函数根据所请求的信息告知用户名是否有效。注意，服务器会返回两个可能的值 available 和 not available，因此仅须检查这两个可能的值之一。

```
function checkUsername_callBack(sResponseText)
{
    var userValue = document.getElementById("username").value;

    if (sResponseText == "available")
    {
        alert("The username " + userValue + " is available!");
    }
```

```
        else
        {
            alert("We're sorry, but " + userValue + " is not available.");
        }
    }
```

如果服务器的响应是 available，函数就告诉用户，所申请的用户名有效。否则警告框就提示，用户名已被占用。

检查电子邮件是否有效采用几乎相同的过程。checkEmail()函数获取输入到 Email 字段中的文本，并将该信息发送给服务器应用程序。

```
function checkEmail()
{
    var emailValue = document.getElementById("email").value;

    if (emailValue == "")
    {
        alert("Please enter an email address to check! ");
        return;
    }

    var url = "formvalidator.php?email=" + emailValue;

    var request = new HttpRequest(url, checkEmail_callBack);
    request.send();
}
```

函数 checkEmail_callBack()使用的逻辑与 checkUsername_callBack()函数相同，但它基于 Email 字段的值。

```
function checkEmail_callBack(sResponseText)
{
    var emailValue = document.getElementById("email").value;

    if (sResponseText == "available")
    {
        alert("The email " + emailValue + " is currently not in use! ");
    }
    else
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user. ");
    }
}
```

同样，如果该函数发现服务器的响应是 available，则通知用户，e-mail 地址可以使用。如果该地址不可用，则用另一个消息通知用户，e-mail 无效。

14.5 注意事项

使用 JavaScript 在客户端和服务器之间通信，极大地扩展了语言的功能。但是，这种强大功能是有代价的，其中最重要的两个问题是安全性和可用性。

14.5.1 安全性问题

安全性是目前 Internet 上的一个热门话题，Web 开发人员必须考虑 Ajax 上的安全限制。了解 Ajax 的安全性问题可以节省开发和调试时间。

1. 同源策略

早在 Netscape Navigator 2.0 时代，JavaScript 就不能访问不同来源的脚本或者文档。这是浏览器厂商采取的一个安全措施。否则，恶意代码的编写者就能在任何地方执行代码。同源策略提出，仅当两个页面的协议(HTTP)、端口号(默认为 80)和主机名相同时，它们才是同源的。

考虑以下两个页面：

- 页面 1 位于 http://www.site.com/folder/mypage1.htm
- 页面 2 位于 http://www.site.com/folder10/mypage2.htm

根据同源策略，这是两个同源页面。它们有相同的主机(www.site.com)，使用相同的协议(HTTP)，并在相同的端口上访问(这两个页面都没有指定端口，因此都使用 80 端口)。由于这是两个同源页面，因此一个页面中的 JavaScript 可以访问另一个页面。

再考虑下面的两个页面：

- 页面 1 位于 http://www.site.com/folder/mypage1.htm
- 页面 2 位于 https://www.site.com/folder/mypage2.htm

这两个页面是不同源的。它们的主机相同，但协议和端口号不同。页面 1 使用 HTTP 协议(端口号 80)，而页面 2 使用 HTTPS 协议(端口号 443)。这一差异虽然小，但足以使这两个页面不同源。因此，其中一个页面的 JavaScript 无法访问另一个页面。

这与 Ajax 技术有什么关系？因为 JavaScript 在 Ajax 中占很大比例。例如，由于同源策略的限制，XMLHttpRequest 对象将无法访问任何不同来源的文件或文档。这一问题很容易解决：使用一个同源的页面作为代理(proxy)，获取另一个非同源服务器上的数据。同源策略还影响到隐藏的 frame/iframe 技术，即使两个不同来源的页面位于同一个框架集中，JavaScript 也无法与这两个页面交互。

2. ActiveX

XMLHttpRequest 的一个弊端在于它是一个 ActiveX 控件，只适用于 Windows 上的 IE。尽管 IE 是当前市场占有率最高的浏览器，而且短期内似乎不会有什么改变，但这终归是个问题。过去几年中，围绕 ActiveX 产生了很多的安全性问题，特别是很多广告软件和间谍软件使用该技术安装在用户的计算机中。

由于人们对安全性问题越来越重视，所以 Microsoft 和用户在采取措施，以限制对 ActiveX 插件和对象的访问，使浏览器更加安全，并避免恶意的攻击。如果用户完全禁用 ActiveX，或者站点被标记为某个安全区域，就无法创建 ActiveX 对象，使基于 XMLHttp-Request 的 Ajax 应用程序无法正常工作。

14.5.2　可用性问题

Ajax 打破了传统 Web 应用程序和网页的模式。它允许开发人员创建以更传统的、非 Web 方式执行的应用程序。但是，Ajax 技术也存在一些缺点，Internet 已经存在了很多年，某些用户已经习惯了传统的网页。

因此，开发人员需要确保用户能够以希望的方式使用网页，且没有挫折感。

1. 浏览器的 Back 按钮

XMLHttpRequest 的一个优点是易于使用。我们只需创建对象，发送请求，等待服务器的响应。但是，这个对象有一个缺点：大多数浏览器都不能在其历史记录中保存使用该对象发出的请求，因此，XMLHttpRequest 实际上使浏览器的 Back 按钮失效。这是一些支持 Ajax 的应用程序或组件希望出现的副作用，但它可能给用户带来严重的可用性问题。

注意：

编写本书时，只有 IE8 能在其历史记录中保存使用 XMLHttpRequest 对象发出的请求，必须注意，Internet 和浏览器的存在时间都比 Ajax 长得多，用户在单击 Back 和 Forward 按钮时会期望执行某些操作。这些按钮失效会给用户带来挫折感，这是负责任的开发人员在设计应用程序时必须考虑的问题。

2. 使用 iframe 创建支持 Back 和 Forward 按钮的表单

使用旧式的 Ajax 技术可以避免使浏览器的导航按钮失效，即使用隐藏框架/内置框架进行客户端和服务器之间的通信。只有使用两个框架时，这种方法才能正常工作。其中一个框架是隐藏的，另一个框架是可见的。

隐藏可见技术包含 4 个步骤：

(1) 用户启动对隐藏框架的 JavaScript 调用。为此，用户可以单击可见框架中的链接，或者执行其他形式的交互操作。这个调用通常只是将隐藏框架重定向到另一个网页上。这个重定向会自动触发第二步。

(2) 向服务器发送请求，服务器会处理数据。

(3) 服务器将响应(一个网页)发送回隐藏框架。

(4) 浏览器在隐藏框架中加载网页，执行联系可见框架所需的 JavaScript 代码。

本节的例子将基于本章前面创建的表单验证程序，但使用隐藏框架来替代 XMLHttp Request 对象，实现浏览器与服务器之间的通信。在编写代码之前，先讨论从服务器上接收到的数据。

提示：

下面的例子不适用于 Safari 2，因为 Safari 2 不记录 iframe 的浏览历史。

3. 服务器响应

使用 XMLHttpRequest 对象从服务器获取数据时，服务器的响应仅包含很少几个词语。但本例的响应不同，它必须包含两项内容：

- 数据，必须是 HTML 格式。
- iframe 接收到 HTML 响应时联系父文档的机制。

下面的代码是响应 HTML 页面的示例：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Returned Data</title>
</head>
<body>
    <script type="text/javascript">
        //more code here
    </script>
</body>
</html>
```

这个简单的 HTML 页面在文档正文中包含一个<script/>元素。这个脚本块中的 JavaScript 代码将由 PHP 应用程序生成，用于调用可见框架中的函数 checkUsername_callBack()或者 checkEmail_callBack()，并将 available 或者 not available 作为它们的参数。因此，下面的 HTML 文档是 PHP 应用程序返回的有效响应。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Returned Data</title>
</head>
<body>
    <script type="text/javascript">
        top.checkUsername_callBack("available", "some_username");
    </script>
</body>
</html>
```

在这个示例响应中，用户名是有效的。因此，HTML 页面调用父窗口中的 checkUsername_callBack()函数，并给它传送字符串 available。另外，把要查找的用户名或 e-mail 发送回客户端，因为单击 Back 或 Forward 按钮时，客户端应用程序会显示正确的用户名或 e-mail。采用这种格式的响应，大部分 JavaScript 代码可以保持不变。

试一试 iframe 智能表单

这个修订过的智能表单的代码与前面 XMLHttpRequest 例子使用的代码非常类似，但有几个变化。打开文本编辑器，输入下列代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Form Field Validation</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }
        .submit
        {
            text-align: right;
        }
        #hiddenFrame
        {
            display: none;
        }
    </style>
    <script type="text/javascript">
        function checkUsername()
        {
            var userValue = document.getElementById("username").value;

            if (userValue == "")
            {
                alert("Please enter a user name to check!");
                return;
            }

            var url = "iframe_formvalidator.php?username=" + userValue;

            frames["hiddenFrame"].location = url;
        }

        function checkUsername_callBack(data, userValue)
        {
            if (data == "available")
            {
                alert("The username " + userValue + " is available!");
            }
            else
            {
                alert("We're sorry, but " + userValue + " is not available.");
            }
        }

        function checkEmail()
        {
```

```
            var emailValue = document.getElementById("email").value;
            if (emailValue == "")
            {
                alert("Please enter an email address to check!");
                return;
            }
            var url = "iframe_formvalidator.php?email=" + emailValue;
            frames["hiddenFrame"].location = url;
            }
            function checkEmail_callBack(data, emailValue)
            {
                if (data == "available")
                {
                    alert("The email " + emailValue + " is currently not in use! ");
                }
                else
                {
                    alert("We're sorry, but " + emailValue
                        + " is in use by another user. ");
                }
            }
        </script>
    </head>
    <body>
        <form>
            <table>
                <tr>
                    <td class="fieldname">
                        Username:
                    </td>
                    <td>
                        <input type="text" id="username" />
                    </td>
                    <td>
                        <a href="javascript: checkUsername()">Check
                                                    Availability</a>
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Email:
                    </td>
                    <td>
                        <input type="text" id="email" />
                    </td>
                    <td>
```

```
                <a href="javascript: checkEmail()">Check Availability</a>
            </td>
        </tr>
        <tr>
            <td class="fieldname">
                Password:
            </td>
            <td>
                <input type="text" id="password" />
            </td>
            <td />
        </tr>
        <tr>
            <td class="fieldname">
                Verify Password:
            </td>
            <td>
                <input type="text" id="password2" />
            </td>
            <td />
        </tr>
        <tr>
            <td colspan="2" class="submit">
                <input type="submit" value="Submit" />
            </td>
            <td />
        </tr>
    </table>
</form>
<iframe src="about:blank" id="hiddenFrame" name="hiddenFrame" />
</body>
</html>
```

将这个文件保存为 validate_iframe_form.htm，放在 Web 服务器的根目录下。另外从代码下载中找到 iframe_formvalidator.php 文件，并放在这个目录中。

打开 Web 浏览器，导航到 http://localhost/validate_iframe_form.htm。结果如图 14-8 所示。

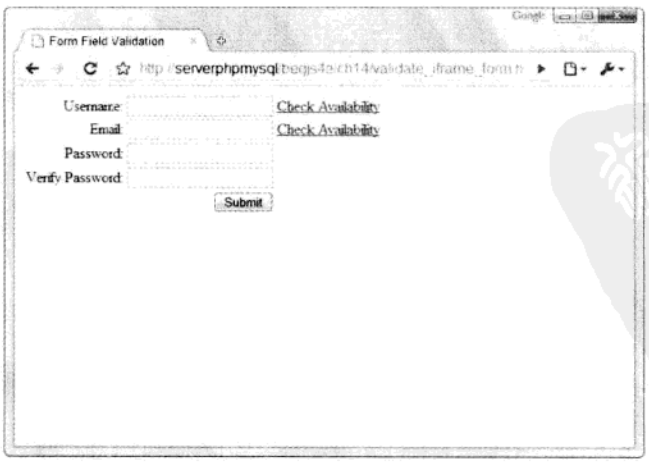

图　14-8

输入三个用户名和e-mail地址进行检查。关闭了最后一个警告框后，单击浏览器的Back按钮几次。前面输入的信息将按倒序出现。文本框中的文本并没有改变，但是警告框将显示所输入的用户名和 e-mail。也可以单击 Forward 按钮进行测试。

页面正文中的 HTML 没有变化，只是结束标记<form/>后添加了<iframe/>标记。

```
<iframe src="about:blank" id="hiddenFrame" name="hiddenFrame" />
```

该框架初始化为一个空白的 HTML 页面，其 name 和 id 属性都包含 hiddenFrame 的值。后面使用 name 属性的值，从 BOM 的 frames 集合中获取这个框架。接下来设置该框架的 CSS。

```
#hiddenFrame
{
    display: none;
}
```

该规则仅包含一个样式声明，以便隐藏<iframe/>。

注意：

通过 CSS 来隐藏<iframe/>，以便在需要调试服务器端应用程序时方便地显示<iframe/>。

下面是 JavaScript 代码。

```
function checkUsername()
{
    var userValue = document.getElementById("username").value;

    if (userValue == "")
    {
        alert("Please enter a user name to check! ");
        return;
    }

    var url = "iframe_formvalidator.php?username=" + userValue;

    frames["hiddenFrame"].location = url;
}
```

checkUsername()函数只有一点小小的修改：使用 iframe 替代 XMLHttpRequest，来发出请求。该函数首先获取 Username 文本框的值，并检查用户是否在文本框中输入了数据。如果文本框为空，则警告框会显示一条消息，告诉用户需要输入用户名。如果文本框不为空，函数就会构造请求 URL。最后一步使用 frames 集合和 location 属性把 URL 加载到隐藏的 iframe 中。

第二个函数 checkUsername_callBack()也略有修改。它现在接收两个参数：第一个包含 available 或者 not available；第二个包含在请求中发送的用户名。

```
function checkUsername_callBack(data, userValue)
{
    if (data == "available")
```

```
        {
            alert("The username " + userValue + " is available! ");
        }
        else
        {
            alert("We' re sorry, but " + userValue + " is not available. ");
        }
    }
```

该函数首先检查用户名是否有效，如果有效，就用警告框告诉用户，该用户名可用。否则，警告框告诉用户，用户名不可用。

搜索 e-mail 地址的函数与搜索用户名的函数非常类似。

```
function checkEmail()
{
    var emailValue = document.getElementById("email").value;

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return;
    }

    var url = "iframe_formvalidator.php?email=" + emailValue;

    frames["hiddenFrame"].location = url;
}

function checkEmail_callBack(data, emailValue)
{
    if (data == "available")
    {
        alert("The email "+ emailValue + " is currently not in use!");
    }
    else
    {
        alert("We're sorry, but "+ emailValue + " is in use by another user. ");
    }
}
```

checkEmail()函数获取文本框的值，并检查用户是否输入了数据。然后构造 URL，在 iframe 中加载该 URL。

checkEmail_callBack()函数的修改与 checkUsername_callBack()类似。该函数现在也接收两个参数，检查 e-mail 是否可用，并显示相应的信息。

2. 处理延迟

Web 浏览器与传统的应用程序一样，其用户界面(UI)会提示浏览器在执行某个任务。用户单击一个链接时，会运行一个跳动的动画，光标的旁边会出现一个沙漏图标(在 Windows 上)，状态栏通常会显示浏览器加载页面的进度。

这正是 Ajax 技术特别是 XMLHttpRequest 没有解决的另一个问题。但这个问题非常容易解决：只需添加 UI 元素，告诉用户某个任务正在运行，在该任务完成时删除 UI 元素。考虑下面的代码：

```
function requestComplete(sResponseText)
{
    //do something with the data here

    document.getElementById("divLoading").style.display = "none";
}

var myRequest = new HttpRequest("http://localhost/myfile.txt", requestComplete);
document.getElementById("divLoading").style.display = "block";//show that we're
loading
myRequest.send();
```

这段代码使用前面创建的 HttpRequest 模块请求文本文件。在发送请求前，在文档中获取 id 为 divLoading 的 HTML 元素。这个 元素告诉用户，数据正在加载。请求完成时，就隐藏 元素，让用户了解到加载过程已经完成。

给用户提供这个信息，可以让用户知道应用程序正根据他的请求执行某个操作。否则，用户单击某个对象后没有看到任何及时的反应，就可能会怀疑应用程序是否在正常工作。

3. 在 Ajax 失败时正常退出

在理想情况下，编写的代码每次都能正常运行。遗憾的是，支持 Ajax 的网页并不总是发挥了 Ajax 的长处，因为用户在浏览器上关闭了 JavaScript。

解决这个问题的唯一办法是：创建一个旧式的网页，使用旧式的表单、链接和其他 HTML 元素。然后，关闭这些 HTML 元素的默认行为，并使用 JavaScript 来添加相应的 Ajax 功能。考虑下面的超链接：

```
<a href="http://www.wrox.com" title="Wrox Publishing">Wrox Publishing</a>
```

这是一个非常普通的超链接。用户单击它时，就会进入 http://www.wrox.com。使用 JavaScript，可以覆盖这个操作，代之以我们自己的操作。

```
<a href="http://www.wrox.com" title="Wrox Publishing"
   onclick="return false;">Wrox Publishing</a>
```

这个功能的关键在于上述代码中突出显示的 onclick 事件处理程序，并返回 false。可以在该事件处理程序中执行任何代码，只是最后必须返回 false。这将告诉浏览器，单击超链接时不执行其默认动作。如果用户禁用了 JavaScript，则忽略 onclick 事件处理程序，超链接将恢复其正常行为。

作为一个经验法则，应先创建网页，再添加 Ajax。

14.6　小结

本章介绍了 Ajax 及其许多用途。

- 讨论了 XMLHttpRequest 对象及其在 IE 5、IE 6 与其他浏览器中的差异。学习了如何给服务器发出同步和异步请求，如何使用 onreadystatechange 事件处理程序。
- 建立了自定义的 Ajax 模块，使异步 HTTP 请求更便于编码。
- 在智能表单中使用新的 Ajax 模块，检查用户名和 e-mail 是否已被使用。
- 讨论了 XMLHttpRequest 对象如何使浏览器的 Back 和 Forward 按钮失效，以及如何使用隐藏的 iframe 来重建智能表单，发出请求，从而解决这个问题。
- 论述了 Ajax 的一些弊端：安全性问题和一些出乎意料的问题。

14.7　习题

附录 A 给出了这些问题的答案。

习题 1：

扩展 HttpRequest 模块，以便在已有异步请求的基础上增加同步请求。必须对代码进行一些调整，以便添加这个功能(提示：给模块创建 async 属性)。

习题 2：

如本章前面所述，智能表单可以改为不使用超链接。修改使用 HttpRequest 模块的表单，使之在用户提交表单时检查用户名和 e-mail 字段。如果用户名和 e-mail 已被占用，就使用表单的 onsubmit 事件处理程序并且取消提交操作，再通过习题 1 中更新的 HttpRequest 模块，来使用同步请求。只有用户名和 e-mail 已被占用，才需要警告用户，所以如果用户名和 e-mail 符合要求，就应返回 true。

第15章

JavaScript 框架

从本书的几个例子(尤其是后几章中的例子)可以看出，客户端开发的问题是开发人员必须考虑许多不同的 Web 浏览器。虽然可以编写事件驱动的代码或 Ajax 应用程序，但有时浏览器之间会出现不兼容的问题。

许多专业开发人员发现，跨浏览器的日常开发非常耗时、麻烦，于是开始开发框架或库，以辅助跨浏览器的开发。一些框架作者公开发布了其框架，其中的几个框架受到了许多人的欢迎，例如 jQuery、Prototype 和 MooTools。

本章介绍可从 Internet 上获得的 3 个 JavaScript 框架，学习如何使用它们，以极大地简化跨浏览器的开发。

但在开始之前，应注意：毫无疑问，JavaScript 框架的确可以节省开发时间、简化开发过程，但要进行跨浏览器的开发，必须牢固掌握 JavaScript 语言，深刻理解各种不同的浏览器的本质。框架和库只是临时使用的工具，而你掌握的知识是永久的。

15.1 选择框架

在过去几年中，网络上出现了许多 JavaScript 框架，它们一般可以分为两类：通用和专用。

通用框架的目标是创建一个统一的新 API，来执行一般任务，例如 DOM 操作和 Ajax 功能，以消除浏览器之间的差别。另一方面，专用框架注重的是专用功能，例如动画。所以用户需要确定自己想要什么样的框架，并据此选择一个框架。例如，如果仅需要开发动画，则可以选择 script.aculo.us 框架(http://script.aculo.us/)。

本章主要讨论通用框架，但即使是通用框架，其目标也各不相同。在确定使用哪个框架之前，应先查看框架的浏览器支持、文档和参与的团体。本章介绍的框架都存在了数年，比较稳定、流行，且与所有主流的现代浏览器(甚至比较旧的浏览器，例如 IE6)兼容。这些框架如下：

- jQuery：这个框架主要关注的是使用 CSS 选择器选择和使用 DOM 对象的能力。它还提供了一个插件架构以及一个附带的 UI 框架(http://jquery.com)。
- Prototype：这个框架提供了一个执行 Web 任务的简单 API。尽管 Prototype 提供了操作 DOM 的方式，但它的主要目标是提供类定义和继承，以增强 JavaScript 语言(http://www.prototypejs.org)。
- MooTools：这个框架的目标是提供一个便于完成常见任务的简单 API，且尽可能紧凑。与 Prototype 一样，MooTools 也旨在增强 JavaScript 语言——而不仅仅是简化 DOM 和 Ajax 的操作。它还包含一个轻型效果组件，这个组件最初称为 moo.fx (http://www.mootools.net)。

这 3 个框架仅是可以在网页中使用的框架范例，本章没有介绍的其他通用框架有：

- Yahoo! User Interface Framework (YUI)：这个框架从最基本的 JavaScript 工具，到完整的 DHTML 小组件，应有尽有，有一个团队专门负责开发 YUI(http://developer.yahoo.com/yui/)。
- Ext JS：这个框架最初是 YUI 的一个扩展，它提供了可定制的 UI 小组件，以创建内容丰富的 Internet 应用程序(http://www.extjs.com)。
- Dojo：这个工具集是围绕包系统来设计的，它的核心功能类似于其他框架(DOM 操作、事件规范化和 DHTML 小组件等)，但它还允许通过添加更多的包来添加更多的功能(http://www.dojotoolkit.org)。
- MochiKit：这个框架经过广泛的测试(根据 MochiKit 网站的数据，该框架通过了数百个测试)，还与其他 JavaScript 框架和库兼容(http://www.mochikit.com)。

15.2　入门

一旦选择了开发所使用的框架，就需要安装并验证该框架，之后就可以使用该框架工作了。

安装框架与在计算机上安装应用程序大不相同，它没有安装程序，其安装也不会改变系统的任何部分。基本上，只需下载一个文件，在网页上引用该文件即可。下面几节将介绍这个过程。

15.2.1　安装框架

首先需要获取框架的 JavaScript 文件。大多数框架至少有两个版本：压缩版和未压缩版。

- 压缩版：这些版本被压缩(删除了所有的注释和不必要的空白)，以使其尺寸尽可能小，当有人访问网页时，这种压缩版会使网页的下载比较快。但如果在文本编辑器中打开压缩版，压缩过程会使 JavaScript 代码难以看懂。但在生产环境下，进行压缩是一个合理的选择。
- 未压缩版：这些版本没有压缩，是正常的 JavaScript 代码文件，其中的注释和空白都没有改变。使用未压缩的 JavaScript 文件是非常不错的，因为它们比压缩文件更容易阅读，可以从中学到设计和开发这些框架的高手们的经验。但是，如果打算

铺开使用框架的网页，则必须确保下载并使用压缩版，因为压缩版的文件尺寸较小，下载速度较快。

1. 下载框架

与从网络上下载其他内容不同，jQuery、Prototype 和 MooTools 的可下载文件不使用压缩算法(例如 ZIP 或 RAR)，而是下载 JavaScript 文件本身。这些文件的扩展名是.js。

首先，获取 jQuery 的一个副本，如下所示：

(1) 打开浏览器，进入 http://jquery.com。

(2) 找到网站主页上的 Download 按钮。

(3) 单击 Download 按钮之前，选择需要的版本——生产版本(压缩版)或开发版本(未压缩版)。无论选择什么版本，jQuery 的工作都是相同的。编写本书时，jQuery 项目下载实际上封装在 Google 代码中，所以单击 Download 链接会退出 jquery.com。

(4) 下载 JavaScript 文件，将其保存在一个易于访问的位置。在 http://code.google.com/p/jqueryjs/downloads/list 上还有一个包含所有可下载文件的列表。

注意：

Wrox 的下载代码中提供了 jQuery 1.3.2 的生产版本。

(5) 现在把浏览器指向 Prototype 的下载页面 http://www.prototypejs.org/download。在这个页面上，可以下载 bleeding edge 版本，这个版本仍处于测试阶段，也可以下载最新的稳定版本。如果愿意，可以下载正在测试中的版本，但本书的示例使用稳定版本。

注意：

Wrox 的下载代码中提供了 Prototype 1.6.0.3 的稳定版本。

现在下载 MooTools。MooTools 的下载 URL 是 http://www.mootools.net/download。与 jQuery 一样，MooTools 的技术人员也为其框架提供了压缩版和未压缩版，还可以选择用 YUI Compressor 或 JSMin 压缩的版本。编写本书时的当前版本是 MooTools 1.2.3，尺寸最小的下载版本是 YUI Compressor 版，所以下载它。一般尺寸越小越好，因为下载时间较短。

注意：

Wrox 的下载代码中提供了 MooTool 1.2.3 的 YUI 压缩版本。

2. 在页面上添加框架

获得了每个框架的 JavaScript 框架文件后，就应把它们添加到网页上，以便使用它们。这只需在页面上添加一个 HTML 元素<script/>即可。

注意不要把所有框架添加到同一个页面上。让所有的框架无缝地协同工作，以同时获得所有框架的功能肯定是非常吸引人的，可惜，每个框架都各有特色，所以应给每个框架创建一个单独的 HTML 页面。

下面的 HTML 页面 ch15_examp1_jq.htm 说明了如何把 jQuery 或其他框架添加到网页上：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 1 jQuery</title>
    <script type="text/javascript" src="jquery-1.3.2.min.js"></script>
</head>
<body>

</body>
</html>
```

只要使用<script/>元素的 src 属性，将其值设置为 JavaScript 文件的路径即可。在这个示例中，jquery-1.3.2.min.js 文件与 HTML 文档位于同一目录中。src 属性的值必须正确，否则浏览器就找不到框架文件，也就不能下载和加载它——这意味着代码不能工作。

输入这个页面的代码并保存。然后为 MooTools 和 Prototype 分别创建类似的页面，命名为 ch15_examp1_mt.htm 和 ch15_examp1_p.htm。不要忘了把<script/>元素的 src 属性指向对应的 JavaScript 文件。

15.2.2　测试框架

下载了每个库，并把它们添加到各自的页面上后，就应进行一个快速测试，以确保它们能正常工作。由于每个库都不相同，所以必须为每个库执行类似的测试。从 jQuery 开始。

1. 测试 jQuery 安装

jQuery 的核心是$()函数(称为 jQuery 函数)，它返回 jQuery 对象。jQuery 函数非常强大，因为它允许通过传送 CSS 选择器来选择元素，通过传送 HTML 来创建元素，通过传送要添加功能的 DOM 对象，来把 jQuery 功能封装到 DOM 对象中。

为了演示 jQuery 函数和它返回的对象，假定在页面加载时要运行一些代码。在普通的 JavaScript 代码中，可以指定一个函数来处理 window 的 load 事件。jQuery 中对应的功能却完全不同：

```
function document_ready()
{
    alert("Hello, jQuery World!");
}

// the normal way
onload = document_ready;

// the jquery way
$(document).ready(document_ready);
```

注意上述代码中的最后一行，它调用 jQuery 函数，并给它传送了 DOM 对象 document。jQuery 函数返回一个 jQuery 对象，其中 ready()是一个方法。给 jQuery 函数传送 DOM 对象，实际上是创建一个新对象，并把它封装在 DOM 对象中。下面的代码较容易理解：

```
var jDocument = $(document);
jDocument.ready(document_ready);
```

这段代码的作用与上一段相同，只是把 jQuery 对象包含在 jDocument 变量中，以便加以重用。

注意：

jQuery 对象不能用来替代 DOM 对象。在上面的示例中，jQuery 函数返回的对象与传送给该函数的 document 对象完全不同。

这种重用——分配一个变量，并在以后重用——非常适用于 jQuery 对象，jQuery 还增加了方法链调用的功能。如果读者不熟悉方法链，可考虑下面的代码：

```
$(document.body).attr("bgColor","yellow").html("<h1>Hello,jQuery World</h1>");
```

jQuery 对象有许多方法，几乎所有的方法都返回当前的 jQuery 对象。因此，可以一个接一个地调用方法，从而提高可读性，且代码较少。这行代码使用了 jQuery 函数，并给该函数传送了 DOM 对象 document.body。在调用 jQuery 函数之后，马上又调用了 attr()方法，把 bgColor 属性设置为黄色。由于 jQuery 函数返回一个 jQuery 对象，该对象封装了对 DOM 对象 document.body 的引用，所以属性在 DOM 对象 document.body 上设置。

在调用 attr()方法之后，又调用另一个方法，因为 attr()返回同一个封装了 DOM 对象 document.body 的 jQuery 对象。第二个方法是 html()，它接受一个写入页面的 HTML 字符串参数。html()把 HTML 写入页面的方式与 document.write()不同。jQuery 的 html()方法使用 DOM 属性 innerHTML 在 DOM 对象(这里是 document.body)内部设置 HTML。

从这个示例可以看出，方法链是在一个对象上执行多个任务的简便方法，jQuery 就建立在这个概念的基础之上。

使用这段代码测试 jQuery 安装。打开 ch15_examp1_jq.htm，在第二个<script/>元素上添加以下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 2 jQuery</title>
    <script type="text/javascript" src="jquery-1.3.2.min.js"></script>
    <script type="text/javascript">
    function document_ready($)
    {
        $(document.body).attr("bgColor", "yellow")
            .html("<h1>Hello, jQuery World!</h1>");
    }

    $(document).ready(document_ready);
    </script>
</head>
<body>
```

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

把这段代码保存为 ch15_examp2_jq.htm，在浏览器中打开它。结果应如图 15-1 所示。

图　15-1

如果页面的背景色不是黄色，就看不到 Hello World 消息，这表示安装可能有问题。确保 jQuery 的.js 文件与 HTML 文件在同一目录下。

下面测试 Prototype 安装，不必担心，我们不久之后就会再次研究 jQuery。

2. 测试 Prototype 安装

Prototype 库的最大一部分是其 DOM 扩展。与 jQuery 一样，Prototype 也提供了许多有帮助的工具函数，来简化 DOM 编程；它甚至有自己的$()函数(与 jQuery 不同，Prototype 没有给这个函数指定特殊的名称，只是简单地称为美元函数)。

Prototype 的$()函数仅接受元素 id 属性值或 DOM 元素对象作为参数，来为 DOM 对象选择和添加额外的功能。

注意：

Prototype 不允许使用 CSS 选择器来选择元素，稍后将讨论。

与 jQuery 的另一个区别是，Prototype 没有与 ready()对应的方法来替代 window.onload，而可以使用 Prototype 添加到浏览器和 DOM 中的许多扩展之一：Event.observe()方法。

```
function window_onload()
{
    alert("Hello, Prototype World!");
}
Event.observe(window, "load", window_onload);
```

Event.observe()方法接受 3 个参数：第一个是要在其中添加事件处理程序的 DOM 或 BOM 对象，第二个是要处理的事件，第三个是事件触发时要调用的函数。Event.observe()方法可以用于给任何能处理事件的 DOM 或 BOM 对象添加事件处理程序。本章后面将介

绍这个方法，以及在 Prototype 中处理事件的其他方式。

与 jQuery 一样，可以在$()函数创建的封装对象上把方法调用链接起来，但其方法名称比 jQuery 繁琐。

```
function window_onload()
{
     $(document.body).writeAttribute("bgColor", "yellow").
         insert("<h1>Hello, Prototype World!</h1>");
}

Event.observe(window, "load", window_onload);
```

window_onload()函数的新函数体把页面的背景改为黄色，并把 HTML 添加到页面上。writeAttribute()方法设置一个属性值，它接受两个参数：属性名和属性值。下一个方法调用是 insert()，它把所提供的内容添加到 document.body 元素上，document.body 元素由传送给最初的$()函数的 DOM 对象指定。

使用这段代码测试 Prototype 安装。打开 ch15_examp1_p.htm，在第二个<script/>元素上添加下列代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 2 Prototype</title>
    <script type="text/javascript" src="prototype-1.6.0.3.js"></script>
    <script type="text/javascript">
    function window_onload()
    {
        $(document.body).writeAttribute("bgColor", "yellow")
            .insert("<h1>Hello, Prototype World!</h1>");
    }

    Event.observe(window, "load", window_onload);
    </script>
</head>
<body>

</body>
</html>
```

把这段代码保存为 ch15_examp2_p.htm，在浏览器中打开它。结果应如图 15-2 所示。否则，确保 Prototype 的 JavaScript 文件与 HTML 文件在同一目录下。

稍后会深入探讨 Prototype。下面测试最后一个框架 MooTools，确保它的安装是正确的。

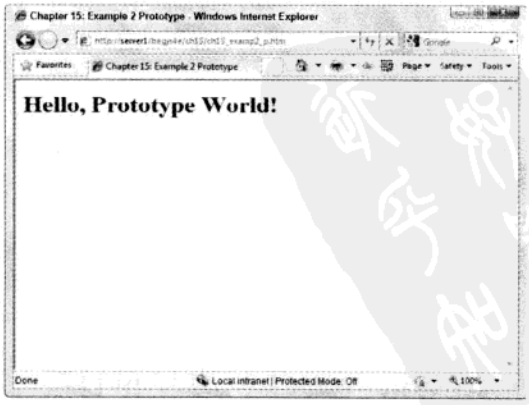

图 15-2

3. 测试 MooTools 安装

测试 MooTools 安装看起来与测试 jQuery 和 Prototype 类似。与这两个框架一样，MooTools 也定义了美元函数，其功能与 Prototype 的美元函数类似。

```
$("myDiv")
```

与 Prototype 一样，MooTool 的美元函数也接受一个包含元素 id 或 DOM 元素的字符串参数，并返回该元素或 DOM 对象和一组扩展的方法。其中一个方法是 addEvent()。这个方法接受两个参数：第一个是要监控的事件，第二个是事件触发时调用的函数。MooTools 框架给 window 和 document 对象添加了 addEvent()方法。只要调用 addEvent()方法，就可以添加一个 domready 事件处理程序，如下所示：

```
function window_domready()
{
    alert("Hello, MooTools World!");
}

window.addEvent("domready", window_domready);
```

domready 事件同样由 MooTools 框架添加，它在 DOM 加载完毕时触发。

与 jQuery 和 Prototype 一样，也可以把 MooTools 方法链接起来，在一个元素上执行多个操作，其代码要比不使用框架时少。例如，下面的代码：

```
function window_domready()
{
    $(document.body).setProperty("bgColor", "yellow")
        .set("html", "<h1>Hello, MooTools World!</h1>");
}

window.addEvent("domready", window_domready);
```

window_domready()的新函数体把 document.body 对象传送给美元函数，然后使用 setProperty()方法，把 bgColor 属性设置为黄色。接着调用 set()方法，把"html"字符串作为第一个参数传送，将第二个参数值设置为页面正文内部的 HTML。

使用这段代码测试 MooTools 安装。打开 ch15_examp1_mt.htm 文件，修改代码，如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 2 MooTools</title>
    <script type="text/javascript" src="mootools-1.2.3-core-yc.js"></script>
        <script type="text/javascript">
        function window_domready()
        {
            $(document.body).setProperty("bgColor", "yellow")
```

```
                .set("html", "<h1>Hello, MooTools World!</h1>");
        }

        window.addEvent("domready", window_domready);
    </script>
</head>
<body>

</body>
</html>
```

把这段代码保存为 ch15_examp2_ mt.htm，在浏览器中加载它。页面应如图 15-3 所示。

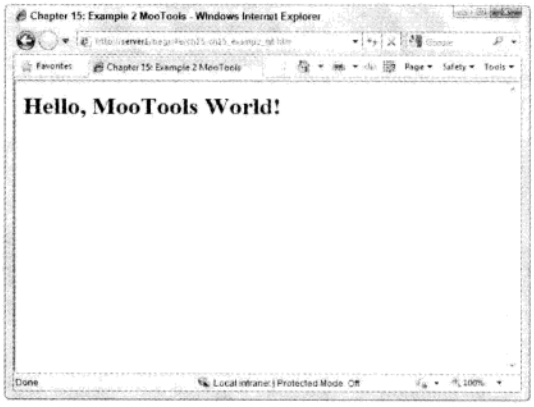

图 15-3

如果结果如图 15-3 所示，表示 MooTools 安装正确。否则，确保 MooTools 的 JavaScript 文件与 HTML 页面在同一目录中。

现在 jQuery、Prototype、MooTools 已安装完毕，并能正确运行。可以开始深入探讨这些框架了，以更好地理解如何使用它们，以从中获益。

下面就开始研究 jQuery、Prototype 和 MooTools。

15.3 深入分析 jQuery

前面用于测试 jQuery 安装的代码让我们初步了解了该框架能做什么。可以看出，修改属性值以及给页面添加 HTML 都非常容易，而且所有操作都非常简单。无论是创建 HTML 元素，并将它们添加到页面上，还是向服务器发出 Ajax 调用，jQuery 都提供了简单的方式。

jQuery 是以 DOM 为中心的框架，但要进行任何 DOM 操作，都必须先定位和检索特定的元素。

15.3.1 选择元素

W3C DOM 标准提供了 getElementById()和 getElementsByTagName()方法，来查找和检

索 DOM 中的元素。这些方法都非常好用，但它们最明显的缺点是限制了选择元素的方式。只能根据 id 属性值或标记名称来选择元素。用户有时希望根据元素的 CSS 类名或它们与其他元素的关系来选择元素。

而 jQuery 在选择元素方面就很有优势。你可以使用 CSS 选择器，根据元素的 CSS 类名、它们与其他元素的关系、id 属性值或标记名来选择元素。下面的代码很简单：

