本书配带的光盘堪称信息金矿：

- 29个附赠章节
- 300多个可运行脚本

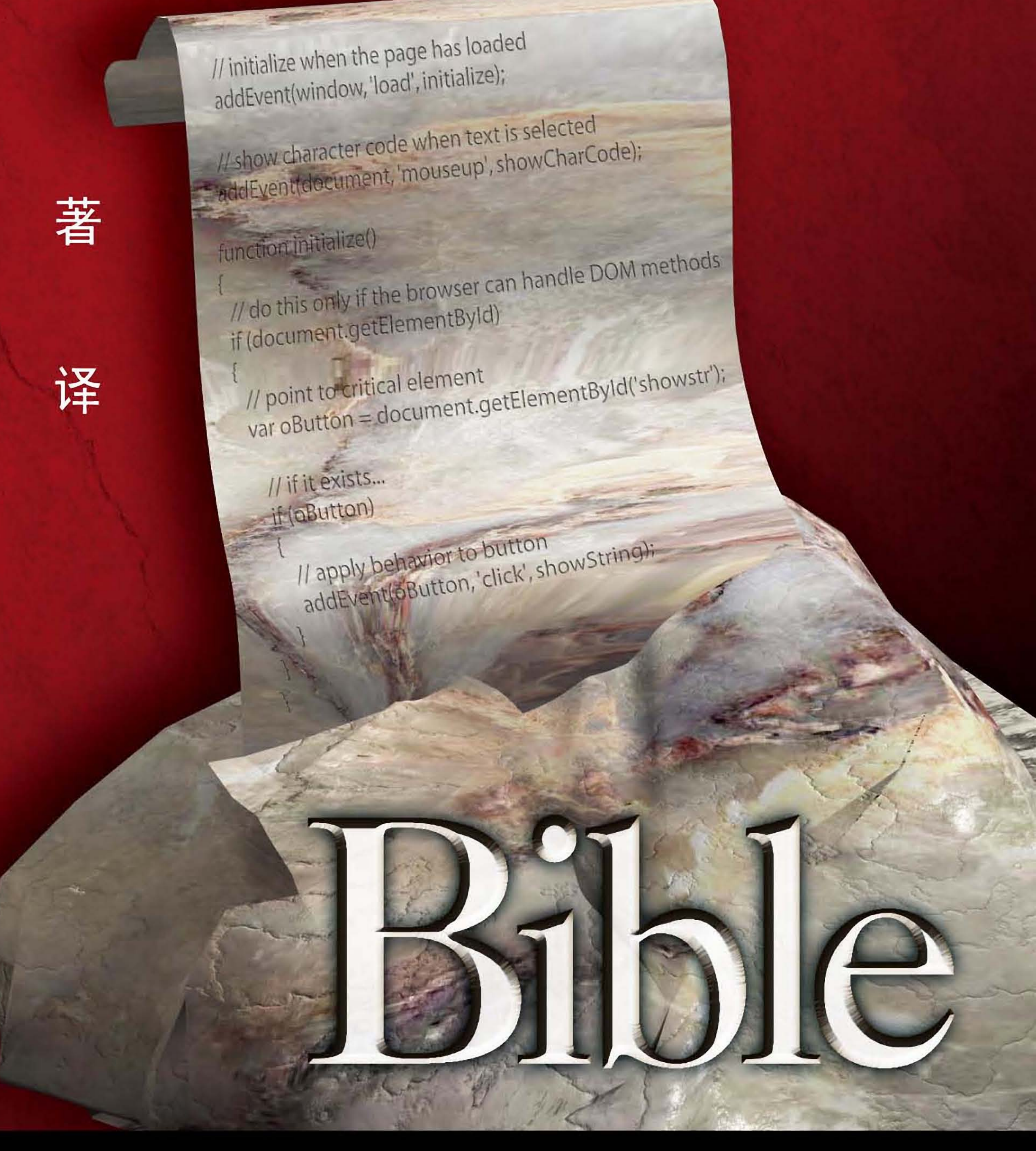
JavaScript Bible, Seventh Edition

JavaScript宝典

(第7版)

[美] Danny Goodman
Michael Morrison
Paul Novitski
Tia Gustaff Rayl 著

杨岳湘 普杰 高宇辉 译



- 创建富有吸引力的交互网站
- 为当今浏览器创建精彩纷呈的动态内容
- 深入理解“文档对象模型”概念

本书为您铺就成功路！

清华大学出版社

JavaScript 宝典

(第 7 版)

[美] Danny Goodman
Michael Morrison
Paul Novitski
Tia Gustaff Rayl 著

杨岳湘 普 杰 高宇辉 译

清华大学出版社

北 京

PDG

更多资源请访问稀酷客(www.ckook.com)

Danny Goodman, Michael Morrison, et al.
JavaScript Bible, Seventh Edition
EISBN：978-0-470-52691-0
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：01-2011-7135

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

图书在版编目(CIP)数据

JavaScript 宝典(第 7 版)/(美)古德曼(Goodman, D.)等著；杨岳湘，普杰，高宇辉 译. —北京：清华大学出版社，2013.1

书名原文：JavaScript Bible, Seventh Edition

ISBN 978-7-302-30322-0

Ⅰ. ① J… Ⅱ. ①古… ②杨… ③普… ④高… Ⅲ. ①JAVA 语言—程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2012)第 238348 号

责任编辑：王 军 韩宏志
装帧设计：牛艳敏
责任校对：成凤进
责任印制：王静怡

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
经 销：全国新华书店
开 本：185mm×260mm **印 张**：64.5 **字 数**：1657 千字
(附光盘 1 张)
版 次：2013 年 1 月第 1 版 **印 次**：2013 年 1 月第1 次印刷
印 数：1～3000
定 价：128.00 元

产品编号：036204-01

更多资源请访问稀酷客(www.ckook.com)

作者简介

Danny Goodman撰写了许多颇受欢迎的畅销书，包括*The Complete HyperCard Handbook*、*Danny Goodman's AppleScript Handbook*、*Dynamic HTML: The Definitive Reference*和*JavaScript & DHTML Cookbook*。他是声誉卓著的计算机脚本语言权威人士和专家级教师。他的写作风格和教育方式会继续为他赢得全球各地读者和教师的赞美。

Michael Morrison是一位作家、开发人员、玩具发明家，还是多部Java、C++、Web脚本、XML、游戏开发和移动设备等图书的作者，Michael撰写的一些著名图书有*Faster Smarter HTML and XML*、*Teach Yourself HTML & CSS in 24 Hours*和*Beginning Game Programming*。Michael还是Stalefish实验室(www.stalefishlabs.com)的创始人，这是一家专门开发非常游戏、玩具和互动产品的娱乐公司。

Paul Novitski自从1981年就开始作为一位自由职业的程序员编写软件。他曾经自学BASIC，来编写机器语言反汇编器，所以他可以砍掉一些Wang's OIS微码。他自从90年代后期开始专注于互联网编程。他的公司Juniper Webcraft开发的HTML-strict网站便于访问，使用语义标记、分隔的开发层和直观的用户界面。在生活中，他享受着甜美的安比拉琴音乐和抚养双胞胎儿子的快乐。

Tia Gustaff Rayl是一位数据库和Web技术的开发和培训顾问。最近她发布了XHTML、CSS、JavaScript和SQL的课件。她在获得佛罗里达大学的英语和教育博士学位后开始了其软件生涯。与大多数这个领域的新手一样，她最初的计算任务是维护软件。她在软件业呆的时间很长，完全了解了软件的整个生命周期、应用程序和数据库开发、项目管理、PC和大型机环境的培训。90年代中期，她开始开发早期的支持Web的数据库应用程序，并在其代码中添加JavaScript。她继续承接开发项目，以磨练自己的编程技巧。她梦想着可以利用业余时间与丈夫及两条狗一起周游世界。

更多资源请访问稀酷客(www.ckook.com)

技术编辑简介

Benjamin Schupak获得了计算机科学的硕士学位，在大公司和美国联邦政府部门的专业编程经验超过11年。他居住在纽约metro地区，喜欢外出旅游。

致　　谢

十分荣幸有机会撰写这本巨作，在此郑重感谢Carol Long of Wiley的编辑John Sleeva、Waterside Productions的Carole Jelen、我在Juniper Webcraft的搭档Julian Hall以及始终不渝地支持我的爱妻Tanya Wright，编写本书的周期很长，他们一直在默默地忍受着，使我不受打扰。Danny Goodman及其以前的合作者在前几个版本中进行的研究和阐述非常了不起，没有这个坚实的基础，我就不可能把这么多知识向前推进一步。Tia Gustaff Rayl睿智而幽默，就像长了翅膀，能及时飞到我身旁给予我有力的帮助。

—Paul Novitski

我在编写本书的过程中得到了许多人的帮助，他们对我的工作发表评论，并鼓励我继续下去。没有丈夫Edward的大力支持，我就不可能完成本书。我永远爱他，感谢上帝把他带进我的生活。亲友们大度宽容，有耐心，他们能够容忍我在接电话和招呼来客时采用“嗨，我爱你，现在不要跟我说话”的方式。我的丈夫和亲朋好友为本书牺牲得最多。谢谢你们。还要感谢Paul与我的合作，感谢编辑John Sleeva，最后还要感谢Rebecca Anderson、Maraya Cornell和Miss Bigelow给予我的帮助。

—Tia Gustaff Rayl

更多资源请访问稀酷客(www.ckook.com)

前　言

在新闻组中，经常可以看到各个层次的脚本编写人员提出的疑问、遇到的困难和挑战，所以笔者根据15年来在编写日常JavaScript脚本，以及主持新闻组的过程中积累的知识和经验，编写了本书的第7版，希望读者能通过本书，避免笔者在脚本浏览器的多个版本中遇到的困难和挫折。

本书的最早版本主要介绍当时的主流浏览器Netscape Navigator，但随着时间的推移，浏览器的市场有了许多变化。Microsoft推出的Internet Explorer多年来一直在此领域占据领头羊的位置，而近年来，用户开始在计算机中使用其他支持业界标准的浏览器。所以，内容开发人员面临着一个极富挑战的任务：他们设计的脚本必须能在标准浏览器和专有环境中正常运行。本书不仅指出了标准浏览器和专有环境中有区别的地方，还说明了如何编写能够适应不同情况的脚本，使它们能在更广泛的浏览器中访问网站或Web应用程序。通过本书的学习，读者将能设计和编写出可在多个浏览器上运行的优秀脚本。本书主要介绍了业界标准，还涉及一些专有功能，确保编写出来的脚本能成功运行在当今和将来的浏览器上。

0.1　本书的编排方式与特点

第7版与此前的三个版本一样，也包含了极其丰富的信息，以至于不能完全包含在一本图书中。本书中仅包含JavaScript基础知识和常见引用，高级内容则在配书光盘上。下面是本书的结构。

第Ⅰ部分　JavaScript入门

第1章比较了JavaScript与Java，并阐述了JavaScript在万维网中的作用。自从JavaScript问世以来，Web浏览器和脚本都发生了巨大变化，因此脚本编写者必须在标准飞速发展的同时，为单平台和跨平台浏览器的用户开发应用程序。第2章讨论了脚本的编写策略，第3章简要介绍一些用来编写页面和脚本的工具，第4章深入探讨如何在各种浏览器上使用JavaScript。

第Ⅱ部分　JavaScript教程

第Ⅱ部分是为JavaScript初学者准备的，共有10章，循序渐进地讲解了浏览器原理、基本编程技巧和实际的浏览器脚本，并重点强调当今多数脚本浏览器支持的业界标准。每章后的练习有助于巩固理解刚学到的知识，并应用这些知识，光盘上的附录D提供了习题答案。阅读了

PDG

更多资源请访问稀酷客(www.ckook.com)

本部分后，读者就能快速掌握编写简单脚本页面的基础知识，并更加方便地理解本书其他部分的深入论述和各种示例。

第III部分　JavaScript 核心语言参考

第III部分是 JavaScript 核心语言的参考资料。其中每个对象和对象特征都列出了支持它的浏览器版本。

第IV部分　文档对象参考

第IV部分最长，深入探讨了目前浏览器中的文档对象模型，包括现代 Ajax 应用程序使用的对象。与第III部分一样，这些 DOM 章节中的每个对象和对象特征都列出了支持它的浏览器版本。其中第 26 章的内容最多，第IV部分的其余章节大多引用了这一章的内容。

第V部分　附录

附录 A 提供了 JavaScript 和浏览器对象快速参考。

第VI部分　文档对象参考(续)

从这部分开始的内容都放在配书光盘上，第VI部分继续第IV部分的文档对象参考，包含 13 章。注意，附赠的章节均为英文内容。

第VII部分　JavaScript 编程的更多内容

第 46 章~第 51 章讨论了高级 JavaScript 编程技术，包括数据输入的验证、调试和安全问题。

第VIII部分　应用程序

本书最后 10 章给出了一些示例应用程序，包括日历、智力游戏等。

第IX部分　附录(续)

最后 3 个附录也提供了一些有用信息。附录 C 是 JavaScript 保留字列表；附录 D 是第 II 部分教程练习的答案；附录 E 是 Internet 资源。

配书光盘

配书光盘是一座信息金矿。这个版本包括 29 个附加章节。

光盘中的 Listings 文件夹是另一个宝库，其中包含 300 多个现成的 HTML 文档，是第III和第IV部分中大多数文档对象模型和 JavaScript 术语的使用示例；该文件夹中还包括所有附赠章节的例子。所有支持 JavaScript 的浏览器都可以运行这些例子，但一定要通过 Listings 文件夹中的 index.html 页面来运行这些清单示例。该文件夹还有一些没有指明具体使用环境的小代码段示例，它们可能非常简单，但有助于理解成熟的 HTML 文档如何运用某些概念。最好手工输入本书第 II 部分教程中的脚本代码，以养成在文档中输入脚本的习惯，因为即使仅仅模仿本书中的程序清单，也可以学到许多东西。

本书第 4 章的程序清单文件为 evaluator.html。在交互平台 The Evaluator 上输入第III部分和

更多资源请访问稀酷客(www.ckook.com)

第Ⅳ部分的许多代码段示例，The Evaluator 会立即显示执行这些代码的结果，这样可以尝试使用对象模型或语言特征，并很快学会这些特征的工作方式——这是本书的特色之一。

附录 A 的快速参考在光盘上使用 PDF 格式，如果需要，可以打印出来，以便随时查阅。

0.2 学习 JavaScript 的必要条件

本书不要求读者拥有丰富的编程经验，但如果读者有使用 HTML 创建网页的经验，就更容易理解 JavaScript 与页面上常用元素的交互方式。有时，脚本的编写需要修改 HTML 标记，如果很熟悉那些标记，就可以轻松掌握 JavaScript 的增强功能。

学习 JavaScript 并不需要了解服务器脚本编程知识，也不必了解如何通过表单向服务器提交信息。本书的重点是客户端脚本编程，包含 JavaScript 脚本的 HTML 页面在加载到浏览器中后，它的操作就与服务器完全无关。但是，即使没有 JavaScript，公共 Web 页面也应能正常运行，所以在执行查找或修改页面内容等动态功能时，应能与服务器端脚本交互操作。在建立了基本的 HTML 页面后，就可以添加 JavaScript，使页面执行更加便捷，能给人带来更多乐趣。尽管学习 JavaScript 并不需要了解服务器端脚本编程，但对于重要的 Web 工作，除了 JavaScript 之外，还是应学习某种服务器端脚本语言，例如 PHP，或者请服务器端程序员来补足客户端脚本功能。

学习 JavaScript 需要熟悉当前 HTML 标准的基本术语，还应熟悉最新的文档标记标准，比如 XHTML 和层叠样式表(Cascading Style Sheets，CSS)。在 Web 上搜索这些术语，可以找到许多相关主题的教程。

0.2.1 以前从未编过程序的读者

JavaScript 并不是世界上最容易学习的语言，但它比纯编程语言(比如 Java 或 C)简单得多。JavaScript 并不用于开发成熟的、功能单一的应用程序，比如在商店中购物的高效率程序，而可以编写简短的 JavaScript 代码段来完成重要任务。每个脚本浏览器都内置了 JavaScript 解释器，来自动完成大量技术工作。

其实，编程只不过是编写一系列指令，让计算机按指令操作。人们无时无刻不在执行指令，以至于我们都意识不到这一点。比如去朋友家，就是执行一系列指令：走过 3 个街区，向左拐，再向右拐，就到达了目的地。在这些指令中，有时需要作出决策：如果交通灯为红色，就停步；如果是绿色，就行进；如果是黄色，就把汽车加速器调至最低档。有时，一些操作必须重复几次，比如，不停地在某街区绕圈，直到找到停车位为止。计算机程序不仅包含主要的执行步骤，还预见到到达目的地的过程中，要做什么决策或重复执行什么操作，比如，如何处理各种交通灯的情况，有人抢先占了停车位该怎么办。

学习编程时，首先要熟悉程序语言将文字和数字组织成指令的规则。这些规则叫做语法，和生活中的语言一样。假如与计算机通信时，不使用计算机能理解的语言，计算机就不能领会你的意图。和某人说话时，如果句子的语法有错误，对方还可能理解，但计算机却做不到这一点。只要语法不正确(即使它在可纠正的语言范围内)，计算机都会报告语法错误。

即使是经验丰富的程序员，也会犯下语法错误。最好记录下自己所犯的语法错误，并重写

PDG
更多资源请访问稀酷客(www.ckook.com)

语句，来更正错误。这样就可以丰富自己的语言知识。

0.2.2 以前编写过少量程序的读者

如果读者以前编过程序，但使用的是 BASIC 等过程式语言，即使对语法有精准的认识，也可能给学习 JavaScript 带来障碍，而不会有所帮助。因为 JavaScript 是面向对象的，而过程式语言编写程序的基本概念与面向对象编程有本质区别。这也与 JavaScript 脚本完成的典型任务(在网页内执行某种操作，来响应用户的动作)有一定的关系。

0.2.3 以前是 C 程序员的读者

在过程式程序中，程序员一般负责处理屏幕和后台上的每个操作。程序首次运行时，要使用大量代码建立可视化环境，使屏幕显示几个文本输入域或可单击按钮。接着，如果用户单击了某个按钮，程序就必须提取单击处的坐标，并和屏幕上的所有按钮坐标比较，最后执行单击该处对应的指令。

面向对象编程和这个过程刚好相反。面向对象编程把按钮看成一个有形的对象，而对象有属性，如标签、大小、对齐方式等。对象还可能包含脚本。如果用户执行了某个操作，系统软件和浏览器就一起把消息发送给对象，来触发脚本。比如，用户单击了一个文本输入域，系统/浏览器就告诉该域，某人单击了它(使该域具有焦点)，然后该域就触发脚本，此时就该脚本大显身手了。连接到该域上的脚本包含一些指令，在用户激活该域后执行这些指令。还有一套指令可以控制当用户键入一项、按下 Tab 键、在域外单击时要执行的操作，从而改变该域的内容。

一些脚本的结构可能是过程式的：包含按顺序执行的一系列简短指令。但当处理表单元素的数据时，这些指令使用了 JavaScript 的面向对象特性。在面向对象编程中，这个表单是一个对象；每个单选按钮或文本框也是对象。脚本通过操作这些对象的属性来执行任务。

从过程式编程转到面向对象编程是最困难的。刚接触面向对象编程这个概念时，可能不会立即领会它。但明白其内涵后，很多事情就变得一目了然了。从那时起，面向对象几乎就成为编程的唯一方法。

JavaScript 借用了 Java 的语法，而 Java 派生于 C 和 C++，所以 JavaScript 与 C 有许多相同的语法，熟悉 C 的程序员会觉得得心应手。JavaScript 的操作符、条件结构和重复循环都采用 C 的风格，但 JavaScript 不像 C 那样注重数据类型，JavaScript 中变量的数据类型是可变的。

以前使用 C 语言的程序员将很熟悉 JavaScript 的许多语法，所以可以集中精力学习全新的文档对象模型。但仍需要具有良好的 HTML 基础，以发挥 JavaScript 的特长。

0.2.4 以前是 Java 程序员的读者

尽管 JavaScript 和 Java 名称相似，但这两种语言的相同点甚少：只有循环和条件结构、类似 C 的“句点”对象引用、用来分组语句的花括号、几个关键字和其他几个特性是相同的。它们在变量声明方面的差异很大，因为 JavaScript 是弱类型化语言。JavaScript 变量可以在一个语句中包含整数值，在下一个语句中包含字符串(这不能算是良好的程序风格)。Java 中的方法，在 JavaScript 中叫做方法(是预定义对象的方法)或函数(脚本开发者定义的动作)。JavaScript 方法和函数不必预先声明数据类型，就可以返回任意类型的值。

更多资源请访问稀酷客(www.ckook.com)

在编写 JavaScript 时，不能使用某些 Java 概念，主要是一些面向对象的概念，如类、继承、实例化和消息传递，但这些在编写脚本时都不是问题。同时，JavaScript 的设计者知道，一些用户已经养成了一些根深蒂固的习惯。比如，JavaScript 不要求在每个语句行后加分号，但假如在 JavaScript 源代码中键入分号，JavaScript 解释器也认可。

0.2.5　以前写过脚本(或宏)的读者

用其他工具编写脚本或使用高效程序编写宏的经验，非常有助于掌握许多 JavaScript 概念。最重要的 JavaScript 概念是组合少量语句，针对一些数据执行特定任务。比如，如果另一台计算机上的公司财务报表包含一些常见图形，就可以在 Microsoft Excel 中编写一个宏，对这些图形进行数据转换。把该宏放在 Macro 菜单中，则导入一组新的图形时，选择该菜单项，就可运行该宏。

一些现代编程环境，如 Visual Basic，在某些方面与脚本环境类似。它们给程序员提供了一个界面构建器，显示与用户交互的屏幕对象。程序员需要编写一些代码，在用户与那些屏幕对象交互时，就执行这些代码。使用 JavaScript 编写脚本的工作就采用这种模式。实际上，那些环境在另一个方面也类似于脚本浏览器环境：它们提供了一组有限的预定义对象，这些对象具有确定的属性和行为集合。这种可预见性更便于学习整个环境和设计应用程序。

0.3　格式和命名约定

本书的脚本清单以等宽字体显示，使它们与其他正文区分开。由于本书的页宽有限，脚本清单常常会断行，此时，脚本的剩余部分就显示在下一行，与清单的左边缘齐平，就像在打开自动换行功能的文本编辑器中一样。在文档中输入脚本清单时，假如这些断行引发了问题，最好在配书光盘上找到相应的清单，看看脚本应该是什么样子。

在本书的第III部分，在阅读对象模型或者语言功能(它们需要某个浏览器的指定最低版本)之前，不可能编写出多个页面。在需要特定的浏览器或浏览器版本时，为了更便于在文中阅读，多数浏览器引用由缩写和版本号组成。比如：WinIE5 表示运行于 Windows 系统的 Internet Explorer 5；NN4 表示运行于任何操作系统的 Netscape Navigator 4：Moz 表示现代 Mozilla 浏览器(它派生了 Firefox、Netscape 6 和以后版本，以及 Camino)；Safari 表示 Apple 用于 Mac OS X 的浏览器。如果浏览器的某个版本引入了一个功能，而且在后续版本中都支持，就在这个版本号后面加一个“+”符号。例如，标记为 WinIE5.5+的功能，表示该功能至少需要 Windows 环境的 Internet Explorer 5.5，WinIE8 和将来的 WinIE 版本也支持该功能。如果在现代浏览器的第 1 版中实现了某功能，就在这个浏览器系列名称的后面加上加号(+)，比如 Moz+表示所有基于 Mozilla 的浏览器。有时，某功能或一些特殊行为只应用于一个浏览器。例如，某功能标记为 NN4，表示只是在 Netscape Navigator 4.x 中有这个功能。减号(例如，WinIE-)表示浏览器不支持当前讨论的内容.

本版书中的 HTML 标记格式符合 HTML5 的编码约定，也遵循许多 XHTML 标准，例如标记和特性名都使用小写形式。

“注意”、“提示”、“警告”、“交叉引用”这几个图标在本书中随处可见，用于标记重点内容，或者告诉你在哪里可以找到更详细的信息。

更多资源请访问稀酷客(www.ckook.com)

目　录

第 I 部分　JavaScript 入门

第 1 章　JavaScript 在万维网和其他领域所起的作用……3
1.1　Web 流量的竞争……4
1.2　其他 Web 技术……4
1.2.1　超文本标记语言(HTML 和 XHTML)……5
1.2.2　CSS……7
1.2.3　服务器编程……7
1.2.4　辅助程序和插件程序……8
1.3　JavaScript 是一门综合性语言……9
1.3.1　LiveScript 蜕变成 JavaScript……10
1.3.2　微软的 JavaScript 版本……10
1.3.3　JavaScript 版本……10
1.3.4　核心语言标准 ECMAScript……11
1.4　JavaScript：灵活易用的工具……12
第 2 章　脚本开发策略……13
2.1　浏览器的竞争……13
2.2　相互包容……14
2.3　当今存在的兼容性问题……14
2.3.1　将核心 JavaScript 语言从文档对象中独立出来……15
2.3.2　核心语言标准……15
2.3.3　文档对象模型……16
2.3.4　通过标记打下良好的基础……17
2.3.5　层叠样式表……17
2.3.6　标准兼容模式(DOCTYPE 转换)……18
2.3.7　动态 HTML 和定位……19
2.4　开发脚本编写策略……19
2.4.1　功能降低和渐进增强……19
2.4.2　开发层的分离……20
2.4.3　延伸阅读……21
第 3 章　选择和使用工具……23
3.1　软件工具……23
3.1.1　选择文本编辑器……23
3.1.2　选择浏览器……24
3.2　建立编写环境……24
3.2.1　Windows……25
3.2.2　Mac OS X……25
3.2.3　重载问题……26
3.3　验证……26
3.4　创建第一个脚本……27
3.4.1　第一步：静态 HTML……27
3.4.2　第二步：连接 JavaScript……28
3.4.3　第三步：用 CSS 指定样式……29
第 4 章　JavaScript基础……31
4.1　合并 JavaScript 和 HTML……31
4.1.1　<script>标记……31
4.1.2　旧式内联 JavaScript……35
4.1.3　容纳不支持 JavaScript 的用户代理……35
4.1.4　隐藏脚本……39
4.1.5　给不同的浏览器编写脚本……40
4.2　兼容性设计……44

更多资源请访问稀酷客(www.ckook.com)

4.2.1 处理 beta 版浏览器 ······ 44
4.2.2 参考章节中的兼容性等级 ······ 45
4.3 资深程序员的语言基础 ······ 46

第 II 部分 JavaScript 教程

第 5 章 第一个 JavaScript 脚本 ······ 53
5.1 第一个脚本的功能 ······ 53
5.2 输入第一个脚本 ······ 54
5.2.1 第一步：HTML 文档 ······ 54
5.2.2 第二步：添加 JavaScript ······ 57
5.2.3 第三步：添加样式 ······ 63
5.3 进行改动 ······ 65
5.4 习题 ······ 65
第 6 章 浏览器对象和文档对象 ······ 67
6.1 脚本运行初步 ······ 67
6.2 使用 JavaScript 的场合 ······ 68
6.3 文档对象模型 ······ 69
6.3.1 HTML 结构和 DOM ······ 69
6.3.2 浏览器窗口中的 DOM ······ 70
6.4 文档的载入 ······ 71
6.4.1 简单文档 ······ 72
6.4.2 添加段落元素 ······ 72
6.4.3 添加段落文本 ······ 72
6.4.4 生成新元素 ······ 73
6.5 对象引用 ······ 73
6.5.1 对象命名 ······ 74
6.5.2 引用特定对象 ······ 74
6.6 节点术语 ······ 75
6.6.1 节点 ······ 75
6.6.2 父子节点 ······ 76
6.7 对象的定义 ······ 76
6.7.1 属性 ······ 76
6.7.2 方法 ······ 77
6.7.3 事件 ······ 79
6.8 习题 ······ 80
第 7 章 脚本和 HTML 文档 ······ 83
7.1 把脚本连接到文档上 ······ 83
7.1.1 script 标记的位置 ······ 84
7.1.2 非 JavaScript 的浏览器和 XHTML ······ 85
7.2 JavaScript 语句 ······ 86
7.3 脚本语句的执行时间 ······ 87
7.3.1 文档载入时即刻执行 ······ 87
7.3.2 延时脚本 ······ 88
7.4 查找脚本错误 ······ 90
7.5 脚本和编程 ······ 91
7.6 习题 ······ 92
第 8 章 程序设计基础(一) ······ 93
8.1 JavaScript 语言 ······ 93
8.2 处理信息 ······ 93
8.3 变量 ······ 94
8.3.1 创建变量 ······ 94
8.3.2 变量的命名 ······ 95
8.4 表达式和求值 ······ 95
8.4.1 脚本中的表达式 ······ 96
8.4.2 表达式和变量 ······ 97
8.5 数据类型转换 ······ 97
8.5.1 将字符串转换成数值 ······ 98
8.5.2 将数字转换成字符串 ······ 99
8.6 操作符 ······ 99
8.6.1 算术操作符 ······ 99
8.6.2 比较操作符 ······ 100
8.7 习题 ······ 100
第 9 章 程序设计基础(二) ······ 103
9.1 决策和循环 ······ 103
9.2 控制结构 ······ 103
9.2.1 if 结构 ······ 104

更多资源请访问稀酷客(www.ckook.com)

9.2.2 if ... else 结构 ······ 104
9.3 重复循环 ······ 105
9.4 函数 ······ 106
9.4.1 函数的参数 ······ 107
9.4.2 变量的作用域 ······ 108
9.5 大括号 ······ 109
9.6 数组 ······ 110
9.6.1 创建数组 ······ 110
9.6.2 访问数组的数据 ······ 111
9.6.3 关联数组 ······ 111
9.6.4 数组中的 document 对象 ······ 113
9.7 习题 ······ 114
第 10 章 window 和 document 对象 ·· 115
10.1 顶层对象 ······ 115
10.2 window 对象 ······ 115
10.2.1 访问窗口的属性和方法 ······ 116
10.2.2 创建窗口 ······ 117
10.3 window 对象的属性和方法 ······ 119
10.3.1 window.alert()方法 ······ 119
10.3.2 window.confirm()方法 ······ 119
10.3.3 window.prompt()方法 ······ 120
10.3.4 load 事件 ······ 120
10.4 location 对象 ······ 121
10.5 navigator 对象 ······ 122
10.6 document 对象 ······ 122
10.6.1 document.getElementById()方法 ······ 123
10.6.2 document.getElementsByTagName()方法 ······ 123
10.6.3 document.forms[ ]属性 ······ 124
10.6.4 document. images[ ]属性 ······ 124
10.6.5 document.createElement()和document.createTextNode()方法 ······ 125
10.6.6 document.write()方法 ······ 126
10.7 习题 ······ 130
第 11 章 表单和表单元素 ······ 131
11.1 form 对象 ······ 131
11.1.1 将表单作为对象和容器 ······ 133
11.1.2 访问表单属性 ······ 134
11.1.3 form.elements[ ]属性 ······ 135
11.2 将表单控件作为对象 ······ 136
11.2.1 与文本相关的输入对象 ······ 136
11.2.2 按钮输入对象 ······ 139
11.2.3 复选框输入对象 ······ 139
11.2.4 单选输入对象 ······ 141
11.2.5 select 对象 ······ 143
11.3 用 this 向函数传递元素 ······ 146
11.4 提交和预验证表单 ······ 149
11.5 习题 ······ 152
第 12 章 String、Math 和 Date 对象 ·· 155
12.1 核心语言对象 ······ 155
12.2 String 对象 ······ 155
12.2.1 连接字符串 ······ 156
12.2.2 字符串方法 ······ 157
12.3 Math 对象 ······ 159
12.4 Date 对象 ······ 160
12.5 日期计算 ······ 161
12.6 习题 ······ 163
第 13 章 编写框架和多窗口脚本 ······ 165
13.1 框架：父框架和子框架 ······ 165
13.2 家庭成员之间的引用 ······ 167
13.2.1 父到子的引用 ······ 167
13.2.2 子到父的引用 ······ 167
13.2.3 子到子的引用 ······ 168
13.3 有关框架脚本编程的提示 ······ 168
13.4 iframe 元素简介 ······ 169

更多资源请访问稀酷客(www.ckook.com)

13.5 突出显示脚注：框架集脚本示例 ······ 169
13.6 多窗口引用 ······ 175
13.7 习题 ······ 178

第 14 章 图像和动态 HTML ······ 181
14.1 image 对象 ······ 181
14.1.1 可互换的图像 ······ 182
14.1.2 图像的预缓存 ······ 182
14.1.3 图像变换的创建 ······ 184
14.2 无需脚本的图像变换 ······ 189
14.3 JavaScript：伪 URL ······ 192
14.4 主流的动态 HTML 技术 ······ 193
14.4.1 样式表设置的修改 ······ 193
14.4.2 通过 W3C DOM 节点实现动态内容 ······ 193
14.4.3 通过 innerHTML 属性实现动态内容 ······ 194
14.5 习题 ······ 194

第III部分 JavaScript 核心语言参考

第 15 章 String 对象 ······ 199
15.1 字符串以及数值数据类型 ······ 199
15.1.1 简单字符串 ······ 199
15.1.2 建立长字符串变量 ······ 200
15.1.3 连接字符串字面量和变量 ······ 200
15.1.4 特殊的内嵌字符 ······ 201
15.2 String 对象 ······ 202
15.2.1 语法 ······ 202
15.2.2 关于 String 对象 ······ 203
15.2.3 属性 ······ 204
15.2.4 解析方法 ······ 207
15.3 字符串使用函数 ······ 231
15.4 URL 字符串编码及解码 ······ 236

第 16 章 Math、Number 和 Boolean 对象 ······ 237
16.1 JavaScript 中的数值 ······ 237
16.1.1 整数和浮点数 ······ 237
16.1.2 十六进制和八进制整数 ······ 240
16.1.3 将字符串转换成数值 ······ 241
16.1.4 将数值转换成字符串 ······ 242
16.1.5 数值不是数值型时 ······ 243
16.2 Math 对象 ······ 243
16.2.1 语法 ······ 243
16.2.2 关于 Math 对象 ······ 243
16.2.3 属性 ······ 244
16.2.4 方法 ······ 244
16.2.5 创建随机数 ······ 245
16.2.6 Math 对象的快捷引用 ······ 246
16.3 Number 对象 ······ 246
16.3.1 语法 ······ 247
16.3.2 关于 Number 对象 ······ 247
16.3.3 属性 ······ 247
16.3.4 方法 ······ 248
16.4 Boolean 对象 ······ 250
16.4.1 语法 ······ 250
16.4.2 关于 Boolean 对象 ······ 250

第 17 章 Date 对象 ······ 251
17.1 时区和 GMT ······ 251
17.2 Date 对象 ······ 252
17.2.1 创建 date 对象 ······ 253
17.2.2 内部对象的属性和方法 ······ 254
17.2.3 日期方法 ······ 254
17.2.4 处理时区 ······ 257
17.2.5 字符串日期 ······ 257
17.2.6 用于以前浏览器的日期格式 ······ 258
17.2.7 更多转换 ······ 259
17.2.8 日期和时间运算 ······ 260
17.2.9 计算天数 ······ 262

更多资源请访问稀酷客(www.ckook.com)

17.2.10 早期浏览器中日期的错误和漏洞 ······266
17.3 在表单中验证日期项 ······267
第 18 章 Array 对象 ······273
18.1 结构化的数据 ······273
18.2 创建空数组 ······274
18.3 填充数组 ······274
18.4 JavaScript 数组创建功能的增强 ······276
18.5 删除数组项 ······276
18.6 并行数组 ······277
18.7 多维数组 ······281
18.8 模拟 Hash 表 ······282
18.9 Array 对象的属性和方法 ······284
18.9.1 Array 对象属性 ······285
18.9.2 Array 对象的方法 ······286
18.10 数组包含 ······311
18.11 解构赋值 ······312
18.12 与旧浏览器的兼容性 ······313
第 19 章 JSON — Native JavaScript Object Notation ······315
19.1 JSON 的工作原理 ······315
19.2 收发 JSON 数据 ······317
19.3 JSON 对象 ······318
19.4 安全限制 ······319
第 20 章 E4X — Native XML Processing ······321
20.1 XML ······321
20.2 ECMAScript for XML (E4X) ······322
20.2.1 使用 XML 对象 ······322
20.2.2 在 HTML 中嵌入 E4X ······328
20.2.3 方法 ······328
第 21 章 控制结构和异常处理 ······331
21.1 if 和 if...else 判定语句 ······331
21.1.1 简单判定 ······331
21.1.2 (condition)表达式 ······332
21.1.3 复杂判定语句 ······333
21.1.4 嵌套的 if...else 语句 ······334
21.2 条件表达式 ······336
21.3 switch 语句 ······337
21.4 重复(for)循环 ······340
21.4.1 使用循环计数器 ······342
21.4.2 跳出循环 ······343
21.4.3 使用 continue 继续循环 ······344
21.5 while 循环 ······345
21.6 do-while 循环 ······346
21.7 遍历属性(for-in) ······346
21.8 with 语句 ······348
21.9 标签语句 ······349
21.10 异常处理 ······352
21.10.1 异常及错误 ······352
21.10.2 异常机制 ······353
21.11 使用 try-catch-finally 结构 ······353
现实的异常 ······356
21.12 抛出异常 ······356
21.13 error 对象 ······361
21.13.1 语法 ······361
21.13.2 关于 error 对象 ······362
21.13.3 属性 ······362
21.13.4 方法 ······363
第 22 章 JavaScript 操作符 ······365
22.1 操作符的类别 ······365
22.2 比较操作符 ······366

更多资源请访问稀酷客(www.ckook.com)

22.3 不同数据类型的相等比较……367
22.4 结合操作符……369
22.5 赋值操作符……371
22.6 布尔操作符……373
22.6.1 布尔运算……374
22.6.2 使用布尔操作符……375
22.7 按位操作符……377
22.8 对象操作符……377
22.9 其他操作符……382
22.10 操作符的优先级……384

第 23 章 函数和自定义对象……387

23.1 Function 对象……387
23.1.1 语法……387
23.1.2 关于 Function 对象……388
23.1.3 创建函数……388
23.1.4 嵌套函数……389
23.1.5 函数的参数……390
23.1.6 属性……391
23.1.7 方法……395
23.2 函数应用的注意事项……396
23.2.1 调用函数……396
23.2.2 变量的作用域：全局作用域还是局部作用域……397
23.2.3 参数变量……401
23.2.4 递归函数……402
23.2.5 创建函数库……403
23.2.6 封闭区间……404
23.3 使用面向对象的 JavaScript 创建自定义对象……406
23.3.1 对象的具体细节……407
23.3.2 OO 例子：行星对象……409
23.3.3 进一步的封装……412
23.3.4 创建对象数组……412
23.3.5 利用嵌套对象……414
23.3.6 创建对象的最新方法……415
23.3.7 定义对象属性的提取器和设置器……415
23.4 面向对象的概念……416
23.4.1 增加原型……417
23.4.2 原型继承……418
23.4.3 嵌套对象和原型继承……418
23.5 Object 对象……420
23.5.1 语法……420
23.5.2 关于该对象……421
23.5.3 属性……422
23.5.4 方法……423

第 24 章 全局函数和语句……425

24.1 函数……426
24.2 语句……435
24.3 WinIE 对象……438
24.3.1 ActiveXObject……438
24.3.2 Dictionary……439
24.3.3 Enumerator……440
24.3.4 VBArray……441

第Ⅳ部分 文档对象参考

第 25 章 文档对象模型基础……445

25.1 对象模型层次结构……445
25.1.1 作为路径图的层次结构……446
25.1.2 第一个浏览器文档对象路径图……446
25.2 产生文档对象的过程……447
25.3 对象的属性……448
25.4 对象的方法……449
25.5 对象事件处理程序……450
25.6 对象模型概述……451
25.7 基本对象模型……452
25.8 附加图像的基本对象模型……452
25.9 仅用于 Navigator 4 的扩展……453

更多资源请访问稀酷客(www.ckook.com)

25.9.1 事件捕获模型……453
25.9.2 层……453
25.10 Internet Explorer 4+扩展……454
25.10.1 HTML 元素对象……454
25.10.2 元素包含层次结构……454
25.10.3 层叠样式表……455
25.10.4 事件冒泡……456
25.11 Internet Explorer 5+扩展……456
25.12 W3C DOM……457
25.12.1 DOM 层……457
25.12.2 规范中恒定不变的部分……458
25.12.3 W3C DOM 不具备的特性……458
25.12.4 新的 HTML 惯例……459
25.12.5 新 DOM 概念……459
25.12.6 W3C DOM 的静态 HTML 对象……467
25.12.7 双向事件模型……469
25.13 脚本编程的发展趋势……470
25.13.1 将内容与脚本分离……470
25.13.2 尽量使用 W3C DOM……471
25.13.3 处理事件……471
25.14 标准兼容模式(DOCTYPE 切换)……472
25.15 小结……473
第 26 章 通用 HTML 元素对象……475
第 27 章 window 对象和 frame 对象……657
27.1 window 对象术语……657
27.2 框架……658
27.2.1 创建框架……658
27.2.2 框架对象模型……658
27.2.3 引用框架……660
27.2.4 top 和 parent……660
27.2.5 防止在其他 Web 站点的框架中显示自己的页面……660
27.2.6 确认页面载入框架集……661
27.2.7 从有框架转换为无框架……661
27.2.8 继承性和封装性……661
27.2.9 框架的同步……662
27.2.10 空白框架……662
27.2.11 查看框架源代码……663
27.2.12 框架和 frame 元素对象……663
27.3 window 对象属性……663
27.3.1 语法……665
27.3.2 关于 window 对象……665
27.3.3 属性……667
27.3.4 方法……700
27.3.5 事件处理程序……754
27.4 frame 元素对象……759
27.4.1 语法……759
27.4.2 关于 frame 对象……759
27.4.3 属性……760
27.5 frameset 元素对象……765
27.5.1 语法……765
27.5.2 关于 frameset 对象……766
27.5.3 属性……767
27.6 iframe 元素对象……771
27.6.1 语法……771
27.6.2 关于 iframe 对象……772
27.6.3 属性……772
27.7 popup 对象……776
27.7.1 语法……776
27.7.2 关于 popup 对象……777
27.7.3 属性……777
27.7.4 方法……778
第 28 章 location 对象和 history 对象……781
28.1 location 对象……781
28.1.1 语法……782

更多资源请访问稀酷客(www.ckook.com)

28.1.2 关于 location 对象 ············782
28.1.3 属性 ············784
28.1.4 方法 ············795

28.2 history 对象 ············798
28.2.1 语法 ············798
28.2.2 关于 history 对象 ············798
28.2.3 属性 ············799
28.2.4 方法 ············800

第 29 章 document 对象和 body 对象 ············805

29.1 document 对象 ············806
29.1.1 语法 ············808
29.1.2 关于 document 对象 ············808
29.1.3 属性 ············809
29.1.4 方法 ············848
29.1.5 事件处理程序 ············870

29.2 body 元素对象 ············871
29.2.1 语法 ············872
29.2.2 关于 body 对象 ············872
29.2.3 属性 ············873
29.2.4 方法 ············877
29.2.5 事件处理程序 ············879

29.3 TreeWalker 对象 ············879
29.3.1 语法 ············879
29.3.2 关于 TreeWalker 对象 ············879
29.3.3 属性 ············880
29.3.4 方法 ············881

第 30 章 link 和 anchor 对象 ············883

第 31 章 image、area、map 和 canvas 对象 ············891

31.1 image 和 img 元素对象 ············891
31.1.1 语法 ············892
31.1.2 关于 image 对象 ············893
31.1.3 属性 ············894
31.1.4 事件处理程序 ············908

31.2 area 元素对象 ············910
31.2.1 语法 ············910
31.2.2 关于 area 对象 ············911
31.2.3 属性 ············912