```
$("a")
```

这行代码选择页面中所有的<a/>元素，并把它们返回到一个数组中。由于是一个数组，所以可以使用 length 属性确定选择了多少个元素，如下所示：

```
alert($("a").length);
```

jQuery 简化了 DOM 操作，由于采用了这种设计思想，所以可以一次修改多个元素。例如，假定建立了一个网页，其中有超过 100 个链接。忽然某一天希望把这些链接的 target 属性设置为_blank，在一个新窗口中打开它们。这是一项很艰巨的任务，但使用 jQuery 却可以很容易完成。因为可以通过调用$(“a”)，检索出文档中所有的<a/>元素，再调用 attr()方法设置 target 属性。如下面的代码所示：

```
$("a").attr("target", "_blank");
```

调用$("a")会得到一个 jQuery 对象，但这个对象也兼作数组。在这个 jQuery 对象上调用的任何方法都会对数组中的所有元素执行相同的操作。执行这行代码，会把页面上每个<a/>元素的 target 属性设置为_blank，甚至不需要使用循环！

下一种选择元素的方式是使用 CSS id 语法，即在元素的 id 属性值前加上英镑符号(#)。使用 DOM 的 getElementById()方法可以执行相同的任务，但使用 jQuery 函数需要的输入量较少，还会返回一个 jQuery 对象。

```
$("#myDiv")
```

即使知道使用这行代码只能选择一个元素，也可以使用 length 属性确保在页面上找到了元素。如果找不到该元素，length 就是 0。可以使用相同的逻辑根据 CSS 类名来选择元素。只需把 CSS 选择器传送给 jQuery 函数，如下所示：

```
$(".myCssClass")
```

因为 jQuery 使用 CSS 选择器选择元素，所以很容易根据元素的层次结构选择元素。考虑下面的 HTML：

```
<p>
    <div>Div 1</div>
    <div>Div 2</div>
    <span>Span 1</div>
</p>
<span>Span 2</span>
```

这段 HTML 代码定义了一个<p/>元素，它包含两个 元素和一个元素。在

<p/>元素的外部是另一个元素。如果使用传统的 DOM 方法和属性，就必须编写几行代码来识别和检索<p/>元素内部的元素，而使用 jQuery，只需编写一行代码即可：

```
$("p > span")
```

这行代码使用 CSS 选择器语法 parent > child，选择出<p/>元素的所有子元素。

注意：

Internet Explorer 6 没有内置对这种 CSS 选择器的支持，但如果在该浏览器中运行上述代码，会发现这行代码依然有效。JQuery 和其他框架都支持非常广泛的 CSS 选择器——有时浏览器并不支持这些 CSS 选择器。要查阅支持的 CSS 选择器的完整列表，可查看框架的网站。

jQuery 函数还允许在一个函数调用中使用多个选择器。如下面的代码：

```
$("a, #myDiv, .myCssClass, p > span")
```

用逗号分隔每个选择器即可。这行代码会检索出所有<a/>元素、id 为 myDiv 的元素、CSS 类为 myCssClass 的元素以及<p/>元素的所有子元素。如果希望把这些元素的文本颜色设置为红色，就可以使用如下代码：

```
$("a, #myDiv, .myCssClass, p > span").attr("style", "color:red; ");
```

这并非修改元素样式的最佳方法。实际上，jQuery 提供了许多修改元素样式的方法。

注意：

要查看支持的选择器的完整列表，请参阅 http://docs.jquery.com/Selectors。

15.3.2 修改样式

与 DOM 一样，通过修改单个 CSS 属性或修改元素的 CSS 类，就可以改变元素的样式。为此，需要使用 css()方法。这个方法接受两个参数：属性名和属性值。

```
$("#myDiv").css("color", "red");
```

这行代码把 color 属性设置为红色，所以把文本的颜色改为红色。传送给 css()方法的属性名可以使用样式表格式或脚本格式。这表示，如果希望修改元素的背景色，可以给该方法传送 background-color 或 backgroundColor，如下所示：

```
$("#myDiv").css("background-color", "yellow"); // CORRECT!!!
$("#myDiv").css("backgroundColor", "yellow"); // CORRECT, TOO!!!
```

但一般情况下，如果想修改元素的样式(不是基于动画的)，最好改变元素的 CSS 类，而不是单个样式属性。这样可以使样式内容依然位于其所属的样式表中。

1. 使用多个 CSS 类

jQuery 对象为操作元素的 className 属性提供了多个方法。在介绍它们之前，要知道

一个元素有多个 CSS 类是合法的。分析下面的 HTML：

```
<div class="myClass1 myClass2">
    My div with two CSS classes!
</div>
```

为了把两个或多个 CSS 类应用于一个元素，只需用空格分隔类名。在这个 HTML 段中，把两个 CSS 类 myClass1 和 myClass2 的样式应用于 元素。之所以介绍这个概念，是因为 jQuery 中操作类名的方法是建立在这个概念的基础之上的。

第一个方法 addClass()把指定的 CSS 类添加到元素上。

```
$("#myDiv").addClass("myClass1")
    .addClass("myClass2");
```

这段代码把 myClass1 和 myClass2 两个 CSS 类添加到元素上。可以缩短这段代码，方法是在一次调用中把两个类名传送给方法 addClass()：

```
$("#myDiv").addClass("myClass1 myClass2");
```

确保用空格分隔开类名。需要从元素中删除某个类或某些类时，可以使用 removeClass()方法：

```
$("#myDiv").removeClass("myClass2");
```

这行代码从元素中删除了 CSS 类 myClass2，这样就只有 myClass1 是应用于该元素的 CSS 类。传送给 removeClass()方法的参数是可选的，如果不给该方法传送任何参数，就会删除所有 CSS 类。下面的代码就删除了元素中的所有 CSS 类：

```
$("#myDiv").removeClass();
```

2. 使用 toggleClass 方法

下一个方法是 toggleClass()。与前面的 CSS 类方法不同，这个方法只接受一个类名作为参数。它检查指定的类是否存在，如果存在，就删除它。如果该类不存在，就把该类添加到元素上。下面是一个示例：

```
$("#myDiv").addClass("myClass1 myClass2")
    .toggleClass("myClass2")
    .toggleClass("myClass2");
```

这段代码首先把 CSS 类 myClass1 和 myClass2 添加到元素上。第一个 toggleClass()调用从元素中删除了 myClass2，第二个 toggleClass()调用又把 myClass2 添加回来。需要在元素中添加或删除指定的类时，toggleClass()方法很方便。例如，下面的代码是很普通的旧 JavaScript 和 DOM 代码，它根据事件类型添加或删除指定的 CSS 类：

```
if (e.type == "mouseover")
{
    eSrc.className = "mouseover";
}
```

```
else if (e.type == "mouseout")
{
    eSrc.className = "";
}
```

使用 toggleClass()方法，可以把这段代码缩减为下面的 4 行代码：

```
if (e.type == "mouseover" || e.type == "mouseout")
{
    $(eSrc).toggleClass("mouseover");
}
```

使用 toggleClass()方法可以减少代码量，从而提高效率和下载速度，而这常常是我们追求的一个终极目标。

3. 使用 hasClass 方法

最后一个 CSS 类方法是 hasClass()，它根据是否把指定的 CSS 类应用于元素，而返回 true 或 false。

```
$("#myDiv").addClass("myClass1 myClass2")
    .hasClass("myClass1");
```

与 toggleClass()一样，hasClass()方法也只接受一个类名作为参数。在这段代码中，hasClass()返回 true，因为元素应用了 CSS 类 myClass1。这个示例没有什么实际用处，因为我们知道给元素指定了什么类，它仅用于演示如何使用 hasClass()方法。

注意：

因为 hasClass()返回 true 或 false，所以在调用它之后，就不能再链接其他方法了。

JQuery 简化了其他类型的 DOM 操作，后面介绍在 DOM 中创建、添加和删除对象时你将体会到这一点。

15.3.3 创建、追加和删除元素

第 12 章介绍了如何创建元素以及如何把元素追加到页面上。下面的代码复习这些内容：

```
function window_onload()
{
    var a = document.createElement("a");
    a.id = "myLink";
    a.setAttribute("href", "http://jquery.com");
    a.setAttribute("title", "jQuery's Website");

    var text = document.createTextNode("Click to go to jQuery' s website");

    a.appendChild(text);
    document.body.appendChild(a);
}
```

```
onload = window_onload;
```

这段代码定义了 window_onload()函数，浏览器加载完页面时就调用这个函数。执行 window_onload()时，它会创建一个<a/>元素，给它指定 id，并设置 href 和 title 属性。接着创建一个文本节点，并把它赋予 text 变量。最后把文本节点追加到<a/>元素中，再把<a/>元素追加到文档的<body/>元素中。

1. 创建元素

这段代码在技术上没有什么错误，它是标准的 DOM 元素创建、填充和插入代码。但这段代码很长，比较繁琐。使用 jQuery 完成相同的操作时，输入量比较少，如下面的代码所示：

```
function document_ready()
{
    var a = $(document.createElement("a"));
    $(document.body).append
    (
        a.attr("id", "myLink")
        .attr("href", "http://jquery.com")
        .attr("title", "jQuery's Website")
        .text("Click here to go to jQuery's website.")
    );
}

$(document).ready(document_ready);
```

下面分析 document_ready()函数，以更清晰地理解其执行过程。首先用 document.create Element()方法创建<a/>元素：

```
var a = $(document.createElement("a"));
```

这行代码不是简单地把元素赋予一个变量，而是把它传送给 jQuery 函数，以便使用 jQuery 方法给它填充属性和文本。接着给 jQuery 函数传送 document.body 对象，并调用 append()方法：

```
$(document.body).append
(
```

2. 追加元素

append()方法类似于 DOM 的 appendChild()方法，因为它也给 DOM 对象追加子节点。append()方法接受 DOM 对象、jQuery 对象或包含 HTML 内容的字符串作为参数。无论为 append()传送什么参数，append()都会把参数的内容追加到 DOM 对象中。在这段代码中，传送的是 jQuery 对象，它引用了前面创建的<a/>元素，并给元素指定了属性。

```
a.attr("id", "myLink")
.attr("href", "http://jquery.com")
.attr("title", "jQuery's Website")
```

指定了 id、href 和 title 属性后，又给链接添加了文本，可通过两种方法添加文本：可以使用 append()方法，并给它传送文本；也可以使用 text()方法。这两个方法会得到相同的结果，但这里使用了 text()，仅是因为我们还没有使用过它。

```
        .text("Click here to go to jQuery's website.")
);
```

在 jQuery 中，可以把方法调用链接在一起的关键是大多数方法都返回 jQuery 对象，而这些方法都是在这个 jQuery 对象上调用的。所以 text()方法返回 jQuery 对象，该 jQuery 对象引用了<a/>元素对象，该对象传送给 append()方法，而 append()方法是在 jQuery 对象上调用的，该 jQuery 对象又引用了 document.body 对象。

可以用另外两种方式重写这段代码。首先把这段代码重写为：

```
function document_ready()
{
    var a = $(document.createElement("a"))
          .attr("id", "myLink")
          .attr("href", "http://jquery.com")
          .attr("title", "jQuery's Website")
          .text("Click here to go to jQuery's website.");

    $(document.body).append(a);
}
```

这段代码的执行结果与前面相同，但更容易理解。但如果希望再少写一些代码，可以输入下面的代码：

```
function document_ready()
{
    $(document.body).append($(document.createElement("a"))
        .attr("id", "myLink").attr("href", "http://jquery.com")
        .attr("title", "jQuery's Website")
        .text("Click here to go to jQuery's website."));
}
```

这段代码更加简洁。如果在 Internet 上查看基于 jQuery 的代码，会发现它们大多属于这种类型(这也是作者使用的代码类型)，但可读性略差。重要的是应以自己觉得舒服的方式编码。如果把几个方法链接在一起，可能会令人迷惑。因此，后面的代码示例把某个方法调用放在单独一行上。

3. 删除元素

使用 jQuery 从 DOM 中删除元素比使用传统的 DOM 方法容易许多。使用传统的 DOM 方法至少要在 DOM 树中找到两个元素：要删除的元素及其父元素。而使用 jQuery，只需找到要删除的元素，再调用 remove()方法即可，如下所示：

```
$("#myLink").remove();
```

这行代码找到前一个代码示例创建的<a/>元素，从 DOM 中删除它。调用 empty()方法还可以删除某个元素的所有子节点。

```
$(document.body).empty();
```

这行代码会清空<body/>元素，因此删除了页面中的所有内容。

我们所做的大多数 DOM 修改都对应于用户执行的操作，例如，用户把鼠标移动到某个元素上或者在页面的某个位置单击。所以，我们必须处理对应的事件。

15.3.4　jQuery 事件模型和事件的处理

所有 jQuery 对象都有一个 bind()方法，用于把事件处理程序赋予指定的事件。

```
function myButton_click(event)
{
    alert("You clicked me! ");
}

$("#myButton").bind("click", myButton_click);
```

这段代码指定 myButton_click()函数来处理 id 值为 myButton 的元素的 click 事件。

jQuery 把一个 jQuery.Event 对象传给处理事件的函数。由于 IE 和 W3C DOM 事件模型之间的差异很大，所以 jQuery 的创建者 John Resig 决定把这两个事件模型合并到自己的事件模型上，该事件模型基于 W3C 事件模型的 Event 和 MouseEvent 对象。这意味着，我们不需要检查 window.event 或使用 IE 事件模型的任何属性，jQuery.Event 对象提供了与 window.event 相同的信息，使用 W3C 的 Mouse 和 MouseEvent 对象的属性就可以获得此信息。

也就是说，我们有一个统一的对象可以在所有支持的浏览器上工作，来处理事件。为了说明这一点，可以编写下面的代码，它可以在所有支持的浏览器上工作：

```
function myButton_click(event)
{
    alert(event.target.tagName + " clicked at X: " + event.pageX
        + " and Y: " + event.pageY);
}

$("#myButton").bind("click", myButton_click);
```

图 15-4 显示了这段代码在 IE 中的执行结果，图 15-5 显示了这段代码在 Firefox 中的执行结果。

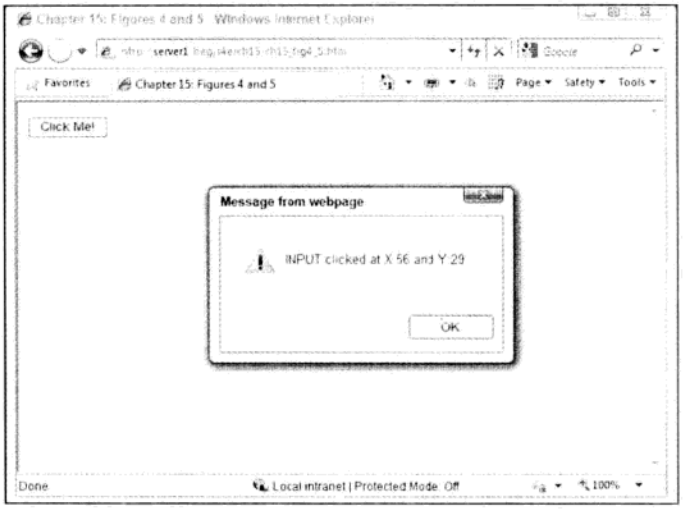

图 15-4

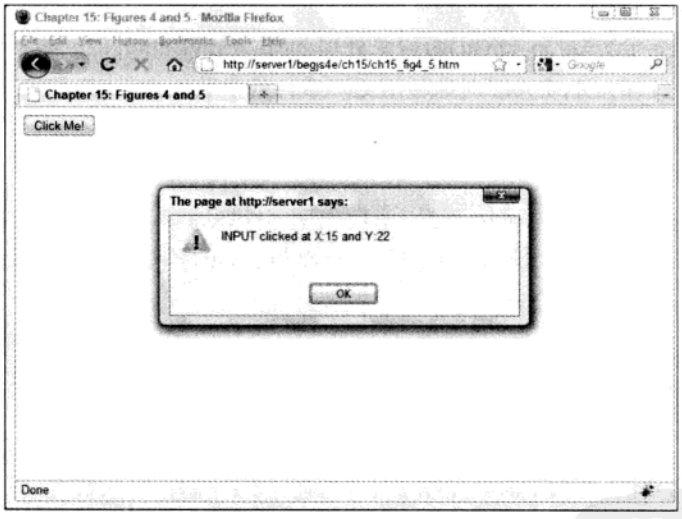

图 15-5

由于有了 jQuery.Event 对象，所以使用 W3C 模型编码，它可以在所有浏览器上工作。

注意：

要查看支持的事件的完整列表，请参阅 jQuery 网站 http://docs.jquery.com/Events。

15.3.5 用 jQuery 重写 DHTML 工具栏

前面学习了如何在 DOM 中检索元素，如何通过添加和删除类来改变元素的样式，如

何在页面上添加和删除样式，以及如何通过 jQuery 使用事件。

下面运用这些新知识来重构第 12 章的 DHTML 工具栏(可作为第 12 章末尾第 2 题的答案)。

试一试　**用 jQuery 重构工具栏**

打开文本编辑器，输入下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 3 with jQuery</title>
    <style type="text/css">
    .tabStrip
    {
       background-color: #E4E2D5;
       padding: 3px;
       height: 22px;
    }

    .tabStrip-tab
    {
       float: left;
       font: 14px arial;
       cursor: pointer;
       padding: 2px;

       border: 1px solid transparent;
    }

    .tabStrip-tab-hover
    {
       border-color: #316AC5;
       background-color: #C1D2EE;
    }

    .tabStrip-tab-click
    {
       border-color: #facc5a;
       background-color: #f9e391;
    }
    </style>
    <script type="text/javascript" src="jquery-1.3.2.min.js"></script>
    <script type="text/javascript">
    var currentNum = 0;
    function handleEvent(e)
    {
        var el = $(e.target);

        if (e.type == "mouseover" || e.type == "mouseout")
```

```
        {
            if (el.hasClass("tabStrip-tab") && !el.hasClass("tabStrip-tab-click"))
            {
                el.toggleClass("tabStrip-tab-hover");
            }
        }

        if (e.type == "click")
        {
            if (el.hasClass("tabStrip-tab-hover"))
            {
                var id = e.target.id;
                var num = id.substr(id.lastIndexOf("-") + 1);

                if (currentNum != num)
                {
                    deactivateTab();

                    el.toggleClass("tabStrip-tab-hover")
                        .toggleClass("tabStrip-tab-click");
                    showDescription(num);
                    currentNum = num;
                }

            }
        }
    }

    function showDescription(num)
    {
        var div = $(document.createElement("div"));

        $("#descContainer").append
        (
            div.attr("id", "tabStrip-desc-" + num)
                .text("Description for tab " + num)
        );
    }

    function deactivateTab()
    {
        var descEl = $("#tabStrip-desc-" + currentNum);

        if (descEl.length > 0)
        {
            descEl.remove();

            $("#tabStrip-tab-"+ currentNum).toggleClass("tabStrip-tab-click");
        }
    }

    $(document).bind("click mouseover mouseout", handleEvent);
    </script>
</head>
```

```
<body>
    <div class="tabStrip">
        <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
        <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
        <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
    </div>
    <div id="descContainer"></div>
</body>
</html>
```

保存为 ch15_examp3.htm，把它与 ch12_q2.htm 进行比较。在 jQuery 支持的任意浏览器中打开 ch15_examp3.htm，注意它的行为与 ch12_q2.htm 相同。

在这个新版本中，第一个重要变化是 CSS，因为 jQuery 的 CSS 类方法比简单地修改 className 属性更适于添加和删除 CSS 类。这种方法可以给样式建立层次，因此不必使用 className 属性完全改变元素的样式，而只需添加或删除一个样式层次。这减少了需要编写的 CSS 代码量。

```
.tabStrip-tab
{
    float: left;
    font: 14px arial;
    cursor: pointer;
    padding: 2px;
    border: 1px solid transparent;
}

.tabStrip-tab-hover
{
    border-color: #316AC5;
    background-color: #C1D2EE;
}

.tabStrip-tab-click
{
    border-color: #facc5a;
    background-color: #f9e391;
}
```

第一个显著的变化是删除了.tabStrip div 选择器，其样式属性移到 tabStrip-tab 类中。第二个变化是 tabStrip-tab 类把 padding 减少为 2 个像素(原先为 3 个像素)，并添加了 1 个像素宽的透明边框。最后一个 CSS 变化是 tabStrip-tab-hover 和 tabStrip-tab-click 类。以前，这两个类会添加边框，改变背景色，把 padding 减为 2 个像素。而由于 tabStrip-tab 现在把它自己的边框应用于元素，所以 tabStrip-tab-hover 和 tabStrip-tab-click 类仅需改变边框和背景色即可。

现在分析 JavaScript 代码。DHTML 工具栏的这个版本仍使用 currentNum 全局变量，来跟踪活动选项卡的编号。

```
var currentNum = 0;
```

接着是 handleEvent()函数。注意在最初的版本中，必须针对 IE 和 W3C DOM 事件模型编写代码，而这里不需要这么做。

```
function handleEvent(e)
{
    var el = $(e.target);
```

这里只需创建一个 jQuery 对象，它引用 jQuery.Event.target 属性——该元素会引发事件。接着确定鼠标是否移动到一个元素的上面。

```
if (e.type == "mouseover" || e.type == "mouseout")
{
    if (el.hasClass("tabStrip-tab") && !el.hasClass("tabStrip-tab-click"))
    {
        el.toggleClass("tabStrip-tab-hover");
    }
}
```

这段代码也不同于脚本的最初版本。在最初的版本中，需要分别检查 mouseover 和 mouseout 事件，而这个新版本由于改变了 CSS 设计，所以不需要检查这两个事件。

但是，仍需要确定元素是否把 CSS 类 tabStrip-tab 作为它的一个类，以确保这个元素是 tabstrip 中的一个选项卡。由于 handleEvent()函数处理文档对象的 mouseover、mouseout 和 mouseclick 事件，所以这些事件可以在页面的任何元素中触发。实际上，我们希望确保触发事件的元素是一个选项卡，为此，可以检查该元素是否应用了 CSS 类 tabStrip-tab。

我们还希望确保该元素没有 CSS 类 tabStrip-tab-click，否则，鼠标离开应用了 tabStrip-tab-click 类的元素时，jQuery 的 toggleClass()方法就会添加 CSS 类 tabStrip-tab -hover。所以通过确保没有给元素应用 tabStrip-tab-click 类，就可以确保 tabStrip-tab-hover 类正确切换：把鼠标指针指向元素时，就会添加 tabStrip-tab-hover 类，使鼠标指针离开该元素时，就删除该类。

下面看看在选项卡上单击鼠标会发生什么：

```
if (e.type == "click")
{
    if (el.hasClass("tabStrip-tab-hover"))
    {
        var id = e.target.id;
        var num = id.substr(id.lastIndexOf("-") + 1);

        if (currentNum != num)
        {
            deactivateTab();
```

没有什么新鲜的事情发生。声明了一个新变量 id，并给它赋予事件目标的 id。这个改变主要是为了方便，因为 id 比 e.target.id 更容易输入。后面的代码行则是相同的：

```
el.toggleClass("tabStrip-tab-hover")
    .toggleClass("tabStrip-tab-click");
```

```
            showDescription(num);
            currentNum = num;
        }
    }
}
```

这里唯一的区别是将 jQuery 的 toggleClass()方法使用了两次。第一次 toggleClass()调用从元素删除了 CSS 类 tabStrip-tab-hover，第二次 toggleClass()调用添加了 tabStrip-tab-click 类。

showDescription()中的代码完全改写了，但它执行相同的操作：创建一个 元素，给它指定文本，把它追加到 id 为 descContainer 的 元素中。

```
function showDescription(num)
{
    var div = $(document.createElement("div"));

    $("#descContainer").append
    (
        div.attr("id", "tabStrip-desc-" + num)
           .text("Description for tab " + num)
    );
}
```

使用 jQuery 对象执行这些操作会减少需要编写的代码量，下面的 deactivateTab()函数也证明了这一点：

```
function deactivateTab()
{
    var descEl = $("#tabStrip-desc-" + currentNum);

    if (descEl.length > 0)
    {
```

第一行使用 jQuery 函数选择包含选项卡描述的 元素，然后使用 length 属性确保 jQuery 找到一个元素。这么做可确保不会从 DOM 中删除不存在的对象，否则会出错。

如果找到一个元素，就使用 jQuery 的 remove()方法从 DOM 中删除该元素，如下所示：

```
        descEl.remove();

        $("#tabStrip-tab-"+ currentNum).toggleClass("tabStrip-tab-click");
    }
}
```

接着选择活动选项卡的 元素，使用 toggleClass()方法删除 CSS 类 tabStrip-tab-click。

最后，在文档对象上指定 handleEvent()函数来处理 mouseover、mouseout 和 click 事件。如下面的代码所示：

```
$(document).bind("click mouseover mouseout", handleEvent);
```

从这个示例可以看出，jQuery 可以简化 DOM 操作，且需要的输入量更少。在这个示例中，少写了 28 行 CSS 和 JavaScript 代码，所以花些时间学习框架还是值得的。

DOM 操作不是 jQuery 等框架的唯一应用领域。实际上，它还可以大幅减少创建 XMLHttpRequest 对象和发出请求所需的工作量。

15.3.6 把 jQuery 用于 Ajax

上一章详细介绍了模块的创建，以便创建和使用 XMLHttpRequest 对象从 Web 服务器中检索数据。前面创建的模块肯定会使 Ajax 请求更便于编码，而使用 jQuery 编码 Ajax 请求还要更简单一些。

1. 理解 jQuery 函数

jQuery 函数($())是进入 jQuery 的大门，本章多次使用了它。但是，这个函数还有其他用途。本书仅提到一次，而且是一个旁注，但函数也是对象。如果回顾第 5 章末尾，会发现我们创建了自己的对象和引用类型。那时，使用了 prototype 对象，它是 Function 对象的一个属性。

与其他所有对象一样，使用 object.property 或 object.method()语法可以访问 Function 对象的属性和方法。同样，jQuery 的 Ajax 功能是由$函数对象的方法提供的。例如，为了向服务器发出请求，可以使用 get()方法，如下面的代码所示：

```
$.get("textFile.txt");
```

这行代码向服务器发出一个请求，要检索 textFile.txt 文本文件，但这是没有用的，因为不能对检索出来的数据执行任何操作。所以与上一章建立的 HttpRequest 模块一样，$.get()方法允许指定一个回调函数，在请求成功传送给服务器，并检索到指定的数据时，就调用这个回调函数。

```
function get_callBack(data, status)
{
    alert(data);
}

$.get("textFile.txt", get_callBack);
```

这段代码添加了一个 get_callBack()函数，jQuery 在请求成功时会调用它。jQuery 执行 Ajax 回调函数时，会给它传送两个参数。第一个参数 data 是从服务器请求的数据，第二个参数 status 是请求的状态。由于 jQuery 仅在请求成功时才调用回调函数，所以 status 总是 success。

注意：

许多开发人员总是放弃使用第二个参数 status，因为此时它的值只能是 success。读者也可以放弃使用它。

2. 使用 jQuery 的 Ajax 事件处理

jQuery 的 Ajax 事件处理与我们的期望大相径庭。Ajax 有本地事件(指定请求对象的事件)和全局事件。全局事件更易于使用，可以用来添加 UI 内容，以改善用户的体验。可以在任何有效的 DOM 对象上触发全局事件。一个特定的全局事件是 ajaxError，使用 jQuery 对象的 ajaxError()方法可以把它设置为调用一个事件处理程序，也可以使用 bind()方法。下面的代码演示了 ajaxError()方法的用法：

```
function request_ajaxError(event, request, settings)
{
    alert("An Ajax error occurred.");
}

$(document).ajaxError(request_ajaxError);
```

给 Ajax 事件处理程序传送的参数有 3 个：

- event 对象，它带有事件的信息。
- XMLHttpRequest 对象，用于发出请求。
- 包含用于请求的设置的对象。使用 settings 对象，可以检索请求的 URL、其 HTTP 方法、请求是否以异步模式发送等。

可以在 request 和 settings 参数之间建立一些精心设计的错误消息。

注意：
请参见 http://docs.jquery.com/Ajax，来查看 Ajax 事件的完整列表。

还记得上一章的示例吗？我们创建了一个表单，使用 Ajax 检查用户名和电子邮件地址是否可用，并把这些值作为 URL 中的参数发送给服务器。例如，要测试用户名时，可以使用 username 参数，如下所示：

```
phpformvalidator.php?username=jmcpeak
```

使用$.get()方法，给该方法传送一个包含键/值对的对象，就可以完成相同的任务：

```
var parms = new Object();
parms.username = "jmcpeak";

function get_callBack(data, status)
{
    alert(data);
}

$.get("phpformvalidator.php", parms, get_callBack);
```

在这段代码中，创建了一个新对象 parms，给该对象添加了属性 username，并给属性赋值 jmcpeak。接着编写函数 get_callBack()，之后调用$.get()，并传送 URL、parms 对象和回调函数。

3. 传送多个参数

只要给对象添加多个属性，就可以给 URL 发送多个参数，如下所示：

```
var parms = new Object();
parms.username = "jwmcpeak";
parms.email = "someone@xyz.com";
```

可以根据需要在单个请求中发送任意多个参数。

默认情况下，$.get()方法以异步模式发送参数。在大多数情况下，这是很理想的。但有时需要使用同步通信，例如第 14 章的第 2 题。不能使用$.get()指定希望使用的通信模式类型，但 jQuery 提供了$.ajaxSetup()方法。这个方法接受一个对象参数，该对象包含许多属性，可以设置影响所有 Ajax 请求的选项。其中一个选项是 async：

```
var options = new Object();
options.async = false;

$.ajaxSetup(options);
```

这段代码把所有 Ajax 调用设置为使用同步模式。注意，除非绝对必要，才把这个选项设置为 false。在大多数情况下(99.9%)，都需要使用异步通信。

下面回顾上一章使用 XMLHttpRequest 的表单验证器脚本，用 jQuery 的 Ajax 功能替代 HttpRequest 模块代码。

注意：

$.get()方法的用法非常简单，它提供了基本功能。而 jQuery 提供了更高级的低层 Ajax 功能，更多信息可参见 http://docs.jquery.com/Ajax。

试一试　再次访问表单验证器

打开文本编辑器，输入下面的代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Chapter 15: Example 4 with jQuery</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
```

```
<script type="text/javascript" src="jquery-1.3.2.min.js"></script>
<script type="text/javascript">
function checkUsername()
{

    var userValue = $("#username").val();

    if (userValue == "")
    {
        alert("Please enter a user name to check!");
        return;
    }

    var parms = new Object();
        parms.username = userValue;

        $.get("formvalidator.php", parms, checkUsername_callBack);
    }

    function checkUsername_callBack(data, status)
    {
        var userValue = $("#username").val();

        if (data == "available")
        {
            alert("The username " + userValue + " is available! ");
        }
        else
        {
            alert("We're sorry, but " + userValue + " is not available. ");
        }
    }

    function checkEmail()
    {
        var emailValue = $("#email").val();

        if (emailValue == "")
        {
            alert("Please enter an email address to check! ");
            return;
        }

        var parms = new Object();
        parms.email = emailValue;

        $.get("formvalidator.php", parms, checkEmail_callBack);
    }

    function checkEmail_callBack(data, status)
    {
        var emailValue = $("#email").val();

        if (data == "available")
```

```
        {
            alert("The email " + emailValue + " is currently not in use! ");
        }
        else
        {
            alert("I'm sorry, but"+ emailValue +" is in use by another user.");
        }
    }

    function request_error(event, request, settings)
    {
        alert("An error occurred with the following URL:\n"
             + settings.url +".\nStatus code: " + request.status);
    }

    $(document).ajaxError(request_error);
        </script>
    </head>
    <body>
        <form>
            <table>
                <tr>
                    <td class="fieldname">
                        Username:
                    </td>
                    <td>
                        <input type="text" id="username" />
                    </td>
                    <td>
                        <a href="javascript: checkUsername()">Check
                                                    Availability</a>
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Email:
                    </td>
                    <td>
                        <input type="text" id="email" />
                    </td>
                    <td>
                        <a href="javascript: checkEmail()">Check
                                                    Availability</a>
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Password:
                    </td>
                    <td>
                        <input type="text" id="password" />
```

```
                    </td>
                    <td />
                </tr>
                <tr>
                    <td class="fieldname">
                        Verify Password:
                    </td>
                    <td>
                        <input type="text" id="password2" />
                    </td>
                    <td />
                </tr>
                <tr>
                    <td colspan="2" class=vsubmit">
                        <input type="submit" value="Submit" />
                    </td>
                    <td />
                </tr>
            </table>
        </form>
    </body>
    </html>
```

保存为 Web 服务器根目录下的 ch15_examp4.htm。与上一章使用 XMLHttpRequest 的示例一样，只有将这个文件放在 Web 服务器上，才能正常工作。打开 Web 浏览器，指向 http://yourserver/ch15_examp4.htm。在 Username 字段中输入 jmcpeak，单击它旁边的 Check Availability 链接，此时会显示一个警告框，说明这个用户名已被使用。

现在，在 Email 字段中输入 someone@xyz.com，单击它旁边的 Check Availability 链接，此时会再次显示一个警告框，说明这个电子邮件地址已在使用。现在在这些字段中输入用户自己的用户名和电子邮件地址，并单击对应的链接。此时显示的警告框会说明，这个用户名和/或电子邮件地址是可用的(应用程序只使用了用户名 jmcpeak 和 pwilton、电子邮件地址 someone@xyz.com 和 someone@zyx.com)。

这个页面的工作方式完全类似于第 14 章的 validate_form.htm，代码也基本相同。

第一个变化是删除了 httprequest.js，而包含了 jQuery JavaScript 文件 jquery-1.3.2.min.js。

第二个变化在 checkUsername()函数中。

```
function checkUsername()
{
    var userValue = $("#username").val();

    if (userValue == "")
    {
        alert("Please enter a user name to check! ");
        return;
    }
```

第一行检索出了 Username 文本框的值，方法是选择该元素，并调用了 val()方法。val()方法会检索出<input/>元素的 value 属性。接着进行检查，以确保用户在文本框中输入了内

容，如果用户没有输入用户名，就礼貌地要求用户输入用户名。

然后发出如下请求：

```
    var parms = new Object();
    parms.username = userValue;

    $.get("formvalidator.php", parms, checkUsername_callBack);
}
```

首先创建一个 parms 对象，再创建一个 username 属性。把这个对象、要回调的函数的 URL 和名称传送给$.get()方法。

请求成功后，jQuery 就会调用 checkUsername_callBack()函数。传送给这个函数的参数会反映出请求成功这一状况。

```
function checkUsername_callBack(data, status)
{
    var userValue = $("#username").val();

    if (data == "available")
    {
        alert("The username " + userValue + " is available! ");
    }

    else
    {
        alert("We're sorry, but "+ userValue +" is not available. ");
    }
}
```

除了参数之外，这个函数的唯一变化是使用 val()方法检索文本框的值。其他都没有变化：该函数比较从服务器返回的数据，并向用户显示一个消息，说明用户名查询的结果。

checkEmail()的变化类似于 checkUsername()。

```
function checkEmail()
{
    var emailValue = $("#email").val();

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return;
    }

    var parms = new Object();
    parms.email = emailValue;

    $.get("formvalidator.php", parms, checkEmail_callBack);
}
```

这段代码使用 jQuery 函数选择 Email 文本框，再使用 val()方法检索其值。接着确定用户是否输入了数据，如果没有输入，就请求用户输入。接着创建 parms 对象，再创建 email

属性，并给它赋值，然后用$.get()方法发出请求。

请求成功后，就执行 checkEmail_callBack()函数，这个函数的变化与 checkUsername_callBack()相同。

```
function checkEmail_callBack(data, status)
{
    var emailValue = $("#email").val();

    if (data == "available")
    {
        alert("The email " + emailValue + " is currently not in use! ");
    }
    else
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user.");
    }
}
```

万一出现了 Ajax 错误，就给 ajaxError 事件注册一个 Ajax 事件处理程序。该事件处理函数是 ajax_error()，它向用户显示一个错误消息。其定义如下：

```
function ajax_error(event, request, settings)
{
    alert("An error occurred with the following URL:\n"
          + settings.url +".\nStatus code: " + request.status);
}

$(document).ajaxError(ajax_error);
```

该消息告诉用户出现的错误、请求的 URL 以及服务器返回的 HTTP 状态码。有了这些信息，就可以开始调试错误。

jQuery 是一个扩展框架，提供的信息大大超出了本节的范围。但是，jQuery 文档非常不错，可以在 http://docs.jquery.com 上访问它。jQuery 的网站也列出了各种教程，可以在 http://docs.jquery.com/Tutorials 上查看它们。其中包含的效果/动画组件和可选的 UI 库使 jQuery 成为一个多用途的框架。

15.4　Prototype

jQuery 可能是目前最流行的框架了，但框架的王冠却戴在 Prototype 的头上。与 jQuery 不同，Prototype 的重点是提供类和继承，增强用 JavaScript 编程的方法。它还提供了一个可靠的工具集，以使用 DOM 和增加 Ajax 支持。

前面简要介绍了 Prototype，下面将深入探讨这个库，分析它可以为跨浏览器脚本编程提供什么帮助。

15.4.1 检索元素

前面在测试 Prototype 安装时，介绍过美元函数$()。这个函数与 jQuery 函数不同，它只是通过添加许多新方法，扩展了要检索的元素。如果传送元素的 id 值，该函数就会检索该元素，并用更多方法和属性扩展它。

```
$("myDiv")
```

这行代码从 DOM 中检索出 id 为 myDiv 的元素，并扩展它。所以可以像使用其他 Element 对象那样使用这个扩展了的对象。

```
alert($("myDiv").tagName);
```

该还可以给$()函数传送一个 Element 对象，这会得到这个元素的扩展版本。下面的代码把 document.body 对象传送给美元函数：