31.3 map 元素对象 ············913
31.3.1 语法 ············914
31.3.2 关于 map 对象 ············914
31.3.3 属性 ············914

31.4 canvas 元素对象 ············917
31.4.1 语法 ············918
31.4.2 关于 canvas 对象 ············918
31.4.3 属性 ············921
31.4.4 方法 ············923

第 32 章 event 对象 ············927

32.1 事件 ············927
32.1.1 事件的内容和事件发生时间 ············928
32.1.2 静态 event 对象 ············928

32.2 事件传播 ············929
32.2.1 仅用于 NN4 的事件传播 ··929
32.2.2 IE4+事件传播 ············931
32.2.3 W3C 事件传播 ············935

32.3 引用事件对象 ············941

32.4 绑定事件 ············942
32.4.1 使用标记特性绑定事件 ····942
32.4.2 使用对象特性绑定事件 ····943
32.4.3 使用 IE 附加功能绑定事件 ············944
32.4.4 通过 W3C 监听器绑定事件 ············944
32.4.5 跨浏览器的事件绑定解决方案 ············945

32.5 事件对象兼容性 ············946

32.6 事件模型详析 ············948
32.6.1 以跨平台方式检查修改键 ············948

更多资源请访问稀酷客(www.ckook.com)

32.6.2 以跨平台方式捕获按键…950

32.7 事件类型……951

32.7.1 IE4+和 NN6+/W3C 中的事件类型……952

32.7.2 语法……954

32.7.3 关于 event 对象……955

32.7.4 属性……955

32.8 NN6+/Moz 的 event 对象……974

32.8.1 语法……975

32.8.2 关于 event 对象……975

32.8.3 属性……976

32.8.4 方法……994

附录 A JavaScript 和浏览器对象快速参考……997

附录 B 本书配套光盘内容……1011

更多资源请访问稀酷客(www.ckook.com)

第Ⅰ部分

JavaScript 入门

本部分内容

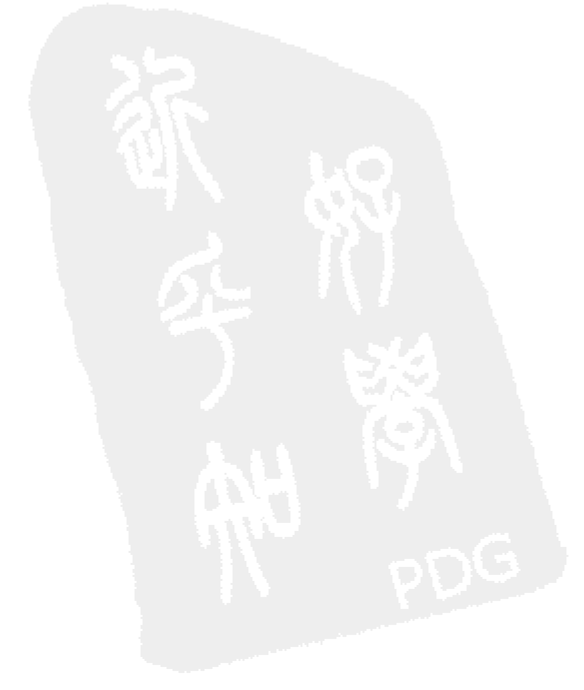
第 1 章　JavaScript 在万维网和其他领域所起的作用

第 2 章　脚本开发策略

第 3 章　选择和使用工具

第 4 章　JavaScript 基础

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第 1 章

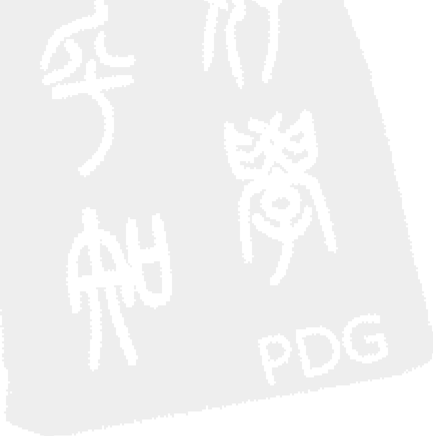
JavaScript 在万维网和其他领域所起的作用

本章包含哪些内容?
JavaScript 与其他 Web 编写技术的结合方式
JavaScript 简史
JavaScript 能完成和不能完成的工作

刚开始时，World Wide Web 只是通过网络发布静态文本和图像内容的媒介，但网站内容的设计者一直在探索、推动和发展 Web。现在，许多 Web 技术早就远远超出了其最初的目标。开发团体一旦拥有一项技术，就常常应用它来完成振奋人心的新工作。花费了大量的精力在服务器和客户机之间建立连接和传输数据后，内容开发人员和程序员开始利用该连接提供令用户耳目一新的体验，生成实用的应用程序。目前，每个人都很容易接触到大量的 Web 技术，尤其是 JavaScript 的浏览器编程，导致 Web 空前的爆炸式发展，把万维网从乏味的发布媒体变成了交互性极高、与操作系统无关的设计平台。

JavaScript 语言以及相关的浏览器功能是一种 Web 增强型技术。在客户计算机上使用该语言，可将静态内容页面转换为引人入胜的交互式智能体验。例如，如果客户计算机所在的时区是早晨，即使此时服务器处在晚饭时间，这种智能技术也会向网站访问者问候“早上好!”。JavaScript 还可实现更加显眼的效果，比如在页面下载时显示幻灯片的内容，而在整个演示过程中，JavaScript 控制着幻灯片的隐藏、显示和切换。

当然，JavaScript 不是给呆板的 Web 内容赋予活力的唯一技术。因此，最好结合使用 JavaScript 与一系列标准、工具和其他技术。本章还将介绍 HTML、CSS(Cascading Style Sheet，层叠样式表)、服务器程序和插件程序等技术。即使某些技术在满足交互式需求方面的宣传有点夸大其词，但在多数情况下，JavaScript 可与其他技术共同工作。最后概述 JavaScript 的起源，以及它在当今最先进的 Web 浏览器中所起的作用。



更多资源请访问稀酷客(www.ckook.com)

1.1　Web 流量的竞争

Web 页面发布者总是希望吸引尽可能多的访问者。不管 Web 页面点击数是否精确，每周获得 10 000 次点击的网站显然比每周获得 1 000 次点击的网站要受欢迎得多。即使不知道精确数字，受欢迎的相对程度也是一个极具价值的评价标准。另外，连接到某网站的外部链接有多少也是一个有效的衡量依据。受欢迎的网站会有许多到它的其他网站的链接，这是在 Web 搜索中该网站获得更高关注度的关键。

Web 发布者追求的终极目标是使人们频繁访问网站，其竞争相当激烈。Web 就像一个有 5 千万个频道的电视，用各种各样的信息吸引观众的注意力，包括屏幕上的各种交互式多媒体信息。

观众喜欢观看娱乐节目，查阅多媒体式的百科全书，也喜欢用鼠标浏览其他丰富多彩、极富魅力的内容，这些节目的质量很高，常常具备一流的图形、动画、现场录像和同步声响效果。相比之下，仅符合最低业界标准的 Web 页面就相形见绌了，这种 Web 页面即便使用了 Dynamic HTML 和样式表，其图片和文本的布局相对于桌面发布文档而言还是受到了很大的限制。即使 HTML 文档内容的质量很高，如果没有事先精心设计其布局，其吸引力也是有限的，在最好的情况下，用户也只能利用设计者通过超文本链接或表单进行导航，有时表单的内容会莫名其妙地消失。

1.2　其他 Web 技术

可以通过许多方法使网站和网页活跃起来，竞争对手也会努力使他们的网站更加吸引人。因此，除非你是极受欢迎的信息的唯一提供者，否则你将必须不断地提供新内容来留住老访问者，并吸引新的访问者。对于内部网，则需要不断地提高工作人员使用内部网站完成工作的效率。

这些是使用多个 Web 技术突出自己的网页的原因，下面分析一下应该了解的主要技术。

图 1-1 是构成典型动态网站的部件，其核心是服务器(Web 主机)和客户(浏览器)之间的一个对话；通过该对话，客户请求数据，服务器做出响应。最简单的模型只包含服务器、一个文档和一个浏览器，但实际上网站会使用这里列出的甚至没有在这里列出的其他部件。

这个过程开始于浏览器向服务器请求一个页面。服务器从其硬盘上直接发送静态的 HTML 页面，动态页面是由在语言解释器中执行的脚本生成的，例如 PHP，但与服务器发送给客户的静态信息一样，动态信息通常也使用 HTML 标记的形式。例如，服务器端数据库可以存储某类别的项，PHP 脚本可以查找这些数据记录，并在浏览器请求它们时，把它们标记为 HTML。

下载的页面可能包含其他部件的地址：样式表、JavaScript 文件、图像和其他资源。浏览器从服务器上请求这些资源，把它们合并到最终的页面上。

在浏览器接收到 JavaScript 程序之前，该程序是无效的——它就像是另一块正在下载的数据。接收到该程序之后，浏览器会验证其语法，编译它，准备在 HTML 页面或用户调用时执行它。接着它就成为网页的一部分，并放在服务器-客户对话中的客户端。JavaScript 可以向服务器发出请求，如后面所述，但它不能直接访问它所在页面之外和运行该程序的浏览器之外的信息。

PDG

更多资源请访问稀酷客(www.ckook.com)

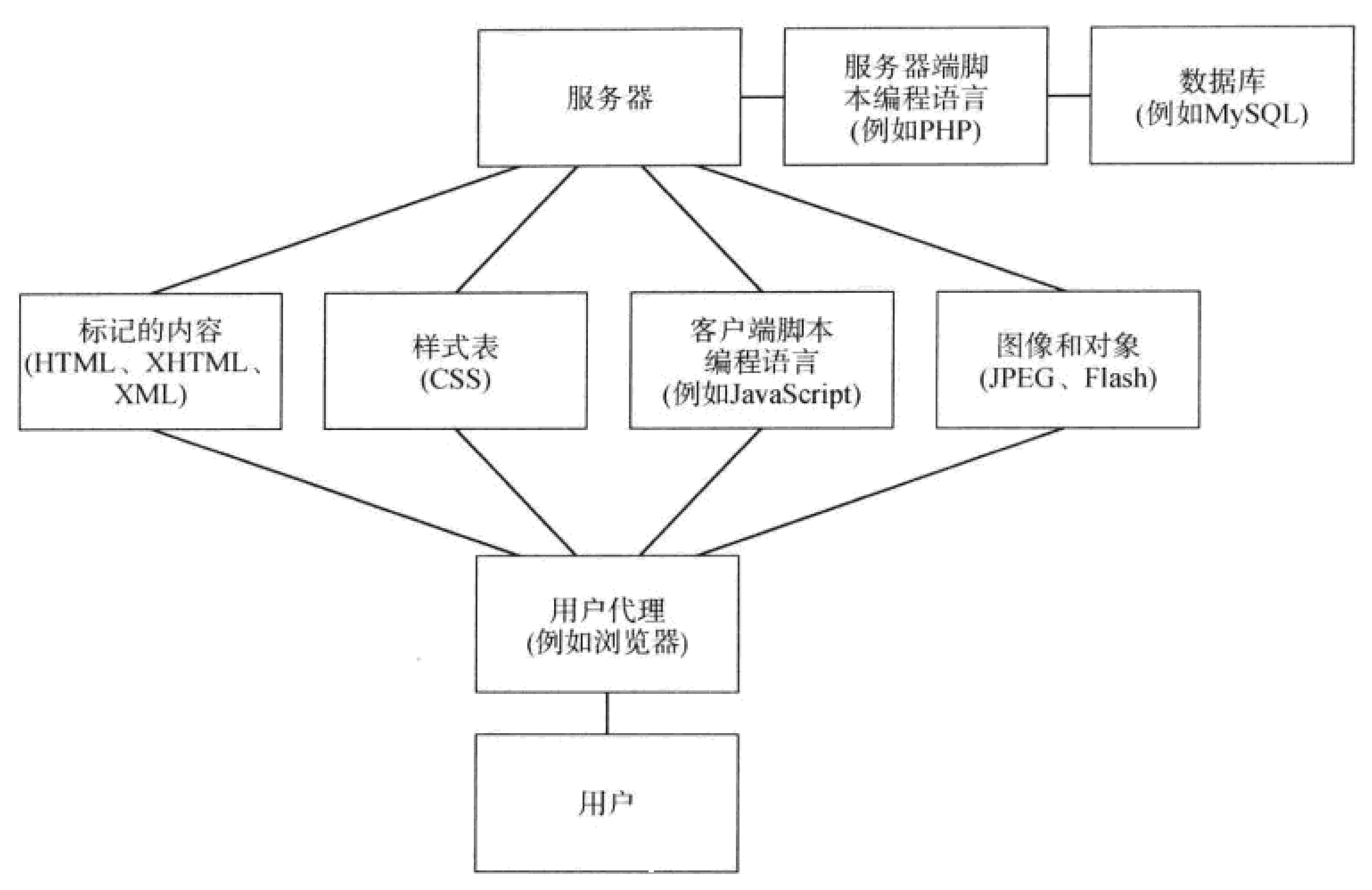


图 1-1 典型动态网站的部件

1.2.1 超文本标记语言(HTML 和 XHTML)

HTML 是 SGML(Standard Generalized Markup Language，标准通用标记语言)的派生物，它给页面的内容带来了结构。这个结构允许采用几种重要的方式来处理内容：

- 标记把海量无差别的文本转换为多个独立的部分，例如标题、段落、列表、数据表、图像和输入控件。结构在不同部分之间建立关系，突出了内容的含义：在标题和子标题之间；在列表的各个项之间；在页面的各个部分之间，例如标题、页脚和内容列。页面的语义对每个读者都非常重要：视力正常的人使用可视化浏览器，视力有障碍的人使用有声和 Braille 浏览器，也可以使用搜索引擎以及其他解析页面的软件来查找可理解的结构。
- 浏览器把一些标记的结构转换为具有特定行为的对象。图像以视觉上有意义的模式排列像素行。表单控件接受用户的输入，按钮把这些表单控件值提交给服务器。超链接可以把新页面加载到浏览器窗口中。这些不同类型的对象不使用某种形式的标记是不可能的。
- 页面在屏幕上、在 Braille 上或在演讲中的呈现方式取决于样式表，样式表用于页面上的元素，并给元素指定了外观、重点和其他属性。每个浏览器都使用一个默认的样式表，使标题、主体文本、超链接和表单控件按特定方式显示出来。Web 开发人员可以创建自己的样式表来覆盖浏览器的默认样式表，使页面用希望的方式显示出来。样式表通过指定文档的标记元素及其特性告诉浏览器如何显示 HTML 页面。
- 对手头的主题而言，最重要的是 HTML 标记给 JavaScript 提供了定位和操作部分页面的方式。脚本可以收集库中的所有图像，或者进入特定的段落，因为文档是用 HTML 标记的。

显然，HTML 标记和处理它的方式对文档的结构和含义、显示方式以及与 JavaScript 的成功交互都是至关重要的。以最佳方式使用 HTML 的内容超出了本书的讨论范围，但显然读者希

更多资源请访问稀酷客(www.ckook.com)

望在学习 JavaScript 的过程中，提高自己的标记技能。如果使用错误的、草率的或没有计划的标记，甚至世界上最好的脚本也会失败。

在 Web 开发的早期，也就是世纪之交(这个时期现在称为“20 世纪 90 年代”)，Web 设计人员对标记的语义和可访问性还所知不多。标记页面的目的仅是生成期望的可视化结果，这依赖于浏览器的默认样式，而没有考虑到标记的作用。开发人员可能选择 h4 标记，仅仅是为了生成某种大小和样式的字体，而没有考虑到文档的整体结构；或者给非表格内容使用数据表标记，仅仅是为了迫使页面内容排成一行。必要的空格和人为的间隔图像零碎地插入到真实的内容中，仅仅为了改变页面的外观：页面外观的重要性完全超越了页面的内容。幸好，目前这种方式已经过时了，可惜仍有许多人采用这种方式生成页面，那个时代还遗留下上百万个页面，它们拖住今天的人们，试图说服人们回到它们那个病态的时代。本书的目标之一就是鼓励读者采用现代的灵活方式编码，而放弃以前那套僵化的方法。

把 HTML 归类为标记语言，不仅不利于建立一流的网页，也不利于用户与网页的交互。其实，引导用户与数字信息进行交互的命令集合和其他语法就是程序设计。Web 页面设计人员利用 HTML 控制用户处理内容的方式，就像设计 Excel 的工程师构思用户与电子表格内容和功能交互的方式。

HTML 4.0 及其以后发布的标准致力于把 HTML 的用途定义为给内容指定上下文，而外观由样式表单独定义。也就是说，HTML 的任务不是将一些文本表示为斜体，而是揭示为什么要把它们定义为斜体。比如，通过<em>标记指定一些要强调的文本，或者标记这些文本的作用，而不必考虑如何用样式表格式化它们。HTML 标记和 CSS 表示之间的这个分工是一个极其强大的概念，它使网站的修改和重设计比以前使用内联样式和标记要快得多。

XHTML 是最近对 HTML 的更新改写，它遵循由 XML(eXtensible Markup Language，可扩展标记语言)标准确立的样式约定。XHTML 没有提供新的标记，但它强化了用标记表示文档结构和内容的思想。几年来，XHTML 被看成 HTML 向文档表示的圣杯——通用 XML 标记——演变的下一阶段，但这个期望已经落空了，部分原因是 Microsoft 在浏览器市场占据巨大的份额，而 Microsoft 一直拒绝把 XHTML 正确纳入为 application/xhtml+xml。目前，几乎所有 XHTML 文档都看成 text/html，这意味着浏览器仅把它们看作另外一些 HTML“标记”。也许使用 XHTML 标记的唯一好处是 W3C HTML Validator (http://validator.w3.org)所使用的更严格的规则集，这些规则会进行检查，以确保所有标记都在 XHTML 文档中关闭，不像 HTML 因其定义较松散而常被人诟病。

把 XHTML 用作 text/html 的相关文章，可参阅 Ian Hickson 发表的 Sending XHTML as text/html Considered Harmful (http://hixie.ch/advocacy/xhtml)和 David Hammond 发表的 Beware of XHTML(http://www.webdevout.net/articles/beware-of-xhtml)。

最近，HTML5 在 World Wide Web Consortium(W3C)的培养中成长起来。尽管现代浏览器刚刚开始实现 HTML5 的一些功能，但 HTML5 提供了比 HTML4 更丰富的标记词汇，更容易匹配将 HTML 用于实际的标记语言。

HTML 4.01 是目前 Web 的主流标记语言，所以本书中的 HTML 示例都符合 HTML 4.01 标准，但演示新 HTML5 元素(如 canvas)的示例除外。正确使用 DOCTYPE，示例应验证为 HTML5。通过几个简单的转换，它们很容易转换为 XHTML1.0：修改 DOCTYPE，给 HTML 元素添加 lang 特性，用/>关闭空元素，例如 input 和 link。希望这些示例有助于读者提高 HTML 或 XHTML

更多资源请访问稀酷客(www.ckook.com)

技能，并使本书更适用于将来的 HTML5。

1.2.2 CSS

指定 Web 页面的外观和操作方式是样式表的任务。如果文档的结构由其 HTML 标记来指定，样式表就定义了布局、颜色、字体、声音以及用于呈现内容的其他视觉和声音特性。将不同的 CSS(层叠样式表)定义集应用到同一文档可使其外观和音频效果完全不同，尽管文字和图像是相同的。

CSS 2.1 是目前用户代理支持的、应用最广泛的 W3C 样式表版本。音频样式表可以给 Web 页面的标记赋予声音和其他音频效果，它是 CSS 3 规范的一个模块，在撰写本书时，只有 Opera 和 Firefox 的扩展版 FireVox 支持它，但其他浏览器以后肯定也会支持它。

为了掌握 CSS 的精髓，需要花费时间，也需要进行反复试验，但这些付出是值得的。现在人们不再用 HTML 表格和透明的“间隔”图像来产生精美的多栏布局。每个 Web 开发人员都应该扎实掌握 CSS 基础知识。