```
$(document.body)
```

这样，就可以使用内联的 DOM 方法和属性，也可以使用 Prototype 提供的方法。

注意：

如果找不到指定的元素，Prototype 的美元函数返回 null。这与 jQuery 的$()函数不同，因为 Prototype 返回的是一个扩展的 DOM 元素对象，即使它是一个扩展对象，它也是 DOM 元素对象。

1. 用 CSS 选择器选择元素

Prototype 和 jQuery 的美元函数的另一个区别是，Prototype 的$()函数不接受 CSS 选择器，而只接受元素的 id 值和 Element 对象。但 Prototype 另有一个函数$$()，它与 jQuery 的$()函数类似。

可以给$$()函数传送几个选择器类型，以定位和检索与选择器匹配的元素。例如，下面的代码会检索出页面上的所有 元素，把它们返回为一个数组，这样就可以使用 length 属性了：

```
$$("div")
```

$$()函数总是返回一个数组，即使使用 id 选择器，也会在找到对应的元素时得到一个只包含一个元素的数组。$$()函数的一个缺点是它返回一个包含扩展元素的数组。如果希望对数组中的每个元素执行操作，就必须对它们执行循环，或者使用 Prototype 提供的 each()方法迭代它们。

2. 对用$$()选择出来的元素执行操作

each()方法类似于第 5 章介绍的新方法 Array.every()，它接受一个函数作为参数，对数组中的每个元素执行该函数。下面的代码演示了这个过程：

```
function insertText(item)
```

```
{
    item.insert("This text inserted using the each() method.");
}

$$("div").each(insertText);
```

注意：

jQuery 对象也有一个 each()方法，它也执行相同的函数。

通过$$()函数，可以使用几个 CSS 选择器类型来选择元素，还可以根据多个选择器来选择元素，但这不同于 jQuery：

```
$$("#myDiv", "p > span, .myCssClass");
```

除了传送一个用逗号分隔各个选择器的字符串之外，还可以传送多个字符串，每个字符串都包含一个选择器。一旦检索出了元素，就可以开始处理它们，例如改变其样式。

注意：

有关 Prototype 支持的 CSS 选择器，详见 http://www.prototypejs.org/api/utility/dollar-dollar。

15.4.2　处理样式

Prototype 提供了几个方法，可用于改变元素的样式，它们与 jQuery 框架中的对应方法并不相同。

可以使用 setStyle()方法来设置单个样式属性。要采用这种方式设置样式，必须创建一个对象，再创建这个对象的属性，这些属性的名称就是对应 CSS 属性的名称。例如，下面的代码把元素的文本颜色设置为红色，并给文本加下划线：

```
var styles = new Object();
styles.color = "red";
styles.textDecoration = "underline";

$("myDiv").setStyle(styles);
```

如前面介绍 jQuery 的一节所述，采用这种方式改变元素的样式并不理想，因为样式应在页面的样式表中定义。较好的方法是操作元素的 CSS 类，Prototype 提供的 addClassName()、removeClassName()、toggleClassName()和 hasClassName()方法很容易完成这个操作。

第一个方法 addClassName()会给元素添加一个 CSS 类名。只需把类名传送给该方法，该类就会应用到元素上：

```
$("myDiv").addClassName("someClass");
```

第二个方法 removeClassName()会从元素中删除指定的类。下面的代码添加了一个类名，然后删除它：

```
$("myDiv").addClassName("someClass")
    .removeClassName("someClass");
```

这段代码并不实用，但说明了这两个方法的用法。接下来是 toggleClassName()方法，这个方法检查指定的类是否应用到元素上，如果是，就删除它。如果没有找到该类，它就把这个类应用到元素上。

```
$("myDiv").hasClassName("someClass");
$("myDiv").toggleClassName("someClass");
$("myDiv").hasClassName("someClass");
```

这段代码演示了 toggleClassName()方法和第四个方法 hasClassName()。第一行代码检查 CSS 类 someClass 是否应用到元素上，因为 CSS 类 someClass 没有应用到元素上，所以这个方法返回 false。第二行代码调用 toggleClassName()方法，把 CSS 类 someClass 添加到元素上。最后一行再次调用 hasClassName()，现在它返回 true，因为 someClass 已在上一行添加到元素上。

这些 CSS 方法与 jQuery 的对应方法非常类似。但 Prototype 中其他类型的 DOM 操作与 jQuery 大不相同，例如创建和插入元素。只是删除 DOM 对象的操作是十分类似的。

15.4.3 创建、插入和删除元素

用 Prototype 操作 DOM 是很简单的。该框架扩展了 Element 对象，允许使用构造函数创建元素，给它填充数据，以及从 DOM 中删除它。

1. 创建元素

Element 对象的构造函数接受两个参数：标记名以及包含属性及其值的对象。下面的代码演示了如何创建<a/>元素，并将其添加到文档正文中：

```
var attributes = new Object();
attributes.id = "myLink";
attributes.href = "http://www.prototypejs.org";
attributes.target = "_blank";

var a = new Element("a", attributes);
```

这段代码的开头几行创建了一个对象 attributes，然后创建了 id、href 和 target 属性，并给它们赋值。接着使用 Element 对象的构造函数创建<a/>元素，我们把字符串“a”作为第一个参数传送，把 attributes 对象作为第二个参数传送。

2. 插入元素

update()和 insert()方法都可以把内容添加到 Element 对象中。其区别是，update()会替换所有已有的内容，而 insert()仅把新内容添加到已有内容上。这两个方法都接受一个字符串值作为参数，该字符串包含简单文本、HTML 或 Element 对象。

下面的代码创建了上一个代码示例中的<a/>元素，给它添加内容，再将其插入页面：

```
var attributes = new Object();
attributes.id = "myLink";
attributes.href = "http://www.prototypejs.org";
```

```
attributes.target = "_blank";

var a = new Element("a", attributes).update("Go to Prototype' s Website");
$(document.body).insert(a);
```

这段代码调用 update()方法，用传送给它的数据指定的新内容替换元素已有的内容(在这个示例中，没有已有的内容)。由于 update()方法返回前面创建的 Element 对象，所以把该对象赋予变量 a，然后把 a 传送给 document.body 对象的 insert()方法。

3. 删除元素

从 DOM 中删除元素更简单，实际上这与 jQuery 中的删除操作相同。首先找到要从 DOM 中删除的元素，然后调用 remove()方法。

```
$("myLink").remove();
```

这段代码找到 id 为 myLink 的元素，然后从 DOM 中删除它。不需要查找该元素的父元素，再调用 removeChild()方法。

15.4.4　使用事件

用$()函数扩展 Element 对象时，可以使用 observe()方法，它会在 DOM 元素上注册一个事件处理程序。这个方法接受两个参数：要监控的事件名和触发事件时调用的函数。

```
function myDiv_click(event)
{
    // do something
}

$("myDiv").observe("click", myDiv_click);
```

这段代码注册了 myDiv_click()函数，以处理 id 为 myDiv 的元素上的单击事件。这不是指定事件处理程序的唯一方式，还可以使用 Event.observe()方法。下面的代码使用 Event.observe()编写了上面的代码：

```
function myDiv_click(event)
{
    // do something
}

Event.observe("myDiv", "click", myDiv_click);
```

Event.observe()的第一个参数可以是一个字符串值，该字符串可以包含要为其指定事件处理程序的 BOM/DOM 对象(例如 window 或 document)或元素的 id。这个方法特别适合于 window 等对象；我们不能把 window 传送给美元函数，再使用 observe()方法，因为浏览器会抛出一个错误，必须改用 Event.observe()。

与 jQuery 不同，Prototype 没有模拟 W3C DOM 事件模型，实际上，它根本不打算创建一个单独的、统一的事件模型，而是扩展 IE 和 W3C DOM 这两个浏览器的 event 对象，

给用户提供一组工具方法，来获得所需的信息。

这些方法都是所扩展的浏览器的 Event 对象。例如，element()方法接受 IE 或 W3C 的 event 对象作为参数，分别返回 IE 和 W3C DOM 浏览器的 srcElement 和 target 属性值。

例如，下面的代码获取触发事件的元素，并切换一个 CSS 类 someClass：

```
function myDiv_click(event)
{
    var eSrc = event.element();
    eSrc.toggleClassName("someClass");
}

$("myDiv").observe("click", myDiv_click);
```

为了使这段代码工作，事件处理函数接受的参数必须是 event。这样，调用 element()方法时，就会引用 IE 的 window.event 对象，W3C DOM 浏览器也会把 event 对象传送给函数。

15.4.5 用 Prototype 重写 DHTML 工具栏

现在我们知道如何用 Prototype 检索元素、改变元素的样式、在 DOM 中添加和删除元素式、关联事件、获取触发事件的元素了。下面运用这些知识重写 DHML 工具栏。

试一试 用 Prototype 重写工具栏

打开文本编辑器，输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 5 with Prototype</title>
    <style type="text/css">
    .tabStrip
    {
        background-color: #E4E2D5;
        padding: 3px;
        height: 22px;
    }

    .tabStrip-tab
    {
         float: left;
        font: 14px arial;
        cursor: pointer;
        padding: 2px;
        border: 1px solid transparent;
    }

    .tabStrip-tab-hover
    {
```

```
    border-color: #316AC5;
    background-color: #C1D2EE;
}
.tabStrip-tab-click
{
    border-color: #facc5a;
    background-color: #f9e391;
}
</style>
<script type="text/javascript" src="prototype-1.6.0.3.js"></script>
<script type="text/javascript">
var currentNum = 0;
function handleEvent(event)
{
    var el = event.element();

    if (event.type == "mouseover" || event.type == "mouseout")
    {
        if (el.hasClassName("tabStrip-tab") &&
            !el.hasClassName("tabStrip-tab-click"))
        {
            el.toggleClassName("tabStrip-tab-hover");
        }
    }

    if (event.type == "click")
    {
        if (el.hasClassName("tabStrip-tab-hover"))
        {
            var id = el.id;
            var num = id.substr(id.lastIndexOf("-") + 1);

            if (currentNum != num)
            {
                deactivateTab();

                el.toggleClassName("tabStrip-tab-hover")
                   .toggleClassName("tabStrip-tab-click");
                showDescription(num);
                currentNum = num;
            }
        }
    }
}
function showDescription(num)
{
    var attributes = new Object();
    attributes.id = "tabStrip-desc-" + num;

    var div = new Element("div", attributes)
        .update("Description for tab " + num);
```

```
        $("descContainer").update(div);
    }

    function deactivateTab()
    {
        var currentTab = $("tabStrip-tab-"+ currentNum);

        if (currentTab)
        {
            currentTab.toggleClassName("tabStrip-tab-click");
        }
    }

    $(document).observe("click", handleEvent);
    $(document).observe("mouseover", handleEvent);
    $(document).observe("mouseout", handleEvent);
</script>
</head>
<body>
    <div class="tabStrip">
        <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
        <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
        <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
    </div>
    <div id="descContainer"></div>
</body>
</html>
```

把这个文件保存为 ch15_examp5.htm，并加载到浏览器中。注意它的结果与第 12 章第 2 题的答案相同，与用 jQuery 重写的版本也相同。

由于 CSS 和标记与 ch15_examp3.htm(jQuery 版本)相同，所以我们只关注 JavaScript 函数是如何变化的。首先分析 handleEvent()函数。

```
function handleEvent(event)
{
    var el = event.element();
```

对于这个函数，首先要注意的是参数名改为 event。这么做是为了调用 element()方法。如果为参数使用另一个名称，就需要确定用户的浏览器是什么，然后在 window.event 对象和 W3C 的事件对象上调用 element()方法。换言之，需要做更多工作，第 12 章的跨浏览器版本就是如此。

接着确定所触发的事件类型。首先检查 mouseover 和 mouseout 事件：

```
if (event.type == "mouseover" || event.type == "mouseout")
{
    if (el.hasClassName("tabStrip-tab") &&
        !el.hasClassName("tabStrip-tab-click"))
    {
```

如果触发了这两个事件中的一个，就必须检查触发事件的元素是否是 tab strip 中的一

个选项卡。为此，要用 hasClassName()方法确定该元素是否包含 tabStrip-tab 类名。还需要确保该元素不包含 tabStriptab-click 类名，否则就会引起不正确的样式变化。

如果样式满足所有条件，就切换 tabStrip-tab-hover 类名。

```
            el.toggleClassName("tabStrip-tab-hover");
        }
    }
```

可以使用 toggleClassName()方法执行这个任务。现在把鼠标指针移到一个选项卡元素上，tabStrip-tab-hover 类就会应用于该元素。鼠标指针离开该元素时，代码就会删除 tabStrip-tab-hover 类，使该元素返回初始状态。

下面确定是否触发了 click 事件，如果触发了，就确定该元素是否有 tabStrip-tab-hover 类。

```
        if (event.type == "click")
        {
            if (el.hasClassName("tabStrip-tab-hover"))
            {
```

如果有，则说明该元素应把其样式改为被单击的选项卡样式。为了启动这个过程，需要获取与该选项卡元素关联的编号：

```
                var id = el.id;
                var num = id.substr(id.lastIndexOf("-") + 1);
```

为此，获取元素的 id 值，再使用 substr()方法检索最后一个连字符后面的所有文本，把 substr()的结果赋予 num 变量。

然后比较 num 和 currentNum，确定被单击的元素是否是 tab strip 中的另一个选项卡。

```
                if (currentNum != num)
                {
                    deactivateTab();
```

如果它是另一个选项卡元素，则调用 deactivateTab()函数，然后使用 toggleClassName()方法删除 tabStrip-tab-hover 类，并给该元素添加 tabStrip-tab-click 类。

```
                    el.toggleClassName("tabStrip-tab-hover")
                        .toggleClassName("tabStrip-tab-click");
```

然后调用 showDescription()函数，给它传送元素的编号值。接着给 currentNum 变量赋予变量 num 包含的值。这么做会告诉脚本：这个新选项卡元素是当前活动选项卡。

```
                    showDescription(num);
                    currentNum = num;
                }
            }
        }
    }
```

下面分析 showDescription()函数，它给页面添加选项卡的描述信息。在这个函数中，首

先创建 attributes 对象，再创建 id 属性：

```
function showDescription(num)
{
    var attributes = new Object();
    attributes.id = "tabStrip-desc-" + num;
```

接着使用 Element 对象的构造函数创建新的 元素：指定要创建一个 元素，并传送 attributes 对象。在调用构造函数之后，链接一个 update()方法的调用，给元素指定一些文本。

```
    var div = new Element("div", attributes)
        .update("Description for tab " + num);
```

这个函数的最后一步是把元素添加到页面上。这需要使用 update()方法：

```
    $("descContainer").update(div);
}
```

这里使用 update()方法，意味着不必删除当前在 deactivateTab()中显示的描述信息，因为 update()方法会删除当前所有内容，代之以传给该方法的内容。

由于在 showDescription()中使用了 update()方法，所以简化了 deactivateTab()的函数体。

```
function deactivateTab()
{
    var currentTab = $("tabStrip-tab-"+ currentNum);

    if (currentTab)
    {
        currentTab.toggleClassName("tabStrip-tab-click");
    }
}
```

首先尝试根据 id 检索当前活动的选项卡元素。如果可以在文档中找到该元素，就执行 if 块中的代码，从元素中删除 tabStrip-tab-click 类。如果在文档中找不到该元素，currentTab 就是 null，函数退出，不做任何其他工作。

与 jQuery 一样，Prototype 的作用不限于 DOM 操作和语言增强方面，它还提供了易于学习和使用的 Ajax 功能。

15.4.6 使用 Ajax 支持

Prototype 中的 Ajax 支持不如 jQuery 中的高级方法$.get()直接。Prototype 的 Ajax 功能以其 Ajax 对象为核心，它包含可以进行 Ajax 调用的各种方法。这个对象类似于内部的 Math 对象，因为不需要创建 Ajax 对象的实例，只需使用该对象提供的方法即可。

Ajax 对象的核心是 Ajax.Request()构造函数。这个构造函数接受两个参数，第一个是发出请求的 URL，第二个是一个包含一组选项的对象，对象在发出请求时使用这些选项。options 对象可以包含各种选项属性，以改变 Ajax.Request()的行为。表 15-1 描述了其中的一些选项。

表　15-1

选　　项	说　　明
asynchronous	确定 XMLHttpRequest 对象是否以异步模式发出请求，默认为 true
method	用于请求的 HTTP 方法，默认是 post，get 是另一个有效值
onSuccess	成功完成请求时调用的回调函数
onFailure	请求完成，但得到一个错误状态码时调用的回调函数
parameters	包含随请求一起发送的参数的字符串，或者包含参数及其值的对象

注意：

要查看选项的完整列表，请参阅 http://www.prototypejs.org/api/ajax/options 上的 Prototype 文档。

执行所有的回调函数，并传送一个包含 XMLHttpRequest 对象的参数，该对象用于发出请求。用 Prototype 发出请求的代码如下：

```
function request_onsuccess(request)
{
    alert(request.responseText);
}

function request_onfailure(request)
{
    alert("An error occurred! HTTP status code is " + request.status);
}

var options = new Object();
options.method = "get";
options.onSuccess = request_onsuccess;
options.onFailure = request_onfailure;

new Ajax.Request("someTextFile.txt", options);
```

开头几行代码定义了 request_onsuccess()和 request_onfailure()函数。这些函数都接受一个参数 request。这个参数的值应是用于发出请求的 XMLHttpRequest 对象。

注意：

Prototype 实际上把一个 Ajax.Response 对象传送给所有 Ajax 请求回调，这非常类似于 XMLHttpRequest 对象，但增加了额外的功能。要直接访问 XMLHttpRequest 对象，可以使用 transport 属性，如 request.transport。在函数定义后，创建一个 options 对象。我们设置的第一个选项是 method，它设置为 get。下一个选项是 onSuccess，把 request_onsuccess()函数指定给这个选项。最后一个选项是 onFailure，给它指定 request_onfailure()函数。

做好所有准备工作后，就发出针对 someTextFile.txt 文件的请求，并把 options 对象传给 Ajax.Request()构造函数(别忘了 new 关键字！)。

如果需要随请求一起传送参数，在调用 new Ajax.Request()之前就必须多做一些准备工作。与 jQuery 一样，可以创建一个对象，其属性名匹配参数名。例如，如果 URL 需要传

送两个参数 name 和 state，就可以编写如下代码：

```
var parms = new Object();
parms.name = "Jeremy";
parms.state = "Texas";

options.parameters = parms;
```

添加了参数，调用了 new Ajax.Request()后，就会将参数添加到 URL 上，之后把请求发送给服务器。

学习了 Prototype 的 Ajax 函数后，就可以修改第 14 章的表单验证器脚本，用 Prototype 替代 HttpRequest 模块。

试一试　用 Prototype 重写表单验证器

打开文本编辑器，输入如下代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Chapter 15: Example 6 with jQuery</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
    <script type="text/javascript" src="prototype-1.6.0.3.js"></script>
    <script type="text/javascript">
    function checkUsername()
    {
        var userValue = $F("username");

        if (userValue == "")
        {
            alert("Please enter a user name to check!");
            return;
        }

        var parms = new Object();
        parms.username = userValue;

        var options = getBasicOptions();
        options.onSuccess = checkUsername_callBack;
        options.parameters = parms;
```

```
    new Ajax.Request("formvalidator.php", options);
}
function checkUsername_callBack(request)
{
    var userValue = $F("username");

    if (request.responseText == "available")
    {
        alert("The username " + userValue + " is available!");
    }
    else
    {
        alert("We're sorry, but " + userValue + " is not available.");
    }
}

function checkEmail()
{
    var emailValue = $F("email");

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return;
    }

    var parms = new Object();
    parms.email = emailValue;

    var options = getBasicOptions();
    options.onSuccess = checkEmail_callBack;
    options.parameters = parms;

    new Ajax.Request("formvalidator.php", options);
}

function checkEmail_callBack(request)
{
    var emailValue = $F("email");

    if (request.responseText == "available")
    {
        alert("The email " + emailValue + " is currently not in use!");
    }
    else
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user.");
    }
}

function request_onfailure(request)
{
    alert("An error occurred. HTTP Status Code: " + request.status);
```

```
        }

        function getBasicOptions()
        {
            var options = new Object();
            options.method = "get";
            options.onFailure = request_onfailure;

            return options;
        }
        </script>
    </head>
    <body>
        <form>
            <table>
                <tr>
                    <td class="fieldname">
                        Username:
                    </td>
                    <td>
                        <input type="text" id="username" />
                    </td>
                    <td>
                        <a href="javascript: checkUsername()">Check Availability</a>
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Email:
                    </td>
                    <td>
                        <input type="text" id="email" />
                    </td>
                    <td>
                        <a href="javascript: checkEmail()">Check Availability</a>
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Password:
                    </td>
                    <td>
                        <input type="text" id="password" />
                    </td>
                    <td />
                </tr>
                <tr>
                        <td class="fieldname">
                            Verify Password:
                        </td>
                        <td>
```

```
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submit" value="Submit" />
                </td>
                <td />
            </tr>
        </table>
    </form>
</body>
</html>
```

在 Web 服务器的根目录下把这个文件保存为 ch15_examp6.htm，因为这个文件只有放在 Web 服务器上，才能正常工作。使浏览器指向 http://youserver/ch15_examp6.htm，测试该表单。

这个页面的工作过程与 ch15_examp4.htm(jQuery 版本)和最初的 validate_form.htm 相同，但这个版本有几个变化。第一个重要变化是添加了两个新函数 getBasicOptions()和 request_onfailure()。

第一个函数用来创建一个对象，其中包含了 Ajax.Request()构造函数需要的基本选项，以检查用户想要使用的用户名和电子邮件。

```
function getBasicOptions()
{
    var options = new Object();
    options.method = "get";
    options.onFailure = request_onfailure;

    return options;
}
```

这个函数首先创建一个 options 对象，再添加一个 method 属性，给它赋予包含字符串“get”的值。

接着添加另一个属性 onFailure，给它赋予指向 request_onfailure()函数的指针。把 options 对象返回给调用程序，以便为 Ajax.Request()构造函数创建基本选项。使用这个函数主要是为了方便：不必两次输入这些代码(一次在 checkUsername()中输入，另一次在 checkEmail()中输入)，而只需在这里输入一次，然后调用这个函数来返回这些选项。

第二个新函数 request_onfailure()用于用户名和电子邮件检查请求。如果请求因某种原因失败，就执行函数 request_onfailure()。

```
function request_onfailure(request)
{
    alert("An error occurred. HTTP Status Code: " + request.status);
}
```

这是一个简单函数，它通知用户发生了一个错误。

下面分析 checkUsername()函数，因为对它做了许多修改。

```
function checkUsername()
{
    var userValue = $F("username");
```

该函数的第一行代码就有变化。它现在使用 Prototype 的$F()函数，这个函数用于表单元素，把元素的 id 传送给该函数，该函数就会检索元素的值。所以$F("username")等价于 document.getElementById("username").value。前者肯定更容易输入。

现在确定用户是否在 Username 字段中输入了信息。只需比较 userValue 变量和空字符串。

```
    if (userValue == "")
    {
        alert("Please enter a user name to check! ");
        return;
    }
```

如果用户未在文本框中输入数据，就在一个警告框中请求用户输入，并从函数中返回。

接着收集准备阶段需要的信息，以便向服务器发出请求。创建一个 parms 对象，建立 username 属性，为其赋予 userValue 包含的值。

```
    var parms = new Object();
    parms.username = userValue;
```

创建要传送给 Ajax.Request()构造函数的 options 对象：

```
    var options = getBasicOptions();
    options.onSuccess = checkUsername_callBack;
    options.parameters = parms;
```

首先调用 getBasicOptions()，来创建基本选项。然后创建 onSuccess 和 parameters 属性。把 onSuccess 的值设置为指向 checkUsername_callback()的指针，把 parameters 属性设置为前面创建的 parms 对象。

```
    new Ajax.Request("formvalidator.php", options);
}
```

在这个函数的最后一步中，调用 Ajax.Request()构造函数，前面加上 new 关键字，并传送 formvalidator.php 的 URL 和 options 对象。

如果这个请求失败，则执行 request_onfailure()。而 checkUsername_callback()会在请求成功时执行，这个函数也有一些变化。

第一个变化是参数名，它现在不是 data，而是 request，更好地反映了包含 XMLHttp Request 对象这一事实——类似于 Ajax.Response。

```
function checkUsername_callBack(request)
{
```

接下来，使用$F()函数，并给它传送字符串 username，以获取 Username 字段的值。

```
    var userValue = $F("username");
```

然后使用 XMLHttpRequest 对象的 responseText 属性获得服务器应用程序的响应，检查用户名是否可用。

```
    if (request.responseText == "available")
    {
        alert("The username " + userValue + " is available! ");
    }
    else
    {
        alert("We're sorry, but " + userValue + " is not available. ");
    }
}
```

根据 if 语句的结果，通知用户他们想要使用的用户名是否可用。

checkEmail()函数的变化与 checkUsername()类似。

```
function checkEmail()
{
    var emailValue = $F("email");

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return;
    }
```

这段代码检索 Email 文本框的值，确定用户是否在该文本框中输入了信息。如果没有，就显示一个警告框，要求用户输入信息，并退出函数。

接着进入用 Prototype 发送请求的准备步骤。首先创建 parms 对象：

```
    var parms = new Object();
    parms.email = emailValue;
```

创建一个 email 属性，为其赋予 emailValue 变量包含的值。然后创建 options 对象：

```
    var options = getBasicOptions();
    options.onSuccess = checkEmail_callBack;
    options.parameters = parms;
```

再次调用 getBasicOptions()函数，为这个请求创建基本选项。接着创建 onSuccess 和 parameters 选项属性并赋值。

在退出函数前，调用 Ajax.Request()构造函数来执行请求。

```
    new Ajax.Request("formvalidator.php", options);
}
```

请求成功时，Prototype Ajax 组件会调用 checkEmail_callBack()，并给它传送用于向服

务器发出请求的 XMLHttpRequest 对象。

```
function checkEmail_callBack(request)
{
    var emailValue = $F("email");

    if (request.responseText == "available")
    {
        alert("The email " + emailValue + " is currently not in use!");
    }
    else
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user.");
    }
}
```

首先获取 Email 文本框的值，并将其存储在 emailValue 变量中。接着比较请求的 responseText 属性和字符串“available”，以确定该电子邮件是否可由用户使用。然后根据 if 语句的结果，向用户显示一个消息，告诉他们电子邮件是否可用。

Prototype 是一个功能强大的框架，提供了一组丰富的工具，可改变编写 JavaScript 的方式。与 jQuery 类似，像这样的一小节远不足以全面介绍该框架的内容，Prototype 及其提供的工具的更多信息可参见 www.prototypejs.org/api 上的 API 文档和 http://www.prototypejs.org/learn 上的教程。

15.5 MooTools

初看起来，MooTools 似乎与 Prototype 相同，的确如此。最初开发 MooTools 时就是为了与 Prototype 一起使用，所以 MooTools 提供的一些工具与 Prototype 的几乎相同就没有什么可惊讶的了。

但是，就 DOM 操作而言，MooTools 更多的是 jQuery 和 Prototype 之间的交集。与 Prototype 一样，MooTools 的目标是增强编写 JavaScript 的方法，提供工具来编写类，并继承这些类。MooTools 像 Prototype 一样也增加了一组丰富的扩展，以简化 DOM 操作。在 MooTools 中选择 DOM 对象与 Prototype 完全相同。但读者在后面的小节中会发现，扩展方法名和使用它们的方式与 jQuery 相同。

15.5.1 查找元素

测试 MooTools 安装时，使用了美元函数，它类似于 Prototype 的美元函数。其实它们完全相同。它们都查找元素并扩展它，只是它们带有不同的方法，如后面的小节所述。

```
$("myDiv")
```

这行代码查找 id 为 myDiv 的元素，用 MooTools 的方法扩展它，并返回扩展的元素。可用使用 DOM 对象 Element 的方法和属性，以及 MooTools 提供的方法。

MooTools 也提供了$$()函数，使用它可以通过 CSS 选择器检索元素，还可以把多个选择器作为一个参数传送给$$()函数。

```
$$(".myClass");
$$("div", ".myClass", "p > div")
```

MooTools 和 Prototype 的一个显著区别是，MooTools 不必在返回的数组上执行迭代操作。

```
$$("div", "a").setStyle("color", "red");
```

这行代码选择页面上的所有 和<a/>元素，把它们的文本颜色设置为红色。这与 Prototype 不同，如下面的示例代码所示：

```
function changeColor(item)
{
    var styles = new Object();
    styles.color = "red";

    item.setStyle(styles);
}

$$("div", "a").each(changeColor);
```

所以该$$()函数是 Prototype 的$$()函数和 jQuery 的$()函数的交集。

注意：

MooTools 也有 each()方法，如果希望在数组的每个元素上执行操作，就可以使用该方法。

15.5.2　修改样式

前面的 MooTools 代码示例使用了 setStyle()方法，它接受两个参数：第一个是 CSS 属性，第二个是其值。与 jQuery 一样，可以使用样式表中的 CSS 属性，或者脚本中的驼峰式大小写版本(camel-case version)：

```
$("myDiv").setStyle("background-color", "red");
$("myDiv").setStyle("backgroundColor", "red");
```

这两行代码都把元素的背景色设置为红色，所以可以使用任一个属性名来设置各个样式属性。

当然，这不是修改元素样式的理想方式。MooTools 给 Element 对象增加了 addClass()、removeClass()、toggleClass()和 hasClass()方法。

根据名称就可以推断 addClass()和 removeClass()方法的作用，它们给元素添加或删除指定的类，如下面的代码所示：

```
$("myDiv").addClass("someClass").removeClass("someClass");
```

toggleClass()方法的工作方式与我们希望的相同：如果元素有所传送的参数指定的 CSS 类，toggleClass()就从元素中删除该 CSS 类，如果元素没有这个 CSS 类，toggleClass()就给元素添加该类。

```
$("myDiv").toggleClass("myClass").toggleClass("myClass");
```

这行代码首先给元素添加 CSS 类 myClass，因为前一个示例删除了它。第二个 toggleClass()调用再次删除了它，因为我们刚才把它添加到元素上。

hasClass()方法根据元素是否有 CSS 类，返回 true 或 false。

```
$("myDiv").hasClass("myClass");
```

这行代码返回 false，因为 CSS 类 myClass 没有应用到元素上。

改变元素样式仅是 DOM 操作的一部分，MooTools 还可以在 DOM 中创建、插入和删除元素。

15.5.3 创建、插入和删除元素

用 MooTools 创建元素非常类似于在 Prototype 中创建元素。只需使用 Element 对象的构造函数，并给它传送要创建的元素类型和包含该元素应有的属性的对象即可。下面的代码创建了<a/>元素，指定了 id、href 和 target 属性，并把该元素添加到文档中：

```
var attributes = new Object();
attributes.id = "myLink";
attributes.href = "http://www.prototypejs.org";
attributes.target = "_blank";

var a = new Element("a", attributes).appendText("Go to Prototype's Website");
$(document.body).adopt(a);
```

这段代码的前 4 行创建了一个 attributes 对象及其 id、href 和 target 属性，并指定了属性的值。接着用 Element 对象的构造函数创建一个新的<a/>元素，用 appendText()方法给该元素添加了文本。最后用 adopt()方法将这个元素追加到 document.body 对象上。

从 DOM 中删除对象非常简捷明了，只需调用 dispose()方法即可。下面的代码说明了这一点：

```
$("myLink").dispose();
```

这行代码查找 id 为 myLink 的元素，再调用 dispose()方法从 DOM 中删除它。

15.5.4 使用和处理事件

用$()函数扩展 Element 对象时，MooTools 给元素添加了 addEvent()方法。这个方法把一个事件处理程序关联到元素上，以处理特定的事件。下面的代码是使用该方法的一个示例：

```
function myDiv_click(event)
```

```
{
    alert("You clicked me!");
}

$("myDiv").addEvent("click", myDiv_click);
```

addEvent()方法接受两个参数：第一个是要监控的事件，第二个是事件触发时处理该事件的函数。

注意：

MooTools 会给 window 和 document 对象自动添加 addEvent()方法。

使用 addEvents()方法可以一次注册多个事件处理程序。这个方法接受一个对象作为参数，该对象的属性名匹配事件类型的名称，属性的值就是要用于处理事件的函数。

例如，下面的代码在一个元素上注册了 mouseover 和 mouseout 事件的处理程序：

```
function eventHandler(e)
{
    // do something with the event here
}

var handlers = new Object();
handlers.mouseover = eventHandler;
handlers.mouseout = eventHandler;

$("myDiv").addEvents(handlers);
```

触发事件时，如果处理程序是通过 addEvent()或 addEvents()方法设置的，MooTools 就会把它自己的 Event 对象传送给事件处理函数。这个对象有自己专用的一组属性，其中一些属性类似于 W3C 的 Event 和 MouseEvent 对象，一些属性不同于 IE 或 W3C 事件模型中的任何属性，尽管它们提供相同的信息。表 15-2 列出了 MooTools 的 Event 对象提供的一些属性。

表　15-2

属　　性	说　　明
page.x	鼠标相对于浏览器窗口的水平位置
page.y	鼠标相对于浏览器窗口的垂直位置
client.x	鼠标相对于客户区的水平位置
client.y	鼠标相对于客户区的垂直位置
Target	事件目标
relatedTarget	相对于事件目标的元素
Type	调用事件处理程序的事件类型

注意：

要查看 MooTools 的 Event 对象的完整属性列表，请参阅 http://mootools.net/docs/core/Native/Event。

例如，下面的代码给 id 为 myDiv 的元素注册了 click 事件的处理程序：

```
function myDiv_onclick(e)
{
    var eSrc = $(e.target).addClass("myClass");

    alert("You clicked at X: " + e.client.x + " Y: " + e.client.y);
}

$("myDiv").addEvent("click", myDiv_onclick);
```

触发事件，执行 myDiv_onclick()时，会将 MooTools 的 Event 对象传送给该函数。该函数的第一行扩展了事件目标元素，并调用 addClass()方法，把 CSS 类 myClass 添加到元素上。addClass()方法返回扩展的 Element 对象，并存储在 eSrc 变量中。然后使用一个警告框，通过 client.x 和 client.y 属性显示鼠标指针相对于视区的坐标。

15.5.5 用 MooTools 重写 DHTML 工具栏

前面介绍了 MooTools 操作 DOM 的功能，下面使用它们来两次重写第 12 章的 DHTML 工具栏。

试一试 用 MooTools 重写 DHTML 工具栏

打开文本编辑器，输入如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 15: Example 7 with MooTools</title>
    <style type="text/css">
    .tabStrip
    {
        background-color: #E4E2D5;
        padding: 3px;
        height: 22px;
    }

    .tabStrip-tab
    {
         float: left;
        font: 14px arial;
        cursor: pointer;
        padding: 2px;
        border: 1px solid transparent;
    }

    .tabStrip-tab-hover
    {
        border-color: #316AC5;
```

```
    background-color: #C1D2EE;
}

.tabStrip-tab-click
{
    border-color: #facc5a;
    background-color: #f9e391;
}
</style>
<script src="mootools-1.2.3-core-yc.js" type="text/javascript">
</script>
<script type="text/javascript">
var currentNum = 0;
function handleEvent(e)
{
    var el = $(e.target);

    if (e.type == "mouseover" || e.type == "mouseout")
    {
        if (el.hasClass("tabStrip-tab") &&
            !el.hasClass("tabStrip-tab-click"))
        {
            el.toggleClass("tabStrip-tab-hover");
        }
    }

    if (e.type == "click")
    {
        if (el.hasClass("tabStrip-tab-hover"))
        {
            var id = el.id;
            var num = id.substr(id.lastIndexOf("-") + 1);

            if (currentNum != num)
            {
                deactivateTab();

                el.toggleClass("tabStrip-tab-hover")
                    .toggleClass("tabStrip-tab-click");
                showDescription(num);
                currentNum = num;
            }
        }
    }
}

function showDescription(num)
{
    var attributes = new Object();
    attributes.id = "tabStrip-desc-" + num;

    var div = new Element("div", attributes)
        .appendText("Description for tab " + num);
```

```
        $("descContainer").adopt(div);
    }

    function deactivateTab()
    {
        var descEl = $("tabStrip-desc-" + currentNum);

        if (descEl)
        {
            descEl.dispose();
            $("tabStrip-tab-" + currentNum)
                .toggleClass("tabStrip-tab-click");
        }
    }

    var handlers = new Object();
    handlers.mouseover = handleEvent;
    handlers.mouseout = handleEvent;
    handlers.click = handleEvent;

    document.addEvents(handlers);
    </script>
</head>
<body>
    <div class="tabStrip">
        <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
        <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
        <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
    </div>
    <div id="descContainer"></div>
</body>
</html>
```

把这个文件保存为 ch15_examp7.htm，在 MooTools 支持的任意浏览器中打开它。注意这个页面与其他所有版本的工作过程是相同的。

下面直接跳到代码上，从 handleEvent()函数开始：

```
function handleEvent(e)
{
    var el = $(e.target);
```

MooTools 把它的 Event 对象传送给这个函数。这个对象的一个属性是 target，与 W3C 事件模型的 Event 对象一样，MooTools 的 Event 对象也包含触发事件的元素。我们把事件目标传送给美元方法，以添加 MooTools 的扩展，再把得到的扩展 Element 对象保存到 el 变量中。

接着检查触发的事件类型。在这个示例中，主要对 mouseover 和 mouseout 事件感兴趣。

```
    if (e.type == "mouseover" || e.type == "mouseout")
    {
        if (el.hasClass("tabStrip-tab") &&
                !el.hasClass("tabStrip-tab-click"))
```

```
        {
            el.toggleClass("tabStrip-tab-hover");
        }
    }
```

对于这两个事件，检查事件目标元素是否有 CSS 类 tabStrip-tab，以确定事件目标是否是一个选项卡元素。如果是，且只有该选项卡没有 tabStrip-tab-click 类时，才切换 tabStrip-tab-hover 类。

接着确定单击事件是否导致事件处理程序的执行，然后确定事件目标是否为应用了 CSS 类 tabStrip-tab-hover 的选项卡。

```
    if (e.type == "click")
    {
        if (el.hasClass("tabStrip-tab-hover"))
        {
```

如果是，就检索与选项卡关联的编号，把它赋予 num 变量，如下面的代码所示：

```
            var id = el.id;
            var num = id.substr(id.lastIndexOf("-") + 1);
```