CSS 比较难掌握，因为不同的浏览器对其许多功能的支持并不一致。很容易把大量时间花费在触发浏览器的标准模式上，其实更应该遵循 W3C CSS 规范。建议在标记顶部使用正确的 DOCTYPE，来触发该标准模式，例如：

```
HTML 4.01 Strict:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/REC-html40/strict.dtd">

HTML 5:

<!DOCTYPE html>

XHTML 1.0 Strict:

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

有关 DOCTYPE 切换的详细解释，可参阅 Eric Meyer 的文章 Picking a Rendering Mode (http://www.ericmeyeroncss.com/bonus/render-mode.html)。

1.2.3 服务器编程

如果网站依赖数据库访问，或需要非常频繁地改变其内容，就应使用服务器编程技术。这种技术可以为浏览器生成 HTML 输出，或处理站点访问者在页面上填写的表单。即使是提交简单的登录或搜索表单，也会触发一些服务器进程，并将结果发送到用户的浏览器上。服务器编程技术有很多种，浏览 Web 开发站点，就会看到它们的名称，其中最常用的是 PHP、ASP、.NET、JSP 和 Coldfusion。相关的编程语言包括 Perl、Python、Java、C++、C#、Visual Basic，在某些环境下甚至有服务器端的 JavaScript。

无论使用哪种语言，Web 页面作者肯定都需要能控制服务器，或提供后台程序(如数据库)来计算结果或处理用户信息。即使可以使用基于服务器的新 Web 站点设计工具，负责设计内容

更多资源请访问稀酷客(www.ckook.com)

的 HTML 作者也需要把服务器脚本编程任务交给经验更丰富的程序员。

客户端的 JavaScript 并不能替代服务器端的脚本编程。即使 JavaScript 在浏览器中用于改进用户的体验，动态响应用户操作或保存数据的网站也都必须由服务器端脚本驱动。原因很简单：第一，JavaScript 本身不能写入服务器上的文件。它可以帮助用户做出选择，并准备要上传的数据，但之后，它只能把数据交给一个服务器端脚本来更新数据库。第二，非所有用户代理都运行 JavaScript。屏幕阅读器、移动设备、搜索引擎和安装在某些公司中的浏览器都不使用 JavaScript，或者在接收网页时会剔除 JavaScript。因此，网站应在关闭 JavaScript 后仍能正常工作。使用 JavaScript 可使浏览速度更快、更具美感，但不应让网站在不运行 JavaScript 时崩溃。

虽然服务器端脚本编程的功能很强大、很高效，但它与用户的交互速度受到服务器和用户代理之间的 Internet 连接速度的限制。显然，需要更新服务器上数据的任何进程都必须包含某个客户-服务器对话，但在 JavaScript 的帮助下，用户交互的许多方面完全可以在浏览器上进行，表单验证和拖放操作就是两个例子，再在响应密集型过程完成后更新服务器。

服务器编程和浏览器脚本共同工作的一种方式是所谓的 *Ajax*—Asynchronous JavaScript and XML。“异步”部分在浏览器中运行，向完全在后台的服务器端脚本请求 XML 数据或将数据发布到服务器，然后，服务器返回的 XML 数据由浏览器中的 JavaScript 检查，以更新 Web 页面的相应部分。这就是许多基于 Web 的流行电子邮件用户界面以及可拖放的 Google Maps(http://maps.google.com)卫星照片特写镜头的工作方式。

服务器编程和浏览器脚本共同工作，就能编写出精美的应用程序。可以用某种服务器端语言编写，例如 PHP，或者与使用这种语言的人组成团队，给要创建的支持 JavaScript 的页面打好基础。

1.2.4 辅助程序和插件程序

在 WWW 的早期阶段，浏览器只能提供几类数据，桌面操作系统的浏览器内置了显示文本(使用 HTML 标记)和图像(流行的格式如 GIF 和 JPEG)的功能。开发人员不希望受到这些数据类型的束缚，就努力扩展浏览器，使其他格式的数据可以在客户计算机上显示。然而，以前建立的浏览器不能下载和显示任何声音文件格式。

要解决这个问题，应允许浏览器在识别特定类型的入站文件时，在客户机上启动特定的应用程序来显示文件的内容。只要把这个辅助应用程序安装在客户机上(并在浏览器的配置中设置为关联该辅助程序)，浏览器就会启动该程序，并把入站文件发送给该程序。因此，可以给 MIDI 声音文件安装一个辅助程序，给动画文件安装另一个辅助程序。

从 1996 年初的 Netscape Navigator2(简称 NN2)开始，浏览器的软件插件程序就允许开发人员扩展浏览器的功能，而不用修改浏览器。与辅助应用程序不同，插件程序可将外部内容完美无缺地融入文档。

最常见的插件程序可以播放服务器上的音频和视频。音频包括音轨(在访问页面时在后台播放)和现场的直播解说音频(流，类似于广播站)。视频和动画在通过插件程序播放时，会显示在页面的某个地方(插件程序知道如何处理此类数据)。

当今的浏览器带有解码最常见声音文件类型的插件程序。开发人员为 Windows 操作系统的 Internet Explorer(简称 IE)开发插件程序时，一般将插件程序实现为 ActiveX 控件，这对操作系

更多资源请访问稀酷客(www.ckook.com)

统来说很重要，但对用户来说并不重要。

插件程序和辅助程序不仅仅用于播放视频和音频。一个流行的辅助应用程序是 Adobe Acrobat Reader，其 Acrobat 文件的显示效果与其打印效果一样。对于交互性，当今的开发人员通常利用 Macromedia 公司的 Flash 插件程序。使用 Flash 设计环境创建的 Flash 应用程序可以包含可单击区域、可拖动元素、动画和嵌入视频。一些程序设计者还在 Flash 中设计漂亮的视频游戏和动画故事。安装了 Flash 插件程序的浏览器在嵌入浏览器页面的矩形区域中显示内容。JavaScript 的一个变体称为 ActionScript，它允许 Flash 与用户和 HTML 页面的部件交互操作。与 JavaScript 相同，ActionScript 也可以向服务器发出数据的读写请求，以访问外部资源。YouTube.com 就是大量集成了 Flash 的一个流行网站。

在 Flash 或相似环境中设计交互式内容的一个潜在缺点是，假如访问者没有安装正确的插件程序版本，就需要花费一定的时间来下载插件程序。此外，一旦安装了插件程序，将高质量的图像和交互内容下载到客户机上所需的时间，就长于用户期望的等待时间(特别是拨号连接)，此时必须在网页下载速度和访问者需要的交互内容之间进行平衡。

另一种客户端技术 Java applet 在 20 世纪 90 年代后期一度流行，但由于一些技术原因和企业政治方面的原因已经失宠。不过，Java 目前仍旧是一种流行的服务器语言，甚至用于移动电话编程，这远远超出了创建该语言的公司 Sun Microsystems 的业务范围。

1.3 JavaScript 是一门综合性语言

Sun 的 Java 语言派生于 C 和 C++，但它是一门新颖的语言，它的主要用户是富有经验的程序员，而不是 Web 页面设计人员。Java 于 1995 年首次发布的说明书令人沮丧，某些程序员和脚本编写员对一些设计工具非常熟悉，比如 Apple 公司风光一时的 HyperCard 和 Microsoft 公司的 Visual Basic，他们会很快采纳某种更合适的语言。在这些可访问的开发平台上，非专业的程序员会设计一些创造性的应用程序，常常用于完成一些专业程序员不愿意完成的任务。人们常常为了满足个人的需要，在教室、办公室、小房间或车库里进行开发，但 Java 并不是这种综合性语言。

1995 年 11 月，听说 Netscape Communications 公司正在酝酿一种脚本语言，它命名为 LiveScript，与 Netscape 的 Web 服务器软件新版本一同开发。这种语言用相同的语法实现两个目标：一个是作为脚本语言，由 Web 服务管理员用于管理服务器，并将网页与其他服务相连，比如后台数据库以及用于查询信息的搜索引擎。为了发展壮大 Live 这个品牌，Netscape 将服务器上使用 LiveScript 进行数据库连接的部分称为 LiveWire。

在客户端的 HTML 文档中，程序设计者可使用这种新语言编写脚本，以在很多方面润色 Web 页面。比如，程序设计者可使用 LiveScript，确保用户在必填的文本字段中填入电子邮件地址或信用卡号。这里不是强迫服务器或数据库验证数据(这需要在客户浏览器和服务器之间交换数据)，而让用户的计算机处理所有计算工作——这利用了原本可能闲置的计算资源。总之，LiveScript 为用户提供 HTML 层次的交互。

更多资源请访问稀酷客(www.ckook.com)

1.3.1　LiveScript蜕变成JavaScript

1995年12月上旬，在Navigator 2正式发布之前，Netscape和Sun Microsystems发表联合声明，这种编程语言以后更名为JavaScript。虽然Netscape采用这个名字有几个市场原因，但没有预料到这一变动在Java编程界和HTML脚本领域带来的混乱。

在发表此声明之前，该语言已经在某些方面与Java联系到了一起。其众多的基本语法元素都使人联想到Java风格。但对客户端脚本，该语言的目标与Java有很大的区别——它是一种集成到HTML文档中的编程语言，而不是用来编写占据页面上固定矩形区域的applet(这些applet一般不考虑页面上的其他内容)的语言。Java有成熟的编程语言术语(和概念上更难掌握的面向对象方法)，而JavaScript的术语较少，编程模型也更容易理解。

其实真正的困难在于清晰地表明Java和JavaScript之间的区别。许多计算机媒体暗示说，JavaScript提供了建立Java applet的更简单方式，这是完全错误的。直到今天，一些刚入行的程序新手还误以为JavaScript与Java语言是相同的，所以将一些Java问题发送到专为JavaScript设置的Internet新闻组和邮件列表中。

其实，客户端Java和JavaScript之间的差异比其相似之处要多得多，这两种语言用两种完全不同的解释器来执行其代码。

1.3.2　微软的JavaScript版本

虽然JavaScript语言起源于Netscape，但是Microsoft看到了这种语言的潜在能力和流行趋势，在IE 3中实现了它(其名称是JScript)。虽然Microsoft希望人们使用它在IE的Windows版本中提供的VBScript(Visual Basic Script)语言，但JavaScript适用于更多的浏览器和操作系统，因此很多客户端脚本程序员选择它为众多不同的用户设计页面。

由于脚本编程基于Microsoft和Netscape开发的主流浏览器，以后的浏览器厂商自动为JavaScript提供支持。因此可以在Opera或Apple Safari(后者基于开源浏览器KHTML)等浏览器上使用基本的脚本服务。并非所有浏览器都按相同的方式处理每个细节——这是本书要给客户端脚本解决的一个重要问题。

1.3.3　JavaScript版本

JavaScript语言有自己的版本编号系统，它完全独立于浏览器的版本号。Netscape浏览器开发组创建了JavaScript，其继任者Mozilla Foundation将继续成为JavaScript版本编号系统的驱动者。

在逻辑上，第一个版本是JavaScript 1.0。它在Navigator 2和Internet Explorer 3的第1版中实现。随着后续浏览器版本的开发，JavaScript版本号以很小的单位增加。JavaScript 1.2是使用时间最长、最为稳定的一个版本，Internet Explorer 7当前支持该版本。基于Mozilla的浏览器和其他浏览器则支持JavaScript 1.5 (Mozilla 1.0和Safari)、JavaScript 1.6 (Mozilla 1.8 浏览器)和JavaScript 1.7(Mozilla 1.8.1及以后版本)中的一些新功能。

JavaScript的每个新版本都新增了一些语言特性。例如，在JavaScript 1.0中，数组没有完全开发出来，所以数组不能跟踪其中的元素个数。JavaScript 1.1就填补了这个漏洞，提供了一

PDG

更多资源请访问稀酷客(www.ckook.com)

个构造函数来生成数组，还给所有生成的数组提供了一个固有的 length 属性。

在浏览器中实现的 JavaScript 版本并非总能很好地预测该浏览器的核心语言特性。例如，JavaScript 1.2(由 Netscape 在 Netscape Navigator 4 中实现)包含对正则表达式的广泛支持，但这些功能并没有完全包含在 Microsoft Internet Explorer 4 的对应 JScript 版本中。同样，Microsoft 在 Internet Explorer 5 的 JScript 中实现了 try-catch 错误处理功能，但 Netscape 没有包含这个功能，直到基于 Mozilla 的 Netscape Navigator 6 实现了 JavaScript 1.5。因此，在确定可以使用什么语言特性时，语言的版本号并非可靠的预测器。

1.3.4 核心语言标准 ECMAScript

尽管 Netscape 首先开发了 JavaScript 语言，但 Microsoft 在 Internet Explorer 3 中就合并了该语言。Microsoft 不希望从其商标拥有者 Sun Microsystems 那里请求 Java 许可，所以该语言在 Internet Explorer 环境中称为 JScript。除了一些非常罕见的例外和新引入功能的步调不一致之外，这两种语言是完全相同的。对于核心语言，同一 JavaScript 版本的浏览器品牌之间的兼容级别非常高(而对象模型的实现有巨大的区别，详见第 25 章)。

如前所述，人们正在建立一些浏览器厂商都会遵循的行业推荐标准，以使开发人员更方便地开展工作。核心语言是达到标准状态的首要部件。欧洲标准机构 ECMA 协商并建立了该语言的正式标准。标准组把该语言的第一个规范称为 ECMAScript，它大致与 Netscape Navigator 3 中的 JavaScript 1.1 相同。标准(ECMA-262)定义了各种数据类型的处理方式、操作符的工作方式、专用数据的特定语法和其他语言特性。新版本(版本 3)对核心语言做了许多改进(版本 2 仅修正了版本 1 中的错误)。

ECMAScript 规范的当前版本称为 ECMAScript 第 5 版，在 www.ecma-international.org 上发布。ECMA 认为，“第 5 版编纂了浏览器实现方案中普遍采用的语言规范的标准解释，支持自从第 3 版发布以来出现的新特性。这些特性包括访问器属性、反射的创建、对象的检查、特性属性的程序控制、新增的数组操作函数、对 JSON 对象编码格式的支持，并提供改进了的错误检查和程序安全性的严格模式。”

如果读者在学习编程语言，就会觉得文档很吸引人，如果只希望编写页面，就可能发现有许多难以置信的细节。

所有主流浏览器开发人员确保器浏览器符合 ECMA 标准。Navigator 3 以及后续版本和 Internet Explorer 4 以及后续版本都遵循大多数 ECMAScript 标准，ECMA 标准添加新特性时，它们也会进入较新的浏览器。ECMAScript 的最新版本是版本 5。在过去几年中，所有主流浏览器都支持 ECMAScript 以前的版本 3。

注意：

甚至 ECMAScript 第 5 版处于筹备阶段时，Mozilla Foundation 和 Microsoft 也在分别实现对应的 JavaScript 2.0 和 JScript 版本。ECMAScript 的一个扩展是 E4X(ECMAScript for XML)，它在 2005 年后期完成，并在基于 Mozilla 1.8.1 或更新版本(例如 Firefox 2.0)的浏览器中实现。用于开发 Flash 动画的 Adobe ActionScript 3 语言完全支持 E4X。E4X 是 JavaScript 的一个重要扩展，因为它使 XML 成为该语言语法中的一个内置数据类型，更便于处理 XML 格式的数据。XML 是许多数据交换过程使用的数据格式，包括 Ajax (参见第 39 章)。

更多资源请访问稀酷客(www.ckook.com)

1.4 JavaScript：灵活易用的工具

知道什么任务需要什么编写工具来完成，是成为全能 Web 页面设计者的一个重要方面。忽略 JavaScript 的 Web 页面设计者就像使用钳子而不使用扳手的管道工人一样，会弄伤自己的指关节的。

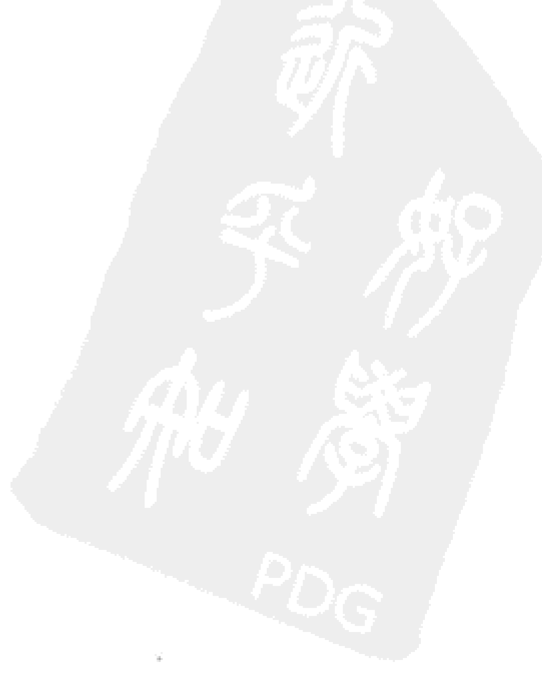
同样，JavaScript 也非万能的，对 JavaScript 的作用和限制了解得越多，就越知道在何种场合使用它。下面的几种情形特别适合使用 JavaScript：

- 使用表单元素(输入域、文本区、按钮、单选按钮、复选框、选择列表)和超文本链接，让 Web 页面直接响应用户。
- 发布类似于数据库的一小组信息，并提供友好的数据界面。
- 根据用户在 HTML 文档中的选择来控制多框架导航、插件或 Java applet。
- 提交给服务器之前，在客户端预处理数据。
- 在现代的浏览器上动态、实时改变内容和风格，以响应用户的交互操作。
- 从服务器上请求文件，向服务器端脚本发出读写请求。

同时，弄明白 JavaScript 不能做什么是非常重要的。一些脚本开发人员白白浪费了许多时间尝试完成 JavaScript 不能完成的任务，其实许多限制是故意设置的，以禁止访问者侵犯他人的隐私或非法访问其桌面计算机。因此，除非访问者使用现代浏览器，并且明确赋予他人权限来访问计算机中受保护的部分，否则 JavaScript 不能秘密地执行下面的动作：

- 设置或检索浏览器的首选项设置、主窗口的外观特性、动作按钮和打印功能
- 在客户计算机上启动应用程序
- 在客户计算机上读写文件或目录(cookies 例外)
- 在服务器的文件中直接写入
- 重新传输从服务器上捕获的实时数据流
- 从 Web 站点访问者处向用户发送机密的电子邮件(也可以给能发送邮件的服务器端脚本发送数据)

Web 网站设计者一直在搜寻一些工具，以使用最少的精力使网站更富有魅力，对于更适合通过编写文档、图形设计和页面布局来创建，而不是通过编程来创建的网站，就更是如此。不是每个 Web 管理员都拥有大批资深程序员为网站编写特殊的自定义功能，也不是每个 Web 设计者都可以控制包含许多 HTML 文件和图形文件的 Web 服务器。即使服务器在电话线另一端的黑盒子里，JavaScript 也能使熟悉 HTML 的人拥有编程能力。



更多资源请访问稀酷客(www.ckook.com)

第2章

脚本开发策略

只有了解了脚本浏览器的历史，才能发现 JavaScript 的优劣势。

优势在于 Microsoft、Mozilla Foundation(使用 Firefox、Netscape 和 Camino 等品牌名)、Apple 和其他公司出品的成熟浏览器产品提供了卓越的功能；而劣势在于无法体会给旧版本浏览器编写脚本的痛苦，这些浏览器错误百出，有时由于缺乏标准，浏览器之间存在许多不兼容之处。如果脚本应用程序是为自己使用的浏览器而编写的，就可能收到网站访问者发来的大量电子邮件，抱怨页面在不同种类、不同版本、不同操作平台的浏览器上运行时，出现了各种脚本错误。

本章包含哪些内容？
浏览器开发竞争给 Web 开发人员带来的好处和压力
将核心 JavaScript 语言与文档对象分离
开发跨浏览器策略的重要性

如果应用程序的用户使用多种不同的浏览器，会带来很多变化，加大页面设计者的工作量。在开始介绍如何输入 JavaScript 代码，编写脚本网页前，本章将呈明这些问题，否则这些问题可能会成为读者进一步学习 JavaScript 及其强大功能的绊脚石。本书作者也是开发人员，从最早的 JavaScript 演示版就开始使用它，所以不会掩饰今天的脚本程序员可能遇到的问题。其实，理解这些问题反而有助于学习这门语言。理解了编写浏览器脚本的宏观环境后，就更容易在 Web 应用程序的开发中成功使用 JavaScript。

2.1 浏览器的竞争

从最早的 Web 淘金热开始，设计人员就需要考虑浏览器的兼容性。这一问题在 JavaScript 出现之前就早已存在了。尽管浏览器开发者和其他感兴趣的团体在创建统一标准的过程中提出了各自的观点，并建立了一套标准标记来定义标题等级和行中断符，但 HTML 的设计者并不能使文档在所有的客户计算机上都具有相同的显示效果，标记中的内容在不同操作系统的不同品牌浏览器上的显示格式很难保持一致。

由于竞争愈演愈烈，迫使人们创立了全新的特性和标记，以创建更加灵活多变、绚丽迷人的网页，最终使 Web 浏览器的开发转变为盈利行业。这个盈利行业同许多计算机相关行业一样，标准的研究进程很快落后于 Web 浏览器的发展。浏览器厂商会把新的 HTML 特性嵌入浏览器中，

PDG

更多资源请访问稀酷客(www.ckook.com)

并建议相应的标准收入该特性。Web 设计者通常在这些标准正式发布之前，就开始使用这些特性了(有时是浏览器演示版)。

用来接收数据的客户机有一个解释引擎，设计者主要根据解释引擎来开发内容，这就会出现一个问题：Web 内容的提供者必须依靠内嵌在浏览器中的功能完成任务，而不依赖该引擎的独立计算机程序可以随意扩展其功能，甚至发明新功能，并让这个新功能运行在每个人的计算机上(至少对于给定的操作系统)。于是，问题就来了：访问某网站时，假如并不是所有的浏览器都支持某个 HTML 特性，是否只有支持该特性的浏览器才能提供某种功能？假如新浏览器部署了新特性，如何为使用旧浏览器的访问者提供服务？

在 Web 发展的初期，编写网页的程序员付出了许多努力和心血，才产生了如今获得广泛认可的 HTML 特性，比如表格和框架的概念，HTML 扩展功能的标准等。

目前占统治地位的操作系统是 Microsoft Windows，其浏览器 Internet Explorer 也占据了大部分市场份额，但人们使用的浏览器数量并未减少。最新的几个浏览器都基于开源浏览器 Mozilla，包括现代的 Netscape、Firefox 和 Camino 浏览器。Macintosh 操作系统有自己的 Apple 品牌浏览器 Safari(2003 年发布)。还有一些用户使用独立的 Opera 浏览器。这些非 Microsoft 浏览器厂商促进了世界的进步，为我们带来了更好的浏览器。

2.2　相互包容

今天的浏览器之战与 Web 早期不同，目前已建立了许多 Web 标准，其范围和深度大大增加了浏览器应用程序的种类，甚至图书开发人员也使用这些标准来组织其内容，所以，大多数开发人员都要求，新的浏览器版本应支持更多更深入的标准。但是，日常用户对标准并不关心，他们只希望能便捷地在 Web 上查找信息。大多数用户都不怎么升级其浏览器，除非无法在其古老的浏览器中访问他们最钟爱的站点。

即使制定了标准，Web 开发人员的工作未必能更容易地完成。首先，最新浏览器支持的标准是不均衡的。一些浏览器支持的标准比其他浏览器更进一步；其次，一些标准的细节比较含糊，解释也有差别。有时，在对内容开发人员至关重要的领域，标准并没有提供什么指南。这些问题最终交给了浏览器厂商，他们用专利功能来解决这些问题，并希望这些功能会成为事实标准。

仅仅几年，浏览器战场就发生了巨变：Netscape 曾经拥有的巨大市场份额现在已被 Microsoft 夺走。但是，最近人们开始关注 Windows 平台的隐私和安全性，于是许多用户希望使用更不容易受到攻击的浏览器。到目前为止，Mozilla Firefox 是他们找到的最合适的替代浏览器。在页面编写方面，现在各浏览器的最新版本之间存在许多共同之处，但最大的问题是，所有浏览器对最新功能的实现是不均衡的，定义浏览器之间的共同点可能是编写工作中最棘手的部分。

2.3　当今存在的兼容性问题

下面描述市场份额处于前三位的浏览器系列(Microsoft Internet Explorer、基于 Mozilla 的浏览器和 Apple Safari)之间的兼容性问题。下面几节的讨论没有涉及具体的脚本技术，因为一些用户目前可能还对编程知识知之甚少。本书的许多章节都提供了脚本编写建议，以兼顾各种浏览器。

PDG

更多资源请访问稀酷客(www.ckook.com)

2.3.1 将核心 JavaScript 语言从文档对象中独立出来

早期的 JavaScript 设计者最初将客户端脚本作为一种允许对页面元素进行编程的环境，但随着浏览器的日趋成熟，这种情况已经发生了变化。目前 JavaScript 核心语言规范和文档中编写元素的规范已经存在明显的差别(比如，表单中的按钮和域，如图 2-1 所示)。

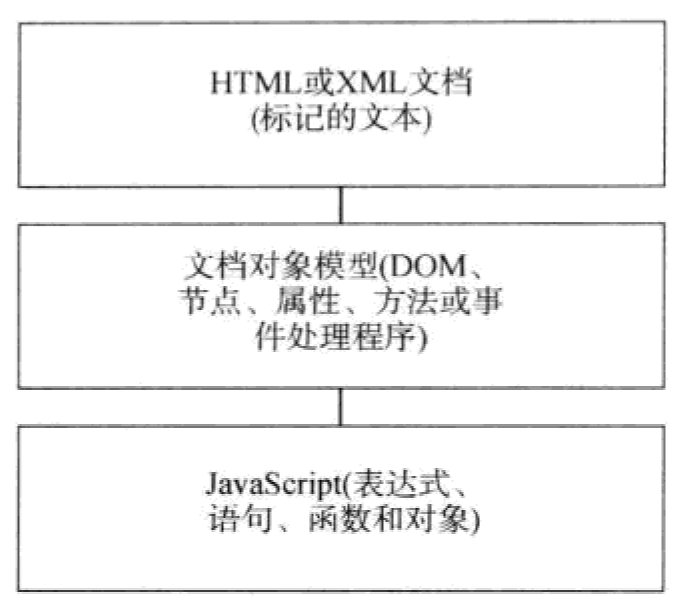


图 2-1 文档对象模型是 HTML 或 XML 文档与 JavaScript 编程语言之间的编程接口

在一个层面讲，这样的差异未尝不是一件好事，这说明，在基本编程概念和语法中存在一个规范，通过该规范可在任意数量的其他环境中生成编程语言。可将核心语言类比作基础电缆，一旦知道了电缆如何工作，就可以把它们连接到各种类型的电器中。而目前的 JavaScript 就是用来在 HTML 文档中连接元素的电缆。未来的操作系统将会使用核心语言来连接需要自动交换数据的桌面应用程序。

JavaScript 电缆连接到浏览器上的一端是页面上的元素，例如，段落(p)、图像(img)和输入域(input)。在编程术语中，这些元素称为文档对象。因为规范已经将文档对象与连接它们的电缆分隔开，所以这些对象可以通过其他类型的电缆(其他语言)连接，这就像设计可以连接各种信号线的电话一样，包括尚未发明的电线类型，目前这些设备可以使用铜线或光纤。在 Internet Explorer 中，这种对象与语言的分离非常清晰，文档对象可以分别用 JavaScript 或 VBScript 编写，它们是相同的对象，只是连线方式不同而已。

核心语言与文档对象的分离使每个概念都有自己的标准和发展步伐。每个概念都有一系列标准，每个浏览器厂商还可以自由扩展这些标准。但设计者必须耗费更大的精力开发能在多个浏览器上使用的页面或脚本，除非脚本遵循某个公共标准(或者使用其他分支技术，让每个浏览器按各自的方式运行)。

2.3.2 核心语言标准

为了追踪 JavaScript 语言版本，就需要了解它的发展历程。JavaScript 的第一个版本(在 Netscape Navigator 2 中)是版本 1，但在使用它时不加该版本号，JavaScript 就是 JavaScript。发布 Navigator 3 时，版本号成了一个问题。与 Navigator 版本相关的 JavaScript 版本是 JavaScript 1.1。Navigator 4 提高了该语言的版本号，它使用 JavaScript 1.2。

Microsoft 的脚本功能使新手不知所措，Internet Explorer 3 是第一个支持脚本的 Internet Explorer

更多资源请访问稀酷客(www.ckook.com)

版本，它几乎与 Navigator 3 同期推出，但脚本编写人员很快就发现，Microsoft 的脚本功能整整落后了一代。Microsoft 并没有被授权使用 JavaScript 商标，因此该公司把它称为 JScript。尽管如此，标记中指定脚本语言的 HTML 标记特性可以是 JScript，也可以是 JavaScript for Internet Explorer，Internet Explorer 3 可以解释为 Navigator 2 编写的 JavaScript 脚本。

在 Navigator 3 和 Internet Explorer 3 占据统治地位的时期，脚本编写新手常常感到困惑，因为他们希望这两种脚本编写语言是相同的。可惜，JavaScript 1.1 中的一些语言特性不能在 Internet Explorer 3 的旧 JavaScript 版本中使用。Microsoft 在 IE3 中改进了 JavaScript，升级了.dll 文件，给 IE 提供了它的 JavaScript 语法。但是，很难确定在访问者的 IE3 中安装了哪个.dll 文件。这种情况在 Internet Explorer 4 中得到了改进。其核心语言达到了 JavaScript 1.2 的水平，与 Navigator 4 的早期版本一样。把脚本加载到 Internet Explorer 4 中，Internet Explorer 4 可以理解 Navigator 4 中的几乎所有新语言特性。Microsoft 仍将该语言称为 JScript。

JavaScript 版本发生这一系列变化的同时，Netscape、Microsoft 和其他相关的团体一起建立了一个核心语言标准，这个标准机构最初是一个在瑞士成立的组织 European Computer Manufacturer's Association(欧洲计算机制造者协会)，现在它简称为 ECMA(通常发音为 ECK-ma)。1997 年中期，该组织协商并发表了第一个正式的语言规范(ECMA-262)，由于 JavaScript 名称的授权有问题，这个机构为该语言起了一个新名字 ECMAScript。

ECMAScript 的第一版几乎和 Navigator 3 使用的 JavaScript 1.1 一样，只是有一些细微差别。Navigator 4 和 Internet Explorer 4 都支持 ECMAScript 标准，也都超越了 ECMAScript 标准，就像商家在支持标准时经常超越标准一样。幸好，这种扩展核心语言的公用标准已得到广泛的认可，减少了程序员编程时遇到的麻烦。

JavaScript 1.3 在 Netscape Navigator 4.06 到 4.7x 中实现，也得到 IE5、5.5 和 6 的支持。在 JavaScript 1.5 中添加的几个新语言特性也在基于 Mozilla 的浏览器中实现了(包括 Navigator 6 及其以后版本)。在 JavaScript 1.6 中又新增了几个核心语言特性，这些特性首次在 Mozilla 1.8 中实现(Firefox 1.5，在 Firefox 3 的 JavaScript 1.8 中还新增了更多的语言特性)。