下一步是确定选项卡是否是当前活动的选项卡，如果不是，就将当前活动的选项卡改为不活动状态。

```
            if (currentNum != num)
            {
                deactivateTab();
```

接着关闭元素的 tabStrip-tab-hover 类，打开 tabStrip-tab-click 类：

```
                el.toggleClass("tabStrip-tab-hover")
                    .toggleClass("tabStrip-tab-click");
```

这个函数的最后一步是显示新单击的选项卡的描述信息，把 num 包含的值存储在 currentNum 变量中：

```
                showDescription(num);
                currentNum = num;
            }
        }
    }
}
```

由于 Prototype 和 MooTools 非常类似，所以 showDescription()函数非常类似于 Prototype 版本。

```
function showDescription(num)
{
    var attributes = new Object();
    attributes.id = "tabStrip-desc-" + num;
```

首先创建 attributes 对象，给 id 属性赋予新元素 所需的值。接着调用 Element 的构造函数，并链接 appendText()方法，以创建 元素。把 appendText()的返回值 Element 对象赋予 div 变量。

```
    var div = new Element("div", attributes)
        .appendText("Description for tab " + num);
```

最后，至少对于这个函数而言，使用 adopt()方法把新元素追加到 id 为 descContainer 的 元素上。

```
    $("descContainer").adopt(div);
}
```

与 showDescription()不同，deactivateTab()函数类似于 jQuery 版本，与 Prototype 版本不同。

```
function deactivateTab()
{
    var descEl = $("tabStrip-desc-" + currentNum);
```

首先获取包含当前活动的选项卡描述信息的 元素，这个元素可能在 DOM 中，也可能不在 DOM 中。为了避免出错，需要检查能否找到该元素。如果找不到该元素，美元函数就返回 null，null 是一个 false-y 值，所以可以把 descEl 变量用作 if 语句的条件。

```
    if (descEl)
    {
        descEl.dispose();
        $("tabStrip-tab-" + currentNum)
            .toggleClass("tabStrip-tab-click");
    }
}
```

如果找到了该元素，就使用 dispose()方法从 DOM 中删除它。接着检索当前活动的选项卡的 元素，用 toggleClass()方法删除 tabStrip-tab-click 类。

最后，注册事件处理程序，来处理 mouseover、mouseout 和 click 事件：

```
var handlers = new Object();
handlers.mouseover = handleEvent;
handlers.mouseout = handleEvent;
handlers.click = handleEvent;

document.addEvents(handlers);
```

这段代码创建了一个 handlers 对象，又创建了其属性，属性名与要处理的事件同名。给这些属性赋予指向 handleEvent()函数的指针，再把 handlers 对象传送给 document 对象的 addEvents()方法。

15.5.6　MooTools 中的 Ajax 支持

MooTools 中的 Ajax 工具与 jQuery 大不相同，与 Prototype 也有些区别。MooTools 的 Ajax 工具是围绕 Request 引用类型建立的。与 Prototype 的 Ajax.Request 一样，需要创建 Request 的一个实例，给它传送包含一组选项的对象。表 15-3 列出了其中的一些选项。

表　15-3

选　项	说　明
async	确定 XMLHttpRequest 对象是否以异步模式发出请求，默认为 true
method	用于请求的 HTTP 方法，默认是 post，get 是另一个有效值
onSuccess	成功完成请求时调用的回调函数
onFailure	请求完成，但得到一个错误状态码时调用的回调函数
url	接收请求的 URL

注意：

要查看选项和回调函数的完整列表，请参阅 http://mootools.net/docs/core/Request/Request。

所有的回调函数都会执行，并接受不同的参数。onSuccess 回调函数需要两个参数，第一个参数是 XMLHttpRequest 对象的 responseText，第二个参数是 responseXML。onFailure 回调函数仅需要一个 XMLHttpRequest 对象参数。使用 Request 引用类型发出请求，如下面的代码所示：

```
function request_onsuccess(text, xml)
{
    alert(text);
}

function request_onfailure(request)
{
    alert("An error occurred! HTTP status code is " + request.status);
}

var options = new Object();
options.method = "get";
options.onSuccess = request_onsuccess;

options.onFailure = request_onfailure;
options.url = "someTextFile.txt";

var request = new Request(options).send();
```

前几行代码定义了 request_onsuccess()和 request_onfailure()函数。定义了函数后，就创建 options 对象。代码设置的第一个选项是 method，它设置为 get。后两个选项是 onSuccess 和 onFailure，给它们分别赋予 request_onsuccess()和 request_onfailure()函数。最后一个选项是 url，给它赋予 someTextFile.txt。

创建了所有选项后，就调用 Request 构造函数，把 options 对象传送给它。接着链接 send()

方法，以发送请求。

可以使用 send()方法发送参数。只需给该方法传送包含参数的字符串即可，如下面的代码所示：

```
var request = new Request(options).send("name=Jeremy");
```

下面使用 MooTools 的 Ajax 工具修改上一章的表单验证器。

试一试　用 MooTools 重写表单验证器

打开文本编辑器，输入下面的代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Chapter 15: Example 8 with MooTools</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
    <script src="mootools-1.2.3-core-yc.js" type="text/javascript"></script>
    <script type="text/javascript">
    function checkUsername()
    {
        var userValue = $("username").value;

        if (userValue == "")
        {
            alert("Please enter a user name to check!");
            return;
        }

        var options = getBasicOptions();
        options.onSuccess = checkUsername_callBack;

        new Request(options).send("username=" + userValue);
    }

    function checkUsername_callBack(text, xml)
    {
        var userValue = $("username").value;

        if (text == "available")
        {
```

```
        alert("The username " + userValue + " is available!");
    }
    else
    {
        alert("We're sorry, but " + userValue + " is not available.");
    }
}

function checkEmail()
{
    var emailValue = $("email").value;

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return;
    }

    var options = getBasicOptions();
    options.onSuccess = checkEmail_callBack;

    new Request(options).send("email=" + emailValue);
}

function checkEmail_callBack(text, xml)
{
    var emailValue = $("email").value;

    if (text == "available")
    {
        alert("The email " + emailValue + "is currently not in use!");
    }
    else
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user.");
    }
}

function request_onfailure(request)
{
    alert("An error occurred. HTTP Status Code: " + request.status);
}

function getBasicOptions()
{
    var options = new Object();
    options.method = "get";
    options.onFailure = request_onfailure;
    options.url = "formvalidator.php";

    return options;
}
</script>
```

```
</head>
<body>
    <form>
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
                <td>
                    <a href="javascript: checkUsername()">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
                <td>
                    <a href="javascript: checkEmail()">Check Availability</a>
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>
                <td />
            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
                <td />
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submit" value="Submit" />
                </td>
                <td />
            </tr>
```

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

在 Web 服务器的根目录下把这个文件保存为 ch15_examp8.htm。打开浏览器，指向 http://yourserver/ch15_examp8.htm，测试该表单。它的工作过程与以前的所有版本都相同。

checkUsername()、checkUsername_callback()、checkEmail()和 checkEmail_callback()都有变化。还增加了两个新函数 request_onfailure()和 getBasicOptions()。

下面先分析这些新函数，从 getBasicOptions()开始。与这个脚本的 Prototype 版本一样，创建这个函数，是为了用所有请求都使用的设置生成一个 options 对象。这可以节省时间，减少输入量，因为这些代码只需输入一次，并可在需要时随时调用它。函数定义如下：

```
function getBasicOptions()
{
    var options = new Object();
    options.method = "get";
    options.onFailure = request_onfailure;
    options.url = "formvalidator.php";

    return options;
}
```

在这个函数中创建了 options 对象以及 method、onFailure 和 url 属性。给这些属性分别赋予如下值：get、request_onfailure 和 formvalidator.php。XMLHttpRequest 对象会发出一个对 formvalidator.php 的 GET 请求，request_onfailure()函数仅在请求对象遇到错误时执行。

request_onfailure()函数仅告诉用户发生了一个错误，如下所示：

```
function request_onfailure(request)
{
    alert("An error occurred. HTTP Status Code: " + request.status);
}
```

之所以显示状态码，是因为如果请求出现了错误，这个状态码有助于调试脚本。

checkUsername()和 checkUsername_callback()函数的变化与前面的版本相同，所以只需查看发生了变化的代码行。checkUsername()中的第一个变化是检索输入到 Username 字段中的信息的方式：

```
var userValue = $("username").value;
```

在 MooTools 中，只需使用美元函数检索元素，再使用 value 属性即可。

下一个变化位于函数末尾发出请求的地方。首先使用 getBasicOptions()函数返回一个 options 对象，以及要用于请求的预定义选项。接着添加 onSuccess 选项，给它赋予指向 checkUsername_ callback()函数的指针。

```
var options = getBasicOptions();
options.onSuccess = checkUsername_callBack;
```

然后发出请求，具体做法是用构造函数创建一个新的 Request 对象，给它传送 options 对象，再用 username 参数发送请求。

```
new Request(options).send("username=" + userValue);
```

请求成功时，会调用 checkUsername_callback()函数。这个函数的第一个变化是参数。MooTools 分别把 XMLHttpRequest 的 responseText 和 responseXML 属性传送给 Request 对象的回调函数。

```
function checkUsername_callBack(text, xml)
```

这里把它们简称为 text 和 xml。接着再次使用美元函数和 value 属性检索 Username 文本框中的文本。

```
var userValue = $("username").value;
```

这个函数的最后一个变化是 if 语句的条件：

```
if (text == "available")
```

比较 text 参数的值和字符串"available"，并显示一个消息，告诉用户，用户名是否可用。

checkEmail()和 checkEmail_callback()函数的变化分别与 checkUsername()和 checkUsername_callback()相同。这里仅讨论这些变化，首先分析 checkEmail()。

这个函数的第一个变化是 Email 文本框的检索：

```
var emailValue = $("email").value;
```

使用美元函数，给它传送<input/>元素的 id，再使用 value 属性检索文本。下一个变化是发出请求，但在发出请求之前，必须先创建 options 对象，给它填充基本设置，再添加 onSuccess 选项。

```
var options = getBasicOptions();
options.onSuccess = checkEmail_callBack;
```

设置了所有选项后，就调用 Request 构造函数，给它传送 options 对象，再向服务器发送 email 参数及其值。

```
new Request(options).send("email=" + emailValue);
```

请求成功后，Request 对象就调用 checkEmail_callback()，给它传送 XMLHttpRequest 对象的 reponseText 和 responseXML 属性。

```
function checkEmail_callBack(text, xml)
```

再次选择元素，使用 value 属性，从 Email 文本框中检索其值。

```
var emailValue = $("email").value;
```

接着确定来自服务器的响应是否指出，该电子邮件地址可以由用户使用。

```
if (text == "available")
```

MooTools 是一个流行的框架，因为它提供了类似于 jQuery 的工具，同时还像 Prototype 那样保留了传统 DOM 编程的一些方面。MooTools 也有动画/效果组件，因此是一个全方位的框架。本节很难覆盖这个框架的所有方面，所以读者应访问 http://www.mootools.net/docs/core 上的 API 文档。

15.6　小结

本章带读者进入了 JavaScript 框架这个相当庞大的领域：

- JavaScript 框架是跨浏览器开发的解决方案。
- 有两种 JavaScript 框架：通用框架和专用框架。本章还列出了目前可用的一些流行框架。
- 本章指出了可从哪里获得使用 jQuery、Prototype 和 MooTools 框架所需的文件。
- 安装 jQuery、Prototype 和 MooTools 框架，并用相同的测试页面测试每个安装。
- 使用 jQuery、Prototype 和 MooTools 框架选择和检索元素、操作 DOM 和处理事件，并用这 3 个框架重写了 DHTML 工具栏脚本。
- 最后，使用 jQuery、Prototype 和 MooTools 框架的 Ajax 组件发出基本的 Ajax 请求，还使用这 3 个框架的 Ajax 功能重写了表单验证器脚本。

15.7　习题

附录 A 给出了这些练习题的答案。

习题 1：

使用 jQuery 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

习题 2：

使用 Prototype 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

习题 3：

使用 MooTools 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

附录 A

参考答案

本附录 A 提供了本书各章习题的参考答案。

第 2 章

习题 1

编写一个 JavaScript 程序，将摄氏温度转换为华氏温度，并将结果保存在一个描述性语句中，输出到页面上。将摄氏温度转换为华氏温度的 JavaScript 等式如下所示：

```
degFahren = 9 / 5 * degCent + 32
```

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var degCent = prompt("Enter the degrees in centigrade",0);
var degFahren = 9 / 5 * degCent + 32;

document.write(degCent + " degrees centigrade is " + degFahren +
   " degrees Fahrenheit");

</script>

</body>
</html>
```

保存为 ch2_q1.htm。

习题 2

下面的代码使用 prompt()函数，从用户的输入中获得两个数值，并将这两个值相加后输出到页面中。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<script language="JavaScript" type="text/javascript">

var firstNumber = prompt("Enter the first number","");
var secondNumber = prompt("Enter the second number","");
var theTotal = firstNumber + secondNumber;
document.write(firstNumber + " added to " + secondNumber + " equals " +
   theTotal);

</script>
</body>
</html>
```

但是，如果执行上面的代码，会发现代码不能正常工作。为什么？请修改代码，使之能正常工作。

参考答案

prompt()函数获得的数据实际上是一个字符串，因此 firstNumber 和 secondNumber 包含的文本是数值字符。使用+符号将两个变量相加时，JavaScript 将认为，由于它们是字符串数据，因此要把两个字符串连接在一起，而不是对它们求和。

为了显式地向 JavaScript 说明，要将两个数值相加在一起，需要使用 parseFloat()函数将数据转换为数值。

```
var firstNumber = parseFloat(prompt("Enter the first number",""));
var secondNumber = parseFloat(prompt("Enter the second number",""));
var theTotal = firstNumber + secondNumber;
document.write(firstNumber + " added to " + secondNumber + " equals " +
   theTotal);
```

保存为 ch2_q2.htm。

现在 prompt()函数返回的数据转换为浮点数，再保存在 firstNumber 和 secondNumber 变量中。最后把它们的和保存在 theTotal 中。JavaScript 作出了正确的推断，因为两个变量都是数值，代码会把两个数值加在一起，而不是连接它们。

一般的规则是，如果表达式仅包含数值数据，则+运算符表示“求和”，如果表达式包含字符串数据，则+表示连接。

第 3 章

习题 1

下面是某个初学者编写的一段代码，它不能正常运行。请指出代码中的错误并改正。

```
var userAge = prompt("Please enter your age");

if (userAge = 0);
{
```

```
   alert("So you' re a baby!");
}
else if ( userAge < 0 | userAge > 200)
   alert("I think you may be lying about your age");
else
{
   alert("That's a good age");
}
```

参考答案

这段初学者的代码有一些错误。下面的代码包含两个错误：

```
if (userAge = 0);
```

首先，在 if 的条件中，只有一个等号，而不是两个，这表示给 userAge 赋予 0，而不是比较 userAge 和 0。第二个错误是句末的分号，if、for 和 while 等语句不需要分号。一般规则是，如果语句包含一个放在大括号中的代码块，就不需要分号。因此，正确的代码应该是：

```
if (userAge == 0)
The next fault is with these lines:
else if ( userAge < 0 | userAge > 200)
   alert("I think you may be lying about your age");
else
```

该初学者的条件是检查 userAge 是否小于 0，或者是否大于 200。正确的布尔或运算符为||，但是代码中只有一个 |。

习题 2

使用 document.write()，输出 12 的乘法表，输出应如下所示：

```
12 * 1 = 12
12 * 2 = 24
12 * 3 = 36
...
12 * 11 = 132
12 * 12 = 144
```

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

var timesTable = 12;
var timesBy;
```

```
for (timesBy = 1; timesBy < 13; timesBy++)
{
   document.write(timesTable + " * " + timesBy + " = " + timesBy * timesTable +
      "<br />");
}



</script>

</body>
</html>
```

保存为 ch3_q2.htm。

上面的代码使用 for 循环计算从 1*12 到 12*12，并使用 document. write()方法将结果输出到页面上。这里要注意运算符的优先级；连接运算符(+)的优先级比乘法运算符(*)低，因此先计算 timesBy * timesTable，再进行字符串的连接。否则，就需要将运算放在圆括号中，以提高其运算优先顺序。

习题 3

将习题 2 中的代码改为一个函数，其参数是乘法表以及起始值和终止值。例如，计算 4 的乘法表，就是从 4*4 开始，一直计算到 4*9。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

function writeTimesTable(timesTable, timesByStart, timesByEnd)
{
    for (;timesByStart <= timesByEnd; timesByStart++)
    {
       document.write(timesTable + " * " + timesByStart + " = " +
          timesByStart * timesTable + "<br />");
    }
}

writeTimesTable(4,4,9);


</script>

</body>
</html>
```

保存为 ch3_q3.htm。

上面的代码声明了函数 writeTimesTable()，该函数接收 3 个参数。第一是要在其中写入结果的乘法表，第二是乘法表的起始值，第三是乘法表的上限结束值。

这里还修改了 for 循环。首先无须初始化任何变量，因此初始化部分为空，但是仍然需要放置一个分号，但分号之前不包含代码。只要参数 timesByStart 小于等于参数 timesByEnd，就继续执行 for 循环。可以看出，与变量一样，可以修改函数的参数。本例在每次循环时给 timesByStart 加 1。

显示乘法表的代码与前面相同。要执行函数的代码，需要实际调用该函数，如下所示：

```
writeTimesTable(4,4,9);
```

该行代码将输出 4 的乘法表，即从 4*4 开始，到 4*9 结束。

习题 4

修改习题 3 的代码，要求用户输入一个数值，然后显示这个数的乘法表。继续要求用户输入，并显示乘法表，直到用户输入 - 1。另进行检查，确保用户输入有效的值，如果用户输入无效，则提示用户重新输入。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>

<script type="text/javascript">

function writeTimesTable(timesTable, timesByStart, timesByEnd)
{
    for (;timesByStart <= timesByEnd; timesByStart++)
    {
       document.write(timesTable + " * " + timesByStart + " = " +
          timesByStart * timesTable + "<br />");
    }
}

var timesTable;

while ( (timesTable = prompt("Enter the times table",-1)) != -1)
{
   while (isNaN(timesTable) == true)
   {
    timesTable = prompt(timesTable + "is not a valid number, please retry",-1);
   }

   if (timesTable == -1)
   {
      break;
   }

   document.write("<br />The " + timesTable + " times table<br />");
```

```
    writeTimesTable(timesTable,1,12);
}

</script>

</body>
</html>
```

保存为 ch3_q4.htm。

本例的函数没有任何改变，因此只需讨论新代码。对习题 3 所做的第一个改变是声明了一个变量 timesTable，并在第一个 while 语句的条件中初始化它。这看起来似乎有点奇怪，但确实奏效。先执行 while 循环条件的圆括号中的代码：

```
(timesTable = prompt("Enter the times table",-1))
```

因为其运算优先级通过圆括号提高了。该代码返回一个值，该返回值与 - 1 进行比较。如果返回值不等于 - 1，则 while 条件为 true，执行循环体。否则，跳过循环体，页面上不会有任何输出。

嵌套在第一个 while 循环中的第二个 while 循环用 isNaN()函数来检查用户是否输入了有效的数字。如果输入无效，则要求用户重新输入，直到用户输入有效的数字为止。

如果用户开始时输入了无效值，则在第二个 while 循环中检查用户是否输入了 - 1。因此，在该 while 语句后，使用 if 语句来检查 timesTable 是否等于 - 1。如果是，则结束 while 循环，否则就调用 writeTimesTable()函数。

第 4 章

习题 1

本章示例 debug_timestable2.htm 页面中包含一个故意设置的错误。对于每一个乘法表，它仅仅计算从 1 到 11 的乘法。

使用脚本调试器找出原因，并修正该错误。

参考答案

代码的逻辑有问题，但语法正确。逻辑错误更难查找和处理，因为与语法错误不同，浏览器不会提示某行有错误，只是不像期望的那样工作。下面代码的错误是：

```
for (counter = 1; counter < 12; counter++)
```

代码应从 1 循环到 12(包括 12)。而 counter < 12 语句至多循环到 11，当计数器等于 12 时，该语句就是 false，所以漏掉了第 12 次循环。要更正这个问题，可将代码修改为：

```
for (counter = 1; counter <= 12; counter++)
```

习题 2

下面的代码包含了多个常见错误，请找出这些错误。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 4, Question 2</title>
</head>
<body>
<script type="text/javascript">
function checkForm(theForm)
{
    var formValid = true;
    var elementCount = 0;
    while(elementCount =< theForm.length)
    {
        if (theForm.elements[elementcount].type == "text")
        {
            if (theForm.elements[elementCount].value() = "")
               alert("Please complete all form elements")
            theForm.elements[elementCount].focus;
            formValid = false;
            break;
        }
    }
    return formValid;
}
</script>
<form name="form1" onsubmit="return checkForm(document.form1)" action="">
    <input type="text" id="text1" name="text1" />
    <br />
    CheckBox 1<input type="checkbox" id="checkbox2" name="checkbox2" />
    <br />
    CheckBox 1<input type="checkbox" id="checkbox1" name="checkbox1" />
    <br />
    <input type="text" id="text2" name="text2" />
    <p>
        <input type="submit" value="Submit" id="submit1" name="submit1" />
    </p>
</form>
</body>
</html>
```

参考答案

正确的代码如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title>Chapter 4, Question 2: Answer</title>
</head>
<body>
<script type="text/javascript">
function checkForm(theForm)
{
    var formValid = true;
    var elementCount = 0;

    while(elementCount < theForm.length)
    {
        if (theForm.elements[elementCount].type == "text")
        {
            if (theForm.elements[elementCount].value == "")
            {
                alert("Please complete all form elements")
                theForm.elements[elementCount].focus();
                formValid = false;
                break;
            }
        }

        elementCount++;
    }

    return formValid;
}
</script>
<form name="form1" onsubmit="return checkForm(document.form1)" action="">
    <input type="text" id="text1" name="text1" />
    <br />
    CheckBox 1<input type="checkbox" id="checkbox2" name="checkbox2" />
    <br />
    CheckBox 1<input type="checkbox" id="checkbox1" name="checkbox1" />
    <br />
    <input type="text" id="text2" name="text2" />
    <p>
        <input type="submit" value="Submit" id="submit1" name="submit1" />
    </p>
</form>
</body>
</html>
```

下面逐一分析每个错误。

第一个错误是逻辑错误：

```
while(elementCount =< theForm.length)
```

数组从 0 开始，第一个 Form 对象的数组下标为 0，第二个 Form 对象的下标为 1，依此类推。最后一个 Form 对象的下标为 4。但是，theForm.length 返回 5，因为表单包含 5 个元素。因此，elementCount 小于等于 5 时，while 循环会继续执行。但最后一个元素的索引

为 4，while 语句将多执行一次。正确的代码是：

```
while(elementCount < theForm.length)
```

或者：

```
while(elementCount <= theForm.length - 1)
```

这两种代码都有效，第一种形式更简洁。

第二个错误在下面的代码中：

```
if (theForm.elements[elementcount].type == "text")
```

粗略一看，这行代码似乎没有错误，由于 JavaScript 严格区分大小写，因此上面的代码有误：变量名是 elementCount，不是 elementcount，正确的代码如下所示：

```
if (theForm.elements[elementCount].type == "text")
```

下面的语句也有错误：

```
if (theForm.elements[elementCount].value() = "")
```

该语句有两个错误，首先，value 是一个属性，而不是方法，因此其后不需要圆括号。其次，将两个等号写成了一个等号。一个等号代表“使它等于”，两个等号表示“检查它等于”。因此，正确的代码如下所示：

```
if (theForm.elements[elementCount].value == "")
```

下一个错误在于未将 if 的代码块放在大括号中。虽然代码的语法是正确的，JavaScript 不会抛出任何错误，但是其逻辑有错误，因而无法获得预期的结果。添加大括号后，if 语句如下所示：

```
if (theForm.elements[elementCount].value == "")
{
    alert("Please complete all form elements")
    theForm.elements[elementCount].focus;
    formValid = false;
    break;
}
```

倒数第二个错误在下面的代码中：

```
theForm.elements[elementCount].focus;
```

这次是在方法名之后没有圆括号。如果要执行方法，即使该方法没有任何参数，方法名之后也必须有空的圆括号。因此，正确代码为：

```
theForm.elements[elementCount].focus();
```

马上就完成了，还有一个错误。这次不是现有代码有错误，而是缺少了必要的语句：

```
elementCount++;
```

必须在 while 循环中添加这个语句，否则 elementCount 将永远等于 0，while 循环条件永远为 true，循环会永远执行下去：一个典型的死循环。

第 5 章

习题 1

使用 Date 类型，计算距今 12 个月后的日期，并输出在网页上。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Question 1</title>
</head>
<body>
<script type="text/javascript">

var months = new Array("Jan", "Feb","Mar","Apr","May","Jun","Jul","Aug",
                          "Sep", "Oct","Nov","Dec");
var nowDate = new Date();

nowDate.setMonth(nowDate.getMonth() + 12);
document.write("Date 12 months ahead is " + nowDate.getDate());
document.write("" + months[nowDate.getMonth()]);
document.write("" + nowDate.getFullYear());

</script>
</body>
</html>
```

保存为 ch05_q1.htm。

getMonth()方法返回一个 0～11 的数字，用于表示一年中的 12 个月份，而不表示各个月份的名称。因此创建 months 数组来保存每个月份的名称。使用 getMonth()方法可以获得相应月份名称的数组下标。

变量 nowDate 初始化为一个新的 Date 对象。由于没有指定初始值，因此新的 Date 对象包含当前的日期。

要给当前日期加上 12 个月，只需使用 setMonth()。使用 getMonth()方法获取正确的月份值，再加上 12 即可。

最后，将结果输出在页面上。

习题 2

让用户输入一个名字列表，并将名字保存在数组中。继续获取下一个名字，直到用户输入为空为止。然后按升序排列名字顺序，并输出在页面上，每个名字各占一行。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Question 2</title>
</head>
<body>
<script type="text/javascript">
var inputName = "";

var namesArray = new Array();

while ( (inputName = prompt("Enter a name , " ")) != "" )
{
    namesArray[namesArray.length] = inputName;
}

namesArray.sort();

var namesList = namesArray.join("<br/>")
document.write(namesList);
</script>
</body>
</html>
```

保存为 ch05_q2.htm。

上面的代码首先声明了两个变量。其中，inputName 保存用户输入的名字，而 namesArray 是一个 Array 对象，用于保存输入的所有名字。

只要用户的输入不为空，代码就使用 while 循环来获取用户输入的另一个名字。注意，while 条件中的圆括号是非常重要的。圆括号中的如下代码将率先执行，以获取用户输入的名字，并保存在变量 inputName 中。

```
(inputName = prompt("Enter a name",""))
```

接着比较圆括号中代码的返回值(用户输入的内容)与空字符串(表示为“”)。如果它们相等，即用户输入了值，就再次执行循环。

为了对数组进行排序，使用了 Array 对象的 sort()方法：

```
namesArray.sort();
```

最后为了创建一个包含数组中的所有名字、并且每个名字单独占一行的字符串，使用了 HTML
元素，代码如下：

```
var namesList = namesArray.join("<br/>")
document.write(namesList);
```

代码 namesArray.join("<br/>")以 HTML
作为分隔符，创建了一个包含所有数组元素的字符串。最后，使用 document.write()方法将该字符串输出到页面中。

习题 3

本章学习了如何使用 pow()方法来修整数值的小数位数。但是这个函数有一个缺陷。要把 2.1 修整为 3 位小数，正确的 fix()函数应该返回 2.100，但前面的 fix()函数返回 2.1。修改这个 fix()函数，补上所需的 0。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 5: Question 3</title>
    <script type="text/javascript">
    function fix(fixNumber, decimalPlaces)
    {
        var div = Math.pow(10,decimalPlaces);
        fixNumber = new String(Math.round(fixNumber * div) / div);
        if (fixNumber.lastIndexOf(".")==-1)
        {
            fixNumber = fixNumber +".";
        }

        var zerosRequired = decimalPlaces -
            (fixNumber.length - fixNumber.lastIndexOf(".") - 1);

        for (; zerosRequired > 0; zerosRequired-- )
        {
            fixNumber = fixNumber + "0";
        }
        return fixNumber;
    }
    </script>
</head>
<body>
<script type="text/javascript">
var number1 = prompt("Enter the number with decimal " +
    "places you want to fix", " ");
var number2 = prompt("How many decimal places do you " +
    "want?", "");

document.write(number1 + " fixed to " + number2 +
    " decimal places is: ");
document.write(fix(number1,number2));
</script>
</body>
</html>
```

保存为 ch05_q3.htm。

这里的函数声明和第一行代码与本章前面的 fix()函数相同。但其后的代码有所变化。

像以前一样使用 Math.round(fixNumber * div) / div 来创建经过修整的数字，但这里把其结果传送为 String()构造函数的参数，以创建一个新的 String 对象，并保存在 fixNumber 中。

现在获得了指定小数位数的数字，但是该数的形式仍然是 2.1，而不是需要的 2.100。因此，接下来需要添加额外的 0。为此，从 decimalPlaces 指定的小数位数中减去当前截取数的小数位数。首先使用如下表达式计算当前截取数的小数位数：

```
(fixNumber.length - fixNumber.lastIndexOf(".") - 1)
```

对于数字 2.1 来说，fixNumber.length 等于 3，而 fixNumber.lastIndexOf(".")返回 1。注意，第一个字符为 0，第二个字符为 1，依此类推。因此，fixNumber.length - fixNumber.lastIndexOf(".")返回 2。接着再减去 1，得到 1，这就是当前截取数的小数位数。

完整的代码是：

```
var zerosRequired = decimalPlaces -
            (fixNumber.length - fixNumber.lastIndexOf(".") - 1);
```

表达式后半部分(fixNumber.length - fixNumber.lastIndexOf(".") - 1)是 1，而 decimalPlaces 参数为 3，3-1 等于 2，这表示需要添加两个 0。

知道需要添加几个 0，就添加它们：

```
for (; zerosRequired > 0; zerosRequired--)
{
    fixNumber = fixNumber + "0";
}
```

最后，将函数的返回值返回给调用代码。

```
return fixNumber;
```

第 6 章

习题 1

创建一个包含多个链接的页面，并编写代码，触发 window 的 load 事件，以显示页面上每个链接的 href(提示：事件处理程序以 on 开头)。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Question 1</title>
    <script type="text/javascript">
    function displayLinks()
    {
```

```
        var linksCounter;

        for (linksCounter = 0; linksCounter < document.links.length;
linksCounter++)
        {
            alert(document.links[linksCounter].href);
        }
    }
    </script>
</head>
<body onload="displayLinks()">
<a href="link0.htm">Link 0</a>
<a href="link1.htm">Link 2</a>
<a href="link2.htm">Link 2</a>
</body>
</html>
```

保存为 ch06_q1.htm。

在<body>开始标记中，添加一个属性，以连接 window 对象的 onload 事件处理程序。

```
<body onload="displayLinks()">
```

onload 事件触发时，会执行引号中调用 displayLinks()函数的脚本。

displayLinks()函数使用 for 循环来遍历 document 对象的 links 集合中的每个 a 对象。

```
function displayLinks()
{
    var linksCounter

    for (linksCounter = 0; linksCounter < document.links.length; linksCounter++)
    {
        alert(document.links[linksCounter].href);
    }

}
```

上面的代码在条件中使用 links 集合的 length 属性来确定需要循环多少次。接着使用警告框显示每个 a 对象的 href 属性。不能在 load 事件中使用 document.write()，因为 load 事件在页面加载完毕后触发。

习题 2

创建两个页面 ieonly.htm 和 notieonly.htm。每个页面都包含一个标题，以说明当前加载的是哪个页面。例如：

```
<H2>Welcome to the Internet Explorer only page</H2>
```

使用检查浏览器类型的函数，并将该函数连接到 window 对象的 onload 事件处理程序，以检测用户浏览器的类型。如果浏览器加载的不是对应的页面，则重定向到另一个页面。

参考答案

notieonly.htm 页面如下所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example Question 2</title>
    <script type="text/javaScript">
    function getBrowserName()
    {
        var lsBrowser = navigator.userAgent;
        if (lsBrowser.indexOf("MSIE") >= 0)
        {
            lsBrowser = "MSIE";
        }
        else if (lsBrowser.indexOf(Firefox") >= 0)
        {
            lsBrowser = "Firefox";
        }
        else if (lsBrowser.indexOf("Chrome") >= 0)
        {
            lsBrowser = "Chrome";
        }
        else if (lsBrowser.indexOf("Safari") >= 0)
        {
            lsBrowser = "Safari";
        }
        else if (lsBrowser.indexOf("Opera") >= 0)
        {
            lsBrowser = "Opera";
        }
        else
        {
            lsBrowser = "UNKNOWN";
        }
        return lsBrowser;
    }

    function getBrowserVersion()
    {
        var findIndex;
        var browserVersion = 0;
        var browser = getBrowserName();

        browserVersion = navigator.userAgent;
        findIndex = browserVersion.indexOf(browser) + browser.length + 1;
        browserVersion = parseFloat(browserVersion.substring(findIndex,
                findIndex + 3));
```

```
        return browserVersion;
    }

    function checkBrowser()
    {
        if (getBrowserName() == "MSIE")
        {
          window.location.replace("ieonly.htm");
        }
    }
    </script>
</head>
<body onload="checkBrowser()">
<h2>Welcome to the Not-IE only page</h2>
</body>
</html>
```

ieonly.htm 页面非常类似：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Example Question 2</title>
    <script type="text/javaScript">
    function getBrowserName()
    {
        var lsBrowser = navigator.userAgent;
        if (lsBrowser.indexOf("MSIE") >= 0)
        {
            lsBrowser = "MSIE";
        }
        else if (lsBrowser.indexOf("Firefox") >= 0)
        {
            lsBrowser = "Firefox";
        }
        else if (lsBrowser.indexOf("Chrome") >= 0)
        {
            lsBrowser = "Chrome";
        }
        else if (lsBrowser.indexOf("Safari") >= 0)
        {
            lsBrowser = "Safari";
        }
        else if (lsBrowser.indexOf("Opera") >= 0)
        {
            lsBrowser = "Opera";
        }
        else
        {
```

```
            lsBrowser = "UNKNOWN";
        }
        return lsBrowser;
    }

    function getBrowserVersion()
    {
        var findIndex;
        var browserVersion = 0;
        var browser = getBrowserName();

        browserVersion = navigator.userAgent;
        findIndex = browserVersion.indexOf(browser) + browser.length + 1;
        browserVersion = parseFloat(browserVersion.substring(findIndex,
                findIndex + 3));

        return browserVersion;
    }

    function checkBrowser()
    {
        if (getBrowserName() != "MSIE")
        {
          window.location.replace("notieonly.htm");
        }
    }
    </script>
</head>
<body onload="checkBrowser()">
<h2>Welcome to the Internet Explorer only page</h2>
</body>
</html>
```

首先讨论 ieonly.htm 页面，该页面先添加一个 onload 事件处理程序，并连接到 checkBrowser()函数上，因此在加载该页面时，会调用它。

```
<body onload="checkBrowser()">
```

接着在 checkBrowser()函数中使用 getBrowserName()函数确定用户浏览器的类型。如果它不是 IE，则将页面替换为 notieonly.htm 页面。注意，这里用 replace()替代了 href，因为不希望用户单击浏览器的 Back 按钮。但这个方法不太容易确定当前加载的是哪个新页面。

```
function checkBrowser()
{
    if (getBrowserName() != "MSIE")
    {
        window.location.replace("notieonly.htm");
    }
}
```

notieonly.htm 页面与之相同，但 if 语句检查的是浏览器是否为 MSIE，如果是，则页面替换为 ieonly.htm。

```
function checkBrowser()
{
    if (getBrowserName() == "MSIE")
    {
        window.location.replace("ieonly.htm");
    }
}
```

习题 3

在页面上用<img/>元素插入一幅图片。当鼠标指针移到图片上时，该图片应切换成另一幅图片；当鼠标指针从图片上移开时，再切换回原来的图片(提示：使用 mouseover 和 mouseout 事件)。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 6: Question 3</title>
    <script type="text/javascript">
    function mouseOver(that)
    {
        that.src = "Img2.jpg";
    }
    function mouseOut(that)
    {
        that.src = "Img1.jpg";
    }
    </script>
</head>
<body>
<img src="Img1.jpg" name="myImage" onmouseover="mouseOver(this)"
    onmouseout="mouseOut(this)" />
</body>
</html>
```

保存为 ch06_q3.htm。

页面顶部定义了两个函数，分别用于处理 mouseover 和 mouseout 事件。

```
function mouseOver(that)
{
    that.src = "Img2.jpg";
}

function mouseOut(that)
```

```
{
    that.src = "Img1.jpg";
}
```

函数名说明它们处理什么事件。将 img 对象的引用传送给函数，就可以为<img/>元素访问 img 对象。在 mouseover 事件中，将图片的 src 属性改为 Img2.tif，在 mouseout 事件中，再把图片的 src 属性改回 img1.tif，Img1.tif 是加载页面时指定的图片。

在页面中，包含如下<img/>元素：

```
<img src="Img1.jpg" name="myImage" onmouseover="mouseOver(this)"
    onmouseout="mouseOut(this)" />
```

第 7 章

习题 1

使用第 2 章中温度转换例子的代码，创建一个用户界面，并将其连接到已有的代码上，以便用户输入一个华氏温度值，并转换为摄氏温度。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Question 1</title>
    <script type="text/javascript">
    function convertToCentigrade(degFahren)
    {
        var degCent = 5 / 9 * (degFahren - 32);

        return degCent;
    }

    function btnToCent_onclick()
    {
        var calcBox = document.form1.txtCalcBox;

        if (isNaN(calcBox.value) == true || calcBox.value == "")
        {
            calcBox.value = "Error Invalid Value";
        }
        else
        {
            calcBox.value = convertToCentigrade(calcBox.value);
        }
    }
    </script>
</head>
<body>
```

```
    <form action=" " name="form1">
        <p>
            <input type="text" name="txtCalcBox" value="0.0" />
        </p>
        <input type="button" value="Convert to centigrade"
            name="btnToCent" onclick="btnToCent_onclick()" />
    </form>
</body>
</html>
```

保存为 ch07_q1.htm。

界面部分只是一个表单，其中包含一个文本框和一个按钮。用户可以在文本框中输入华氏温度值，并单击按钮，将输入的值转换为摄氏温度。按钮的 onclick 事件处理程序设置为调用 btnToCent_onclick()函数。

btnToCent_onclick()的第一行声明了一个变量，以引用表示文本框的对象：

```
var calcBox = document.form1.txtCalcBox;
```

为什么这么做？因为代码中用到 document.form1.txtCalcBox 的地方，都可以使用简短的 calcBox 来代替，减少了输入量，提高了代码的可读性。

所以

```
alert(document.form1.txtCalcBox.value);
```

可简化为：

```
alert(calcBox.value);
```

函数的其余代码进行必要的检查，如果用户的输入是数字(不是 NotANumber)，且文本框包含值，则调用第 2 章的温度转换函数，将华氏温度转换为摄氏温度，结果用于设置文本框的值。

习题 2

创建一个用户界面，以允许用户根据自己的爱好挑选计算机系统，其原则类似于在因特网上销售计算机的电子商务网站。例如，允许用户选择处理器的类型、速度、内存和硬盘容量，并添加 DVD-ROM、声卡等附件。用户改变其选择时，系统的价格应自动更新，并通过使用警告框或更新文本框的内容，来通知用户所指定的系统的价格。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 7: Question 1</title>
    <script type="text/javascript">
    var compItems = new Array();
    compItems[100] = 1000;
```

```
compItems[101] = 1250;
compItems[102] = 1500;

compItems[200] = 35;
compItems[201] = 65;
compItems[202] = 95;

compItems[300] = 50;
compItems[301] = 75;
compItems[302] = 100;

compItems[400] = 10;
compItems[401] = 15;
compItems[402] = 25;
function updateOrderDetails()
{
    var total = 0;
    var orderDetails = "";
    var formElement;
    var theForm = document.form1;
    formElement = theForm.cboProcessor[theForm.cboProcessor.selectedIndex];
    total = parseFloat(compItems[formElement.value]);
    orderDetails = "Processor : " + formElement.text;
    orderDetails = orderDetails + " $" + compItems[formElement.value] + "\n";

    formElement = theForm.cboHardDrive[theForm.cboHardDrive.selectedIndex];
    total = total + parseFloat(compItems[formElement.value]);
    orderDetails = orderDetails + "Hard Drive : " + formElement.text;
    orderDetails = orderDetails + "$" + compItems[formElement.value] + "\n";

    formElement = theForm.chkCDROM;
    if (formElement.checked == true)
    {
        orderDetails = orderDetails + "CD-ROM : $" +
            compItems[formElement.value] + "\n";
        total = total + parseFloat(compItems[formElement.value]);
    }

    formElement = theForm.chkDVD;
    if (formElement.checked == true)
    {
        orderDetails = orderDetails + "DVD-ROM : $" +
            compItems[formElement.value] + "\n";
        total = total + parseFloat(compItems[formElement.value]);
    }

    formElement = theForm.chkScanner;
    if (formElement.checked == true)
    {
        orderDetails = orderDetails + "Scanner : $" +
            compItems[formElement.value] + "\n";
        total = total + parseFloat(compItems[formElement.value]);
    }
```

```
        formElement = theForm.radCase;
        if (formElement[0].checked == true)
        {
            orderDetails = orderDetails + "Desktop Case : $" +
                compItems[formElement[0].value];
            total = total + parseFloat(compItems[formElement[0].value]);
        }
        else if (formElement[1].checked == true)
        {
            orderDetails = orderDetails + "Mini Tower Case : $" +
                compItems[formElement[1].value];
            total = total + parseFloat(compItems[formElement[1].value]);
        }
        else
        {
            orderDetails = orderDetails + "Full Tower Case : $" +
                compItems[formElement[2].value];
            total = total + parseFloat(compItems[formElement[2].value]);
        }

        orderDetails = orderDetails + "\n\nTotal Order Cost is $" + total;

        theForm.txtOrder.value = orderDetails;
    }

    </script>
</head>
<body>
    <form action=" "name="form1">
        <table>
            <tr>
                <td width="300">
                    Processor
                    <br />
                        <select name="cboProcessor">
                            <option value="100">MegaPro 10ghz</option>
                            <option value="101">MegaPro 12</option>
                            <option value="102">MegaPro 15ghz</option>
                        </select>
                        <br />
                        <br />
                        Hard drive
                        <br />
                        <select name="cboHardDrive">
                            <option value="200">30tb</option>
                            <option value="201">40tb</option>
                            <option value="202">60tb</option>
                        </select>
                        <br />
                        <br />
                        CD-ROM
```

```
                    <input type="checkbox" name="chkCDROM" value="300" />
                    <br />
                    DVD-ROM
                    <input type="checkbox" name="chkDVD" value="301" />
                    <br />
                    Scanner
                    <input type="checkbox" name="chkScanner" value="302" />
                    <br />
                    <br />
                    Desktop Case
                    <input type="radio"name="radCase"checked value="400"/>
                    <br />
                    Mini Tower
                    <input type="radio" name="radCase" value="401" />
                    <br />
                    Full Tower
                    <input type="radio" name="radCase" value="402" />
                    <br />
                    <br />
                    <input type=" button" value="Update" name="butUpdate"
                        onclick="updateOrderDetails()"/>
                </td>
                <td>
                    <textarea rows="20" cols="35" id="txtOrder"
                        name="txtOrder"></textarea>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

保存为 ch07_q2.htm。

上面的代码给出了处理这个问题的一种方法，也许读者已经想到了更好的方法。

本例把用户选择的结果在 textarea 框中显示为文本，每一项及其价格占一行，最后显示总价。

每个表单元素都把 value 属性设置为库存 ID 号。例如，立式机箱(full tower)的库存 ID 为 402。该配件的实际价格保存在页面开头定义的数组中。为什么不直接将价格保存在每个表单元素的 value 属性中？因为使用 ID 更灵活。目前数组只包含每个配件的价格信息，但可以把它改为包含更多数据，例如价格、描述、库存数量等。另外，如果将表单提交给服务器，则所传送的值是库存 ID，可用于查询库存数据库。如果 value 属性设置为价格，并提交表单，就无法确定客户预定了什么，只知道其价格。

本答案包含更新按钮，单击该按钮时，会更新 textarea 框中的订单信息。也可以为每个表单元素添加事件处理程序，以便在发生改变时进行更新。

下面分析显示订单汇总信息的 updateOrderDetails()函数。该函数包含了大量代码，显得比较复杂。但实际上它非常简单。大部分代码都很相似，仅仅做了少量的修改。

为了减少输入并提高代码的可读性，本答案声明了两个变量：theForm 包含 Form 对象，formElement 依次引用表单中的各个元素，以获取库存 ID 和价格。在声明变量后，确定用户选择了哪种处理器，计算价格，并把信息添加到 textarea 框中。

```
formElement = theForm.cboProcessor[document.form1.cboProcessor.selectedIndex];
total = parseFloat(compItems[formElement.value]);
orderDetails = "Processor : " + formElement.text;
orderDetails = orderDetails + " $" + compItems[formElement.value] + "\n";
```

selectedIndex 属性表示用户在选择控件中选中了哪个 Option 对象，formElement 变量将引用这个对象。

确定选中的硬盘大小时，使用了相同的办法。接下来看看其他可选配件的复选框，首先分析 CD-ROM 复选框：

```
formElement = theForm.chkCDROM
if (formElement.checked == true)
{
    orderDetails = orderDetails + "CD-ROM : $" +
        compItems[formElement.value] + "\n";
    total = total + parseFloat(compItems[formElement.value]);
}
```

同样，使用 formElement 变量来引用 chkCDROM 复选框对象。如果该复选框被选中，则将一个 CD-ROM 添加到订单信息中，并更新总价格。DVD 和扫描仪复选框也采用同样的处理办法。

最后分析机箱类型，用户只能选择一种机箱，所以使用了单选按钮组。但是单选按钮没有像复选框那样的 selectedIndex 属性。因此，必须依次遍历每个单选按钮，以便检查选中了哪个单选按钮。

```
formElement = theForm.radCase
if (formElement[0].checked == true)
{
    orderDetails = orderDetails + "Desktop Case : $" +
        compItems[formElement[0].value];
    total = total + parseFloat(compItems[formElement[0].value]);
}
else if (formElement[1].checked == true)
{
    orderDetails = orderDetails + "Mini Tower Case : $" +
        compItems[formElement[1].value];
    total = total + parseFloat(compItems[formElement[1].value]);
}
else
{
    orderDetails = orderDetails + "Full Tower Case : $" +
        compItems[formElement[2].value]
    total = total + parseFloat(compItems[formElement[2].value]);
}
```

上面的代码检查用户选中了哪个单选按钮，并将它的详细信息添加到 textarea 中，将其价格添加到总价。如果代码块开头定义的库存数组包含更多信息，例如描述和价格，则可以遍历单选按钮数组，根据 CompItems 数组添加信息。

最后，把 textarea 设置为用户选中的计算机系统信息。

```
orderDetails = orderDetails + "\n\nTotal Order Cost is " + total;
theForm.txtOrder.value = orderDetails;
```

第 8 章

习题 1

在第 7 章的习题中，创建了一个表单，允许用户选配计算机系统。用户可以查看系统的信息及其总价，单击按钮，就会把信息写入一个 textarea。请将该例子改为基于框架的网页：该应用程序不把信息写入 textarea，而是写入另一个框架。

提示：使用 about:blank 作为要写入的框架的 src 属性，使用 document 对象的 close()和 open()方法清空信息框架中以前写入的数据。

参考答案

这个答案包含了一个框架集，将页面划分为左右框架。左框架显示了一个表单，供用户选择计算机系统的配件。用户单击 Update 按钮时，配件的汇总信息将显示在右框架中。

第一个页面框架集页面，浏览器先加载该页面。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Chapter 8: Question 1 Answer</title>
</head>
<frameset cols="55%,*">
    <frame src="ch8_q1_picksystem.htm" name="pickSystem">
    <frame src="about:blank" name="systemSummary">
</frameset>
</html>
```

保存为 ch8_q1_frameset.htm。

最后是左框架中加载的页面，它允许用户选择自己的计算机系统和配件。该页面与第 7 章中习题 2 的答案非常类似，因此本例仅仅列出修改的部分，所有的修改都在 updateOrderDetails()函数中进行。下面突出显示了修改的代码：

```
function updateOrderDetails()
{
    var total = 0;
    var orderDetails = "";
    var formElement;
```

```
var theForm = document.form1;
formElement = theForm.cboProcessor[theForm.cboProcessor.selectedIndex];
total = parseFloat(compItems[formElement.value]);
orderDetails = "Processor : " + formElement.text;
orderDetails = orderDetails + "$" + compItems[formElement.value] + "<br />";

formElement = theForm.cboHardDrive[theForm.cboHardDrive.selectedIndex];
total = total + parseFloat(compItems[formElement.value]);
orderDetails = orderDetails + "Hard Drive : " + formElement.text;
orderDetails = orderDetails + "$" + compItems[formElement.value] + "<br />";

formElement = theForm.chkCDROM;
if (formElement.checked == true)
{
    orderDetails = orderDetails + "CD-ROM : $"
        + compItems[formElement.value] + "<br />";
    total = total + parseFloat(compItems[formElement.value]);
}

formElement = theForm.chkDVD;
if (formElement.checked == true)
{
    orderDetails = orderDetails + "DVD-ROM : $"
        + compItems[formElement.value] + "<br />";
    total = total + parseFloat(compItems[formElement.value]);
}

formElement = theForm.chkScanner;
if (formElement.checked == true)
{
    orderDetails = orderDetails + "Scanner : $"
        + compItems[formElement.value] + "<br />";
    total = total + parseFloat(compItems[formElement.value]);
}

formElement = theForm.radCase;
if (formElement[0].checked == true)
{
    orderDetails = orderDetails + "Desktop Case : $"
        + compItems[formElement[0].value] + "<br />";
    total = total + parseFloat(compItems[formElement[0].value]);
}
else if (formElement[1].checked == true)
{
    orderDetails = orderDetails + "Mini Tower Case : $"
        + compItems[formElement[1].value] + "<br />";
    total = total + parseFloat(compItems[formElement[1].value]);
}

else
{
    orderDetails = orderDetails + "Full Tower Case : $"
```

```
            + compItems[formElement[2].value] + "<br />";
        total = total + parseFloat(compItems[formElement[2].value]);
    }

    orderDetails = orderDetails + "<p>Total Order Cost is $" + total + "</p>";

    top.systemSummary.document.close();
    top.systemSummary.document.open();
    top.systemSummary.document.write(orderDetails);
}
```

删除了页面中的<textarea/>元素，因为不再需要它。将文件保存为 ch8_ql_picksystem.htm，在浏览器中加载 ch8_q1_frameset.htm，以测试代码。

这个版本与第 7 章中习题 2 的版本之间的第一个区别在于，在创建汇总信息时，创建了 HTML，而不是纯文本，所以使用了
元素来替代\n。

主要的改变是下列三行代码：

```
top.systemSummary.document.close();
top.systemSummary.document.open();
top.systemSummary.document.write(orderDetails);
```

第 7 章的习题 2 设置了<textarea/>元素的值，而这里将订单汇总信息输出在 HTML 页面中，该页面包含在右框架 systemSummary 中。首先使用 close()关闭文档，否则后续的更新会把 HTML 追加到页面中，而不是替换当前的内容。接着使用 open()打开文档以准备写入信息，最后输出汇总字符串。

习题 2

本章的第 4 个例子 ch08_examp4.htm 是一个带图书图像的页面，单击某本书的图片，将在一个弹出窗口中显示该图书的信息。修改该例，在弹出窗口中添加一个按钮或链接，单击该按钮或链接时，把该图书添加到购物车中。另外，在主页面中用某种方式打开购物车窗口，其中包含用户购买的所有商品，并允许用户从中删除商品。

参考答案

到目前为止，这是最具有挑战性的题目。本书的最后将使用 JavaScript 创建一个更复杂的应用程序。本题的参考答案包含四个页面：前两个页面显示图书的信息(与本章中创建的页面非常类似)。第三个页面显示图书的图片，并打开新窗口。第四个页面是全新的，包含了购物车。

加载的主页面是 ch8_q2_online_books.htm：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 8: Question 2</title>
    <script type="text/javascript">
```

```
var detailsWindow;
var basketWindow;

var stockItems = new Array();
stockItems[0] = new StockItem("Professional Ajax, 2nd Edition", "$39.99");
stockItems[1] = new StockItem("Professonal JavaScript, 2nd Edition", "$46.99");

function removeItem(stockId)
{
    stockItems[stockId].quantity = 0;
    alert("Item Removed");
    showBasket();
    return false;
}

function addBookToBasket(stockId)
{
    stockItems[stockId].quantity = 1;
    alert("Item added successfully");
    detailsWindow.close();
}

function showDetails(bookURL)
{
    detailsWindow = window.open(bookURL,"bookDetails",
"width=400,height=500");
    detailsWindow.focus();
    return false;
}

function showBasket()
{
    basketWindow = window.open('about:blank', 'shoppingBasket',
        'width=400,height=350');
    basketWindow.document.open();
    var basketItem;
    var containsItems = false;
    basketWindow.document.write("<h4>Your shopping basket contains :</h4>");

    for (var i = 0; i < stockItems.length; i++)
    {
        var stockItem = stockItems[i];

        if (stockItem.quantity > 0)
        {
            basketWindow.document.write(stockItem.title + "at");
            basketWindow.document.write(stockItem.price);
            basketWindow.document.write("    ");
            basketWindow.document.write("<a href=' 'onclick='return"
                + "window.opener.removeItem(" + i + ")'>");
            basketWindow.document.write("Remove Item</a><br />");
            containsItems = true;
        }
```

```
        }

        if (!containsItems)
        {
            basketWindow.document.write("<h4>No items</h4>");
        }

        basketWindow.document.close();
        basketWindow.focus();
    }

    function StockItem(title, price)
    {
        this.title = title;
        this.price = price;
        this.quantity = 0;
    }
    </script>
</head>
<body>
    <h2 align="center">
        Online Book Buyer</h2>
    <form name="form1" action="">
        <input type="button" value="Show Shopping Basket" onclick="showBasket()"
         name="btnShowBasket" />
        </form>
        <p>
            Click any of the images below for more details
        </p>
        <h4>Professional Ajax</h4>
        <p>
            <img src="pro_ajax.jpg" alt="Professional Ajax, 2nd Edition" border="0"
                onclick="showDetails('pro_ajax_details.htm')" />
        </p>
        <h4>Professional JavaScript for Web Developers</h4>
        <p>
            <img src="pro_js.jpg" alt="Professional JavaScript, 2nd Edition"
             border="0"
                onclick="showDetails('pro_js_details.htm')" />
        </p>
</body>
</html>
```

图书的信息保存在 stockItems 数组中，stockItems 数组包含是 StockItem 对象，使用 StockItem()构造函数定义该引用类型。

```
function StockItem(title, price)
{
    this.title = title;
    this.price = price;
    this.quantity = 0;
}
```

这个函数创建的对象有 title、price 和 quantity 属性。前两个属性从同名的参数中获取值，第三个对象初始化为 0。

所以先填充 stockItems 数组，每个元素都包含一个 StockItem 对象，如下所示：

```
var stockItems = new Array();
stockItems[0] = new StockItem("Professional Ajax, 2nd Editionv", "$39.99");
stockItems[1] = new StockItem("Professonal JavaScript, 2nd Edition", "$46.99");
```

代码中定义的第一个函数是 removeItem()。

```
function removeItem(stockId)
{
    stockItems[stockId].quantity = 0;
    alert("Item Removed");
    showBasket();
    return false;
}
```

该函数从购物车中移除一本书。它接收一个参数 stockId，即图书的数组下标，该下标用于把 quantity 属性设置为 0。

接下来的 addBookToBasket()函数将一本书添加到购物车中。

```
function addBookToBasket(stockId)
{
    stockItems[stockId].quantity = 1;
    alert("Item added successfully");
    detailsWindow.close();
}
```

最后一个函数把购物车的内容显示在一个新窗口中。

```
function showBasket()
{
    basketWindow = window.open('about:blank', 'shoppingBasket',
        'width=400,height=350');
    basketWindow.document.open();
    var basketItem;
    var containsItems = false;
    basketWindow.document.write("<h4>Your shopping basket contains :</h4>");

    for (var i = 0; i < stockItems.length; i++)
    {
        var stockItem = stockItems[i];

        if (stockItem.quantity > 0)
        {
            basketWindow.document.write(stockItem.title + "at");
            basketWindow.document.write(stockItem.price);
            basketWindow.document.write("    ");
            basketWindow.document.write("<a href=' ' onclick='return "
                + "window.opener.removeItem(" + i + ")'>");
```

```
                basketWindow.document.write("Remove Item</a><br />");
                containsItems = true;
            }
        }

        if (!containsItems)
        {
            basketWindow.document.write("<h4>No items</h4>");
        }

        basketWindow.document.close();
        basketWindow.focus();
    }
```

该函数先打开一个新窗口，把其 window 对象引用保存在 basketWindow 中。然后向新窗口的 document 中写入一个标题，接着遍历 stockItems 数组中的每个元素，并检查 StockItems 对象的 quantity 属性。如果该属性大于 0，就将该图书的信息输出到购物列表窗口中。还给购物车输出了一个链接，单击该连接，就调用 removeItem()函数。

最后，需要创建图书描述页面。首先创建 pro_ajax_details.htm 页面，它与本章示例的版本相同，只是添加了表单和按钮。单击按钮时，将调用窗口中的 addToBasket()函数，打开这个窗口，即 ch8_q2_online_books.htm。下面突出显示的部分是对这个文件的修改：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Professional ASP.NET 2.0</title>
</head>
<body>
    <h3>Professional Ajax, 2nd Edition</h3>
    <form name="form1" action="">
        <input type="button" value="Add to basket" name="btnAddBook"
            onclick="window.opener.addBookToBasket(0)" />
    </form>
    <strong>Subjects</strong><br />
    Ajax<br />
    Internet<br />
    JavaScript<br />
    ASP.NET<br />
    PHP<br />
    XML<br />
    <hr color="#cc3333" />
    <h3>Book overview</h3>
    <p>
        A comprehensive look at the technologies and techniques used in Ajax,
        complete with real world examples and case studies. A must have for
        any Web professional looking to build interactive Web sites.
    </p>
</body>
</html>
```

最后创建页面 pro_js_details.htm。同样，该页面与本章示例创建中的版本相同，但利用表单和按钮把图书添加到购物车上，与 pro_ajax_details.htm 页面相同。下面突出显示的部分是对这个文件的修改：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Professional JavaScript</title>
</head>
<body>
    <h3>Professional JavaScript, 2nd Edition</h3>
    <form name="form1">
        <input type="button" value="Add to basket" name="btnAddBook"
            onclick="window.opener.addBookToBasket(1)" />
    </form>
    <strong>Subjects</strong>
    ECMAScript<br />
    Internet<br />
    JavaScript
    <br />
    XML and Scripting<br />
    <hr color="#cc3333" />
    <p>
        This book takes a comprehensive look at the JavaScript language
        and prepares the reader with in-depth knowledge of the languages.
    </p>
    <p>
        It includes a guide to the language - when where and how to get
        the most out of JavaScript - together with practical case studies
        demonstrating JavaScript in action. Coverage is bang up-to-date, with
        discussion of compatability issues and version differences, and the
        book concludes with a comprehensive reference section.
    </p>
</body>
</html>
```

第 9 章

习题 1

下面的代码解决了什么问题？

```
var myString = "This sentence has has a fault and and we need to fix it."
var myRegExp = /(\b\w+\b) \1/g;
myString = myString.replace(myRegExp, "$1");
```

现在假定修改代码，以创建如下的 RegExp 对象：

```
var myRegExp = new RegExp(" (\b\w+\b) \1");
```

为什么这行代码不起作用？该如何修正存在的问题？

参考答案

语句包含 has has 和 and and，显然是错误的。很多字处理软件都有自动纠正此类常见错误的功能。这里使用正则表达式来模拟这个功能。

错误的 myString 是：

```
"This sentence has has a fault and and we need to fix it."
```

正确的字符串是：

```
"This sentence has a fault and we need to fix it."
```

下面讨论代码的工作原理，首先是正则表达式：

```
/(\b\w+\b) \1/g;
```

使用圆括号定义了一个分组，即(\b\w+\b)是分组 1。该分组匹配的模式是一个单词分界，后跟一个或多个数字字母字符(即 a-z、A-Z 和 0-9 以及_字符)，最后一个单词分界。该分组的后面是一个空格和\1。\1 表示“精确匹配在模式分组 1 中匹配的字符”。例如，如果分组 1 匹配 has，则\1 也匹配 has。重要的是，由于\1 精确匹配分组 1 匹配的字符，因此，如果分组 1 匹配是 and，则\1 匹配 and，而不是匹配前面的 has。

replace()方法再次使用了这个分组。但这次使用$符号来指定分组，所以$1 匹配分组 1。这将把两个匹配的 has 和 and 替换为一个 has 和 and。

对于习题的第二部分，如何修改下面的代码，使之正常工作？

```
var myRegExp = new RegExp("(\b\w+\b) \1");
```

很简单，上面的代码把一个字符串传送给 RegExp 对象的构造函数，在表示正则表达式语法字符时，需要使用双斜线替换单斜线，如下所示：

```
var myRegExp = new RegExp("(\\b\\w+\\b) \\1","g");
```

注意，还传递了 g 字符作为第二个参数，进行全局匹配。

习题 2

请编写一个正则表达式，以查找下述语句中的所有单词 a，并替换为单词 the：

```
"a dog walked in off a street and ordered a finest beer"
```

替换之后的语句应为：

```
"the dog walked in off the street and ordered the finest beer"
```

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<script language="JavaScript" type="text/javascript">
var myString = "a dog walked in off a street and ordered a finest beer";
var myRegExp = /\ba\b/gi;
myString = myString.replace(myRegExp, "the");
alert(myString);
</script>
</body>
</html>
```

保存为 ch09_q2.htm。

使用正则表达式，得到的常常不仅仅是需要的字符，还有不希望匹配的字符。这里希望匹配字母 a，为什么正则表达式不能是下面的语句？

```
var myRegExp = /a/gi;
```

这个正则表达式也能运行，但是它将替换掉单词 walked 中的 a，这并不是我们希望的。我们希望替换的字母 a 是一个独立的单词，而不是包含在其他单词中的字母 a。一个字母何时会变成一个单词？位于两个单词分界之间的字母就是一个单词。单词分界可以用正则表达式的特殊字符\b 来表示。因此，正则表达式应是：

```
var myRegExp = /\ba\b/gi;
```

结尾的 gi 表示进行不区分大小写的全局匹配。

创建正则表达式后，就可以将其用于 replace()方法的第一个参数：

```
myString = myString.replace(myRegExp,"the");
```

习题 3

假如有一个带有留言板的网站，请编写一个正则表达式，以删除禁用的词汇(可以自己设置禁用的词汇)。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<script language="JavaScript" type="text/javascript">
   var myRegExp = /(sugar )?candy|choc(olate|oholic)?/gi;
   var myString = "Mmm, I love chocolate, I'm a chocoholic. " +
      "I love candy too, sweet, sugar candy";
   myString = myString.replace(myRegExp,"salad");
   alert(myString)
</script>
</body>
</html>
```

保存为 ch09_q3.htm。

本例假定为营养保健站点的留言板编写脚本。在这个留言板中，与糖果相关的文本是禁止出现的，这样的文本要用更健康的选项替换，如色拉。

禁止出现的单词如下所示：

- chocolate
- choc
- chocoholic
- sugar candy
- candy

下面将讨论删除指定单词的正则表达式：

(1) 首先是两个基本的单词，要匹配 choc 或者 candy，可以使用：

```
candy|choc
```

(2) 添加对 sugar candy 的匹配。由于 sugar 是可选的，所以将它放在圆括号中，并在其后添加?，这表示该组匹配 0 次或 1 次”。

```
(sugar )?candy|choc
```

(3) 需要添加可选的 olate 或者 oholic 作为结尾。把它们添加为一个分组，放在 choc 的后面，并使该组可选。使用字符 | 可以匹配组中的任一个结尾子串。

```
(sugar )?candy|choc(olate|oholic)?/gi
```

(4) 正则表达式声明为：

```
var myRegExp = /(sugar )?candy|choc(olate|oholic)?/gi
```

末尾的 gi 表示，该正则表达式将按不区分大小写的全局方式查找和替换单词。

因此，这个正则表达式：

```
/(sugar )?candy|choc(olate|oholic)?/gi
```

表示：

要么匹配 candy 或者 sugar candy，要么匹配 choc、chocolate 或者 chocoholic。

最后，下面的代码：

```
myString = myString.replace(myRegExp,"salad");
```

将禁止出现的单词替换为 salad，把 myString 设置为干净的新版本：

```
"Mmm, I love salad, I'm a salad. I love salad too, sweet, salad."
```

第 10 章

习题 1

创建一个顶部带广告图片的网页。当页面加载时，给广告随机选择一幅图片。每隔 4

秒将广告更换为另一幅图片，并确保选择不同的图片，直到显示了所有广告图片为止。

参考答案

下面的答案显示了3张图片，但通过一个计时器切换它们。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript">

var imagesSelected = new Array(false,false,false);
var noOfImages = 3;
var totalImagesSelected = 0;

function window_onload()
{
   setInterval("switchImage()",4000);
}

function switchImage()
{
   var imageIndex;

   if (totalImagesSelected == noOfImages)
   {
      for (imageIndex = 0; imageIndex < noOfImages; imageIndex++)
      {
         imagesSelected[imageIndex] = false;
      }

      totalImagesSelected = 0;
   }

var selectedImage = Math.floor(Math.random() * noOfImages) + 1;
while (imagesSelected[selectedImage - 1] == true)
{
      selectedImage = Math.floor(Math.random() * noOfImages) + 1;
   }
   totalImagesSelected++;
   imagesSelected[selectedImage - 1] = true;
   document.imgAdvert.src = "AdvertImage" + selectedImage + ".jpg";

}

</script>
</head>
<body onload="window_onload()">
<img src="AdvertImage1.jpg" name="imgAdvert" />
</body>
</html>
```

保存为 ch10_q1.htm。

上面的参考答案基于本章的示例 Adverts.htm，该示例按设定的时间间隔依次显示 3 幅图片。参考答案与该示例的第一个不同点在于，每次随机选择一幅图片，而不是按顺序显示。第二个不同点在于，为了确保同一幅图片在一轮图片显示中不选中两次，使用了 imagesSelected 数组，根据图片以前是否被选中，把该数组的每个元素设置为 true 或 false。显示了所有图片后，就重置该数组，并开始下一轮的随机图片选择。

参考答案与本章示例的第三个不同点在于，使用 setInterval()设置计时器。这样，图片的随机显示过程将一直进行，直到用户进入另一个页面为止。

习题 2

创建一个表单，以获取用户的生日信息。然后根据用户的生日，计算出他生日那天是星期几。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script type="text/javascript">

var days = new Array();
days[0] = "Sunday";
days[1] = "Monday";
days[2] = "Tuesday";
days[3] = "Wednesday";
days[4] = "Thursday";
days[5] = "Friday";
days[6] = "Saturday";

function dayOfWeek()
{

   var form = document.form1;
   var date = parseInt(form.txtDate.value)
   var year = parseInt(form.txtYear.value)

   if (isNaN(date) || isNaN(year))
   {
      alert("Please enter a valid whole number");
   }
   else
   {
      if (date < 1 || date > 31)
      {
         alert("Day of the month must be between 1 and 31");
      }
```

```
        else
        {
            userDate = date + "";
            userDate = userDate +
              form.selMonth.options[form.selMonth.selectedIndex].value;
            userDate = userDate + "" + year;
            var dateThen = new Date(userDate);
            alert(days[dateThen.getDay()]);
        }
    }
}
</script>
</head>
<body>
<p>Find the day of your birth</p>
<p>
<form name="form1">
<input type="text" name="txtDate" size="2" maxlength="2">
<select name=selMonth>
   <option selected value="Jan">Jan</option>
   <option selected value="Feb">Feb</option>
   <option selected value="Mar">Mar</option>
   <option selected value="Apr">Apr</option>
   <option selected value="May">May</option>
   <option selected value="Jun">Jun</option>
   <option selected value="Jul">Jul</option>
   <option selected value="Aug">Aug</option>
   <option selected value="Sept">Sept</option>
   <option selected value="Oct">Oct</option>
   <option selected value="Nov">Nov</option>
   <option selected value="Dec">Dec</option>
</select>
<input type="text" name="txtYear" size="4" maxlength="4" />
<br>
<input type="button" value="Day of the week"
      onclick="dayOfWeek()" name="button1" />
</form>
</p>

</body>
</html>
```

保存为 ch10_q2.htm。

上面的参考答案是非常简单的。只需根据用户输入的日期，创建一个新的 Date 对象，然后使用 Date 对象的 getDay()方法，得到他生日那天是星期几。该方法返回一个数字，所以代码定义了一个匹配该数字的 days 数组，并将 getDay()的值作为 days 数组的下标。

代码还做了一些基本的检查，以确保用户输入了数字，并且输入的日期在 1～31 之间。也可以定义一个 select 元素，其中仅包含 1～31 之间的数字，作为获取日期的方法。当然，这两种方法都没有检查输入的日期是否无效(例如，用户输入了 2 月 31 日)。可以将这个

有效性检查作为附加练习。

提示：

要获得某个月份的最后一天，只需获取下一个月的第一天，再减去 1。

第 11 章

习题 1

创建一个页面，来跟踪用户上个月访问该页面的次数。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<script language="JavaScript"
type="text/JavaScript"
src="CookieFunctions.js"></script>
<script type="text/javascript">
var pageViewCount = getCookieValue("pageViewCount");
var pageFirstVisited = getCookieValue("pageFirstVisited");

if (pageViewCount == null)
{
   pageViewCount = 1;
   pageFirstVisited = new Date();
   pageFirstVisited.setMonth(pageFirstVisited.getMonth());
   pageFirstVisited = pageFirstVisited.toGMTString();
   setCookie("pageFirstVisited",pageFirstVisited, " "," ")
}
else
{
   pageViewCount = Math.floor(pageViewCount) + 1;
}

setCookie("pageViewCount",pageViewCount, " "," ")

</script>
</head>
<body>
<script>
var pageHTML = "You've visited this page " + pageViewCount;
pageHTML = pageHTML + " times since " + pageFirstVisited;
document.write(pageHTML);
</script>
</body>
</html>
```

保存为 ch11_q1.htm。

第 11 章讨论了 Cookie 的功能，下面将直接介绍新代码。

前两行获取了两个 Cookie，并保存在变量中。第一个 cookie 保存了访问次数，第二个 cookie 保存第一次访问页面的日期。

```
var pageViewCount = getCookieValue("pageViewCount");
var pageFirstVisited = getCookieValue("pageFirstVisited");
```

如果不存在名为 pageViewCount 的 cookie，则要么是由于该 cookie 已过期(注意计算的是上个月的访问次数)，要么是由于用户从未访问过网站。不论是哪种情况，都需要将 pageViewCount 设置为 1，并将第一次访问页面的日期加上一个月，并存储在 pageFirstVisited 中。以后在创建名为 pageViewCount 的 cookie 时，将使用这个值设置作为 expires 值，因为无法使用代码确定已有 cookie 的过期日期。

```
if (pageViewCount == null)
{
   pageViewCount = 1;
   pageFirstVisited = new Date();
   pageFirstVisited.setMonth(pageFirstVisited.getMonth() + 1)
   pageFirstVisited = pageFirstVisited.toGMTString();
   setCookie("pageFirstVisited",pageFirstVisited, " ", " ")
}
```

在 else 语句中增加 pageViewCount 的值。

```
else
{
  pageViewCount = Math.floor(pageViewCount) + 1;
}
```

接着将 cookie 设置为跟踪用户访问页面的次数。

```
setCookie("pageViewCount",pageViewCount, " "," ")
```

最后在页面上输出自重置计时器以来页面的访问次数和日期。

```
var pageHTML = "You've visited this page " + pageViewCount;
pageHTML = pageHTML + " times since " + pageFirstVisited;
document.write(pageHTML);
```

习题 2

使用 Cookie，在用户每次访问页面时加载不同的广告。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
```

```
<script language="JavaScript" type="text/JavaScript" src="CookieFunctions.js"></
script>
</head>
<body>
<img src="AdvertImage1.jpg" name="imgAdvert">
<script type="text/javascript">

var imageNumber = getCookieValue("displayedImages");
var totalImages = 3;

if (imageNumber == null)
{
    imageNumber = "1";
}

else
{
    imageNumber = Math.floor(imageNumber) + 1;
}

if (totalImages == imageNumber)
{
    setCookie("displayedImages"," ", " ","Mon, 1 Jan 1970 00:00:00");
}
else
{
    setCookie("displayedImages",imageNumber, " "," ");
}

document.imgAdvert.src = "AdvertImage" + imageNumber + ".jpg";
</script>
</body>
</html>
```

保存为 ch11_q2.htm。

上面的参考答案基于前面章节的类似习题。例如第 10 章的一道习题显示随机选取的图片。本题在每次用户访问页面时显示一幅不同的图片，只要还允许选择图片。

本章讨论了 Cookie 设置和获取函数，因此下面只讨论新代码。

在名为 displayedImages 的 Cookie 中保存前一幅显示的图片编号。下一幅要显示的图片编号是该编号加 1。如果所有的图片都已显示完毕，则再次从 1 开始。如果用户从未访问过网站，就不存在 Cookie，getCookieValue()返回 null，此时将 imageNumber 设置为 1。

大部分代码简单明了，除了下面这几行代码：

```
if (totalImages == imageNumber)
{
  setCookie("displayedImages"," "," ","Mon, 1 Jan 1970 00:00:00")
}
```

其作用是，通过将 cookie 的过期时间设置为一个过去的时间，来删除该 Cookie。

第 12 章

习题 1

下面的 HTML 代码创建了一个表格。请使用 JavaScript 和核心 DOM 对象重建该表格，并生成 HTML。在所有可用的浏览器中测试代码，确保代码正常工作。提示：为所编写的每行代码添加注释，以说明当前元素在树结构中的位置。再为页面中的每个元素创建一个变量(例如，为 9 个 TD 单元格创建 9 个变量，而不是只创建一个变量)。

```
<table>
    <thead>
        <tr>
            <td>Car</td>
            <td>Top Speed</td>
            <td>Price</td>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Chevrolet</td>
            <td>120mph</td>
            <td>$10,000</td>
    </tr>
    <tr>
            <td>Pontiac</td>
            <td>140mph</td>
            <td>$20,000</td>
        </tr>
    </tbody>
</table>
```

参考答案

这似乎是一个很难处理的例子，但其实并不难，只是综合了两个方面，一个是建立树形结构，另一个是导航该树形结构。首先导航到<body/>元素，创建一个<table/>元素。再导航到新建的<table/>元素上，创建一个新的<thead/>元素，依此类推。这是一个很长的重复过程，所以最好给代码添加注释，以确定当前的位置。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Question 1 Answer</title>
</head>
<body>
<script type="text/javascript">
var tableElem = document.createElement("table")
var thElem = document.createElement("thead")
```