实际上，除了少数前沿功能之外，今天使用的许多浏览器都支持 Mozilla 浏览器的所有功能，所以 JavaSctipt 版本号通常并不重要。在核心语言问题之前，人们先遇到了与旧式浏览器的其他兼容性问题。现在可以不考虑解决最旧浏览器中的缺陷的复杂方法。

2.3.3 文档对象模型

主流浏览器的核心 JavaScript 语言兼容性比较高，而文档对象的核心 JavaScript 语言兼容性并不高。Internet Explorer 3 的文档对象模型(DOM)是基于 Netscape Navigator 2 的，它使用相同的浏览器级别作为核心语言的模型。Netscape 在 Navigator 3 的模型中增加了几个新对象，但这给脚本编写新手带来了新的麻烦，因为他们期待这些对象也出现在 Internet Explorer 3 中。在 Internet Explorer 3 中，通常缺少图像对象；图像对象使得用户在图像上移动鼠标指针时(它们通常称为 mouse rollovers，鼠标替换特效)，脚本可以交换图像。

然而，在第 4 代浏览器中，Internet Explorer 的文档对象模型远远超过了 Netscape 在 Navigator 4 中实现的对象模型。IE4 的两个最具创新性的优点是：它可以编写 HTML 文档中的每个元素，内容的改变可以立即反映在页面上。这样，HTML 内容就可以动态显示，而无须浏览器从服务器

更多资源请访问稀酷客(www.ckook.com)

中提取重新整理过的页面。NN4 只实现了这种动态功能的一小部分，它不能编写所有的元素，也不能将页面进行回流处理。它引入的专用层概念在 Navigator 4.x 退出时也被放弃。NN4 中的内联内容不能像 IE4 那样修改，因此 IE4 是一个令人满意的工具。

同时，World Wide Web Consortium (W3C) 开始主持 DOM 标准的协商。脚本编写人员希望确立新标准后，能更加方便地为所有支持新标准的浏览器开发动态内容。最后产生的标准(W3C DOM)能为页面上的每个元素编写脚本，与 IE4 一样。不过它也创造了一个浏览器都没有使用过的全新对象语法。于是，各浏览器支持 W3C DOM 标准的竞赛开始了。

Netscape 公司组建了一个部门 Mozilla.org，来开发一个支持行业标准的全新浏览器。全新的 Navigator 6 以 Mozilla 浏览器的引擎为基础，它融合了 W3C DOM Level 1 的所有内容和 Level 2 的许多内容。Netscape 7 浏览器以 Mozilla 1.01 为基础，而 Netscape 7.1 以 Mozilla 1.4 为基础。2003 年夏，Netscape 的母公司 AOL Time Warner 决定终止进一步开发 Netscape 品牌的浏览器。但是，Mozilla 浏览器的开发工作仍在名为 Mozilla 基金会的独立组织下继续进行。

基于 Mozilla 的浏览器和其他使用该引擎的浏览器(如 Fiefox 和 Camino)继续升级，并且公开发布。Mozilla 引擎提供了对 W3C DOM 标准的最深入支持。

尽管 Microsoft 参与了 W3C DOM 标准的开发，但 IE5 和 5.5 只实现了 W3C DOM 标准的一部分，在某些情况下，只有简单的跨浏览器脚本编程能遵循该标准。Microsoft 在 IE6 中进一步加强了对 W3C DOM 的支持，但忽略了一些重要的部分。尽管 IE6 和 IE7 的发布时间间隔较长，但 IE7 没有提供额外的 W3C DOM 支持，这让 Web 开发人员感到很苦恼。IE8 最终包含了 W3C DOM 规范的一些重大支持，且有一个选项可以让 IE8 变成支持旧脚本的 IE7 兼容模式。

Apple Safari 浏览器的生命期相对较短，但它参与了这个竞争，以提供实质的 W3C DOM 支持。其版本 2 尤其如此，该版本第一次就发布为 Mac OS X 10.4 版的一部分。

这个 DOM 开发和浏览器支持历史相当曲折，直到现在仍充满波折。那么，该如何成功地编写 DOM 脚本程序？尽管目前浏览器能完成众多任务，但在开发过程中肯定会遇到兼容性问题，本书将指导用户解决最常见的问题，并学会解决其他问题。

2.3.4 通过标记打下良好的基础

当页面上的 HTML 标记遵循统一的公共标准时，不同厂商提供的浏览器会采用相似的方式处理这些标记。但为了支持以前数以百万计的网页，浏览器还要尝试猜出设计者使用不同于标准的标记的用意。但这种“猜测”大多没有在 HTML 标准中指定，所以各个浏览器厂商提出了完全不同的解决方案。因此，如果标记很容易解析，错误较少，就会得到较一致的结果，建立更坚固的网站。

最好生成遵循 W3C 规范的标记，例如，http://www.w3.org/TR/html4/上的 HTML 4.01 规范。可使用 W3C HTML Validator (http://validator.w3.org/)等工具依据该标准来检查标记。此外，即使规范比较宽松，也要使自己的标记保持一致，例如，即使规范和 Validator 没有要求，也要关闭所有标记，如 td 和 li。分析并模仿 JavaScript 使用 DOM 方法生成的标记，就可以做到这一点。

2.3.5 层叠样式表

Navigator 4 和 Internet Explorer 4 最先声称自己遵循 W3C 规范(Cascading Style Sheets Level

更多资源请访问稀酷客(www.ckook.com)

1 (CSS1))。这个规范为设计人员提供了一种井然有序的方法以便自定义文档的外观和操作方式(因此最大程序地减少了每个标记中的 HTML)。但在具体实现时，NN4 在许多方面都很粗糙，尤其是合并样式表和表格时。而 IE4 也不完美，尤其是比较样式表在浏览器的 Windows 和 Macintosh 版本中的显示效果时(它们由两个不同的团队开发)，差异非常明显。

CSS Level 2 给标准添加了更多的样式功能；IE6、基于 Mozilla 的浏览器和 Safari 的最新版本都支持 Level 2 的很多方面(尽管并不均衡)，例如 Mozilla 1.8+和 Safari 2+，它们开始支持 CSS Level 3 特性。样式内容的显示效果在浏览器中更加协调一致，这主要归功于样式显示的原则。但复杂的布局仍不时需要加以精细调整，因为浏览器对标准有不同的解释。

JavaScript 在 IE4+、Mozilla 和 Safari 的样式表中起着一定的作用，因为这些浏览器的对象模型允许动态修改与页面内容相关的样式。样式表信息是对象模型的一部分，因此可在 JavaScript 中访问和修改。

2.3.6 标准兼容模式(DOCTYPE 转换)

Microsoft 和 Netscape/Mozilla 都发现，随着时间的推移，它们实现 CSS 特性的方式完全不同于以后发布的标准(通常在工作组成员的多次争吵之后发布)。为了弥补这些差异，重新与标准兼容，主流浏览器厂商决定让页面设计人员选择<!DOCTYPE>标题元素的细节，以确定文档是采用旧方式(有时称为宽松模式)还是标准兼容方式。这种策略在非正式情况下称为 DOCTYPE 转换，在 Internet Explorer 6 和更新版本、用于 Mac 的 Internet Explorer 5 和所有基于 Mozilla 的浏览器中实现。

尽管两种模式的大多数差异都很小，但在 Internet Explorer 6 和以后版本中有一些重大的差异，尤其是样式或 DHTML 脚本依赖于用边框、页边距和补白设计的元素时。Microsoft 最初的盒模型用不同于最终 CSS 标准的方式来衡量元素的维度。

为使受影响的浏览器采用 CSS 标准兼容模式，应在每个文档的顶部添加一个<!DOCTYPE>元素，来指定如下语句之一：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
        "http://www.w3.org/TR/REC-html40/loose.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/REC-html40/frameset.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">

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

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

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```


更多资源请访问稀酷客(www.ckook.com)

但注意，无论是否有<!DOCTYPE>设置，用于 Windows 的 Internet Explorer 旧版本(例如 Internet Explorer 5 或 Internet Explorer 5.5)会忽略标准兼容模式，而使用旧的 Microsoft 宽松模式。但使用标准兼容模式 DOCTYPE 可能会迫使内容和样式表在最新的浏览器中显示相同的效果。

2.3.7　动态 HTML 和定位

在 Netscape 和 Microsoft 的第四代浏览器中，其内部工作方式的最大突破是以 Dynamic HTML (DHTML)这个概念为中心。DHTML 的终极目标是允许文档中的脚本控制内容、内容的定位和内容的显示来响应用户的动作。为此，W3C 组织开发了另一个标准以准确定位页面上的 HTML 元素，这个标准作为 CSS 标准的扩展。CSS 定位建议后来融入到 CSS 标准，它们现在都是 CSS Level 2 的一部分。有了定位功能，就可以在页面上准确指定元素的位置，指定某项是否可见，并指定它在所有相互重叠的各个项中的堆栈顺序。

IE4+支持标准定位语法，让脚本来控制可定位的项。从概念上看，Navigator 4 也支持该标准，但它采用了另一种方法，该方法使用了一个未经批准的全新分层标记。在 Navigator 4 中，也可以通过脚本来控制这种可定位的项，但许多脚本语法都不同于 Internet Explorer 4 中使用的语法。DHTML 设计者高兴地看到，Mozilla 支持 CSS 标准，在语法上更接近 IE4+中的 DHTML 样式属性。

目前人们更感兴趣的是修改网页的内联内容，而无须重新加载整个页面。许多浏览器都支持 W3C DOM Level 1 中的基本标准，包括 IE5+、Mozilla、Safari 和 Opera。在所有这些浏览器上，使用相同的基本语法可以完成许多任务。但仍存在一些问题，本书将讨论它们。

2.4　开发脚本编写策略

如何创建在所有环境下都正常工作的网页？最新一代的浏览器是标准和专用扩展功能的大杂烩。即使给目前的浏览器编写的脚本遵循公共标准，这些脚本代码也可能并不兼容 JavaScript 核心语言和浏览器 DOM 的早期版本。

设计人员面临的真正挑战是确定所编写页面的预期用户。每个新的浏览器版本不仅给用户带来了期望在页面中使用的新功能，还使访问公开页面的访问者进一步分散。用户用浏览器的最新版本替代其旧版本需要数月的时间，一些人甚至很少或从不升级其浏览器，除非新浏览器是通过购买新计算机或操作系统升级得到的。因此，Web 开发人员要为各种浏览器品牌和型号的用户编写页面，包括一些不支持 JavaScript 的最新移动设备。再加上所有非可视化浏览器的用户代理，例如屏幕阅读器和搜索引擎。如何应对所有这些情况？

2.4.1　功能降低和渐进增强

在脚本浏览器历史的这个阶段，大多数 Web 冲浪者都使用至少支持简单 W3C DOM 和 DHTML 功能的浏览器访问网站。但“大多数”并不代表“全部”，所以必须以递增的、增加附加值的方式应用脚本编程，这种方式称为“渐进增强(progressive enhancement)”。这意味着，页面应能将其主要信息传递给不支持脚本编程的浏览器(这些浏览器是为视力或行动有障碍的用户设计的)和内置于最新手机中的功能较少的浏览器。另外，脚本编程可以给使用最新脚本浏览器的访问者提供更愉悦的体验——交互性更强、性能更好、显示效果更具魅力。用户不仅为技

更多资源请访问稀酷客(www.ckook.com)

术发展做贡献，还可以坚持对浏览器脚本编程的独特性见解。

在 Web 演化的早期，许多网站开发人员使用了最新浏览器版本的新功能，而没有考虑未升级其浏览器的用户或从品牌 X 转向品牌 Y 的用户。这意味着，对于许多访问者而言，许多新网站在外观甚至功能上都是有问题的。横幅广告常常解释说："这个网站最好在某个浏览器中查看"。这迫使用户忍受这种混乱局面，或者安装新浏览器。

多年来，网站制作人得到了两个令人激动的新功能，它们改变了获得浏览器支持的方式：在线业务模型和社区道德。把潜在的网站访问者排除在商业网站和非盈利网站之外，在财务方面是不明智的。拒绝辅助技术用户是不公平的——越来越多的行政区允许控告这类行为。Web 开发人员不止关心几个不能运行 JavaScript 的旧浏览器，还有许多其他用户代理也不支持脚本编程，包括搜索引擎、一些移动设备、一些屏幕阅读器和其他辅助 UA。许多人仅仅在其浏览器中关闭 JavaScript 支持功能，而一些公司的防火墙从入站的网页中剔除 JavaScript，这样即使公司中有完全支持 JavaScript 功能的浏览器，也无法看到或运行 JavaScript。Web 统计数据到处都是，但据估计，因为上述原因而不运行 JavaScript 的网站访问者占 10%或更高。即便这个数字只有 1%，其人数也非常庞大。假定打开某个物理店面的前门，让顾客进入，但每 100 个人中拒绝一人进入，这需要花费多长时间？

降低功能是为防止失败而提供回退机制的一种系统设计方法。在 Web 设计中，这意味着如果用户代理不能处理构成页面的高级技术，就提供低级的外观或功能。降低功能的问题是它融入网站制作的方式。以前遇到过挫折的开发人员，以及这些开发人员培训出的新开发人员，继续主要针对新先进、配备精良的浏览器建立网站，在最后一步或改进时，尝试容纳其他浏览器。这种方式给项目的最后阶段增加了许多工作，只是为了容纳少量用户。所以在 Web 的实际设计中，降低功能方式常常因为时间、资金和利他行为的常见限制而牺牲。结果是对于大量潜在的访问者而言，Internet 上的许多网站仍会非常笨拙地失败。

渐进增强是一种充分发挥"降低功能"的新方式。网站首先设计为可以在所有用户代理上运行，再加以改进，如果用户使用更高级的浏览器，就可以加快或丰富用户的访问体验。这确保支持每个读者(人或计算机)的基本功能，也支持新浏览器，且不会忽略较简单的浏览器。它先提供蔬菜，再提供餐后甜点，先提供马，再提供二轮运货马车。它确保汽车有一个有效的引擎，再添加音响系统。

在工作中演示渐进增强原则的一种方式是建立没有 JavaScript 也能工作的页面。在 JavaScript"宝典"中这似乎是一个异端，但其底线是：JavaScript 是用于完成任务的许多工具之一。任何工具都不应抢走真正的表演明星——网站访问者——的风头。

2.4.2 开发层的分离

现代 Web 设计的另一个重要原则是开发层的分离。我们工作的主要层是页面的内容(例如文本和图像名)、HTML 标记中的页面结构、页面在 JavaScript 中的行为和页面在 CSS 中的显示效果(如图 2-2 所示)。这些层的区分越明显，就越便于建立、修改和重用各个任务。

更多资源请访问稀酷客(www.ckook.com)

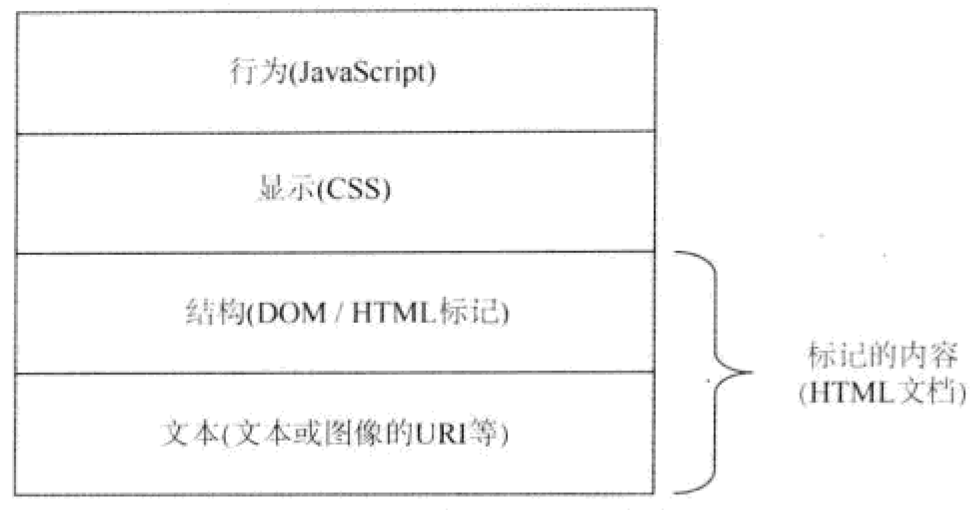


图 2-2　客户端 Web 开发的 4 层

老式网页很容易认出：只要查看 HTML 源代码，就会看到标记中嵌入的 JavaScript 和样式特性。这就像看到房子的水泥墙中嵌入了线路和管道一样。如果希望改变或替换线路，手中就会有一大堆东西。这些水泥墙是多余的，因为不同页面的相同功能需要每次下载相同的标记。因此，它们也会降低下载速度。

如果层是区分开的，则每个部件文件都相当简短，下载速度也很快；如果把相同的样式表或 JavaScript 脚本应用于多个页面(这是很常见的)，响应时间就会进一步缩短，因为浏览器会缓存(存储)已经从服务器中提取的文件。网站开发和重新设计也比较高效，因为几个人可以在不同的层上工作，而不会重复彼此的工作。还可以开发标记、脚本和样式模块，这些模块更容易修改和重用于未来的项目。

几个开发层并非没有关联。实际上正相反，样式表和 JavaScript 代码引用并依赖 HTML 标记结构、标记和特性。它们是手指完美接合起来的手，但它们仍是彼此分离的手。

分离各个层的第一步是给 HTML、JavaScript 和 CSS 分别创建文件(本书中的 HTML 文件已经合并了标记和内容，在 Web 的日常制作过程时，这些文件常常混合了服务器端脚本执行的 HTML 模板和数据库字段)。

层的分离还是一种编写逻辑脚本的方法。JavaScript 可以在一个操作中把一堆标记和文本写入文档，但如果在脚本中把新的结构元素与其内容分开，代码就更容易调试和重用，错误也更少。同样，JavaScript 可直接修改页面上元素的样式，改变其颜色、大小或位置，但如果仅指定样式表中定义的 id 或类，图像设计器就可以改变该显示，而不必修改 JavaScript。

2.4.3　延伸阅读

(1) Debra Chamra 著，Progressive Enhancement: Paving the Way for Future Web Design, http://hesketh.com/publications/articles/progressive-enhancementpaving- the-way-for/

(2) Tommy Olsson 著，Graceful Degradation & Progressive Enhancement, http://accessites.org/site/2007/02/graceful-degradation-progressiveenhancement/

(3) 维基百科上的 *Progressive enhancement*，http://en.wikipedia.org/wiki/Progressive_enhancement

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第3章

选择和使用工具

本章将在计算机上建立一个高效的脚本编写和预览环境，学习在 Web 的什么地方查找有价值的资源。我们还要生成示例 HTML、JavaScript 和 CSS 文件，来演示如何用这些工具创建真正可用的网页。

由于各种个人计算操作系统的操作方式存在差异，这里将介绍两个主流操作系统环境：Windows(95 到 XP)和 Mac OS X。在大多数情况下，无论使用什么操作系统平台(包括 Linux 或 Unix)，JavaScript 编程过程都是相同的，尽管浏览器和操作系统的字体设计可能略有区别，但其信息是一致的。本书中的大多数浏览器输出都是在 Internet Explorer 和 Firefox 的 Windows XP 版本中实现的。如果运行其他浏览器或版本，所显示的内容可能与本书的不大一致。

本章包含哪些内容？

- 如何选择基本的 JavaScript 编写工具
- 如何设置编写环境
- 如何输入简单的脚本来创建网页

3.1 软件工具

学习 JavaScript 的最佳方式是在文本编辑器中给文档输入 HTML 和脚本代码。选择什么编辑器由用户来决定，但这里会提供一些选择原则。

3.1.1 选择文本编辑器

通过本书学习 JavaScript 时，应避免使用 WYSIWYG (所见即所得)网页编写工具，例如 FrontPage 和 Dreamweaver。使用这些工具可能有助于设计页面的内容和布局，但本书的例子主要考虑的是脚本代码(必须手工输入这些内容)，所以不必输入太多的 HTML。

在选择编辑器时，一个重要的考虑因素是它应很容易把标准文本文件保存为.html。在 Windows 中，可将文件默认保存为文本文件，还可将文件扩展名设置为.htm 或.html，这将防止出现许多问题。例如，如果使用 Microsoft Word，该程序就试图把文件保存为二进制的 Word 文件——Web 浏览器不能加载该文件。要把文件保存为.txt 或.html，需要使用 Save As 对话框，这就很麻烦。

PDG

更多资源请访问稀酷客(www.ckook.com)

更重要的是，像 Microsoft Word 这样的字处理程序打开时，会使用许多默认设置，这方便了桌面发布，但可能使程序员受挫，例如给单词插入空格，并自动用直引号替代曲引号。把它的默认值设置为对程序员很友好，会使其不再是有效的字处理程序。

最后，建议不要在 Word 中创建文档，再把它们保存为网页。这会生成一个 HTML 文档，它看上去像是一个字处理文档，但它生成的 HTML 代码是臃肿的、多余的——读者阅读完本书后，应能生成井然有序的、简洁的代码。Word 并不是用于完成这个任务的好工具。

使用非常简单的文本编辑器也是可以的。在 Windows 中，这包括记事本程序或功能更丰富的产品，例如 Visual Studio 或共享编辑器 TextPad。在 Mac OS X 中，其附带的 TextEdit 应用程序也不错。Mac HTML 设计者和脚本编写人员最喜欢的产品包括 BBEdit (Bare Bones Software) 和 SubEthaEdit(www.codingmonkeys.de/subethaedit)。

3.1.2 选择浏览器

学习 JavaScript 所需的另一个组件是浏览器。在浏览器上测试脚本时，不一定要连接到 Internet 上，可以脱机执行所有的测试。所以，可在笔记本电脑上学习 JavaScript，创建美轮美奂的脚本网页——甚至可在大海中飘摇的小船上完成这一切。

使用什么浏览器品牌和版本也由用户来决定。因为本书的教程章节将介绍W3C DOM语法，所以应使用较新的浏览器。下面列出的浏览器都可以用于学习该教程：Internet Explorer 5 或更新版本(Windows 或 Macintosh)、任何基于 Mozilla 的浏览器(包括 Firefox、Netscape 7 或更新版本和 Camino)、Apple Safari 和 Opera 7 或更新版本。

注意：

本书第Ⅲ和第Ⅳ部分的许多示例程序清单演示了只能在特定浏览器和版本上运行的语言或文档对象模型(DOM)特性。检查一下该语言或 DOM 特性的兼容性列表，确保使用了正确的浏览器来加载页面。

3.2 建立编写环境

为了便于测试脚本，文本编辑器和浏览器应同时运行。在检查和更正代码中可能的错误时，需要在编辑器和浏览器之间快速切换。典型的工作流包括如下步骤：

(1) 在文本编辑器中给源文档输入 HTML、JavaScript 和 CSS。

(2) 保存到磁盘上。

(3) 切换到浏览器上。

(4) 执行如下步骤之一：

- 如果是新文档，就通过浏览器的 Open 菜单打开该文件。
- 如果该文档已加载，就把文件重载到浏览器中。

第(2)~(4)步要重复执行，这个 3 步序列称为“保存-切换-重载”序列。在编写脚本时，这个序列必须重复执行，所以很快会成为你的一个习惯动作。根据操作系统的不同，应用程序窗口的安排以及对“保存-切换-重载”序列的影响也是不同的。

在网站制作过程中，在自己的计算机上修改了标记、样式表和脚本后，就要使用 FTP (File

PDG

更多资源请访问稀酷客(www.ckook.com)

Transfer Protocol)程序把它们上传到服务器上。

也可以使用内联编辑器，但现在我们只希望简单一些。在自己的计算机上脱机完成任务的一个优点是，在网站开发阶段，不必有活跃的 Internet 连接。

3.2.1　Windows

不一定要最大化编辑器或浏览器窗口(占满整个屏幕)，才能使用它们。实际上，应调整每个窗口的大小和位置，使两个窗口尽可能大，在使用一个窗口的同时仍能单击另一个窗口。或者，可以让任务栏可见，以便单击所需程序的按钮，切换到相应的窗口上，如图 3-1 所示。分辨率超过 800×600 像素的监视器肯定可以给窗口和任务栏提供更大的屏幕空间。

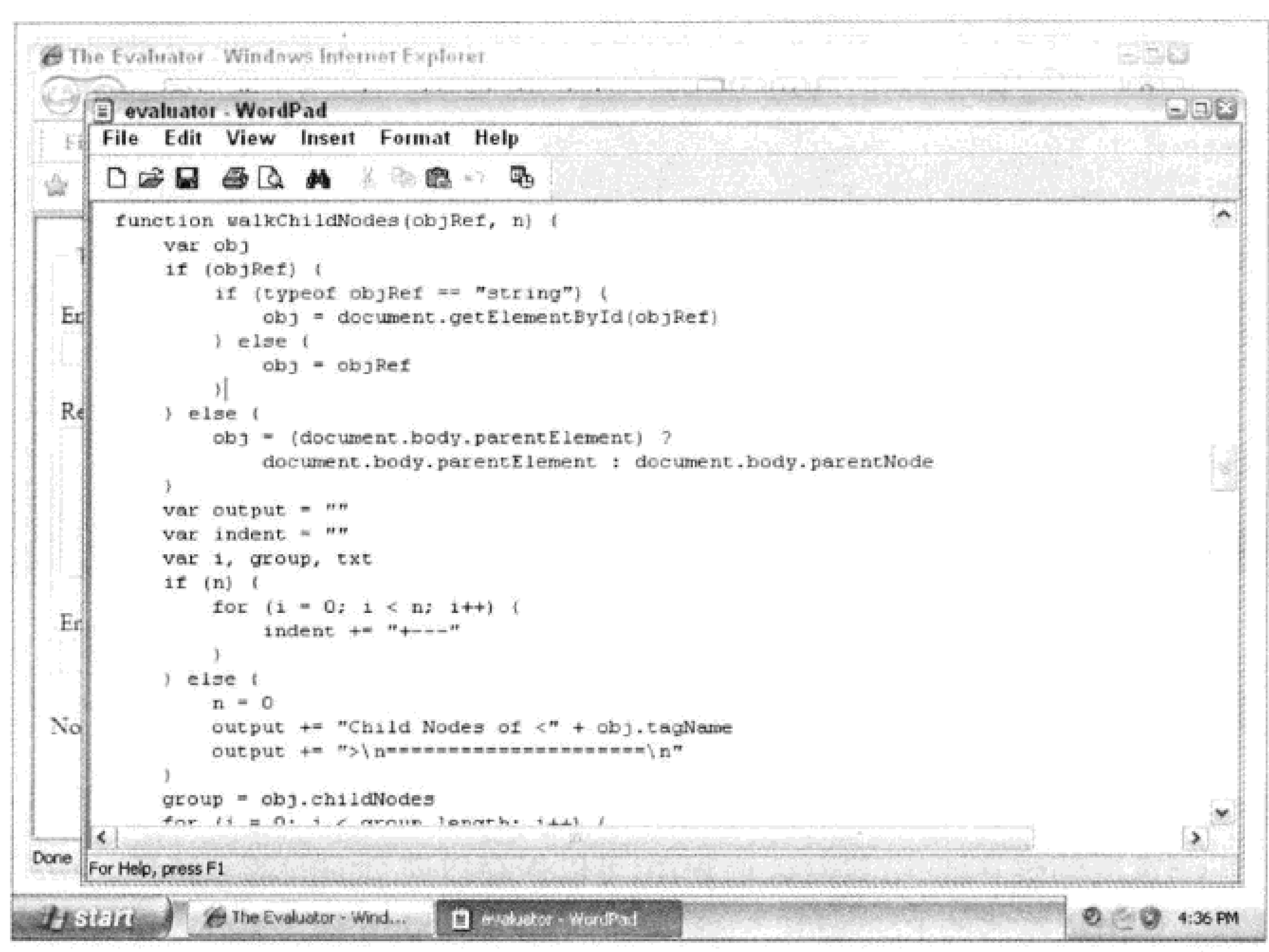


图 3-1　Windows XP 中的编辑器和浏览器窗口排列方式

但实际上，Windows Alt+Tab 任务切换快捷键更容易完成前述的保存-切换-重载序列。如果运行 Windows 兼容的文本编辑器(大都有 Ctrl+S 文件保存快捷键)，就可以用左手在键盘上完成保存-切换-重载序列：Ctrl+S(保存源文件)、Alt+Tab (切换到浏览器)和 Ctrl+R (重载已保存的源文件)。

只要通过 Alt+Tab 任务切换快捷键在浏览器和文本编辑器之间切换，其中一个程序就总是可以显示出来。

3.2.2　Mac OS X

在 Mac OS X 中，通过 Dock 或更便捷的⌘+Tab 键，就可以在文本编辑器和浏览器应用程序之间切换。只要在这两个应用程序中，则按下⌘+Tab 就会切换到另一个程序中，如图 3-2 所示。

有了这个设置，保存-切换-重载序列就很简单了：

更多资源请访问稀酷客(www.ckook.com)

(1) 按下⌘+S (保存源文件)。

(2) 按下⌘+Tab (切换到浏览器)。

(3) 按下⌘+R(重载已保存的源文件)。

要回过头来编辑源文件，可以再次按下⌘+Tab。

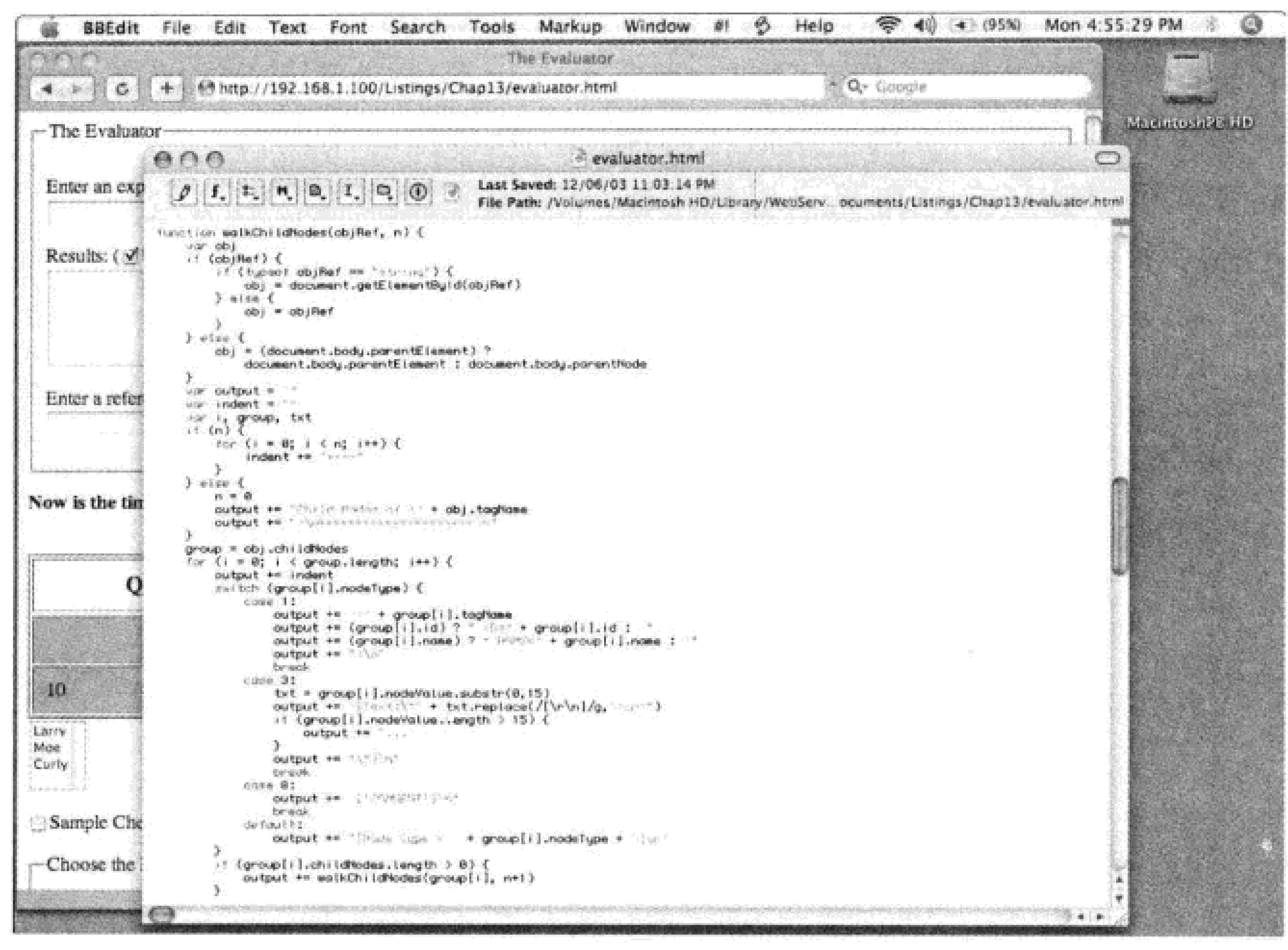


图 3-2 Macintosh 屏幕上的编辑器和浏览器窗口排列

3.2.3 重载问题

在大多数情况下，重载简单的页面，就会立即测试脚本的修订版本。但在重载时，即使改变了源代码，浏览器的缓存(使用其默认设置)也可以保存以前页面的部分特性。为了更彻底地执行重载操作，单击浏览器的 Reload/Refresh 按钮时要按住 Shift 键。另外，可在首选项区域关闭浏览器的缓存，但在常规的 Web 冲浪过程中，该设置对浏览器的整体性能有负面影响。

3.3 验证

检查 HTML，确保其有效，就可以节省很多调试时间。如果标记有误，JavaScript 和 CSS 就可能不像期望的那样工作，因为它们都依赖 HTML 元素(标记)及其特性。标记越接近业界标准规范，在浏览器上就越可能得到一致的结果，这是 Web 标准一致性的终极目标。当然，应总是校对自己的代码，但一个自动验证工具有助于校对庞大而复杂的页面，也可为仍在学习正确标记规则的程序员提供帮助。

World Wide Web Consortium (W3C)编写了 HTML 规范，它也开发出了这样一个验证器。该验证器会根据页面开头的 DOCTYPE 特定规则检查页面。除找出输入错误外，运行验证器的另一个优点是，它有助于学习 HTML 标记的优势。这个验证器是我们的朋友。

更多资源请访问稀酷客(www.ckook.com)

W3C 验证器在 http://validator.w3.org/上。它提供了输入标记的 3 种方式——输入联机 URL，从计算机上传.html 文件，直接把标记复制并粘贴到验证器中。接着单击 Check 按钮，查看结果。一个标记错误常会导致在验证器错误报告中列出几项错误。如果它在标记中找到错误，就进行必要的更正，并再次通过验证器运行代码，直到更正了所有错误为止。

使用验证器可能会让读者对 HTML 规范产生好奇。在 W3C 网站上可以看到 HTML 规范：

- HTML 4.01：http://www.w3.org/TR/html4/
- HTML5：http://www.w3.org/TR/html5/
- XHTML 1.0：http://www.w3.org/TR/xhtml1/

随着时间的推移，读者会在这些网站上发现其他有用的规范和验证器，例如 CSS(Cascading Style Sheets)规范和 Web 可访问性原则：

- CSS 2.1 规范：http://www.w3.org/TR/CSS21/
- CSS 验证服务：http://jigsaw.w3.org/css-validator/
- Web 内容可访问性原则：http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/
- Web 易用性评估工具：http://www.w3.org/WAI/ER/tools/

3.4　创建第一个脚本

为了说明这些工具的用法，下面用 3 个步骤创建一个经典的"Hello, World"页面：先输入普通的 HTML，再添加一些 JavaScript，最后使用 CSS 指定样式。

为简单起见，目前建立的脚本类型会在浏览器加载 HTML 页面后立即自动运行。尽管这里的所有脚本编写和浏览工作都是脱机完成的，但如果把源文件放在一个服务器上，有人通过网络访问它时，页面的行为是相同的。

首先打开文本编辑器和浏览器。再运行 Windows 文件管理器/ Macintosh Finder，创建一个文件夹来保存脚本。

3.4.1　第一步：静态 HTML

创建一个新的文本文件，输入程序清单 3-1 中的标记。

程序清单 3-1　hello-world.html 的源代码

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Hello, World</title>
    </head>
    <body>
        <h1>Hello, World</h1>
        <p>This is HTML.</p>
    </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

在硬盘上把这个文件保存为 hello-world.html，并在浏览器中打开它。之后它会显示在浏览器中，如图 3-3 所示。不同浏览器中显示的页面(如果使用屏幕阅读器，则是声音)可能略有不同，因为此时使用的是浏览器中的默认样式表来显示页面。

图 3-3 显示在浏览器中的静态 HTML 文件

这个页面的 head 包含两个必需的元素：声明 MIME 类型和字符集的 meta 标记和一个 title。body 包含标题和文本段。

3.4.2 第二步：连接 JavaScript

下面给页面添加 JavaScript。在计算机上新建一个文本文件，输入程序清单 3-2。确保所有内容的拼写都正确无误，包括大小写。

程序清单 3-2 hello-world.js 的源代码