```
var trElem1 = document.createElement("tr")
var trElem2 = document.createElement("tr")
var trElem3 = document.createElement("tr")
var tdElem1 = document.createElement("td")
var tdElem2 = document.createElement("td")
var tdElem3 = document.createElement("td")
var tdElem4 = document.createElement("td")
var tdElem5 = document.createElement("td")
var tdElem6 = document.createElement("td")
var tdElem7 = document.createElement("td")
var tdElem8 = document.createElement("td")
var tdElem9 = document.createElement("td")
var tbodyElem = document.createElement("tbody")
var textNodeA1 = document.createTextNode("Car")
var textNodeA2 = document.createTextNode("Top Speed")
var textNodeA3 = document.createTextNode("Price")
var textNodeB1 = document.createTextNode("Chevrolet")
var textNodeB2 = document.createTextNode("120mph")
var textNodeB3 = document.createTextNode("$10,000")
var textNodeC1 = document.createTextNode("Pontiac")
var textNodeC2 = document.createTextNode("140mph")
var textNodeC3 = document.createTextNode("$14,000")

docNavigate = document.documentElement; //Starts with HTML document
docNavigate = docNavigate.firstChild.nextSibling; //Moves to the head
                                                  // then body element
docNavigate.appendChild(tableElem);      //Adds the table element
docNavigate = docNavigate.lastChild;     //Moves to the table element
docNavigate.appendChild(thElem);         //Adds the thead element
docNavigate = docNavigate.firstChild;    //Moves to the thead element
docNavigate.appendChild(trElem1);        //Adds the TR element
docNavigate = docNavigate.firstChild;    //Moves the TR element
docNavigate.appendChild(tdElem1);        //Adds the first TD element in the
                                         // heading
docNavigate.appendChild(tdElem2);        //Adds the second TD element in the
                                         // heading
docNavigate.appendChild(tdElem3);        //Adds the third TD element in the
                                         // heading
docNavigate = docNavigate.firstChild;    //Moves to the first TD element
docNavigate.appendChild(textNodeA1);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeA2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeA3);     //Adds the third text node

docNavigate = docNavigate.parentNode;    //Moves back to the TR element
docNavigate = docNavigate.parentNode;    //Moves back to the thead element
docNavigate = docNavigate.parentNode;    //Moves back to the table element
docNavigate.appendChild(tbodyElem);      //Adds the tbody element
docNavigate = docNavigate.lastChild;     //Moves to the tbody element
docNavigate.appendChild(trElem2);        //Adds the second TR element
```

```
docNavigate = docNavigate.lastChild;     //Moves to the second TR element
docNavigate.appendChild(tdElem4);        //Adds the TD element
docNavigate.appendChild(tdElem5);        //Adds the TD element
docNavigate.appendChild(tdElem6);        //Adds the TD element
docNavigate = docNavigate.firstChild;    //Moves to the first TD element
docNavigate.appendChild(textNodeB1);     //Adds the first text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeB2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeB3);     //Adds the third text node
docNavigate = docNavigate.parentNode;    //Moves back to the TR element
docNavigate = docNavigate.parentNode;    //Moves back to the tbody element
docNavigate.appendChild(trElem3);        //Adds the TR element
docNavigate = docNavigate.lastChild;     //Moves to the TR element
docNavigate.appendChild(tdElem7);        //Adds the TD element
docNavigate.appendChild(tdElem8);        //Adds the TD element
docNavigate.appendChild(tdElem9);        //Adds the TD element
docNavigate = docNavigate.firstChild;    //Moves to the TD element
docNavigate.appendChild(textNodeC1);     //Adds the first text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeC2);     //Adds the second text node
docNavigate = docNavigate.nextSibling;   //Moves to the next TD element
docNavigate.appendChild(textNodeC3);     //Adds the third text node
</script>
</body>
</html>
```

习题 2

前面提到，示例 10 是选项卡的一个不完整的 DHTML 脚本，通过进行如下修改对其进行一定程度的完善：

- 一次只能有一个选项卡是激活的。
- 只有激活的选项卡才显示其描述信息。

参考答案

示例 10 不完整，是因为脚本没有跟踪激活的选项卡。在脚本中添加状态标识的最简单方法是添加一个全局变量，来跟踪上次单击的选项卡编号。这个参考答案就采用了这种方法。修改突出显示的代码行：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 12: Question 2 Answer</title>
    <style type="text/css">
    .tabStrip
    {
        background-color: #E4E2D5;
```

```
        padding: 3px;
        height: 22px;
    }

    .tabStrip div
    {
        float: left;
        font: 14px arial;
        cursor: pointer;
    }

    .tabStrip-tab
    {
        padding: 3px;
    }

    .tabStrip-tab-hover
    {
        border: 1px solid #316AC5;
        background-color: #C1D2EE;
        padding: 2px;
    }

    .tabStrip-tab-click
    {
        border: 1px solid #facc5a;
        background-color: #f9e391;
        padding: 2px;
    }
    </style>
    <script type="text/javascript">
    var currentNum = 0;
    function handleEvent(e)
    {
        var eSrc;

        if (window.event)
        {
            e = window.event;
            eSrc = e.srcElement;
        }
        else
        {
            eSrc = e.target;
        }

        if (e.type == "mouseover")
        {
            if (eSrc.className == "tabStrip-tab")
            {
                eSrc.className = "tabStrip-tab-hover";
            }
```

```
        }

        if (e.type == "mouseout")
        {
            if (eSrc.className == "tabStrip-tab-hover")
            {
                eSrc.className = "tabStrip-tab";
            }
        }

        if (e.type == "click")
        {
            if (eSrc.className == "tabStrip-tab-hover")
            {
                var num = eSrc.id.substr(eSrc.id.lastIndexOf("-") + 1);

                if (currentNum != num)
                {
                    deactivateTab();

                    eSrc.className = "tabStrip-tab-click";
                    showDescription(num);
                    currentNum = num;
                }
            }
        }
    }
    function showDescription(num)
    {
        var descContainer = document.getElementById("descContainer");

        var div = document.createElement("div");
        div.id = "tabStrip-desc-" + num;
        var text = document.createTextNode("Description for tab " + num);

        div.appendChild(text);
        descContainer.appendChild(div);
    }

    function deactiveTab()
    {
        var descContainer = document.getElementById("descContainer");
        var descEl = document.getElementById("tabStrip-desc-" + currentNum);

        if (descEl)
        {
            descContainer.removeChild(descEl);

            document.getElementById("tabStrip-tab-"
                + currentNum).className = "tabStrip-tab";
        }
    }

    document.onclick = handleEvent;
```

```
        document.onmouseover = handleEvent;
        document.onmouseout = handleEvent;
        </script>
    </head>
    <body>
        <div class="tabStrip">
            <div id="tabStrip-tab-1" class="tabStrip-tab">Tab 1</div>
            <div id="tabStrip-tab-2" class="tabStrip-tab">Tab 2</div>
            <div id="tabStrip-tab-3" class="tabStrip-tab">Tab 3</div>
        </div>
        <div id="descContainer"></div>
    </body>
    </html>
```

下面仅逐行讨论新代码。首先分析 showDescription()函数中的新代码行，如下所示：

```
div.id = "tabStrip-desc-" + num;
```

这行代码只是给包含选项卡描述的 元素添加一个 id。这样，在需要单击另一个选项卡，并修改其描述信息时，就很方便地找到这个元素。

下面是脚本的第一行新代码，它添加了一个全局变量 currentNum，并赋值为 0：

```
var currentNum = 0;
```

单击选项卡时，处理 click 事件的代码会检查被单击的选项卡编号是否不同于 currentNum 包含的值。如果不同，就表示被单击的选项卡不是当前激活的选项卡，所以可以禁用当前的选项卡，把新选项卡设置为激活的选项卡。

```
var num = eSrc.id.substr(eSrc.id.lastIndexOf("-") + 1);

if (currentNum != num)
{
    deactivateTab();

    eSrc.className = "tabStrip-tab-click";
    showDescription(num);
    currentNum = num;
}
```

If 语句的第一行调用 deactivateTab()函数，以便禁用当前选项卡(如后面所示)。把选项卡的 className 属性改为新单击的选项卡名称，并在页面中添加描述信息，currentNum 变量改为包含新激活的选项卡编号。

现在分析 deactivateTab()函数。其作用是从页面中删除当前激活的选项卡的描述信息，把激活选项卡的样式改为正常。

```
function deactiveTab()
{
    var descContainer = document.getElementById("descContainer");
    var descEl = document.getElementById("tabStrip-desc-" + currentNum);
```

这个函数的第一行获取 id 为 descContainer 的 元素，第二行使用 showDescription() 函数赋予它的 id 值，尝试定位描述元素。

这里的“尝试”是关键，注意 currentNum 初始化为 0，而没有编号为 0 的选项卡。但第一次单击任意选项卡时，就会调用 deactivateTab()，因为 0 不等于被单击的选项卡号码。所以第一次单击选项卡时，代码尝试查找 id 为 tabStrip-desc-0 的元素，显然该元素不存在，getElementById()如果找不到使用指定 id 的元素，就返回 null。这是一个潜在的问题，因为如果尝试对值为 null 的 descEl 变量执行操作，浏览器就会抛出一个错误。

其解决方法非常简单，检查 descEl 变量是否有值，仅在该变量有值时执行操作。

```
    if (descEl)
    {
        descContainer.removeChild(descEl);

        document.getElementById("tabStrip-tab-"
            + currentNum).className = "tabStrip-tab";
    }
}
```

descEl 包含值时，就从 descContainer 元素中删除它，把激活选项卡的 className 属性改为 tabStrip-tab，把其样式改为正常状态。

第 13 章

习题 1

使用 Quicktime 插件或 ActiveX 控件，创建一个包含三个链接的页面。当用户单击任一个链接时，就播放声音。使用警告框告诉没有安装 QuickTime 的用户，单击链接时，必须安装 Quicktime。

在 IE、Firefox、Safari、Chrome 和 Opera 中，该页面都可以正常运行。告诉 QuickTime 播放哪个文件的方法是 SetURL()。

参考答案

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Chapter 13: Question Answer</title>
    <script type="text/javascript">
    var plugInInstalled = false;
    function play(fileName)
    {
        if (plugInInstalled)
        {
            document.audioPlayer.SetURL(fileName);
            document.audioPlayer.Play();
        }
```

```
        else
        {
            alert("You must have QuickTime installed to play this file.");
        }

        return false;
    }

    function window_onload()
    {
        if (!window.ActiveXObject)
        {
            var pluginsLength = navigator.plugins.length;
            for (var i = 0; i < pluginsLength; i++)
            {
                var pluginName = navigator.plugins[i].name.toLowerCase();
                if (pluginName.indexOf("quicktime") > -1)
                {
                    plugInInstalled = true;
                    break;
                }
            }
        }
        else
        {
            if (document.audioPlayer.readyState == 4)
            {
                plugInInstalled = true;
            }
        }

        if (!plugInInstalled)
        {
            alert("You need Quicktime to play the audio files!");
        }
    }

    onload = window_onload;
    </script>
</head>
<body>
    <object id="audioPlayer"
    classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
    codebase="http://www.apple.com/qtactivex/qtplugin.cab"
    width="320" height="260">
    <param name="src" value="sound1.mp3" />
    <param name="controller" value="false" />
    <param name="autoplay" value="false" />
    <embed name="audioPlayer"
        height="260"
        width="320"
        src="sound1.mp3"
```

```
        type="video/quicktime"
        pluginspage="www.apple.com/quicktime/download"
        controller="false"
        autoplay="false"
    />
    </object>
    <a href="#" onclick="return play('sound1.mp3')">Sound 1</a>
    <a href="#" onclick="return play('sound2.mp3')">Sound 2</a>
    <a href="#" onclick="return play('sound3.mp3')">Sound 3</a>
</body>
</html>
```

保存为 ch13_q1.htm。

上面的参考答案基于本章的 QuickTime 示例。注意，三个声音文件 sound1.mp3、sound2.mp3 和 sound3.mp3 可从本书的代码下载中获得。

第一行 JavaScript 代码声明了 plugInInstalled 变量，这与原示例在 window_onload()函数中声明的变量相同。在 window 的 onload 事件处理程序中调用 window_onload()函数，验证用户是否能使用 QuickTime 插件。

IE 和其他浏览器对插件的支持是不同的。因此，检查插件的方式也是不同的。对于 Firefox、Safari、Chrome 和 Opera，需要遍历 navigator 对象的 plugins 集合，在每个已安装的插件中查找插件 quickTime。如果找到，就表示用户安装了 QuickTime 播放器。

对于 IE，只需用 ActiveX 控件的 readyState 属性，检查 QuickTime 是否正确安装和初始化。

为了播放声音，定义了play()函数，该方法的参数是要播放的声音文件的文件名。这个函数检查是否安装了 QuickTime 插件或 ActiveX 控件。

```
function play(fileName)
{
    if (plugInInstalled)
    {
        document.audioPlayer.SetURL(fileName);
        document.audioPlayer.Play();
    }
    else
    {
        alert("You must have QuickTime installed to play this file.");
    }

    return false;
}
```

如果安装了 QuickTime，函数就使用 QuickTime 播放器的 SetURL()方法设置要播放的声音文件，使用 Play()方法开始播放声音剪辑。如果未安装 QuickTime，就用警告框告诉用户，只有安装 QuickTime 才能听到声音。

每次把不同的文件名指定为 play()函数的参数，就可以为每个链接使用不同的声音。单击链接时，onclick 事件处理程序就开始播放声音。

```
<a href="#" onclick="return play('sound1.mp3')">Sound 1</a>
```

第 14 章

习题 1

扩展 HttpRequest 模块，以便在已有异步请求的基础上增加同步请求。必须对代码进行一些调整，以便添加这个功能(提示：给模块创建 async 属性)。

参考答案

```
function HttpRequest(sUrl, fpCallback)
{
    this.url = sUrl;
    this.callBack = fpCallback;
    this.async = true;
    this.request = this.createXmlHttpRequest();
}

HttpRequest.prototype.createXmlHttpRequest = function () {
    if (window.XMLHttpRequest)
    {
        var oHttp = new XMLHttpRequest();
        return oHttp;
    }
    else if (window.ActiveXObject)
    {
        var versions =
        [
            "MSXML2.XmlHttp.6.0",
            "MSXML2.XmlHttp.3.0"
        ];

        for (var i = 0; i < versions.length; i++)
        {
            try
            {
                oHttp = new ActiveXObject(versions[i]);
                return oHttp;
            }
            catch (error)
            {
                //do nothing here
            }
        }
    }
    return null;
}

HttpRequest.prototype.send = function()
{
    this.request.open("GET", this.url, this.async);
```

```
    if (this.async)
    {
        var tempRequest = this.request;
        var fpCallback = this.callBack;

        function request_readystatechange()
        {
            if (tempRequest.readyState == 4)
            {
                if (tempRequest.status == 200)
                {
                    fpCallback(tempRequest.responseText);
                }
                else
                {
                    alert("An error occurred while attempting to " +
                          "contact the server.");
                }
            }
        }

        this.request.onreadystatechange = request_readystatechange;
    }

    this.request.send(null);

    if (!this.async)
    {
        this.callBack(this.request.responseText);
    }
}
```

可以采用多种办法给 HttpRequest 模块添加同步通信。本例采用的方式修改代码，添加一个新属性 async。async 属性包含 true 或者 false。如果 async 属性为 true，则底层的 XMLHttpRequest 对象使用异步通信方式获取文件。如果 async 属性为 false，该模块就使用同步通信方式。简而言之，这个属性类似于 XML DOM 的 async 属性，以决定 XML 文档的加载方式。

模块的第一个变化是构造函数本身。原构造函数初始化一个 XMLHttpRequest 对象，来发送数据。但是新构造函数仅初始化了所有属性。

```
function HttpRequest(sUrl, fpCallback)
{
    this.url = sUrl;
    this.callBack = fpCallback;
    this.async = true;
    this.request = this.createXmlHttpRequest();
}
```

其中有三个新属性。第一个属性 url 表示 XMLHttpRequest 对象尝试从服务器请求的 URL。callBack 属性包含对回调函数的引用。async 属性决定 XMLHttpRequest 对象采用的

通信类型。在构造函数中，将 async 设置为 true，表示该属性的默认值为 true，因此可以采用异步模式发送请求，无需在外部设置这个属性。

新的构造函数和属性是真正需要的，因为它们允许为多个请求重用相同的 HttpRequest 对象。如果要向另一个 URL 发出请求，只须给 url 属性设置新值。回调函数也是如此。

createXmlHttpRequest()方法保持不变。这是一个辅助方法，与发送请求无关。

模块的主要改变在于 send()方法。模块利用这个方法决定是采用同步通信还是异步通信方式。在发送请求时，这两种通信方式几乎完全不同：异步通信使用 onreadystatechange 事件处理程序，而同步通信在请求完成时，才允许访问 XMLHttpRequest 对象的属性。因此，需要使用代码分支：

```
HttpRequest.prototype.send = function()
{
    this.request.open("GET", this.url, this.async);

    if (this.async)
    {
        //more code here
    }

    this.request.send(null);

    if (!this.async)
    {
        //more code here
    }
}
```

这个方法的第一行代码使用 XMLHttpRequest 对象的 open()方法，并将 async 属性作为该方法的最后一个参数，该参数决定 XMLHttpRequest 对象是否采用异步通信。接下来，if 语句检查 this.async 属性是否为 true，如果是，就在这个 if 块中放置异步代码。随后调用 XMLHttpRequest 对象的 send()方法，向服务器发送请求。最后一个 if 语句检查 this.async 是否为 false，如果是，就把同步代码放在要执行的代码块中。

```
HttpRequest.prototype.send = function()
{
    this.request.open("GET", this.url, this.async);

    if (this.async)
    {
        var tempRequest = this.request;
        var fpCallback = this.callBack;

        function request_readystatechange()
        {
            if (tempRequest.readyState == 4)
            {
                if (tempRequest.status == 200)
                {
                    fpCallback(tempRequest.responseText);
```

```
                }
                else
                {
                    alert("An error occurred while attempting to " +
                        "contact the server.");
                }
            }
        }

        this.request.onreadystatechange = request_readystatechange;
    }

    this.request.send(null);

    if (!this.async)
    {
        this.callBack(this.request.responseText);
    }
}
```

上面的新代码完成了 send()方法的定义。第一个 if 块给新变量 fpCallback 赋予 this.callBack 的值，其原因与使用 tempRequest 变量一样——解决作用域问题——因为 this 指向 request_readystatechange()函数，而不是 HttpRequest 对象。除此之外，其他异步代码保持不变。请求成功时，request_readystatechange()函数处理 readystatechange 事件，并调用回调函数。

第二个 if 块非常简单。仅当采用同步方式时，才执行这个代码块。该代码块只是调用回调函数，并传递 XMLHttpRequest 对象的 responseText 属性。

重构的模块使用起来非常简单。下面的代码异步请求一个假定的文本文件 test.txt。

```
function request_callback(sResponseText)
{
    alert(sResponseText);
}

var oHttp = new HttpRequest("test.txt", request_callback);

oHttp.send();
```

对于异步请求来说，没有什么实质性变化，代码与本章中的示例完全相同。如果要使用同步通信，只需将 async 设置为 false，例如下面的代码：

```
function request_callback(sResponseText)
{
    alert(sResponseText);
}

var oHttp = new HttpRequest("test.txt", request_callback);
oHttp.async = false;

oHttp.send();
```

现在，这个 Ajax 模块可以用同步和异步通信方式请求信息了。

习题 2

如本章前面所述，智能表单可以改为不使用超链接。修改使用 HttpRequest 模块的表单，使之在用户提交表单时检查用户名和 e-mail 字段。如果用户名和 e-mail 已被占用，就使用表单的 onsubmit 事件处理程序并且取消提交操作，再通过习题 1 中更新的 HttpRequest 模块，来使用同步请求。只有用户名和 e-mail 已被占用，才需要警告用户，所以如果用户名和 e-mail 符合要求，就应返回 true。

参考答案

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Form Field Validation</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
    <script type="text/javascript" src="HttpRequest.js"></script>
    <script type="text/javascript">
        var isUsernameTaken;
        var isEmailTaken;

        function checkUsername_callBack(sResponseText)
        {
            if (sResponseText == "available")
            {
               isUsernameTaken = false;
            }
            else
            {
               isUsernameTaken = true;
            }
        }

        function checkEmail_callBack(sResponseText)
        {
            if (sResponseText == "available")
            {
                isEmailTaken = false;
```

```
        }
        else
        {
            isEmailTaken = true;
        }
    }

    function form_submit()
    {
        var request = new HttpRequest();
        request.async = false;

        //First check the username
        var userValue = document.getElementById("username").value;

        if (userValue == "")
        {
            alert("Please enter a user name to check!");
            return false;
        }

        request.url = "formvalidator.php?username=" + userValue;
        request.callBack = checkUsername_callBack;
        request.send();

        if (isUsernameTaken)
        {
            alert("The username " + userValue + " is not available!");
            return false;
        }

        //Now check the email
        var emailValue = document.getElementById("email").value;

        if (emailValue == "")
        {
            alert("Please enter an email address to check!");
            return false;
        }

        request.url = "formvalidator.php?email=" + emailValue;
        request.callBack = checkEmail_callBack;
        request.send();

        if (isEmailTaken)
        {
            alert("I'm sorry, but " + emailValue + " is in use by " +
                "another user.");
            return false;
        }

        //If the code's made it this far, everything's good
        return true;
    }
```

```
        </script>
    </head>
    <body>
        <form onsubmit="return form_submit()">
            <table>
                <tr>
                    <td class="fieldname">
                        Username:
                    </td>
                    <td>
                        <input type="text" id="username" />
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Email:
                    </td>
                    <td>
                        <input type="text" id="email" />
                    </td>
                </tr>
                <tr>
                    <td class="fieldname">
                        Password:
                    </td>
                    <td>
                        <input type="text" id="password" />
                    </td>

                </tr>
                <tr>
                    <td class="fieldname">
                        Verify Password:
                    </td>
                    <td>
                        <input type="text" id="password2" />
                    </td>

                </tr>
                <tr>
                    <td colspan="2" class="submit">
                        <input type="submit" value="Submit" />
                    </td>
                </tr>
            </table>
        </form>
    </body>
    </html>
```

首先分析 HTML：删除了链接和表格的第三列。新 HTML 的主要区别在于开始<form>标记中的 onsubmit 事件处理程序。理想情况下，只有表单字段通过了验证，表单才能提交

数据。因此，onsubmit 事件处理程序设置为 return form_submit()。form_submit()函数返回 true 或 false，如果表单字段通过了验证，浏览器就可以提交表单的数据，否则就不能提交。

JavaScript 代码的变化最大。在这些新的实现代码中，声明了两个全局变量 isUsernameTaken 和 isEmailTaken。如果用户名或电子邮件被占用，这两个变量就是 true，否则就是 false。

```
var isUsernameTaken;
var isEmailTaken;

function checkUsername_callBack(sResponseText)
{
    if (sResponseText == "available")
    {
        isUsernameTaken = false;
    }
    else
    {
        isUsernameTaken = true;
    }
}

function checkEmail_callBack(sResponseText)
{
    if (sResponseText == "available")
    {
        isEmailTaken = false;
    }
    else
    {
        isEmailTaken = true;
    }
}
```

前两个函数 checkUsername_callBack()和 checkEmail_callBack()与最初的版本有些类似。但它们不给用户提供警告信息，而是给 isUsernameTaken 和 isEmailTaken 变量赋值。

完成大多数工作的函数是 form_submit()，它负责向服务器发送请求，并确定表单字段中的数据是否可以提交。

```
function form_submit()
{
    var request = new HttpRequest();
    request.async = false;

    //more code here
}
```

这段代码创建 HttpRequest 对象，并将它设置为同步通信。有时需要使用同步通信，例如表单验证。表单字段的验证是一个顺序处理过程，表单的提交取决于 onsubmit 事件处理程序的结果。使用同步通信会在试图验证表单字段之前，强制函数等待从服务器中接收

的信息。如果使用异步通信，则 form_submit()会在验证用户名和 email 之前执行，并返回一个值。另外请注意，HttpRequest()构造函数没有接收任何参数。这是因为可以使用新版本显式地设置 url 和 callBack 属性。

首先检查 Username 字段：

```
function form_submit()
{
    var request = new HttpRequest();
    request.async = false;

    //First check the username
    var userValue = document.getElementById("username").value;

    if (userValue == "")
    {
        alert("Please enter a user name to check!");
        return false;
    }

    request.url = "formvalidator.php?username=" + userValue;
    request.callBack = checkUsername_callBack;
    request.send();

    if (isUsernameTaken)
    {
        alert("The username " + userValue + " is not available!");
        return false;
    }

    //more code here
}
```

这段代码获取 Username 字段的值，并检查是否在其中输入了信息。如果用户没有输入信息，则弹出一条消息，要求用户输入数据。如果用户在 Username 字段中输入了信息，则代码继续执行。接着给 url 和 callBack 属性赋值，然后把请求发送给服务器。如果所申请的用户名被占用，就将弹出一个警告框通知用户。否则，代码继续执行，检查 e-mail 信息。

```
function form_submit()
{
    var request = new HttpRequest();
    request.async = false;

    //First check the username
    var userValue = document.getElementById("username").value;

    if (userValue == "")
    {
        alert("Please enter a user name to check!");
        return false;
    }
```

```
    request.url = "formvalidator.php?username=" + userValue;
    request.callBack = checkUsername_callBack;
    request.send();

    if (isUsernameTaken)
    {
        alert("The username " + userValue + " is not available!");
        return false;
    }

    //Now check the email
    var emailValue = document.getElementById("email").value;

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return false;
    }

    request.url = "formvalidator.php?email=" + emailValue;
    request.callBack = checkEmail_callBack;
    request.send();

    if (isEmailTaken)
    {
        alert("I'm sorry, but " + emailValue + " is in use by another user.");
        return false;
    }

    //If the code's made it this far, everything's good
    return true;
}
```

检查 e-mail 的处理过程与检查用户名相同。首先获取 Email 字段的值，并检查用户是否在该文本框中输入了值。然后使用该值向服务器发送另一个请求。同样，url 属性和 callBack 属性是显式设置的。如果 isEmailTaken 为 true，则弹出一个警告框通知用户，电子邮件地址已被另一个用户占用，函数返回 false。如果该地址可用，则函数返回 true，允许浏览器提交表单。

第 15 章

习题 1

使用 jQuery 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

参考答案

关键之处是同步通信。没有它，表单验证就会失败。确保设置 async 属性。下面代码中突出显示的部分是使用 jQuery 进行的修改。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
```

```
        "http://www.w3.org/TR/html4/strict.dtd">

    <html>
    <head>
        <title>Chapter 15: Question 1 Answer with jQuery</title>
        <style type="text/css">
            .fieldname
            {
                text-align: right;
            }

            .submit
            {
                text-align: right;
            }
        </style>
        <script type="text/javascript" src="jquery-1.3.2.min.js"></script>
        <script type="text/javascript">
            var isUsernameTaken;
            var isEmailTaken;

            function checkUsername_callBack(data, status)
            {
                if (data == "available")
                {
                    isUsernameTaken = false;
                }
                else
                {
                    isUsernameTaken = true;
                }
            }

            function checkEmail_callBack(data, status)
            {
                if (data == "available")
                {
                    isEmailTaken = false;
                }
                else
                {
                    isEmailTaken = true;
                }
            }

            function form_submit()
            {
                //First check the username
                var userValue = $("#username").val();
                if (userValue == "")
                {
                    alert("Please enter a user name to check!");
```

```
                return false;
            }

            var userParms = new Object();
            userParms.username = userValue;

            $.get("formvalidator.php", userParms, checkUsername_callBack);

            if (isUsernameTaken)
            {
                alert("The username " + userValue + " is not available! ");
                return false;
            }

            //Now check the email
            var emailValue = $("#email").val();

            if (emailValue == "")
            {
                alert("Please enter an email address to check!");
                return false;
            }

            var emailParms = new Object();
            emailParms.email = emailValue;

            $.get("formvalidator.php", emailParms, checkEmail_callBack);

            if (isEmailTaken)
            {
                alert("I'm sorry, but " + emailValue +
                " is in use by another user. ");
                return false;
            }

            //If the code's made it this far, everything's good
            return true;
        }

        var ajaxOptions = new Object();
        ajaxOptions.async = false;

        $.ajaxSetup(ajaxOptions);

        function request_error(event, request, settings)
        {
            alert("An error occurred with the following URL:\n"
             + settings.url + ".\nStatus code: " + request.status);
        }

            $(document).ajaxError(request_error);
    </script>
</head>
<body>
    <form onsubmit="return form_submit()">
```

```
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
            </tr>
            <tr>
                <td class="fieldnameV>
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>

            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submit" value="Submit" />
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

习题 2

使用 Prototype 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

参考答案

与习题 1 一样，同步通信是保证脚本正确工作的关键，所以在请求选项中添加它。下

面代码中突出显示的部分是使用 Prototype 进行的修改。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Chapter 15: Question 2 Answer with Prototype</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }

        .submit
        {
            text-align: right;
        }
    </style>
    <script type="text/javascript" src="prototype-1.6.0.3.js"></script>
    <script type="text/javascript">
        var isUsernameTaken;
        var isEmailTaken;

        function checkUsername_callBack(request)
        {
            if (request.responseText == "available")
            {
                isUsernameTaken = false;
            }
            else
            {
                isUsernameTaken = true;
            }
        }

        function checkEmail_callBack(request)
        {
            if (request.responseText == "available")
            {
                isEmailTaken = false;
            }
            else
            {
                isEmailTaken = true;
            }
        }

        function form_submit()
        {
            //First check the username
            var userValue = $F("username");
```

```
    if (userValue == "")
    {
        alert("Please enter a user name to check!");
        return false;
    }

    var userParms = new Object();
    userParms.username = userValue;

    var options = getBasicOptions();
    options.onSuccess = checkUsername_callBack;
    options.parameters = userParms;

    new Ajax.Request("formvalidator.php", options);

    if (isUsernameTaken)
    {
        alert("The username " + userValue + " is not available!");
        return false;
    }

    //Now check the email
    var emailValue = $F("email");

    if (emailValue == "")
    {
        alert("Please enter an email address to check!");
        return false;
    }

    var emailParms = new Object();
    emailParms.email = emailValue;

    options.onSuccess = checkEmail_callBack;
    options.parameters = emailParms;

    new Ajax.Request("formvalidator.php", options);

    if (isEmailTaken)
    {
        alert("I'm sorry, but " + emailValue +
            " is in use by another user.");
        return false;
    }

    //If the code's made it this far, everything's good
        return true;
    }

    function request_onfailure(request)
    {
        alert("An error occurred. HTTP Status Code: " + request.status);
    }

    function getBasicOptions()
```

```
        {
            var options = new Object();
            options.method = "get";
            options.onFailure = request_onfailure;
            options.asynchronous = false;

            return options;
        }
    </script>
</head>
<body>
    <form onsubmit="return form_submit()">
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>

            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>

            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submit" value="Submit" />
                </td>
```

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

习题 3

使用 MooTools 完成第 14 章的第 2 题，并在 Ajax 请求出错时添加错误报告。

参考答案

关键还是同步通信。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
    <title>Chapter 15: Question 3 Answer with MooTools</title>
    <style type="text/css">
        .fieldname
        {
            text-align: right;
        }
        .submit
        {
            text-align: right;
        }
    </style>
    <script src="mootools-1.2.3-core-yc.js" type="text/javascript"></script>
    <script type="text/javascript">
        var isUsernameTaken;
        var isEmailTaken;

        function checkUsername_callBack(text, xml)
        {
            if (text == "available")
            {
                isUsernameTaken = false;
            }
            else
            {
                isUsernameTaken = true;
            }
        }

        function checkEmail_callBack(text, xml)
        {
            if (text == "available")
            {
                isEmailTaken = false;
```

```
        }
        else
        {
            isEmailTaken = true;
        }
    }

    function form_submit()
    {
        //First check the username
        var userValue = $("username").value;

        if (userValue == "")
        {
            alert("Please enter a user name to check!");
            return false;
        }

        var options = getBasicOptions();
        options.onSuccess = checkUsername_callBack;

        new Request(options).send("username=" + userValue);

        if (isUsernameTaken)
        {
            alert("The username " + userValue + " is not available! ");
            return false;
        }

        //Now check the email
        var emailValue = $("email").value;

        if (emailValue == "")
        {
            alert("Please enter an email address to check! ");
            return false;
        }

        options.onSuccess = checkEmail_callBack;

        new Request(options).send("email=" + emailValue);

        if (isEmailTaken)
        {
            alert("I'm sorry, but " + emailValue +
                " is in use by another user. ");
            return false;
        }

        //If the code's made it this far, everything's good
        return true;
    }

    function request_onfailure(request)
```

```
        {
            alert("An error occurred. HTTP Status Code: " + request.status);
        }

        function getBasicOptions()
        {
            var options = new Object();
            options.method = "get";
            options.onFailure = request_onfailure;
            options.url = "formvalidator.php";
            options.async = false;

            return options;
        }
    </script>
</head>
<body>
    <form onsubmit="return form_submit()">
        <table>
            <tr>
                <td class="fieldname">
                    Username:
                </td>
                <td>
                    <input type="text" id="username" />
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Email:
                </td>
                <td>
                    <input type="text" id="email" />
                </td>
            </tr>
            <tr>
                <td class="fieldname">
                    Password:
                </td>
                <td>
                    <input type="text" id="password" />
                </td>

            </tr>
            <tr>
                <td class="fieldname">
                    Verify Password:
                </td>
                <td>
                    <input type="text" id="password2" />
                </td>
```

```
            </tr>
            <tr>
                <td colspan="2" class="submit">
                    <input type="submitV value="Submit" />
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

附录 B

JavaScript 核心参考

本附录列出了所有 JavaScript 核心语言函数、对象及其属性和方法的语法。如果版本之间有变化，会特别说明。

B1 浏览器参考

表 B-1 列出了当前使用的 JavaScript 版本和 JavaScript 使用的浏览器。注意 Internet Explorer 实现了 Jscript，即 JavaScript 的 Microsoft 版本。但 Jscript 的功能与 JavaScript 大致相同。

表 B-1

JavaScript 版 本	Netscape Navigator	Mozilla Firefox	Internet Explorer	Safari / Chrome	Opera
1.0	2.x		3.0		
1.1	3.x				
1.2	4.0-4.05				
1.3	4.06-4.7x		4.0		
1.4	5.0				
1.5	6.x	1.0	5.5,6,7,8	3.0,3.1	6,7,8,9
1.6	6.2	1.5		3.2,4.0	1.0
1.7		2.0			
1.8		3.0			
1.81		3.5			

B2 保留字

JavaScript 使用各种保留字和符号。这些保留字不能用作变量名，也不能在变量名中使用符号。表 B-2 列出了保留字和符号。

表　B-2

abstract	boolean	break
byte	case	catch
char	class	const
continue	debugger	default
delete	do	double
else	enum	export
extends	false	final
finally	float	for
function	goto	if
implements	import	in
instanceof	int	interface
long	native	new
null	package	private
protected	public	return
short	static	super
switch	synchronized	this
throw	throws	transient
true	try	typeof
var	void	volatile
while	with	
-	!	~
%	/	*
>	<	=
&	^	|
+	?	

其他应避免使用的标识符

最好不要把下面的标识符用作变量名。

1. JavaScript 1.0

abs acos anchor asin atan atan2 big blink bold ceil charAt comment cos Date E escape eval exp fixed floor fontcolor fontsize getDate getDay getHours getMinutes getMonth getSeconds getTime getTimezoneOffset getYear indexOf isNaN italics lastIndexOf link log LOG10E LOG2E LN10 LN2 Math max min Object parse parseFloat parseInt PI pow random round,,c setDate setHours setMinutes setMonth setSeconds setTime setYear sin slice small sqrt SQRT1_2 SQRT2 strike String sub substr substring sup tan toGMTString toLocaleString toLowerCase toUpperCase unescape UTC

2. JavaScript 1.1

```
caller className constructor java JavaArray JavaClass JavaObject
JavaPackage join length MAX_VALUE MIN_VALUE NaN NEGATIVE_INFINITY netscape
Number POSITIVE_INFINITY prototype reverse sort split sun toString valueOf
```

3. JavaScript 1.2

```
arity callee charCodeAt compile concat exec fromCharCode global ignoreCase
index input label lastIndex lastMatch lastParen leftContext match multiline
Number Packages pop push RegExp replace rightContext search shift slice splice
source String test unshift unwatch watch
```

4. JavaScript 1.3

```
apply call getFullYear getMilliseconds getUTCDate getUTCDay getUTCFullYear
getUTCHours getUTCMilliseconds getUTCMinutes getUTCMonth getUTCSeconds
Infinity isFinite NaN setFullYear setMilliseconds setUTCDate setUTCFullYear
setUTCHours setUTCMilliseconds setUTCMinutes setUTCMonth setUTCSeconds
toSource toUTCString undefined
```

B3 JavaScript 运算符

下面几节列出了可以在 JavaScript 中使用的各种运算符。

B3.1 赋值运算符

赋值运算符可以把值赋予变量。表 B-3 列出了可以使用的各种赋值运算符。

表 B-3

名　称	引入版本	含　义
赋值	JavaScript 1.0	把变量 v1 的值设置为变量 v2 的值，var v1 = v2;
相加或连接的简写形式，等同于 v1 = v1 + v2	JavaScript 1.0	v1 += v2
相减的简写形式，等同于 v1 = v1 - v2	JavaScript 1.0	v1 -= v2
相乘的简写形式，等同于 v1 = v1 * v2	JavaScript 1.0	v1 *= v2
相除的简写形式，等同于 v1 = v1 / v2	JavaScript 1.0	v1 /= v2
取余的简写形式，等同于 v1 = v1 % v2	JavaScript 1.0	v1 %= v2
左移位的简写形式，等同于 v1 = v1 << v2	JavaScript 1.0	v1 <<= v2
右移位的简写形式，等同于 v1 = v1 >> v2	JavaScript 1.0	v1 >>= v2
补零右移位的简写形式，等同于 v1 = v1 >>> v2	JavaScript 1.0	v1 >>>= v2
AND 的简写形式，等同于 v1 = v1 & v2	JavaScript 1.0	v1 &= v2
XOR 的简写形式，等同于 v1 = v1 ^ v2	JavaScript 1.0	v1 ^= v2
OR 的简写形式，等同于 v1 = v1 | v2	JavaScript 1.0	v1 |= v2

B3.2　比较运算符

比较运算符可以比较两个变量或值。比较语句返回布尔值。如表 B-4 所示。

表　B-4

名　称	引入版本	含　义
等于	JavaScript 1.0	v1 == v2，如果两个操作数严格相等，或者转换为相同的类型后相等，就返回 true
不等于	JavaScript 1.0	v1 != v2，如果两个操作数不严格相等，或者转换为相同的类型后不相等，就返回 true
大于	JavaScript 1.0	v1 > v2，如果左边的操作数(LHS)大于右边的操作数(RHS)，就返回 true
大于等于	JavaScript 1.0	v1 >= v2，如果左边的操作数大于等于右边的操作数，就返回 true
小于	JavaScript 1.0	v1 < v2，如果左边的操作数小于右边的操作数，就返回 true
小于等于	JavaScript 1.0	v1 <= v2，如果左边的操作数小于等于右边的操作数，就返回 true
严格等于	JavaScript 1.3	v1 === v2，如果两个操作数相等，且类型相同，就返回 true
不严格等于	JavaScript 1.3	v1 !== v2，如果两个操作数不严格相等，就返回 true

B3.3　算术运算符

算术运算符可以对变量或值进行算术运算。如表 B-5 所示。

表　B-5

名　称	引入版本	含　义
加	JavaScript 1.0	v1 + v2，v1 和 v2 之和(如果其中一个操作数是字符串，就连接 v1 和 v2)
减	JavaScript 1.0	v1 - v2，v1 和 v2 之差
乘	JavaScript 1.0	v1 * v2，v1 和 v2 的乘积
除	JavaScript 1.0	v1 / v2，v1 和 v2 相除的商
取模	JavaScript 1.0	v1 % v2，v1 和 v2 整除，取余数
前缀递增	JavaScript 1.0	++v1 * v2 等价于(v1 + 1) * v2，注意在计算后，v1 变成了 v1+1.
后缀递增	JavaScript 1.0	v1++ * v2 等价于 v1 * v2，之后 v1 递增 1
前缀递减	JavaScript 1.0	--v1 * v2 等价于(v1 - 1) * v2，注意在计算后，v1 变成了 v1-1.
后缀递减	JavaScript 1.0	v1-- * v2 等价于 v1 * v2，之后 v1 递减 1

B3.4　按位运算符

按位运算符是把 v1 和 v2 的值转换为 32 位二进制数，再比较这两个二进制数的各个位，结果返回为正常的十进制数。如表 B-6 所示。

表 B-6

名　称	引入版本	含　义
按位与	JavaScript 1.0	v1 & v2，按位与操作数对齐两个操作数的位，对同一个位置上的两个位执行 AND 操作。如果两个位都是 1，这个位置上的结果位就返回 1，如果其中一个位是 0，这个位置上的结果位就返回 0
按位或	JavaScript 1.0	v1 | v2，按位或操作数对齐两个操作数的位，对同一个位置上的两个位执行 OR 操作。如果两个位中有一个位是 1，这个位置上的结果位就返回 1，如果两个位都是 0，这个位置上的结果位就返回 0
按位异或	JavaScript 1.0	v1 ^ v2，按位异或对齐两个操作数的位，对同一个位置上的两个位执行 XOR 操作。只有两个位都是 1，这个位置上的结果位才返回 1，否则，这个位置上的结果位就返回 0
按位非	JavaScript 1.0	v1～v2，反转操作数中的所有位

B3.5 按位移运算符

按位移运算符是把 v1 的值转换为 32 位二进制数，再把二进制数中的位向左或右移动指定的位数，如表 B-7 所示。

表 B-7

名　称	引入版本	含　义
左移	JavaScript 1.0	v1 << v2，把 v1 向左移动 v2 个位置，用 0 填充空隙
传播符号的右移	JavaScript 1.4	v1 >> v2，把 v1 向右移动 v2 个位置，忽略溢出的位
0 填充的右移	JavaScript 1.0	v1 >>> v2，把 v1 向右移动 v2 个位置，忽略溢出的位，在数的左边添加 v2 个 0

B3.6 逻辑运算符

逻辑运算符返回布尔值 true 或 false。但如果 v1 或 v2 不是布尔值，其中一个值也不容易转换为布尔值，例如 0、1、null、空字符串或 undefined，就不返回 true 或 false。如表 B-8 所示。

表 B-8

名　称	引入版本	含　义
逻辑与	JavaScript 1.0	v1 && v2，如果 v1 和 v2 都是 true，就返回 true，否则返回 false。如果 v1 是 false，就不计算 v2
逻辑或	JavaScript 1.0	v1 ‖ v2，如果 v1 和 v2 都是 false，就返回 false；否则返回 true。如果 v1 是 true，就不计算 v2
逻辑非	JavaScript 1.0	!v1，如果 v1 是 true，就返回 false，否则返回 true

B3.7　对象运算符

JavaScript 提供了操作对象的许多运算符，如表 B-9 所示。

表　B-9

名　　称	引 入 版 本	含　　义
delete	JavaScript 1.2	delete obj，删除一个对象、对象的一个属性或者数组中指定下标的元素，也可以删除未用 var 关键字声明的变量
in	JavaScript 1.4	for (prop in somObj)，如果 someObj 有指定的属性，就返回 true
instanceof	JavaScript 1.4	someObj instanceof ObjType，如果 someObj 的类型是 ObjType，就返回 true，否则返回 false
new	JavaScript 1.0	new ObjType()，创建 ObjType 类型的对象的一个新实例
this	JavaScript 1.0	this.property，引用当前对象

B3.8　其他运算符

表 B-10 列出了其他运算符。

表　B-10

名　　称	引 入 版 本	含　　义
条件运算符	JavaScript 1.0	(evalquery) ? v1 : v2; 如果 evalquery 是 true，运算符就返回 v1，否则返回 v2
逗号运算符	JavaScript 1.0	var v3 = (v1 + 2, v2 * 2) 计算两个操作数，并把它们当作一个表达式。返回第二个操作数的值。在这个例子中，v3 保存着 v2 * 2 的结果
typeof	JavaScript 1.1	typeof v1 返回一个字符串，它包含 v1 的类型，但不计算 v1
void	JavaScript 1.1	void(eva1) 计算 eval1，但不返回值

B3.9　运算符的优先级

是计算 1 + 2 * 3 = 1 + (2 * 3) = 7，还是计算(1 + 2) * 3 = 9？

运算符的优先级确定了运算符的计算顺序。例如，乘法运算符的优先级比加法运算符高，因此上面问题的正确答案是 1 + (2 * 3)。

表 B-11 按优先级从高到低的顺序列出了 JavaScript 中的运算符。第三列解释了把 1+2+3+4 读作((1+2)+3)+4 (从左到右)还是 1+(2+(3+(4))) (从右到左)。

表 B-11

运算符类型	运 算 符	类似元素的计算顺序
成员	.或[]	从左到右
创建实例	New	从右到左
函数调用	()	从左到右
递增	++	N/a
递减	--	N/a
逻辑非	!	从右到左
按位非	~	从右到左
一元+	+	从右到左
一元-	-	从右到左
类型	typeof	从右到左
Void	void	从右到左
删除	delete	从右到左
乘	*	从左到右
除	/	从左到右
取模	%	从左到右
加	+	从左到右
减	-	从左到右
按位移	<<,>>,>>>	从左到右
关系	<,<=,>,>=	从左到右
包含	in	从左到右
实例	instanceof	从左到右
相等	==,=!,===,!===	从左到右
按位与	&	从左到右
按位异或	^	从左到右
按位或	|	从左到右
逻辑与	&&	从左到右
逻辑或	||	从左到右
条件	?:	从右到左
赋值	=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=	从右到左
逗号	,	从左到右

B4 JavaScript 语句

下面的表描述了 JavaScript 的核心语句。

B4.1　块

JavaScript 块以左花括号({}开头，以右花括号())结束。块语句使包含在其中的多个语句一起执行，例如函数体或条件，如表 B-12 所示。

表　B-12

语　句	引入版本	说　明
{ }	JavaScript 1.5	用于组合用花括号界定的语句

B4.2　条件

表 B-13 列出了 JavaScript 的条件语句和引入它们的版本。

表　B-13

语　句	引入版本	说　明
if	JavaScript 1.2	如果指定的条件为 true，就执行代码块
else	JavaScript 1.2	If 语句的第二部分。如果 if 语句的结果为 false，就执行代码块
switch	JavaScript 1.2	根据作为参数传送过来的表达式值，执行不同的语句块

B4.3　声明

表 B-14 的关键字在 JavaScript 代码中声明了变量或函数。

表　B-14

语　句	引入版本	说　明
var	JavaScript 1.0	用于声明变量。在声明时还可以根据需要给变量初始化一个值
function	JavaScript 1.0	用指定的参数声明函数，参数可以是字符串、数字或对象。函数要返回值，必须使用 return 语句

B4.4　循环

循环在指定的条件为 true 时执行代码块。如表 B-15 所示。

表　B-15

语　句	引入版本	说　明
do…while	JavaScript 1.2	执行指定的语句，直到 while 后面的测试条件为 false 为止。语句至少执行一次，因为测试条件在最后计算
for	JavaScript 1.0	根据 for 后面括号中用分号分隔开的 3 个可选表达式，创建一个受控的循环。3 个表达式中的第一个是初始化表达式，第二个是测试条件，第三个是递增表达式
for…in	JavaScript 1.0	使用变量迭代对象的所有属性。对于每个属性，执行循环内部的指定语句

(续表)

语　句	引 入 版 本	说　　明
while	JavaScript 1.0	如果测试条件是 true，就执行语句块。循环每次重复都测试条件，如果条件为 false，就停止循环
break	JavaScript 1.0	在 while 或 for 循环中用于停止循环，把程序控制圈传送给循环后面的语句。也可以与 label 一起使用，跳到循环外部的特定程序位置上
label	JavaScript 1.2	可将这个标识符与 break 或 continue 语句一起使用，指定停止循环的执行后，程序从哪里继续执行

B4.5　执行控制语句

可以用许多方式控制代码的执行。除了条件和循环语句之外，表 B-16 也可以控制程序的执行。

表　B-16

语　句	引 入 版 本	说　　明
continue	JavaScript 1.0	用于停止 while 或 for 循环的当前迭代中语句块的执行，继续执行循环的下一次迭代
return	JavaScript 1.0	用于指定函数的返回值
with	JavaScript 1.0	为代码块指定默认对象

B4.6　异常处理语句

错误是编程的一个与生俱来的部分，JavaScript 提供了捕获错误和处理它们的方式。如表 B-17 所示。

表　B-17

语　句	引 入 版 本	说　　明
throw	JavaScript 1.4	抛出用户定义的一个定制异常
try…catch …finally	JavaScript 1.4	执行 try 块中的语句，如果出现了异常，就在 catch 块中处理。Finally 块可以指定在 try 和 catch 语句后执行的语句

B4.7　其他语句

表 B-18 列出了其他 JavaScript 语句和引入它们的版本。

表　B-18

语　句	引 入 版 本	说　　明
comment	JavaScript 1.0	脚本引擎忽略的注释，可以用于解释代码。注释有两类：单行注释和多行注释 // single line comment /* multi line comment */

B5　顶级属性和函数

表 B-19 和表 B-20 介绍核心属性和函数，它们与低层对象无关，但在 ECMAScript 和 Jscript 使用的术语中，它们称为全局对象的属性和方法。

表　B-19

属　性	引 入 版 本	说　　明
Infinity	JavaScript 1.3	返回无限大
NaN	JavaScript 1.3	返回不是数字的值
undefined	JavaScript 1.3	表示没有给变量赋值

表　B-20

函　　数	引 入 版 本	说　　明
decodeURI()	JavaScript 1.5	给用 encodeURI()编码的 URI 解码
decodeURIcomponent()	JavaScript 1.5	给用 encodeURIComponent()编码的 URI 解码
encodeURI()	JavaScript 1.5	用于构造完整 URI 的新版本，替代某些字符的每个实例。它基于字符的 UTF-8 编码
encodeURIComponent()	JavaScript 1.5	用转义序列替代某些字符的每个实例，来构造完整 URI 的新版本。通过字符的 UTF 编码来表示
escape()	JavaScript 1.0	编码 ISO Latin-1 字符集中的字符串，以添加到 URL 中或达到其他目的
eval()	JavaScript 1.0	返回 JavaScript 代码的结果，JavaScript 代码传送为一个字符串参数
isFinite()	JavaScript 1.3	表示参数是否是一个有限的数字
isNaN()	JavaScript 1.1	表示参数是否不是数字
Number()	JavaScript 1.2	把对象转换为数字
parseFloat()	JavaScript 1.0	解析一个字符串，返回为浮点数
parseInt()	JavaScript 1.0	解析一个字符串，返回为整数。可选的第二个参数指定要转换的数字的进制
String()	JavaScript 1.2	把对象转换为字符串
unescape()	JavaScript 1.0	给指定的 16 进制编码值返回 ASCII 字符串

顶级属性在 JavaScript 1.3 中引入，但在以前的版本中，存在 Infinity 和 NaN，它们是 Number 对象的属性。

B6　JavaScript 和 Jscript 核心对象

本节描述可以在 JavaScript 和 Jscript 核心语言中使用的对象，以及对象的方法和属性。

B6.1 ActiveXObject

在 Microsoft 的 JScript 代码中访问时，ActiveXObject 对象表示 ActiveX 对象。它在 Jscript 3.0 中引入，在 ECMAScript 或 JavaScript 中不可用。它用 ActiveXObject 构造函数创建。例如，要创建一个 Microsoft Word 文档，可以编写如下代码：

```
var objActiveX = new ActiveXObject("Word.Document");
```

这个对象的属性和方法就是所创建的 ActiveX 对象的属性和方法。例如，下面的代码打开一个 Word 文档，把一些文本写入该文档和 HTML 页面：

```
var objActiveX = new ActiveXObject("Word.Document");
strText="This is being written both to the HTML page and to the Word document.";
objActiveX.application.selection.typeText(strText);
document.write(strText);
```

B6.2 Array

Array 对象表示一组变量，它在 JavaScript 1.1 中引入。可以用 Array 构造函数创建 Array 对象：

```
var objArray = new Array(10)              // an array of 11 elements
var objArray = new Array("1", "2", "4")   // an array of 3 elements
```

也可以用数组字面语法创建数组：

```
var objArray = [];
```

字面语法是创建数组的首选方法。Array 对象的属性和方法如表 B-21 和表 B-22 所示。

表 B-21

属 性	引 入 版 本	说 明
constructor	JavaScript 1.1	用于引用对象的构造函数
length	JavaScript 1.1	返回数组中的元素个数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表 B-22

方 法	引 入 版 本	说 明
concat(value1 [, value2, …])	JavaScript 1.2	连接两个数组，返回构造出的新数组
every(testFn(element, index, array))	JavaScript 1.6	迭代数组，在每个元素上执行 testFn()。如果所有迭代都返回 true，该方法就返回 true；否则返回 false
filter(testFn(element, index, array))	JavaScript 1.6	迭代数组，在每个元素上执行 testFn()。返回一个新数组，该数组由传送给 testFn()的元素组成

(续表)

方　法	引入版本	说　明
foreach(fn(element, index, array))	JavaScript 1.6	迭代数组，在每个元素上执行 Fn()
indexOf(element [, startIndex])	JavaScript 1.6	如果找到指定的元素，就返回该元素的下标，如果没有找到该元素，就返回-1。如果指定了 startIndex，就从 startIndex 开始查找
join([separator])	JavaScript 1.1	把数组中的所有元素连接起来，成为一个字符串，如果指定了分隔符，就用该分隔符分隔字符串中的各个元素
lastIndexOf(element [, startIndex])	JavaScript 1.6	在最后一个元素开始向后查找，如果找到指定的元素，就返回该元素的下标，如果找不到该元素，就返回-1。如果指定了 startIndex，就从 startIndex 开始查找
map(fn(element, index, array))	JavaScript 1.6	迭代数组，在每个元素上执行 fn()。根据 fn()的结果返回一个新数组
pop()	JavaScript 1.2	取出数组末尾的最后一个元素，并返回该元素
push(value1 [, value2, …])	JavaScript 1.2	把一个或多个元素放在数组的末尾，并返回数组的新长度
reverse()	JavaScript 1.1	逆序排列数组的元素，使第一个元素变成最后一个元素，最后一个元素变成第一个元素
shift()	JavaScript 1.2	从数组的开头删除第一个元素，并返回该元素
slice(startIndex [, endIndex])	JavaScript 1.2	返回数组的一个部分，该部分从 startIndex 下标开始，到 endIndex 下标之前的元素结束
some(testFn(element, index, array))	JavaScript 1.6	迭代数组，在每个元素上执行 testFn()。如果 test-Fn()的结果至少有一个是 true，就返回 true
sort([sortFn(a,b)])	JavaScript 1.1	给数组的元素排序，如果提供 sortFn()了，就执行 sortFn()，以进行排序
splice(startIndex [, length, value1, …)	JavaScript 1.2	从 startIndex 开始删除 length 个元素。用所提供的值替代被删除的元素，返回被删除的元素
toString()	JavaScript 1.1	把 Array 对象转换为字符串
unshift(value1 [, value2, …])	JavaScript 1.2	把元素添加到数组开头，返回数组的新长度
valueOf()	JavaScript 1.1	返回数组的原始值

上表中带方括号([])的参数表示可选参数。

B6.3　Boolean

Boolean 对象用作布尔值的封装器，它在 JavaScript 1.1 中引入，用 Boolean 构造函数创建，该构造函数的参数是对象的初始值(如果参数不是布尔值，就转换为布尔值)。

False-y 是 null、undefined、" "和 0，其他值都是 truth-y。Boolean 对象的属性和方法如表 B-23 和 B-24 所示。

表 B-23

属 性	引 入 版 本	说 明
constructor	JavaScript 1.1	指定创建对象原型的函数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表 B-24

方 法	引 入 版 本	说 明
toString()	JavaScript 1.1	把 Boolean 对象转换为字符串
valueOf()	JavaScript 1.1	返回 Boolean 对象的原始值

B6.4 Date

Date 对象用于表示给定的日期时间，它在 JavaScript 1.0 中引入。Date 对象的属性和方法如表 B-25 和表 B-26 所示。

表 B-25

属 性	引 入 版 本	说 明
constructor	JavaScript 1.1	表示对象的构造函数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表 B-26

方 法	引 入 版 本	说 明
getDate()	JavaScript 1.0	从 Date 对象中获取某月的日期
getDay()	JavaScript 1.0	从 Date 对象中获取星期几
getFullYear()	JavaScript 1.3	从 Date 对象中获取完整的年份
getHours()	JavaScript 1.0	从 Date 对象中获取某天的小时数
getMilliseconds()	JavaScript 1.3	从 Date 对象中获取微秒数
getMinutes()	JavaScript 1.0	从 Date 对象中获取分钟数
getMonth()	JavaScript 1.0	从 Date 对象中获取月份
getSeconds()	JavaScript 1.0	从 Date 对象中获取秒数
getTime()	JavaScript 1.0	从 Date 对象中获取从 1970 年 1 月 1 日 00:00:00 开始到现在的微秒数
getTimezoneOffset()	JavaScript 1.0	获取本地时区和国际协调时间(UTC)之间的分钟差
getUTCDate()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的某月的日期
getUTCDay()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的星期几
getUTCFullYear()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的完整年份
getUTCHours()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的某天的小时数

(续表)

方　法	引入版本	说　明
getUTCMilliseconds()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的微秒数
getUTCMinutes()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的分钟数
getUTCMonth()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的月份
getUTCSeconds()	JavaScript 1.3	从 Date 对象中获取调整到国际协调时间的秒数
getYear()	JavaScript 1.0	从 Date 对象中获取年份
parse(dateString)	JavaScript 1.0	从 Date 对象中获取从本地时间的 1970 年 1 月 1 日 00:00:00 开始到现在的微秒数
setDate(dayOfMonth)	JavaScript 1.0	给 Date 对象设置某月的日期
setFullYear(year [, month, day])	JavaScript 1.3	给 Date 对象设置完整年份
setHours(hours [, minutes, seconds, milliseconds])	JavaScript 1.0	给 Date 对象设置某天的小时数
setMilliseconds(milliseconds)	JavaScript 1.3	给 Date 对象设置微秒数
setMinutes(minutes [, seconds, milliseconds])	JavaScript 1.0	给 Date 对象设置分钟数
setMonth(month [, day])	JavaScript 1.0	给 Date 对象设置月份
setSeconds(seconds [, milliseconds])	JavaScript 1.0	给 Date 对象设置秒数
setTime(milliseconds)	JavaScript 1.0	根据从 1970 年 1 月 1 日 00:00:00 开始到现在的微秒数给 Date 对象设置时间
setUTCDate(dayOfMonth)	JavaScript 1.3	根据国际协调时间给 Date 对象设置某月的日期
setUTCFullYear(year [, month,day])	JavaScript 1.3	根据国际协调时间给 Date 对象设置完整年份
setUTCHours(hours [, minutes, seconds, milliseconds])	JavaScript 1.3	根据国际协调时间给 Date 对象设置某天的小时数
setUTCMilliseconds(milliseconds)	JavaScript 1.3	根据国际协调时间给 Date 对象设置微秒数
setUTCMinutes(mintes [, seconds, milliseconds])	JavaScript 1.3	根据国际协调时间给 Date 对象设置分钟数
setUTCMonth(month [, day])	JavaScript 1.3	根据国际协调时间给 Date 对象设置月份
setUTCSeconds()	JavaScript 1.3	根据国际协调时间给 Date 对象设置秒数
setYear(year)	JavaScript 1.0	给 Date 对象设置年份，已被 setFullYear()取代
toGMTString()	JavaScript 1.0	根据格林威治时间，把 Date 对象转换为字符串，已被 toUTCString 取代
toLocaleString()	JavaScript 1.0	根据本地时区，把 Date 对象转换为字符串
toString()	JavaScript 1.1	把 Date 对象转换为字符串
toUTCString()	JavaScript 1.3	根据国际协调时间，把 Date 对象转换为字符串
UTC(year, month [, day, hours, minutes, seconds, milliseconds])	JavaScript 1.0	从 Date 对象中获取从国际协调时间的 1970 年 1 月 1 日 00:00:00 开始到现在的微秒数
valueOf()	JavaScript 1.1	返回 Date 对象的原始值

B6.5 Function

ActiveXObject 函数表示根据需要调用的 JavaScript 代码块。Function 对象在 JavaScript 1.1 中引入，用 Function 构造函数创建。

可以用许多方式定义函数，使用下面的标准函数语句就可以创建函数：

```
function functionName() {
    // code here
}
```

也可以创建一个匿名函数，再把它赋予一个变量。下面的代码演示了这种方法：

```
var functionName = function() {
    // code here
};
```

尾部的分号并非输入错误，因为这个语句是一个赋值操作，所有的赋值操作都应以分号结尾。

函数是对象，因此有构造函数。也可以使用 Function 对象的构造函数创建函数，如下面的代码所示：

```
var functionName = new Function("arg1", "arg2", "return arg1 + arg2");
```

构造函数的第一个参数是函数的参数名——可以添加任意多个参数。传送给构造函数的最后一个参数是函数体。前面的代码创建了一个函数，它接受两个参数，返回它们的和。

使用 Function 构造函数的实例非常少。最好使用标准函数语句定义函数，或者创建匿名函数，再赋予一个变量。Function 对象的属性和方法如表 B-27 和表 B-28 所示。

表 B-27

属　性	引入版本	说　明
arguments	JavaScript 1.1	包含传送给函数的参数的数组
arguments.length	JavaScript 1.1	返回传送给函数的参数个数
constructor	JavaScript 1.1	表示对象的构造函数
length	JavaScript 1.1	返回函数需要的参数个数，它不同于 arguments.length，arguments.length 返回实际传送给函数的参数个数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表 B-28

方　法	引入版本	说　明
apply(thisObj, arguments)	JavaScript 1.3	调用函数或方法，就好像该函数或方法属于 thisObj，并给该函数或方法传送 arguments。arguments 必须是数组
call(thisObj,arg1, …)	JavaScript 1.3	与 apply()相同，但参数是逐个传送的，而不是作为一个数组传送的
toString()	JavaScript 1.1	把 Function 对象转换为字符串
valueOf()	JavaScript 1.1	返回 Function 对象的原始值

B6.6　Math

Math 对象提供了用于数学计算的方法和属性，在 JavaScript 1.0 中引入。Math 是一个顶级对象，没有构造函数也可以访问它。Math 对象的属性和方法如表 B-29 和 B-30 所示。

表　B-29

属　　性	引 入 版 本	说　　明
E	JavaScript 1.0	返回欧拉常数(自然对数的底，约为 2.718)
LN10	JavaScript 1.0	返回 10 的自然对数(约为 2.302)
LN2	JavaScript 1.0	返回 2 的自然对数(约为 0.693)
LOG10E	JavaScript 1.0	返回以 10 为底的 E 的对数(约为 0.434)
LOG2E	JavaScript 1.0	返回以 2 为底的 E 的对数(约为 1.442)
PI	JavaScript 1.0	返回 pi，即圆周率(圆的周长与其直径之比，约为 3.142)
SQRT1_2	JavaScript 1.0	返回 1/2 的平方根(约为 0.707)
SQRT2	JavaScript 1.0	返回 2 的平方根(约为 1.414)

表　B-30

方　　法	引 入 版 本	说　　明
abs(x)	JavaScript 1.0	返回一个数的绝对值(正值)
acos(x)	JavaScript 1.0	返回一个数的反余弦(以弧度为单位)
asin(x)	JavaScript 1.0	返回一个数的反正弦(以弧度为单位)
atan(x)	JavaScript 1.0	返回一个数的反正切(以弧度为单位)
atan2(y, x)	JavaScript 1.0	返回 x 轴与作为参数传送进来的 y 和 x 坐标表示的位置之间的角度(以弧度为单位)
ceil(x)	JavaScript 1.0	把一个数向上圆整到最近的整数值
cos(x)	JavaScript 1.0	返回一个数的余弦值
exp(x)	JavaScript 1.0	返回以 E 为底、以传送进来的参数为指数计算出来的幂
floor(x)	JavaScript 1.0	把一个数向下圆整到最近的整数值
log(x)	JavaScript 1.0	返回一个数的自然对数(以 E 为底)
max(a, b)	JavaScript 1.0	返回传送进来的两个参数中的较大者
min(a, b)	JavaScript 1.0	返回传送进来的两个参数中的较小者
pow(x, y)	JavaScript 1.0	返回以第一个参数为底、以第二个参数为指数计算出来的幂
random()	JavaScript 1.1	返回 0 到 1 之间的一个伪随机数
round(x)	JavaScript 1.0	把一个数向上或向下圆整到最近的整数值
sin(x)	JavaScript 1.0	返回一个数的正弦值
sqrt(x)	JavaScript 1.0	返回一个数的平方根
tan(x)	JavaScript 1.0	返回一个数的正切值

B6.7 Number

Number 对象是基本数值的封装器，它在 JavaScript 1.1 中引入，使用 Number 构造函数创建，其中初始数字值作为参数传送进来。如表 B-31 和表 B-32 所示。

表 B-31

属　性	引入版本	说　明
constructor	JavaScript 1.1	表示对象的构造函数
MAX_VALUE	JavaScript 1.1	返回 JavaScript 中可以表示的最大值(约为 1.79E+308)
MIN_VALUE	JavaScript 1.1	返回 JavaScript 中可以表示的最小值(5E-324)
NaN	JavaScript 1.1	返回不是数字的值
NEGATIVE_INFINITY	JavaScript 1.1	返回表示负无穷大的值
POSITIVE_INFINITY	JavaScript 1.1	返回表示正无穷大的值
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表 B-32

方　法	引入版本	说　明
toExponential(fractionDigits)	JavaScript 1.5	返回一个包含数字的指数表示法的字符串。参数应在 0 到 20 之间，以确定小数点后的位数
toFixed([digits])	JavaScript 1.5	数字的格式，该数字向上圆整，在小数点后添加 0，以达到期望的小数位数
toPrecision([precision])	JavaScript 1.5	返回一个字符串，表示指定精度的 Number 对象
toString()	JavaScript 1.1	把 Number 对象转换为字符串
valueOf()	JavaScript 1.1	返回 Number 对象的原始值

B6.8 Object

Object 是 JavaScript 对象的基本类型，所有其他对象都派生自 Object(即所有其他对象都继承了 Object 对象的方法和属性)。它在 JavaScript 1.0 中引入，可以用 Object 构造函数创建，如下所示：

```
var obj = new Object();
```

还可以使用对象字面量表示法创建对象，如下所示：

```
var obj = {};
```

对象字面量表示法是创建对象的首选方法。如表 B-33 和表 B-34 所示。

表 B-33

属　性	引入版本	说　明
constructor	JavaScript 1.1	表示对象的构造函数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表　B-34

方　　法	引 入 版 本	说　　明
hasOwnProperty(propertyName)	JavaScript 1.5	检查指定的属性是否是继承的。如果不是继承的，就返回 true，否则返回 false
isPrototypeOf(obj)	JavaScript 1.5	确定指定的对象是否是另一个对象的原型
propertyIsEnumerable(propertyName)	JavaScript 1.5	确定指定的属性是否可以在 for in 循环中访问
toString()	JavaScript 1.0	把 Object 对象转换为字符串
valueOf()	JavaScript 1.1	返回 Object 对象的原始值

B6.9　RegExp

RegExp 对象用于在字符串值中查找模式。如表 B-35 和 B-36 所示可以用两种方法创建该对象：使用 RegExp 构造函数或字面文本。RegExp 对象在 JavaScript 1.2 中引入。表 B-37 列出了正则表达式中使用的特殊字符。

表 B-35 的一些属性有长名称和短名称，短名称是从 Perl 编程语言中派生而来的。

表　B-35

属　　性	引 入 版 本	说　　明
constructor	JavaScript 1.2	表示对象的构造函数
global	JavaScript 1.2	表示是查找字符串中所有的匹配，还是只查找第一个匹配。对应于 g 标记
ignoreCase	JavaScript 1.2	表示匹配是否不区分大小写。对应于 i 标记
input	JavaScript 1.2	正则表达式要匹配的字符串
lastIndex	JavaScript 1.2	指定下一个匹配从字符串的哪个位置开始
multiline	JavaScript 1.2	表示是否在多行上搜索字符串。对应于 m 标记
prototype	JavaScript 1.2	返回对象的原型，对象的原型可用于扩展对象的接口
source	JavaScript 1.2	正则表达式的模式文本

表　B-36

方　　法	引 入 版 本	说　　明
exec(stringToSearch)	JavaScript 1.2	在所传送的字符串参数中搜索匹配
test(stringToMatch)	JavaScript 1.2	在所传送的字符串参数中测试匹配
toString()	JavaScript 1.2	把 RegExp 对象转换为字符串
valueOf()	JavaScript 1.2	返回 RegExp 对象的原始值

表 B-37

字 符	示 例	作 用
\	/n/ 匹配 n; /\n/ 匹配换行符 /^/ 匹配行首 /\^/ 匹配 ^	对于默认处理为正常字符的字符，反斜杠表示下一个字符用特殊的值解释； 对于一般处理为特殊字符的字符，反斜杠表示下一个字符解释为正常字符
^	/^A/匹配A man called Adam中的第一个A，不匹配第二个A	匹配行首或 input 的开头
$	/r$/匹配horror中的最后一个r	匹配行末或 input 的结尾
*	/ro*/匹配right中的r，匹配wrong中的ro，匹配room中的roo	0 次或多次匹配前面的字符
+	/l+/匹配life中的l，匹配still中的ll，匹配stilllife中的lll	0 次或多次匹配前面的字符，例如/a+/匹配 candy 中的 a，匹配 caaaaaaandy 中的所有 a
?	/Smythe?/匹配Smyth和Smythe	0 次或多次匹配前面的字符
.	/.b/匹配blob中的第二个b，不匹配第一个b	匹配除换行符之外的所有字符
(x)	/(Smythe?)/匹配John Smyth and Rob Smythe中的Smyth和Smythe，也可以把子字符串提取为RegExp.$1和RegExp.$2	匹配 x，并存储该匹配。匹配的子字符串可以从匹配结果数组的元素中提取，或者从 RegExp 对象的$1, $2 ... $9 或 lastParen 属性中提取
x|y	/Smith|Smythe/匹配Smith和Smythe	匹配 x 或 y(x 和 y 是字符块)
{n}	/l{2}/匹配still中的ll，也匹配stilllife中的前两个l	精确匹配前面字符的 n 个实例(n 是正整数)
{n,}	/l{2,}/ 匹配still中的ll，也匹配stilllife中的lll	匹配前面字符的 n 个或更多实例(n 是正整数)
{n,m}	/l{1,2}/匹配life中的l，匹配still中的ll，也匹配stilllife中的前两个l	匹配前面字符的 n 个到 m 个实例(n 和 m 是正整数)
[xyz]	[ab]匹配a和b，[a-c]匹配a、b和c	匹配方括号中的任一个字符。使用连字符可以匹配字母表中的某个字符范围
[^xyz]	[^aeiouy]匹配easy中的s，[^a-y]匹配lazy中的z	匹配不包含在方括号中的字符。使用连字符可以指定字母表中的某个字符范围
[\b]		匹配一个退格符
\b	/t\b/匹配 about time 中的第一个 t	匹配一个字边界(例如一个空格或行尾)
\B	/t\Bi/匹配 it is time 中的 ti	匹配没有字边界的位置上的字符
\cX	/\cA/匹配 Ctrl-A	匹配一个控制符
\d	/IE\d/匹配 IE4, IE5 等	匹配一个数字字符，它等同于[0-9]
\D	/\D/匹配 3.142 中的小数点	匹配任意非数字字符，它等同于[^0-9]
\f		匹配换页符
\n		匹配换行符
\r		匹配回车符
\s	/ \s/匹配 not now 中的空格	匹配任意空白字符，包括空格、制表符和换行符等。它等同于[\f\n\r\t\v]
\S	/ \S/匹配"a"中的 a	匹配不是空白字符的任何字符，它等同于[^ \f\n\r\t\v]

(续表)

字　符	示　　例	作　　用
\t		匹配制表符
\v		匹配竖杠字符
\w	/ \w/匹配"O?!"中的 O，匹配"$1"中的 1	匹配任意字母数字字符或下划线，它等同于[A-Za-z0-9_]
\W	/ \W/匹配$10million 中的$，匹配 j_smith@wrox 中的@	匹配任意非字母数字字符(下划线除外)，它等同于[^A-Za-z0-9_]
()\n	/(Joh?n)和\1/匹配 John and John's friend 中的 John and John，但不匹配 John and Jon	在与括号中内容的第 n 个匹配中提取最后一个子字符串，并存储该子字符串(n 是正整数)
\octal \xhex	/\x25/匹配%	匹配与指定的 8 进制或 16 进制转义值对应的字符

B6.10　String

String 对象用于包含一串字符，它在 JavaScript 1.0 中引入，它必须与字符串字面量区分开，但 String 对象的方法和属性也可以由字符串字面量访问，因为在调用它们时，会创建临时对象。

最后一个表中的 HTML 方法不在 ECMAScript 标准中，但它们是 JavaScript 1.0 语言及其以后版本的一部分，它们可用于动态生成 HTML。String 对象的属性和方法如表 B-38 到 B-40 所示。

表　B-38

属　　性	引 入 版 本	说　　明
constructor	JavaScript 1.1	用于引用对象的构造函数
length	JavaScript 1.0	返回字符串中的字符个数
prototype	JavaScript 1.1	返回对象的原型，对象的原型可用于扩展对象的接口

表　B-39

方　　法	引 入 版 本	说　　明
charAt(index)	JavaScript 1.0	返回字符串中指定位置的字符
charCodeAt(index)	JavaScript 1.2	返回字符串中指定位置的字符的 Unicode 值
concat(value1,value2, ...)	JavaScript 1.2	连接作为参数提供的字符串，返回构造出来的新字符串
fromCharCode(value1, value2, ...)	JavaScript 1.2	返回一个字符串，该字符串由用户提供的 Unicode 值表示的字符连接而成
indexOf(substr [, startIndex])	JavaScript 1.0	返回 String 对象中第一个与所提供的子字符串匹配的子字符串位置，如果没有找到子字符串，就返回-1。如果提供了 startIndex，就从 startIndex 开始查找
lastIndexOf(substr [, startIndex])	JavaScript 1.0	返回 String 对象中最后一个与所提供的子字符串匹配的子字符串位置，如果没有找到子字符串，就返回-1。如果提供了 startIndex，就从 startIndex 开始查找

(续表)

方　法	引入版本	说　明
match(regexp)	JavaScript 1.2	在字符串中查找与所提供的模式匹配的子字符串，返回一个数组，如果没有找到匹配，就返回 null
replace(regexp, newValue)	JavaScript 1.2	用新值替代匹配正则表达式的子字符串
search(regexp)	JavaScript 1.2	在正则表达式和字符串之间搜索匹配，返回该匹配的索引，如果没有找到匹配，就返回-1
slice(startIndex [, endIndex])	JavaScript 1.0	返回 String 对象的一个子字符串
split(delimiter)	JavaScript 1.1	把字符串分隔为子字符串，从而把 String 对象分割为一组字符串
substr(startIndex [, length])	JavaScript 1.0	从给定的开始位置返回一个子字符串，其中包含指定数量的字符
substring(startIndex [, endIndex])	JavaScript 1.0	返回字符串中两个位置之间的子字符串，endIndex 处的字符不包含在子字符串中
toLowerCase()	JavaScript 1.0	返回转换为小写的字符串
toUpperCase()	JavaScript 1.0	返回转换为大写的字符串

表　B-40

方　法	引入版本	描　述
anchor(name)	JavaScript 1.0	返回用<a>…</a>标记界定的字符串，并把所传送的参数赋予 name 属性
big()	JavaScript 1.0	把字符串放在<big>…</big>标记中
blink()	JavaScript 1.0	把字符串放在<blink>…</blink>标记中
bold()	JavaScript 1.0	把字符串放在…标记中
fixed()	JavaScript 1.0	把字符串放在<tt>…</tt>标记中
fontcolor(color)	JavaScript 1.0	把字符串放在<font>…</font>标记中，并把参数值赋予 color 属性
fontsize(size)	JavaScript 1.0	把字符串放在<font>…</font>标记中，并把参数值赋予 size 属性
italics()	JavaScript 1.0	把字符串放在<I>…</i>标记中
link(url)	JavaScript 1.0	把字符串放在<a>…</a>标记中，并把参数值赋予 href 属性
small()	JavaScript 1.0	把字符串放在<small>…</small>标记中
strike()	JavaScript 1.0	把字符串放在<strike>…</strike>标记中
sub()	JavaScript 1.0	把字符串放在_…标记中
sup()	JavaScript 1.0	把字符串放在[…]标记中，并把字符串显示为上标

附录 C

W3C DOM 参考

由于 JavaScript 主要用于浏览器编程，以及给网页添加功能，所以有必要介绍 W3C DOM 参考信息。

下面列出 W3C DOM 可用的对象。

C1 DOM 核心对象

本节描述并列出了 DOM 标准定义的对象，从最低级的 DOM 对象开始。所有对象都按字母表顺序介绍。

C1.1 低级 DOM 对象

DOM 规范描述了 Node、NodeList 和 NamedNodeMap 对象，它们是 DOM 中最低级的对象，也是高级对象的主要构建块。

1. Node

Node 对象在 DOM Level 1 上定义，是整个 DOM 的主要数据类型。DOM 中的所有对象都继承自 Node。Node 对象有 12 种不同类型，每种类型都有一个关联的整数值。表 C-1 到表 C-3 列出了 Node 对象的类型值、属性和方法。

表 C-1

类 型 名	整 数 值	引入的层级	关联的数据类型
ELEMENT_NODE	1	Level 1	Element
ATTRIBUTE_NODE	2	Level 1	Attr
TEXT_NODE	3	Level 1	Text
CDATA_SECTION_NODE	4	Level 1	CDATASection
ENTITY_REFERENCE_NODE	5	Level 1	EntityReference

(续表)

类 型 名	整 数 值	引入的层级	关联的数据类型
ENTITY_NODE	6	Level 1	Entity
PROCESSING_INSTRUCTION_NODE	7	Level 1	ProcessingInstruction
COMMENT_NODE	8	Level 1	Comment
DOCUMENT_NODE	9	Level 1	Document
DOCUMENT_TYPE_NODE	10	Level 1	DocumentType
DOCUMENT_FRAGMENT_NODE	11	Level 1	DocumentFragment
NOTATION_NODE	12	Level 1	Notation

表 C-2

属 性 名	说 明	引入的层级
attributes	NamedNodeMap，如果它是一个 Element，就包含这个节点的属性，否则就是 null	Level 1
childNodes	包含这个节点的所有子节点的 NodeList	Level 1
firstChild	获取这个节点的第一个子节点。如果没有子节点，就返回 null	Level 1
lastChild	获取这个节点的最后一个子节点。如果没有子节点，就返回 null	Level 1
localName	返回节点的限定名中的本地部分(使用名称空间时，就是节点限定名的冒号后面的部分)。主要在 XML DOM 中使用	Level 2
namespaceURI	节点的名称空间 URI，如果未指定，就是 null	Level 2
nextSibling	获取紧跟在这个节点后面的节点，如果其后不存在节点，就返回 null	Level 1
nodeName	获取这个节点的名称	Level 1
nodeType	表示这个节点的类型的整数。参见表 C-1	Level 1
nodeValue	根据类型获取这个节点的值	Level 1
ownerDocument	获取包含这个节点的 Document 对象。如果这个节点是 Document，就返回 null	Level 1
parentNode	获取这个节点的父节点。对于当前不在 DOM 树中的节点，返回 null	Level 1
prefix	返回这个节点的名称空间前缀，如果未指定，就返回 null	Level 2
previousSibling	获取正好位于这个节点之前的节点，如果其前不存在节点，就返回 null	Level 1

表 C-3

方 法 名	说 明	引入的层级
appendChild(newChild)	把 newChild 添加到子节点列表的末尾	Level 1
cloneNode(deep)	返回节点的副本。返回的节点没有父节点。如果 deep 是 true，这个方法就复制节点包含的所有子节点	Level 1
hasAttributes()	根据节点是否有属性(节点是否是一个元素)，返回一个布尔值	Level 2

(续表)

方　法　名	说　　明	引入的层级
hasChildNodes()	根据节点是否有子节点，返回一个布尔值	Level 1
insertBefore(newChild, refChild)	在 refChild 引用的已有子节点前面插入 newChild。如果 refChild 是 null，newChild 就添加到子节点列表的末尾	Level 1
removeChild(oldChild)	删除指定的子节点，并返回该节点	Level 1
replaceChild(newChild, oldChild)	用 newChild 替换 oldChild，并返回 oldChild	Level 1

2. NodeList

NodeList 对象是一个有序的节点集合，NodeList 中包含的项可通过从 0 开始的索引来访问。

NodeList 是节点的实时快照。在 DOM 中对节点的任何修改都会立即反映到 NodeList 的每个引用上。NodeList 对象的属性和方法如表 C-4 和 C-5 所示。

表　C-4

属　性　名	说　　明	引入的层级
length	列表中的节点数	Level 1

表　C-5

方　法　名	说　　明	引入的层级
item(index)	返回指定索引的项，如果该索引大于或等于列表长度，就返回 null	Level 1

3. NamedNodeMap

名为 NamedNodeMap 的对象表示可以按名称访问的节点集合。这个对象没有继承 NodeList。元素的属性列表就是 NamedNodeMap 的一个例子。NamedNodeMap 对象的属性和方法如表 C-6 和 C-7 所示。

表　C-6

属　性　名	说　　明	引入的层级
length	映射中的节点数	Level 1

表　C-7

方　法　名	说　　明	引入的层级
getNamedItem(name)	获取指定名称的节点	Level 1
removeNamedItem(name)	删除指定名称的项	Level 1
setNamedItem(node)	把 nodeName 属性用作键，给列表添加一个节点	Level 1

C1.2 高级 DOM 对象

这些对象继承了 Node，是 HTML DOM 指定的更高级 DOM 对象的基础。这些对象镜像了不同的节点类型。

下面的对象按字母顺序列出。本节忽略了 CDATASection、Comment、DocumentType、Entity、EntityReference、Notation 和 ProcessingInstruction 对象。

1. Attr

Attr 对象表示 Element 对象的属性。尽管 Attr 对象继承了 Node，但并不认为它们是所描述的元素的子元素，因此不是 DOM 树的一部分。对于 Attr 对象，parentNode、previousSibling 和 nextSibling 的 Node 属性返回 null。Attr 对象的属性如表 C-8 所示。

表 C-8

属 性 名	说 明	引入的层级
ownerElement	返回属性所关联的 Element 对象	Level 2
name	返回属性的名称	Level 1
value	返回属性的值	Level 1

2. Document

Document 对象表示整个 HTML 或 XML 文档。它是文档树的根。Document 是文档中所有节点的容器，每个 Node 对象的 ownerDocument 属性都指向 Document。Document 对象的属性和方法如表 C-9 和 C-10 所示。

表 C-9

属 性 名	说 明	引入的层级
docType	与这个文档关联的 DocType 对象。对于没有文档类型声明的 HTML 和 XML 文档，返回 null	Level 1
documentElement	返回文档的根元素。对于 HTML 文档，documentElement 是 <html/>元素	Level 1
implementation	与 Document 关联的 DOMImplementation 对象	Level 1

表 C-10

方 法 名	说 明	引入的层级
createAttribute(name)	返回带有指定名称的新 Attr 对象	Level 1
createAttributeNS(namespaceURI, qualifiedName)	返回带有指定限定名称和名称空间 URI 的属性，不用于 HTML DOM	Level 2
createComment(data)	返回带有指定数据的新 Comment 对象	Level 1
createCDATASection(data)	返回其值是指定数据的新 CDATASection 对象	Level 1
createDocumentFragment()	返回一个空的 DocumentFragment 对象	Level 1

(续表)

方　法　名	说　　明	引入的层级
createElement(tagName)	返回带有指定标记名的新 Element 对象	Level 1
createElementNS(namespaceURI, qualifiedName)	返回带有指定的限定名称和名称空间 URI 的元素，不用于 HTML DOM	Level 1
createTextNode(text)	返回包含指定文本的新 Text 对象	Level 1
getElementById(elementId)	返回带有指定 ID 值的 Element。如果该元素不存在，就返回 null	Level 2
getElementsByTagName(tagName)	返回带有指定标记名称的所有 Element 对象的一个 NodeList，Element 对象的顺序就是它们在 DOM 树中的显示顺序	Level 1
getElementsByTagNameNS (namespaceURI,localName)	返回带有指定的本地名称和名称空间 URI 的所有元素的一个 NodeList，元素的顺序就是它们在 DOM 树中的显示顺序	Level 2
importNode(importedNode, deep)	从另一个文档中导入节点。不从其文档中删除或改变源节点，而是创建源节点的一个副本。如果 deep 是 true，则导入源节点的所有子节点，如果 deep 是 false，就只导入源节点	Level 2

3. DocumentFragment

DocumentFragment 对象是一个轻型 Document 对象，其主要作用是提高效率。对 DOM 树进行许多修改是一个很昂贵的过程，例如逐个追加多个节点。可以把 Node 对象追加到 DocumentFragment 对象上，以方便而高效地在 DOM 树中插入包含在 DocumentFragment 中的所有节点。

下面的代码演示了 DocumentFragment 的用法：