```
var today = new Date();
var msg = "This is JavaScript saying it's now " + today.toLocaleString();
alert(msg);
```

把这个文件保存为 hello-world.js，它只包含 3 个 JavaScript 语句：第一个语句获取当前日期，第二个语句构建了一个简短消息，第三个语句显示该消息。

为使这个 JavaScript 程序作用于 HTML 文档，在 HTML 的 head 部分添加一个新的 script 标记(如程序清单 3-3 中突出显示的代码所示)：

程序清单 3-3 修订后的 hello-world.html 源代码

```
...
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Hello, World</title>
      <script type="text/javascript" src="hello-world.js"></script>
   </head>
...
```


更多资源请访问稀酷客(www.ckook.com)

HTML 元素 script 告诉浏览器要与文档合并的脚本文件的类型和名称。因为 src (source)特性不包含路径，所以系统假定它在 HTML 文件所在的文件夹下。与 meta 标记不同，每个<script>标记都必须用</script>结束。

保存新的 JavaScript 文件和修改过的 HTML 文件，在浏览器中重载 HTML 文件，结果如图 3-4 所示。

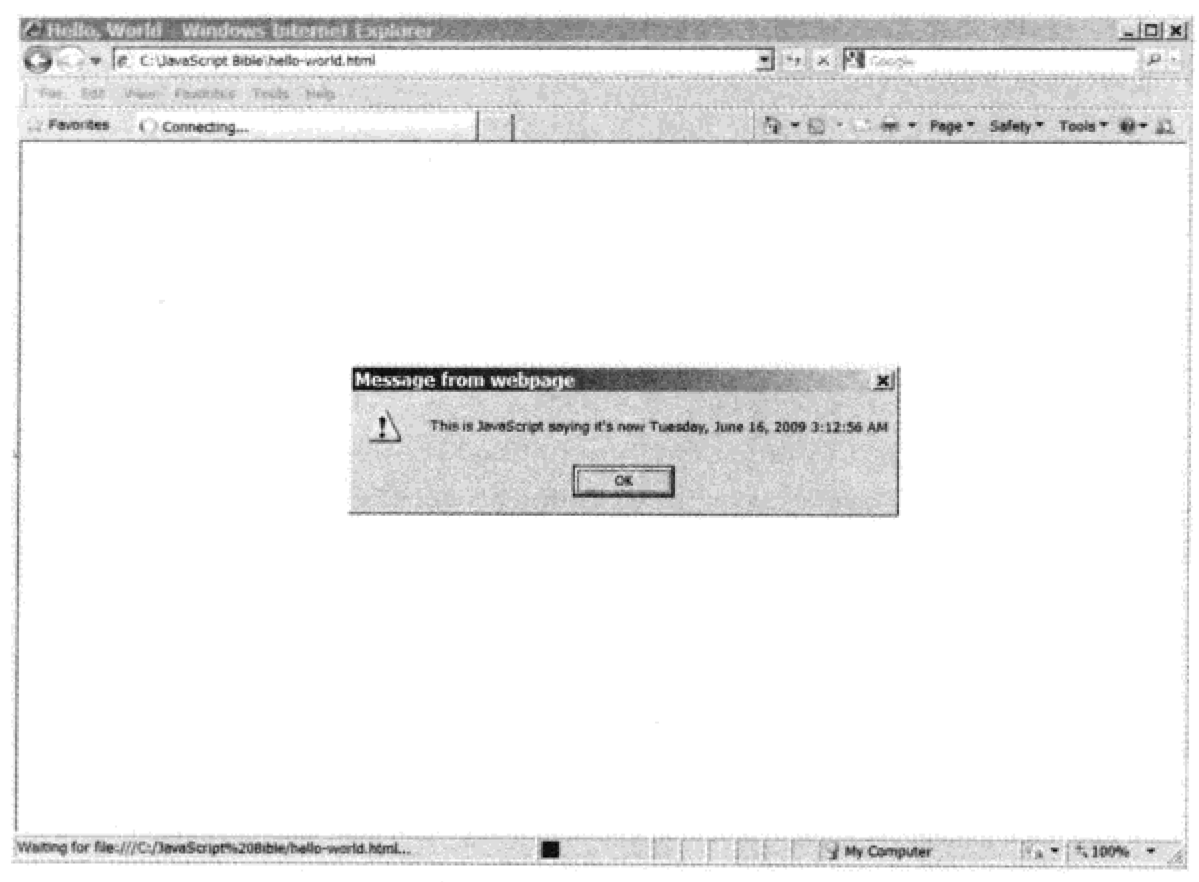


图 3-4　JavaScript 警告

传送给 alert()的文本显示在一个模态对话框中。“模态”表示在关闭该对话框之前，不能在应用程序(浏览器)中执行任何操作。注意 HTML 标题和段落似乎已经消失了！不必担心，只要单击模态对话框中的 OK，它们就会出现。为什么？浏览器会从头至尾地显示 HTML 文件。它遇到 head 中的 script 标记后，就会运行指定的 JavaScript 程序，之后再显示 HTML body。在该例中，由 alert()方法生成的模态对话框暂停了页面其余内容的显示，直到我们响应模态对话框为止。

3.4.3　第三步：用 CSS 指定样式

下面给页面添加一些样式，以便查看 HTML 和 CSS 的交互方式。在计算机上创建一个新的文本文件，输入程序清单 3-4。

程序清单 3-4　hello-world.css 的源代码

```
h1
{
   font: italic 3em Arial;
}
p
{
   margin-left: 2em;
```

更多资源请访问稀酷客(www.ckook.com)

```
    font-size: 1.5em;
}
```

在硬盘上将这个文件保存为 hello-world.css。

这个样式表把特定的字体和颜色应用于页面上的所有 h1 元素，给所有的 p 元素应用了一个左页边距和字号。当然，上面的文档中只有一个 h1 元素和一个 p 元素。

为了使用这个样式表来设置 HTML 文档，在 HTML 的 head 部分添加一个 link 标记(如程序清单 3-5 中突出显示的代码所示)：

程序清单 3-5 修改过的 hello-world.html 源代码

```
...
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Hello, World</title>
        <script type="text/javascript" src="hello-world.js"></script>
        <link type="text/css" rel="stylesheet" href="hello-world.css">
    </head>
...
```

link 元素告诉浏览器要与文档合并的样式表的类型、关系和名称。注意，link 与 meta 标记类似，但与 script 标记不同，link 是一个空标记，没有单独的结束标记。由于 src (source)特性没有包含路径，所以系统假定它在 HTML 文件所在的文件夹下。每个<script>标记都必须用</script>关闭。

保存这些文件，在浏览器中重载 HTML 文档(单击模态对话框中的 OK 按钮)，结果如图 3-5 所示。

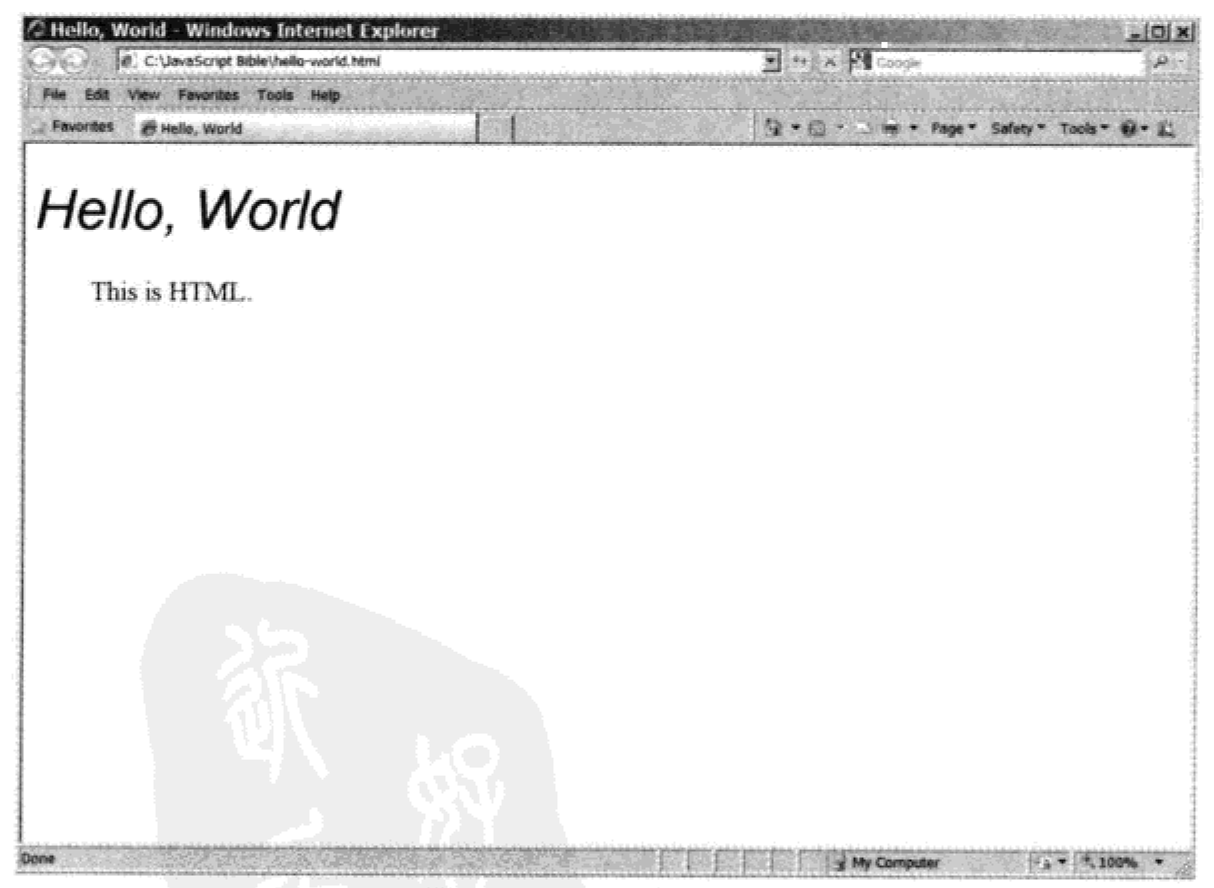


图 3-5 设置了样式的页面

更多资源请访问稀酷客(www.ckook.com)

第 4 章

JavaScript基础

第一次在 Web 浏览器环境中学习 JavaScript 的用法时，很容易混淆 JavaScript 语言的对象和使用 JavaScript 控制的文档对象。将语言和文档对象模型(Document Object Model，DOM)分开是很重要的，这有助于在设计 JavaScript 增强页面时做出重要决策。如果将来把 JavaScript 用于其他对象模型，例如服务器端编程或编写 Flash 动画，你就会赞同这种分离的做法。语言的所有基础知识在所有环境中都是相同的，只有对象是不同的。

本章介绍核心 JavaScript 语言的许多方面，尤其是需要把脚本部署在访问者使用各种浏览器访问的页面中。通过学习本章，你将对该语言有更深入的了解。幸好，现代浏览器在支持 JavaScript (ECMAScript)标准方面更具一致性，使浏览器在 JavaScript 方面的差异大大减少了。第III部分将介绍 JavaScript 核心语言语法的细节。

4.1 合并 JavaScript 和 HTML

脚本浏览器提供了几种方式，以便在 HTML 文档中包含脚本或脚本元素。在目前的最佳实践中，并非所有的方法都值得推荐使用，但学习旧技术，有助于在网络上遇到旧代码时理解它们。

4.1.1 <script>标记

使用<script></script>标记集可以把 JavaScript 代码融入 HTML 文档，该标记集通过 type 特性指定了脚本语言。在文档中这种标记集可以有任意多个。建议将所有脚本都设置为链接到 HTML 上的外部文件：

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

还可以把 JavaScript 直接包含在<script>...</script>标记中，将这些代码嵌入 HTML：

更多资源请访问稀酷客(www.ckook.com)

```
<script type="text/javascript">
// JavaScript code goes here
</script>
```

把外部 JavaScript 链接到 HTML 上比把脚本直接嵌入标记具有明显的优势。开发层的分离原则鼓励把文档的不同部分分开，一般放在多个链接文件中。与把所有技术混合放在一个文件中相比，这么做会使开发更清晰、下载速度更快，代码的模块化程度更高、重用性更高，更容易修改。

1. 链接到脚本库(.js 文件)

当同一个脚本需要运行在站点的多个页面上时，链接到外部 JavaScript 文件上的优点会非常明显。如果脚本嵌入 HTML 标记，每个页面都会增加冗余的内容，若修改了脚本，就必须在多个文件中重复这些修改。相反，若链接到外部文件上，则只需给每个 HTML 文件添加一行，外部 JavaScript 文件也可以只修改一次，就会影响包含它的每个页面。另一个优点是，把脚本代码与 HTML 文档分开，就不必担心隐藏注释或 CDATA 段技巧(如后面所述)。

这个外部脚本文件只包含 JavaScript 代码，不包含<script>标记，也不包含 HTML。所创建的脚本文件必须是纯文本文件。其文件扩展名可以任意指定，但一般使用.js。为了告诉浏览器在常规 HTML 文件的某个特定位置加载外部文件，需要给<script>标记添加 src 特性，如下所示：

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

如果要加载多个外部库，就可以在文档的开头包含一系列此类标记集。

注意<script></script>标记对是必需的，尽管在它们之间什么都没有。起始标记包含 src 特性时，不要把任何脚本语句放在起始和结束标记之间。

在 src 特性中如何引用源文件取决于其物理位置和 HTML 编码样式。在前面的例子中，.js 文件假定位于包含标记的 HTML 文件所在的目录下。也很容易链接到位于服务器上其他目录或其他域上的 JavaScript 文件。与 HTML 文件一样，如果要引用绝对 URL，文件的协议就是 http://。

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

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

给文档使用脚本库的一个重要先决条件是 Web 服务器软件必须知道如何把.js 扩展名文件映射到 application/x-javascript 的 MIME 类型上。测试 Web 服务器，根据需要调整服务器配置。现代服务器大都能正确识别.js 文件。

当用户浏览链接到外部脚本库的页面中的源代码时，即使浏览器把所加载的脚本看作当前文档的一部分，.js 文件的代码也不会显示在窗口中。但是，.js 文件的名称或 URL 通常是可见的(其形式与它在源代码中的形式相同)。任何人都可以在浏览器中(使用 http://协议)打开该文件，来查看.js 文件的源代码。换言之，外部 JavaScript 源文件不再隐藏，而是像直接嵌入 HTML 文件中的 JavaScript 一样显示出来。浏览器下载的、用于显示网页的任何信息都不会隐藏起来。

2. 指定 MIME 类型和语言

每个起始<script>标记都应指定 type 特性。script 是一个一般元素，它指定它包含的语句应

PDG

更多资源请访问稀酷客(www.ckook.com)

解释为可执行的脚本，而不应显示为 HTML。该元素用于包含浏览器知道的任何脚本语言——换言之，它包含一个解释器。例如，Internet Explorer 包含了 JScript (Microsoft's version of JavaScript 的 Microsoft 版本)和 VBScript 的解释器。

```
<script type="text/javascript" src="example.js"></script>
<script type="text/vbscript" src="example.vb"></script>
```

这些脚本都由 IE 解释，但不支持 VBScript 的浏览器(如 Mozilla)只能解释 JavaScript 文件。

3. 指定语言版本

很少需要指定要在文档中使用的 JavaScript 版本。如果脚本依赖的对象和方法是某些旧浏览器不支持的最新 JavaScript 版本，最好直接测试浏览器是否提供了该支持。例如：

```
// if this browser isn't DOM-aware, exit gracefully
if (!document.getElementById) return;

// now it's safe to use the DOM method
var oExample = document.getElementById("example");
```

但是，并不总是可以在运行期间测试语言特性是否存在。如果新特性内置于 JavaScript 的语法中，则脚本中存在该特性，就会让解释器在编译到脚本中不能执行 if 测试的地方停下来。

例如，考虑在 JavaScript 1.6 中引入的 E4X 语法(参阅第 20 章)：

```
var xAuthor = <name>
                  <first>George</first>
                  <last>Sand</last>
              </name>;
```

如果在一个脚本中包含这个语法，而由不知道 E4X 语法的浏览器运行该脚本，则脚本不会运行，在某些情况下，也不会给访问者或开发人员显示有意义的错误消息。该脚本中的任何代码都不会运行，即使浏览器能编译这些代码。

可以给 script 元素使用特定版本的 type 特性，来隔离新语法：

```
<script src="example-all.js" type="text/javascript"></script>
<script src="example-1-6.js" type="text/javascript;version=1.6"></script>
```

或：

```
<script type="text/javascript">
   // script for all browsers goes here
</script>
<script type="text/javascript;version=1.6">
   // script for browsers that know JavaScript 1.6 and later
</script>
```

但注意，把版本指定为特殊的标记并不是标准的，所以并不是所有的浏览器都支持它。对于 E4X 语法，Mozilla 给 type 特性引入了另一个特殊标记：e4x=1 表示“E4X support = true”。

```
<script src="example-e4x.js" type="text/javascript;e4x=1"></script>
```


更多资源请访问稀酷客(www.ckook.com)

于是就出现了问题：如果部分脚本代码不能用于旧浏览器，则该在这些旧浏览器上使用什么代码代替新代码？如果编写这些替代代码，为什么不只编写旧代码？在同一个脚本中用两种不同的方式编写相同的逻辑有什么好处？使用先进的语言特性时，这是一个有必要考虑的问题。

4. 废弃的语言特性

type 特性是 HTML 4 中的<script>标记所必需的，HTML 4 在 1998 年正式发布。早期的浏览器能识别 language 特性，而不是 type。language 特性被废弃了，这表示它是一个过时的语言特性，已被更新的结构替代，在 HTML 的未来版本中可能不再使用。浏览器仍能识别它，只是为了支持旧网页，但没有理由继续在目前符合业界标准的网站上使用它。这里包含它是为了让读者在旧标记中遇到它时，能够理解其含义。

language 特性允许脚本编写人员给特定的 JavaScript 版本或其他语言(对于 Internet Explorer 就是 VBScript)编写脚本。例如，Navigator 3 内置的 JavaScript 解释器知道 JavaScript 1.1 版本，Navigator 4 和 Internet Explorer 4 包含 JavaScript 1.2 版本。对于以后的 JavaScript 版本，可在语言名称的后面加上版本号(没有任何空格)来指定语言的版本，例如：

```
<script language="JavaScript1.1">...</script>

<script language="JavaScript1.2">...</script>
```

从 Internet Explorer 5、Mozilla 和符合 W3C DOM 标准的浏览器开始的浏览器都支持 type 特性。如果需要告诉旧浏览器所使用的脚本语言及其版本，就可以在<script>标记中指定 type 和 language 特性，因为旧浏览器会忽略 type 特性：

```
<script type="text/javascript" language="JavaScript1.5">...</script>
```

当然，如果依赖 JavaScript1.5 的特性，就表示不再使用旧浏览器。此时，只需使用新方法，在脚本的开头测试现代方法即可。

5. 专用的 for 和 event 特性

Internet Explorer 4～7 提供了<script>标记的一个变体，它把脚本语句限制为一个特定的对象和该对象生成的事件。除了 language 特性外，该标记还必须包含 for 和 event 特性(它们都不包含在 HTML 规范中)。赋予 for 特性的值是所需对象的引用，它常常只是赋予对象的 id 特性的标识符。event 特性是希望脚本响应的事件处理程序名称。例如，如果设计了一个脚本，对 ID 为 myParagraph 的段落中的 mousedown 事件执行某个动作，脚本语句就包含在如下标记对中：

```
<script for="myParagraph" event="onmousedown" type="text/javascript">
...
</script>
```

标记对中的语句仅在触发事件时执行。不需要定义函数。

因为 for 和 event 特性是 Microsoft 的专用特性，这种把对象的事件与脚本绑定起来的方式仅确保，在发生指定的事件时，Internet Explorer 会执行脚本。

甚至在当时，script for 标记也不成功。其特性包含许多源代码，放在每个对象脚本的开头，所以把脚本语句与多个对象和事件链接起来时，它从来不是高效的方式。另外，非 Internet

更多资源请访问稀酷客(www.ckook.com)

Explorer 和 Internet Explorer 4 以前的浏览器会在加载页面上执行脚本语句，这使这个专用语言特性在跨浏览器脚本中非常不实用。

4.1.2 旧式内联 JavaScript

在以前的“标记”Web 开发中，常把 JavaScript 语句直接插入 HTML 标记：

```
<select id="nations" name="nations" onchange="doSomething(this);">
```

在这个例子中，控制对象 nations 的 change 事件设置为触发所提供的 JavaScript 语句：一个自定义函数调用。本书将在其他地方说明其工作原理，以帮助理解旧代码，但现在像这样合并标记和脚本一般会被认为“老土”。

采用这种方式将 JavaScript 插入标记有许多缺点：标记和脚本代码混合在一起，则独立修改标记或脚本就非常不便，代价很高；所下载的标记很多，有些是多余的，不能充分利用外部链接脚本的集中化模块，运行各种 JavaScript 版本的用户代理如果不能处理代码，就不能避免脚本。

在现代页面上，这个标记应简化为：