```
var documentFragment = document.createDocumentFragment();

for (var i = 0; i < 1000; i++) {
    var element = document.createElement("div");
    var text = document.createTextNode("Here is test for div #" + i);

    element.setAttribute("id", i);

    documentFragment.appendChild(element);
}

document.body.appendChild(documentFragment);
```

没有 DocumentFragment 对象，这段代码要更新 DOM 树 1000 次，这会降低性能。而有了 DocumentFragment 对象，DOM 树就只更新一次。

DocumentFragment 对象继承了 Node 对象，所以也拥有 Node 的属性和方法，它没有其他属性或方法。

4. Element

元素(而不是文本)是 DOM 中最常见的对象。Element 对象的属性和方法如表 C-11 和表 C-12 所示。

表 C-11

属 性 名	说 明	引入的层级
tagName	返回元素的名称，等同于这个节点类型的 Node.nodeName	Level 1

表 C-12

方 法 名	说 明	引入的层级
getAttribute(name)	根据指定的名称获取属性的值	Level 1
getAttributeNS(namespaceURI, localName)	根据本地名称和名称空间 URI 返回 Attr 对象，不用于 HTML DOM	Level 2
getAttributeNode(name)	返回与指定名称关联的 Attr 对象。如果该名称没有关联的属性，就返回 null	Level 1
getElementsByTagName (tagName)	返回带有指定 tagName 的所有子元素的 NodeList，其中子元素的顺序就是它们在 DOM 中的顺序	Level 1
getElementsByTagNameNS (namespaceURI, localName)	返回带有指定的本地名称和名称空间 URI 的所有子元素的 NodeList，不用于 HTML DOM	Level 2
hasAttribute(name)	根据元素是否有指定名称的属性，返回一个布尔值	Level 2
hasAttributeNS(namespaceURI, localName)	根据元素是否有指定本地名称和名称空间 URI 的属性，返回一个布尔值，不用于 HTML DOM	
removeAttribute(name)	删除带有指定名称的属性	Level 1
removeAttributeNS (namespaceURI, localName)	删除带有指定名称和名称空间 URI 的属性，不用于 HTML DOM	Level 2
removeAttributeNode(oldAttr)	删除指定的属性，并返回该属性	Level 1
setAttribute(name, value)	创建并添加一个新属性，或修改已有属性的值。这个值是一个简单字符串	Level 1
setAttributeNS(namespaceURI, qualifiedName, value)	创建并添加一个带有指定名称空间 URI、限定名称和值的新属性	Level 2
setAttributeNode(newAttr)	给元素添加指定属性。如果该属性存在，就用同名的新属性替换已有的属性	Level 1
setAttributeNodeNS(newAttr)	给元素添加指定的属性	Level 2

5. Text

Text 对象表示 Element 或 Attr 对象的文本内容。其方法如表 C-13 所示。

表 C-13

方 法 名	说 明	引入的层级
splitText(indexOffset)	在指定的编辑位置把 Text 节点分解为两个节点。新节点在 DOM 树中是同级节点	Level 1

C2　HTML DOM 对象

为了与 DOM 充分交互，W3C 扩展了 DOM Level 1 和 DOM Level 2 规范，来描述专用于 HTML 文档的对象、属性和方法。

前端开发人员接触到的大多数对象都包含在本节中。

C2.1　HTMLCollection 对象

HTMLCollection 对象是一个节点列表，类似于 NodeList。它没有继承 NodeList，但 HTMLCollection 是活跃的，像 NodeList 一样。在修改文档时，会自动更新 HTMLCollection。HTMLCollection 对象的属性和方法如表 C-14 和表 C-15 所示。

表　C-14

属　性　名	说　　明	引入的层级
length	返回集合中的元素个数	Level 1

表　C-15

方　法　名	说　　明	引入的层级
item(index)	返回带有指定索引的元素。如果索引大于集合长度，就返回 null	Level 1
namedItem(name)	使用名称返回元素。它先搜索有匹配 id 属性值的元素，如果没有找到，就搜索带有匹配 name 属性值的元素	Level 1

C2.2　HTMLDocument 对象

HTMLDocument 对象是 HTML 文档的根，包含文档的所有内容。其属性和方法如表 C-16 和 C-17 所示。

表　C-16

属性名	说　　明	引入的层级
anchors	返回文档中把值赋予其 name 属性的所有<a/>元素的一个 HTML Collection	Level 1
applets	返回文档中包含 applets 的所有<applet/>元素和<object/>元素的一个 HTMLCollection	Level 1
body	返回包含文档内容的元素。根据文档的不同，返回<body/>元素或最外层的<frameset/>元素	Level 1
cookie	返回与文档相关的 cookie，如果没有这个 cookie，就返回一个空字符串	Level 1
domain	返回处理文档的服务器的域名，如果无法识别域名，就返回 null	Level 1
forms	返回文档中所有<form/>元素的一个 HTMLCollection	Level 1
images	返回包含文档中所有<img/>元素的一个 HTMLCollection 对象	Level 1

(续表)

属性名	说　　明	引入的层级
links	返回包含文档中所有<area/>和<a/>元素(有 href 值)的一个 HTMLCollection	Level 1
referrer	返回链接到页面上的 URL。如果用户直接导航到该页面上，就返回一个空字符串	Level 1
title	文档的<head/>元素中由<title/>元素指定的文档标题	Level 1
URL	文档的完整 URL	Level 1

表　C-17

方　法　名	说　　明	引入的层级
close()	关闭文档，再用 open()打开该文档，会迫使页面重新显示	Level 1
getElementById(elementId)	返回带有给定 elementId 的元素，如果找不到该元素，就返回 null。该方法在 DOM Level 2 中被删除，而是添加到 Document 对象上	Level 1
getElementsByName(name)	返回带有指定 name 属性值的元素的一个 HTML Collection	Level 1
open()	打开文档，用于写入	Level 1
write()	把一个文本字符串写入文档	Level 1
writeln()	把一个文本字符串写入文档，后跟一个换行符	Level 1

C2.3　HTML 元素对象

HTML 元素属性显示为各个 HTML 元素对象的属性。在 HTML 4.0 规范中，它们的数据类型由属性类型来确定。

除 HTMLElement 之外，所有 HTML 元素对象都按照字母顺序来介绍。后面的内容并不是 HTML 元素对象类型的完整列表，而只列出了如下元素对象类型：

- HTMLAnchorElement
- HTMLBodyElement
- HTMLButtonElement
- HTMLDivElement
- HTMLFormElement
- HTMLFrameElement
- HTMLFrameSetElement
- HTMLIFrameElement
- HTMLImageElement
- HTMLInputElement
- HTMLOptionElement
- HTMLParagraphElement
- HTMLSelectElement

- HTMLTableCellElement
- HTMLTableElement
- HTMLTableRowElement
- HTMLTableSectionElement
- HTMLTextAreaElement

1. HTMLElement

HTMLElement 是所有 HTML 元素的基本对象，这与 Node 是所有 DOM 节点的基本对象一样。因此，所有 HTML 元素都具有如表 C-18 所示的属性。

表　C-18

属　性　名	说　　明	引入的层级
className	获取或设置元素的 class 属性值	Level 1
id	获取或设置元素的 id 属性值	Level 1

2. HTMLAnchorElement

表示 HTML <a/>元素。该对象的属性和方法如表 C-19 和表 C-20 所示。

表　C-19

属　性　名	说　　明	引入的层级
accessKey	获取或设置 accessKey 属性值	Level 1
href	获取或设置 href 属性值	Level 1
name	获取或设置 name 属性值	Level 1
target	获取或设置 target 属性值	Level 1

表　C-20

方　法　名	说　　明	引入的层级
blur()	从元素中删除键盘焦点	Level 1
focus()	给元素获取键盘焦点	Level 1

3. HTMLBodyElement

表示<body/>元素。该对象的属性如表 C-21 所示。

表　C-21

属　性　名	说　　明	引入的层级
aLink	获取或设置 alink 属性值，建议不要使用此属性	Level 1
background	获取或设置 background 属性值，建议不要使用此属性	Level 1
bgColor	获取或设置 bgColor 属性值，建议不要使用此属性	Level 1

(续表)

属 性 名	说 明	引入的层级
link	获取或设置 link 属性值，建议不要使用此属性	Level 1
text	获取或设置 text 属性值，建议不要使用此属性	Level 1
vLink	获取或设置 vLink 属性值，建议不要使用此属性	Level 1

4. HTMLButtonElement

表示<button/>元素。表 C-22 列出其属性。

表 C-22

属 性 名	说 明	引入的层级
accessKey	获取或设置 accessKey 属性值	Level 1
disabled	获取或设置 disabled 属性值	Level 1
form	获取包含按钮的 HTMLFormElement 对象，如果按钮不在表单中，就返回 null	Level 1
name	获取或设置 name 属性值	Level 1
type	获取或设置 type 属性值	Level 1
value	获取或设置 value 属性值	Level 1

5. HTMLDivElement

表示 元素。表 C-23 列出其属性。

表 C-23

属 性 名	说 明	引入的层级
align	获取或设置 align 属性值，建议不要再使用此属性	Level 1

6. HTMLFormElement

表示<form/>元素。其属性和方法如表 C-24 和表 C-25 所示。

表 C-24

属 性 名	说 明	引入的层级
action	获取或设置 action 属性值	Level 1
elements	返回一个包含表单中所有表单控件元素的 HTMLCollection 对象	Level 1
enctype	获取或设置 enctype 属性值	Level 1
length	返回表单中的表单控件个数	Level 1
method	获取或设置 method 属性值	Level 1
name	获取或设置 name 属性值	Level 1
target	获取或设置 target 属性值	Level 1

表　C-25

方　法　名	说　　明	引入的层级
reset()	把表单包含的所有表单控件元素重置为其默认值	Level 1
submit()	提交表单，不触发 submit 事件	Level 1

7. HTMLFrameElement

表示<frame/>元素。其属性如表 C-26 所示。

表　C-26

属　性　名	说　　明	引入的层级
contentDocument	获取框架的 Document 对象。如果 Document 对象不可用，就返回 null	Level 2
frameBorder	获取或设置 frameBorder 属性值	Level 1
marginHeight	获取或设置 marginHeight 属性值	Level 1
marginWidth	获取或设置 marginWidth 属性值	Level 1
name	获取或设置 name 属性值	Level 1
noResize	获取或设置 noResize 属性值	Level 1
scrolling	获取或设置 scrolling 属性值	Level 1
src	获取或设置 src 属性值	Level 1

8. HTMLFrameSetElement

表示<frameset/>元素。其属性如表 C-27 所示。

表　C-27

属　性　名	说　　明	引入的层级
cols	获取或设置 cols 属性值	Level 1
rows	获取或设置 rows 属性值	Level 1

9. HTMLIFrameElement

表示<iframe/>元素。其属性如表 C-28 所示。

表　C-28

属　性　名	说　　明	引入的层级
align	获取或设置 align 属性值，建议不要再使用此属性	Level 1
contentDocument	获取框架的 Document 对象。如果 Document 对象不存在，就返回 null	Level 2
frameBorder	获取或设置 frameBorder 属性值	Level 1
height	获取或设置 height 属性值	Level 1

(续表)

属 性 名	说 明	引入的层级
marginHeight	获取或设置 marginHeight 属性值	Level 1
marginWidth	获取或设置 marginWidth 属性值	Level 1
name	获取或设置 name 属性值	Level 1
noResize	获取或设置 noResize 属性值	Level 1
scrolling	获取或设置 scrolling 属性值	Level 1
src	获取或设置 src 属性值	Level 1
width	获取或设置 width 属性值	Level 1

10. HTMLImageElement

表示<img/>元素。其属性如表 C-29 所示。

表 C-29

属 性 名	说 明	引入的层级
align	获取或设置 align 属性值，建议不要再使用此属性	Level 1
alt	获取或设置 alt 属性值	Level 1
border	获取或设置 border 属性值	Level 1
height	获取或设置 height 属性值	Level 1
name	获取或设置 name 属性值	Level 1
src	获取或设置 src 属性值	Level 1
width	获取或设置 width 属性值	Level 1

11. HTMLInputElement

表示<input/>元素。其属性和方法如表 C-30 和 C-31 所示。

表 C-30

属 性 名	说 明	引入的层级
accessKey	获取或设置 accessKey 属性值	Level 1
align	获取或设置 align 属性值，建议不要再使用此属性	Level 1
alt	获取或设置 alt 属性值	Level 1
checked	type 是 checkbox 或 radio 时使用。根据是否选中了复选框或单选按钮，返回一个布尔值	Level 1
default-Checked	type 是 checkbox 或 radio 时使用。获取或设置选中的属性。选中其他复选框或单选按钮时，该值不会变化	Level 1
disabled	获取或设置 disabled 属性值	Level 1
form	获取包含<input/>元素的 HTMLFormElement 对象。如果元素不在表单中，就返回 null	Level 1

(续表)

属　性　名	说　　明	引入的层级
maxLength	获取或设置 maxLength 属性值	Level 1
name	获取或设置 name 属性值	Level 1
readOnly	仅在 type 是 text 或 password 时使用。获取或设置 readonly 属性值	Level 1
size	获取或设置 size 属性值	Level 1
src	如果 type 是 image，就获取或设置 src 属性值	Level 1
type	获取或设置 type 属性值	Level 1
value	获取或设置 value 属性值	Level 1

表　C-31

方　法　名	说　　明	引入的层级
blur()	从元素中删除键盘焦点	Level 1
click()	给类型为 button、checkbox、radio、reset 和 submit 的<input/>元素模拟鼠标点击	Level 1
focus()	给元素获取键盘焦点	Level 1
select()	选择类型为 text、password 和 file 的<input/>元素的内容	Level 1

12. HTMLOptionElement

表示<option/>元素。其属性如表 C-32 所示。

表　C-32

属　性　名	说　　明	引入的层级
defaultSelected	获取或设置 selected 属性值。选择<select/>元素中的其他<option/>元素时，这个属性值不变化	Level 1
disabled	获取或设置 disabled 属性值	Level 1
form	获取包含<option/>元素的 HTMLFormElement 对象。如果元素不在表单中，就返回 null	Level 1
index	获取<option/>元素在包含它的<select/>元素中的索引位置，从 0 开始	Level 1
label	获取或设置 label 属性值	Level 1
selected	根据当前是否选中<option/>元素，返回一个布尔值	Level 1
text	获取<option/>元素包含的文本	Level 1
value	获取或设置 value 属性值	Level 1

13. HTMLOptionCollection

HTMLOptionCollection 对象在 DOM Level 2 中引入，它包含一列<option/>元素。其属性和方法如表 C-33 和表 C-34 所示。

表 C-33

属 性 名	说 明	引入的层级
length	获取列表中的<option/>元素个数	Level 1

表 C-34

方 法 名	说 明	引入的层级
item(index)	获取指定索引处的<option/>元素	Level 2
namedItem(name)	根据指定的名称获取<option/>元素，它先尝试查找带有指定 id 的<option/>元素，如果没有找到，就查找带有指定 name 的<option/>元素	Level 2

14. HTMLParagraphElement

表示<p/>元素。其属性如表 C-35 所示。

表 C-35

属 性 名	说 明	引入的层级
align	获取或设置 align 属性值，建议不要再使用该属性	Level 1

15. HTMLSelectElement

表示<select/>元素。其属性和方法如表 C-36 和表 C-37 所示。

表 C-36

属 性 名	说 明	引入的层级
disabled	获取或设置 disabled 属性值	Level 1
form	获取包含<select/>元素的 HTMLFormElement 对象。如果元素不在表单中，就返回 null	Level 1
length	返回<option/>元素的个数	Level 1
multiple	获取或设置 multiple 属性值	Level 1
name	获取或设置 name 属性值	Level 1
options	返回一个包含<option/>元素列表的 HTMLOptionsCollection 对象	Level 1
selectedIndex	返回当前选中的<option/>元素的索引。如果没有选中任何元素，就返回-1，如果选中了多个元素，就返回第一个选中的<option/>元素	Level 1
size	获取或设置 size 属性值	Level 1
type	获取或设置 type 属性值	Level 1
value	获取或设置当前表单控件的值	Level 1

表 C-37

方 法 名	说 明	引入的层级
add(element[, before])	给<select/>元素添加一个<option/>元素。如果 before 是 null，element 就添加到列表的末尾	Level 1
blur()	从元素中删除键盘焦点	Level 1

(续表)

方　法　名	说　　明	引入的层级
focus()	给元素获取键盘焦点	Level 1
remove(index)	删除给定索引处的<option/>元素，如果 index 超出了范围，就什么也不做	Level 1

16. HTMLTableCellElement

表示<td/>元素。其属性如表 C-38 所示。

表　C-38

属　性　名	说　　明	引入的层级
align	获取或设置 align 属性值，建议不要再使用该属性	Level 1
bgColor	获取或设置 bgColor 属性值，建议不要再使用该属性	Level 1
cellIndex	按照 DOM 树中的顺序，行中某单元格的索引	Level 1
colSpan	获取或设置 colSpan 属性值	Level 1
height	获取或设置 height 属性值，建议不要再使用该属性	Level 1
noWrap	获取或设置 noWrap 属性值，建议不要再使用该属性	Level 1
rowSpan	获取或设置 rowSpan 属性值	Level 1
vAlign	获取或设置 vAlign 属性值	Level 1
width	获取或设置 width 属性值，建议不要再使用该属性	Level 1

17. HTMLTableElement

表示<table/>元素。其属性和方法如表 C-39 和表 C-40 所示。

表　C-39

属　性　名	说　　明	引入的层级
align	获取或设置 align 属性值，建议不要再使用该属性	Level 1
bgColor	获取或设置 bgColor 属性值，建议不要再使用该属性	Level 1
border	获取或设置 border 属性值	Level 1
cellPadding	获取或设置 cellPadding 属性值	Level 1
cellSpacing	获取或设置 cellSpacing 属性值	Level 1
rows	返回包含表中所有行的 HTMLCollection	Level 1
tBodies	返回包含表中已定义<tbody/>元素对象的 HTML Collection	Level 1
tFoot	返回表的<tfoot/>元素对象(HTMLTableSectionElement)，如果它不存在，就返回 null	Level 1
tHead	返回表的<thead/>元素对象(HTMLTableSectionElement)，如果它不存在，就返回 null	Level 1
width	获取或设置 width 属性值	Level 1

表 C-40

方 法 名	说 明	引入的层级
createTFoot()	如果<tfoot/>元素不存在，就创建并返回它，如果<tfoot/>元素存在，就返回已有的<tfoot/>元素	Level 1
createTHead()	如果<thead/>元素不存在，就创建并返回它，如果<thead/>元素存在，就返回已有的<thead/>元素	Level 1
deleteRow(index)	删除指定索引的行	Level 1
deleteTFoot()	删除表的页脚(如果有)	Level 1
deleteTHead()	删除表的页眉(如果有)	Level 1
insertRow(index)	在指定索引处插入一个新行，并返回该行。如果 index 是-1 或等于行数，新行就追加到行列表末尾处	Level 1

18. HTMLTableRowElement

表示<tr/>元素。其属性和方法如表 C-41 和表 C-42 所示。

表 C-41

属 性 名	说 明	引入的层级
align	获取或设置 align 属性值，建议不要再使用该属性	Level 1
bgColor	获取或设置 bgColor 属性值，建议不要再使用该属性	Level 1
cells	返回一个包含行中单元格的 HTMLCollection	Level 1
rowIndex	表中某行的索引	Level 1
sectionRowIndex	行相对于它所属的部分(<thead/>、<tfoot/>或<tbody/>)的索引	Level 1
vAlign	获取或设置 vAlign 属性值	Level 1

表 C-42

方 法 名	说 明	引入的层级
deleteCell(index)	删除指定索引处的单元格	Level 1
insertCell(index)	插入并返回一个空的<td/>元素。如果 index 是-1 或等于行中的单元格数，新单元格就追加到列表末尾	Level 1

19. HTMLTableSectionElement

表示<thead/>、<tbody/>和<tfoot/>元素。其属性和方法如表 C-43 和表 C-44 所示。

表 C-43

属 性 名	说 明	引入的层级
align	获取或设置 align 属性值，建议不要再使用该属性	Level 1
cells	返回一个包含部分中的行的 HTMLCollection	Level 1
vAlign	获取或设置 vAlign 属性值	Level 1

表　C-44

方　法　名	说　　明	引入的层级
deleteRow(index)	删除相对于部分的指定索引处的行	Level 1
insertRow(index)	在相对于部分的指定索引处插入一个新行，并返回该行。如果 index 是-1 或等于行数，新行就追加到行列表的末尾	Level 1

20. HTMLTextAreaElement

表示<textarea/>元素。其方法和属性如表 C-45 和表 C-46 所示。

表　C-45

属　性　名	说　　明	引入的层级
accessKey	获取或设置 accessKey 属性值	Level 1
cols	获取或设置 cols 属性值	Level 1
defaultValue	获取或设置元素的内容。内容变化时，这个值不变	Level 1
disabled	获取或设置 disabled 属性值	Level 1
form	获取包含<textarea/>元素的 HTMLFormElement 对象。如果元素不在表单中，就返回 null	Level 1
name	获取或设置 name 属性值	Level 1
readOnly	仅在 type 是 text 或 password 时使用。获取或设置 readonly 属性值	Level 1
rows	获取或设置 rows 属性值	Level 1
type	获取或设置 type 属性值，总是设置为 textarea	Level 1
value	获取或设置元素的当前值	Level 1

表　C-46

方　法　名	说　　明	引入的层级
blur()	从元素中删除键盘焦点	Level 1
focus()	给元素获取键盘焦点	Level 1
select()	选择元素的内容	Level 1

C3　DOM 事件模型和对象

DOM 事件模型在 DOM Level 2 中引入，它描述了一个事件系统，在该事件系统中，每个事件都有一个事件目标。事件到达其事件目标时，就会触发在事件目标上为该事件注册的所有事件处理程序。下面的对象由 DOM 事件模型描述。

C3.1　EventTarget

EventTarget 对象由 DOM 中的所有 HTMLElement 对象继承。这个对象提供了在事件

目标上注册和删除事件处理程序的方式。其方法如表 C-47 所示。

表 C-47

方 法 名	说 明
addEventListener(type, listener, useCapture)	在元素上注册一个事件处理程序。type 是要监听的事件类型，listener 是触发事件时调用的 JavaScript 函数，useCapture 确定是捕获事件，还是使事件冒泡
removeEventListener(type, listener, useCapture)	从元素上删除监听程序

C3.2 Event

触发事件时，会把 Event 对象传送给指定的事件处理程序。这个对象包含有关事件的上下文信息。其方法和属性如表 C-48 和表 C-49 所示。

表 C-48

属 性 名	说 明	引入的层级
bubbles	指定事件是否是一个冒泡事件	Level 2
cancelable	指定事件可否禁止执行其默认动作	Level 2
currentTarget	指定当前处理其监听程序的 EventTarget	Level 2
target	指定最初触发事件的 EventTarget 对象	Level 2
timeStamp	指定触发事件的时间(以毫秒为单位)	Level 2
type	事件名(不带 on 前缀的事件名)	Level 2

表 C-49

方 法 名	说 明	引入的层级
preventDefault()	只有事件是可取消时，才能取消事件，禁止执行其默认动作	Level 2
stopPropagation()	阻止进一步传播事件	Level 2

C3.3 MouseEvent

MouseEvent 对象提供了与鼠标事件相关的特定信息，MouseEvent 对象不仅包含下面的属性，还包含 Event 对象的属性和方法。MouseEvent 的属性如表 C-51 所示。

有效的鼠标事件如表 C-50 所示。

表 C-50

事 件 名	说 明
click	在一个元素上点击鼠标时发生。click 定义为在同一个屏幕位置上的 mousedown 和 mouseup
mousedown	在一个元素上按下鼠标按钮时发生
mouseup	在一个元素上释放鼠标按钮时发生

(续表)

事　件　名	说　　明
mouseover	鼠标指针在元素上移动时发生
mouseout	鼠标指针离开元素时发生

表　C-51

属　性　名	说　　明	引入的层级
altKey	返回一个布尔值，指定事件触发过程中是否按下了 Alt 键	Level 2
button	指定按下了哪个鼠标按钮。数字 0 表示左键，1 表示中键，2 表示右键。左手配置的鼠标，其数字与上述相反（2 表示左键，1 表示中键，0 表示右键）	Level 2
clientX	相对于客户区的水平坐标	Level 2
clientY	相对于客户区的垂直坐标	Level 2
ctrlKey	返回一个布尔值，指定事件触发过程中是否按下了 Ctrl 键	Level 2
relatedTarget	表示次级 EventTarget。这个属性当前与 mouseover 事件一起使用，表示鼠标指针退出的 EventTarget，这个属性与 mouseout 一起使用，表示指针进入哪个 EventTarget	Level 2
screenX	相对于屏幕的水平坐标	Level 2
screenY	相对于屏幕的垂直坐标	Level 2
shiftKey	返回一个布尔值，指定事件触发过程中是否按下了 Shift 键	Level 2

C4　其他事件

表 C-52 至表 C-56 分别描述了客户端 JavaScript 上可用的鼠标事件、键盘事件、HTML 控件事件、窗口事件和其他。

表　C-52

事　件　名	说　　明
click	用户点击 HTML 控件时发生
dblclick	用户双击 HTML 控件时发生
mousedown	用户按下鼠标按钮时发生
mousemove	用户移动鼠标指针时发生
mouseout	用户将鼠标指针移到 HTML 控件的外部时发生
mouseover	用户将鼠标指针移到 HTML 控件上时发生
mouseup	用户释放鼠标按钮时发生

表 C-53

事 件 名	说 明
keydown	用户按下键盘上的一个键时发生
keypress	用户按下键盘上的一个键时发生，这个事件一直被触发，直到用户释放该键为止
keyup	用户释放被按下的键时发生

表 C-54

事 件 名	说 明
blur	HTML 控件失去焦点时发生
change	HTML 控件失去焦点，且其值改变时发生
focus	在 HTML 控件上设置焦点时发生
reset	用户重置表单时发生
select	用户选择 HTML 控件中的文本时发生
submit	用户提交表单时发生

表 C-55

事 件 名	说 明
load	加载完窗口时发生
resize	用户重置窗口大小时发生
unload	用户退出文档时执行 JavaScript 代码

表 C-56

事 件 名	说 明
abort	用户中止图像的加载时发生
error	加载页面出错时发生

附录 D

Latin-1 字符集

本附录介绍 Latin-1 字符集，以及十进制和十六进制格式的字符码。如第 2 章所述，转义序列\xNN(其中 NN 是 Latin-1 字符集中的一个十六进制字符码)可用于表示不能直接在 JavaScript 中输入的字符。

十进制字符码	十六进制字符码	符　号
32	20	空格
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2

(续表)

十进制字符码	十六进制字符码	符　号
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S

(续表)

十进制字符码	十六进制字符码	符　号
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t

(续表)

十进制字符码	十六进制字符码	符　号
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
160	A0	非中断空格
161	A1	¡
162	A2	¢
163	A3	£
164	A4	¤
165	A5	¥
166	A6	¦
167	A7	§
168	A8	¨
169	A9	©
170	AA	ª
171	AB	«
172	AC	¬
173	AD	软连字符
174	AE	®
175	AF	¯
176	B0	°
177	B1	±
178	B2	²
179	B3	³
180	B4	´
181	B5	µ
182	B6	¶

(续表)

十进制字符码	十六进制字符码	符　号
183	B7	·
184	B8	¸
185	B9	¹
186	BA	º
187	BB	»
188	BC	¼
189	BD	½
190	BE	¾
191	BF	¿
192	C0	À
193	C1	Á
194	C2	Â
195	C3	Ã
196	C4	Ä
197	C5	Å
198	C6	Æ
199	C7	Ç
200	C8	È
201	C9	É
202	CA	Ê
203	CB	Ë
204	CC	Ì
205	CD	Í
206	CE	Î
207	CF	Ï
208	D0	Ð
209	D1	Ñ
210	D2	Ò
211	D3	Ó
212	D4	Ô
213	D5	Õ
214	D6	Ö
215	D7	×

(续表)

十进制字符码	十六进制字符码	符　号
216	D8	Ø
217	D9	Ù
218	DA	Ú
219	DB	Û
220	DC	Ü
221	DD	Ý
222	DE	Þ
223	DF	ß
224	E0	à
225	E1	á
226	E2	â
227	E3	ã
228	E4	ä
229	E5	å
230	E6	æ
231	E7	ç
232	E8	è
233	E9	é
234	EA	ê
235	EB	ë
236	EC	ì
237	ED	í
238	EE	î
239	EF	ï
240	F0	ð
241	F1	ñ
242	F2	ò
243	F3	ó
244	F4	ô
245	F5	õ
246	F6	ö
247	F7	÷
248	F8	ø

(续表)

十进制字符码	十六进制字符码	符　号
249	F9	ù
250	FA	ú
251	FB	û
252	FC	ü
253	FD	ý
254	FE	þ
255	FF	ÿ

[General Information]
书名=JavaScript入门经典
作者=（美）威尔顿，（美）麦克匹克著
页数=653
出版社=北京市：清华大学出版社
出版日期=2011.02
SS号=12850445
DX号=000008060383
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000008060383&d=5E0E415CBF921DC65C3BFCC66958EF02