```
<select id="nations" name="nations">
```

事件处理程序则完全在 JavaScript 中进行：

```
AddEvent("nations", "change", doSomething);
```

其中 AddEvent()函数可在不同的浏览器中采用不同的事件模型，可与标记完全分开，以保护不能处理脚本的用户代理，并可以通过防御式 if 测试进行防火墙保护，防止不能很好处理现代代码的旧解释器调用该函数。

4.1.3 容纳不支持 JavaScript 的用户代理

有几类用户代理不能处理或不支持 JavaScript，据统计，这些用户代理在互联网上目前约占 10%～15%。这些用户代理包括搜索引擎蜘蛛、移动设备浏览器；另外，政府和公司防火墙中的浏览器出于安全的原因会剔除脚本、辅助技术，其他一些浏览器出于各种原因其用户会关闭脚本功能。目前仍有少量旧浏览器不能运行 JavaScript，或者仅支持其早期版本，但大多数不使用 JavaScript 的互联网访问者使用的都是较新的软硬件。

Web 开发人员应认识到，JavaScript 是一个选项，而不是已知条件，才能确保页面在没有 JavaScript 的情况下也能运行。我们的任务是使用 JavaScript 改进访问者的体验，而不是提供页面必须有的重要元素，没有这些元素，页面就会崩溃或彻底失败。必须确保 HTML 标记提供了所有的基本内容、超链接和表单字段，而服务器端脚本应完成客户和服务器之间的对话，这是 HTML 请求模型的核心。于是，在 JavaScript 运行时，它可以改变页面，美化用户界面，缩短响应时间。这是现代 Web 脚本编程的双赢模型。

问题在于，如何把 JavaScript 添加到 HTML 文档中，使页面在不运行脚本解释器、没有脚本解释器或脚本解释器的版本较旧的情况下仍能正常运转？这有几个策略：

更多资源请访问稀酷客(www.ckook.com)

- 脚本一开始就测试现代 DOM 方法(如 document.getElementById)是否存在，以避免使用 JavaScript 解释器的旧版本。
- 把 JavaScript 导出到外部文件中，用 script 标记把它们链接到 HTML 文档上。不运行脚本解释器的用户代理甚至不会执行 script 标记。
- 由上面可以推知，在 HTML 标记的 event 特性中忽略 JavaScript，但使用 JavaScript 把事件处理程序赋予页面元素。
- 如果无法避免在 HTML 文档中包含 JavaScript，则把脚本放在 HTML 注释中，使不支持脚本的浏览器不显示脚本。

下面详细讨论最后一点。

1. 注释掉 HTML 中的脚本

不支持脚本的浏览器不知道<script>标记。一般情况下，浏览器会忽略它们不理解的标记。如果标记只是一行 HTML，那就很好，但<script>...</script>标记对可以包含任意多行脚本语句。旧的简单浏览器不知道有</script>结束标记，因此，它们自然倾向于显示起始<script>标记后面的所有内容。任何访问者看到 JavaScript 代码出现在页面中，肯定会感到迷惑。

但是，可以试着使用一种技术，它会使大多数不支持脚本的浏览器忽略脚本语句：用 HTML 注释标记把<script>标记对中的脚本语句包含进来。HTML 注释用<! —开头，用— >结束。因此，应使用以下格式将这些注释标记嵌入脚本：

```
<script type="text/javascript">
<!--
Script statements here
//-->
</script>
```

JavaScript 解释器知道应忽略以 HTML 起始注释序列<!--开头的代码行，但在处理结束序列时需要一些帮助。HTML 注释的结尾以 JavaScript 注释序列(//)开头，这会告诉 JavaScript 忽略该行，但不支持脚本的浏览器看到结束 HTML 符号后，会开始用下一个 HTML 标记或文档中的其他文本显示页面。旧浏览器识别</script>标记，所以会忽略该标记，显示该标记后面的内容。

2. 把 XHTML 中的脚本注释为字符数据

如果用 XHTML 标记文档，注释语法就大不相同：

```
<script type="text/javascript">
<!--//--><![CDATA[//><!--
    // JavaScript code goes here
//--><!]]>
</script>
```

这相当古怪，需要好好解释一下。

XHTML 解析器不能接受不是元素开头的<符号和不是 HTML 实体开头的&符号，但这些字符是常见的 JavaScript 操作符。解决方案是把脚本语法放在所谓的 CDATA (发音为“see-day-tah”)段中：

更多资源请访问稀酷客(www.ckook.com)

```
<![CDATA[
    XHTML permits any character here including < and &
]]>
```

接着就可以添加 HTML 注释，禁止旧浏览器显示脚本了：

```
<script type="text/javascript">
<!--<![CDATA[
    // script statements here
//--]]>
</script>
```

但是，这里不能阻止旧浏览器显示脚本，因为 XHTML 也使用<!—和-->注释标记，支持 XHTML 的浏览器会忽略 CDATA 段，且不执行脚本。所以只有在 CDATA 开始之前关闭注释标记，才能符合 XHTML 的要求：

```
<!-- --><![CDATA[
```

使用//注释语法，让大多数 HTML 解析器忽略该行的剩余部分：

```
<!--//--><![CDATA[
```

一些 HTML 解析器把//后面的行看成注释，而其他浏览器用-->关闭注释区域，所以不能让 CDATA 起始标记显示出来。这里使用的技术是让 HTML 解析器认为它是一个结束标记：

```
<![CDATA[>
```

但为了不让 XHTML 将其视为结束标记，应添加另一个注释标记：

```
<![CDATA[//>
```

并再次打开块注释标记，禁止某些浏览器尝试把 JavaScript 代码显示为纯文本：

```
<!--//--><![CDATA[//><!--
```

最后给 HTML 解析器注释掉 CDATA 结束标记：

```
//--]]>
```

这是一些 Web 开发人员在运用多个标准时必须掌握的内容。把 JavaScript 导出到外部文件中要简洁得多！

3. noscript 标记

noscript 元素与 script 相反。如果关闭了脚本功能，或者浏览器不支持在文档中通过 script 元素调用某种脚本语言，支持 JavaScript 的浏览器就会显示 noscript 块的内容(注意，根本不支持脚本、不能识别与脚本相关的标记的旧浏览器会同时显示 script 和 noscript 块的内容，因此要注释掉 script 块的内容。因此，所有不支持脚本的浏览器都会显示 noscript 块)。

在 noscript 标记对中可以显示任意标准 HTML，它是一个块级元素，可包含段落、列表、分部和标题等。

更多资源请访问稀酷客(www.ckook.com)

noscript 的常见用法是提供不支持脚本时要显示的 HTML 内容——一个消息，例如“如果运行 JavaScript，这个页面会更有趣”，或者一个页面的 HTML 超链接，该页面提供了通过 Javascript 插入的内容。

近年来，noscript 的这个用途由在 HTML 中仅提供非脚本标记的方式替代，这种方式用脚本替代或改进了结构。不是告诉访问者应升级浏览器或启用脚本功能，而是一开始就假定他们使用需要的用户代理来查看页面，并为他们提供相当好的、功能完整的页面；再使用脚本给支持 JavaScript 的浏览器提供更好的页面。今天的用户似乎不大可能使用故意关闭脚本功能的浏览器，或者在剔除了脚本的公司防火墙后面运行，或者使用不支持脚本的移动设备，却不知道互联网必须通过 JavaScript 才能提供额外的功能。升级警告可能在 90 年代是有用的建议，但今天它似乎有点蔑视他人，是多余的。

程序清单 4-1 是在支持脚本和不支持脚本的浏览器上显示不同内容的例子。HTML 包含一个所有人都能看到的普通段落，另一个段落放在 noscript 元素中，只有通过不支持脚本、关闭了 JavaScript，或者运行不支持 DOM 的旧 JavaScript 解释器的用户代理才能看到它。附带的 JavaScript 插入了另一个段落，如果用户代理运行的 JavaScript 解释器支持 W3C DOM 方法 appendChild()和 createTextNode()，就可以看到这个段落。

程序清单 4-1 在支持脚本和不支持脚本的浏览器上显示不同内容

HTML: jsb-04-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Presenting Different Content for Scriptable and
      Nonscriptable Browsers</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-04-01.js"></script>
   </head>
   <body>
      <h1>Presenting Different Content for Scriptable and Nonscriptable
      Browsers</h1>

      <p>This text will be revealed to everyone.</p>

      <noscript>
         <p>Only user agents that don't or can't support W3C DOM
         scripting will reveal this text.</p>
      </noscript>

   </body>
</html>
```

JavaScript: jsb-04-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', addText);
```


更多资源请访问稀酷客(www.ckook.com)

```
function addText()
{
      // ensure a DOM-aware user agent
      if (!document.appendChild || !document.createTextNode) return;

   // create a text node
   var oNewText = document.createTextNode("Only DOM-aware user agents ↵
running JavaScript will reveal this text.");

   // create a paragraph
   var oParagraph = document.createElement('p');

   // insert the text into the paragraph
   oParagraph.appendChild(oNewText);

   // insert the paragraph at the end of the document
   document.body.appendChild(oParagraph);
}
```

4.1.4　隐藏脚本

JavaScript 编程新手的一个常见问题是如何对页面的访问者隐藏脚本，也许是为了保护某些代码。可惜这是不可能的。页面要运行，客户端 JavaScript 必须随页面一起下载，因此在浏览器的源代码视图中是可见的。外部链接脚本的 URL 可以粘贴到浏览器的地址栏中。

如果担心其他脚本编写人员窃取自己的脚本，可在源代码中添加版权声明。该声明不仅可以在用户的页面中可见，在窃取者的脚本中也可以看到。这样，当某人逐字窃取脚本时，用户就很容易发现。当然，总是有人能复制脚本，并删除脚本拥有者的版权声明。

按照这些原则，我们最大的希望就是混淆代码，使代码阅读起来很难、很耗时、很无聊。混淆技术包括删除回车换行符、不必要的空格和制表符，使用无意义的函数和变量名，使用变量保存常见的特定对象，使用布尔逻辑替代 if 和其他分支结构，编写自动生成的代码，使用递归，把逻辑序列分割为多个嵌套的函数调用，用 10 进制、16 进制或 8 进制表达式给文本编码。但目前，全局替代、仔细阅读和耐心可以解读任何基于文本的脚本中的逻辑。JavaScript 混淆器最卓有成效的用法是使用尽可能少的字符生成脚本的“压缩”副本，以快速下载大量脚本。但是，即使是这个用途也存在问题，因为外部脚本在第一次下载后，浏览器就会缓存它。

如果主要关心的是使源代码私有化，就应考虑使用另一种把源代码编译为机器码的编程语言。因为存在把机器语言转换回源代码的反编译器，所以编译可以看作另一种形式的混淆，但这肯定会减少愿意解码它的人数。

然而，几乎不可能隐藏重要的信息。即使优秀的程序员不能阅读脚本，他们也可以看出软件的执行方式，并编写一个新脚本来完成同样的任务。测试隐藏脚本所花费的时间可以用于编写令人兴奋的新代码！

建议用户考虑与此相反的方式。不是尝试隐藏编程代码，而是公开它。给源代码添加一些有益的注释，在博客和编程网站上发布它，帮助尽可能多的人学习和使用它(添加一个 Creative Commons 版权，鼓励人们复制这些代码，并免费公开使用)。如果得到了技术高超、乐于助人的程序员的名声，人们就会向你请教更多知识。

更多资源请访问稀酷客(www.ckook.com)

4.1.5　给不同的浏览器编写脚本

每个 JavaScript 程序员都非常关注跨浏览器的兼容性。大多数最新的浏览器都能很好地提供编写脚本的最少公共特性，但仍然必须考虑从最先进的浏览器到最落后的浏览器对 JavaScript 的支持。甚至最新的 Firefox 和最新的 Internet Explorer 之间的差异也非常重要。规划兼容性的第一步就是为各类用户代理确立目标。

1. 建立目标

大多数网页的目标都可以归结为“发布特别的内容”。一个网站与其他网站的区别就是它发布的特定内容以及其“外观和操作方式”——页面的图形化外观和访问者用于请求更多信息的用户界面的样式。

渐进增强的原则指出，首先应把服务器的内容通过 HTML 标记发布到客户端上，不必依赖 JavaScript 的核心功能。网站应是可供使用的、可以导航的，其公共内容会展示给所有的用户代理，包括不支持脚本的用户代理，例如搜索引擎。之后，再使用 JavaScript 改进访问者的体验，访问者使用支持 JavaScript 的浏览器访问网站会更便捷、更有趣。

在规划过程中最后考虑 JavaScript，而不是最先考虑 JavaScript 的一个结果是，我们原本以为需要使用 JavaScript 才能实现的许多网站功能，常常完全可以使用其他获得更广泛支持的技术来实现。JavaScript 应只用于执行超出浏览器核心功能(例如 HTML 显示和 CSS 样式)的功能。例如，无论 JavaScript 是否添加了某些有趣的成分，导航链接都应标记为有效的超链接，多级菜单、弹出式对话框和滚动图像常常可以仅使用 HTML 和 CSS 来实现，因此可供更广泛的访问者使用。

Ajax (Asynchronous JavaScript and XML)是一个有趣的技术。开发人员甚至客户常常选择 Ajax 发布内容，仅因为 Ajax 比通常的页面导航更有用、更快捷高效。读取服务器的内容需要花费一定的时间；如果页面中的所有图像都已下载并缓存，则读取新页面的开销就仅是读取 HTML 标记，而 HTML 标记是页面中最小的一块。导航到各个页面上，显示不同的内容就意味着，访问者很容易为特定的内容块添加书签，并共享其链接，但 Ajax 检索的内容不是这样。仅因为 JavaScript 可以用于某个特定的目的，并不意味着必须或应该使用 JavaScript。

一旦使用服务器端技术和 HTML/CSS 下载技术详细规划好核心页面或网站，就可以用 JavaScript 设计出用户界面的其他部分——餐后甜点！

JavaScript 可用于给静态显示内容添加动画，用 Ajax 下载的附加内容来替换导航链接，通过拖放操作增强按钮单击的编辑过程，装饰表单等。添加这些“有了更好”的部分时，要确保通过 JavaScript 提供的所有内容不需要编写脚本就可以访问。如果给网站添加的重要内容只能使用 JavaScript 来检索，就做得过火了。后退一步，给核心网站添加新内容，再重新进行 JavaScript 规划。

2. 选择自己的战场

任何公共网站都可以使用任何已有的用户代理来查看。支持 JavaScript 的用户代理非常多，从早期的基本用户代理到高级的用户代理，所以为每种有 JavaScript 解释器的浏览器编写脚本就非常困难，基本上是不可能的。因此，Web 开发规划的一部分可以看作“选择自己的战场”

更多资源请访问稀酷客(www.ckook.com)

或者“一种分类”——在要支持的解释器和不支持的解释器之间、要使用的语言特性和避免使用的语言特性之间画出一条界限。根据这种方式，不应把时间花费在编写要在古老浏览器上运行的脚本，而只是禁止它们进入这个“战场”，集中注意力给现代浏览器开发代码。

建议在设计网站时，所有的核心内容和功能都通过从服务器上传来的 HTML 来提供，禁止老式浏览器运行脚本。放心吧，每个人都能使用该网站。把自己支持的 JavaScript 解释器限制为现代流行的浏览器，而排除老式浏览器，就不必为非常少量的访问者执行痛苦的开发工作，这个决策对确保在最后期限之前构建好网站，而且使预算在合理范围内都是有益的。

例如，本书的大多数脚本都开始于一个简单的 if 测试：

```
if (!document.getElementById) return;
```

换言之，“如果在这个浏览器中运行的 JavaScript 解释器不知道 document 对象的 getElementById 方法，就停止运行这个脚本”。这个语句把不知道现代 DOM 方法的老式浏览器排除在外，允许仅为一个用户群编写大多数代码。另一方面，应避免使用禁止大多数当代浏览器运行的高级 JavaScript 语法，例如 E4X。中间部分——支持 ECMAScript 3 和 W3C DOM2 的浏览器——是目前人们使用的大多数浏览器，它们具备相当一致的语言支持。

在规划应用程序或网站时，发布一个可以通过所有用户代理查看的内容流要比在不同模式下重复该内容更有意义。过去，有时会使用后一种方法，例如给使用框架的网站和不支持框架的并行版本使用该方法。万一所创建的应用程序或网站可以采用多条路径查看相同的内容，但不要忘了，随着网站的发展，其维护工作会随之增加。你有时间和预算，并打算更新所有的路径吗？根据经验判断，无论新网站的设计人员有多好的心愿，能妥善维护的网站可以快速降低维护工作的复杂性。

提供多个渠道未必意味着重复工作。网站可以标记为导航到不同的页面上，发布不同的内容，但如果支持脚本的编写，网站就可以使用 Ajax 发布相同的内容。因为这两种渠道都显然来自于服务器，所以可以使用 Ajax 请求每个独立页面所请求的相同数据流。服务器上的数据改变时，例如修改了数据库中的文本，则两个独立的页面和注入的 Ajax 内容也随之改变。

3. 对象检测

实现浏览器版本分支的方法称为对象检测，其原理很简单：如果一个对象类型存在于浏览器的对象模型中，它就可以安全地执行处理该对象的脚本语句。

在许多旧脚本中，对象检测的一个例子是使用 images 集合。只有运行在浏览器上的 JavaScript 版本能把图像识别为对象，脚本才能改变图像的 src 特性——用另一个从服务器上读取的图像替代它。实现图像的对象模型总是包含一组属于 document 对象的图像对象。document.images 数组总是存在，页面上没有图像时，该数组的长度就为 0。因此，如果将图像变换语句放在一个 if 结构内(这个 if 结构仅允许包含 document.images 数组的浏览器执行其中的语句)，则旧式浏览器就会跳过这些语句：

```
function imageSwap(imgName, url)
{
   if (document.images)
   {
```


更多资源请访问稀酷客(www.ckook.com)

```
        document.images[imgName].src = url;
    }
}
```

把对象检测用于测试需要使用的同一对象时，它最有效。把它用于推测未经测试的浏览器版本和对语言特性的支持时，就会出问题。

例如，Internet Explorer 4 引入了一个 document 对象数组 document.all，它常常用于建立对 HTML 元素对象的引用。但 Netscape Navigator 4 没有实现该数组，而是有一个文档级的数组对象 layers，Internet Explorer 4 不支持 layers 对象。可是许多脚本开发人员把这些数组对象是否存在作为浏览器版本的决定因素，而不是作为寻找这些对象的先决条件。他们设置全局变量，如果 document.all 存在，就把最小版本指定为 Internet Explorer 4，如果 document.layers 存在，就把最小版本指定为 Netscape Navigator 4。因为无法知道某浏览器的后续版本是否采用其他浏览器对象，或者删除某个语言特性，所以这是很危险的。实际上，当基于 Mozilla 的 Netscape 版本首次发布时，就去掉了所有 layers 对象，代之以基于 W3C 标准的特性。Web 上的大量脚本以 document.layers 的存在与否作为条件，来区分使用和未使用 document.1ayers 的 Netscape 代码。这样，使用 Netscape 6 或 7 的访问者发现脚本要么崩溃要么不能工作，而实际上浏览器完成该工作绰绰有余。

因此对象检测不能针对浏览器版本，而要针对对象的可用性，如前面图像的例子。此外，安全执行对象检测的条件是所有主流浏览器产品(和 W3C DOM 建议)都采用这个对象，而且将来在任何地方载入页面时，对象的行为都是可预知的。

对象检测技术包括测试对象的方法是否可用。对象方法的引用会返回一个值，所以这个引用可在条件语句中使用。例如，在下列代码段中，一个函数接收包含元素字符串 ID 的参数，并根据三个不同的 DOM，把字符串转换为有效的对象引用：

```
function myFunc(elemID)
{
    var obj;
        if (document.getElementById)
        {
            obj = document.getElementById(elemID);
        }
        else if (document.all)
        {
            obj = document.all(elemID);
        }
        else if (document.layers)
        {
            obj = document.layers[elemID];
        }
        if (obj)
        {
            // statements that work on the object
        }
}
```


更多资源请访问稀酷客(www.ckook.com)

根据这个对象检测模式，哪个浏览器产品、版本或操作系统能将元素 ID 转换为对象引用并不重要。不管浏览器支持 document 对象的哪个特性或方法，都使用这个特性或方法完成转换。假如浏览器支持多个特性或方法，就选择第一个。假如浏览器一个都不支持，就不执行这个函数中的其他语句。然而，在现代浏览器中，这个例子中的第一个方法足以从 ID 中获取所有对象，这也是推荐使用的方法。

假如所有浏览器的对象模型都不包含某个对象，而脚本又希望检查该对象的特性或方法是否存在，则必须预先检查该对象是否存在。在条件表达式中引用一个不存在的对象的特性，会产生脚本错误。为了避免这个错误，可使用&&操作符来级联条件测试。下面的代码段测试 document.body 对象和 document.body.style 特性是否存在：

```
if (document.body && document.body.style)
{
    // statements that work on the body's style property
}
```

这等价于：

```
if (document.body)
{
    if (document.body.style)
    {
        // statements that work on the body's style property
    }
}
```

在这两个例子中，如果对 document.body 的测试失败，JavaScript 将忽略第二个测试。

使用条件表达式测试对象特性是否存在有一个潜在的错误：如果特性存在，但它的值是 0 或空字符串，条件测试就会报告特性不存在。为了消除这个潜在的问题，条件表达式可以检查值的数据类型，来保证特性真正存在。如果对象的特性不存在，就报告 undefined 数据类型。还可以使用 typeof 操作符(详见第 20 章)来测试特性是否有效：

```
if (document.body && typeof document.body.scroll != "undefined")
{
    // statements that work on the body's scroll property
}
```

建议把脚本设计为用对象检测来替代根据浏览器名称字符串和版本号进行分支处理。脚本编写功能逐渐进入各种非传统计算设备的浏览器中。这些浏览器可能不会使用我们现在了解到的名称和版本系统，但可以解释脚本。测试浏览器功能，可能减少将来的脚本维护工作。

4. 浏览器版本检测

多年前，对象检测技术还未开发出来，确定脚本能得到哪些语言支持的常见方法是浏览器检测(browser sniffing)。如第 42 章所述，脚本可以检查 navigator 对象，确定当前浏览器的品牌和版本。检查了浏览器声称的名称和版本号后，脚本就可以进入专用于当前各种浏览器的逻辑分支。现在不再使用这个技术，主要是因为浏览器可以就其名称和型号撒谎，而浏览器版本检

更多资源请访问稀酷客(www.ckook.com)

测是不会过时的技术，它是一次网络攻击——它测试一个方面，以处理另一个方面。

浏览器检测像是让一个自称为飞行员的人驾驶飞机。他们可能并不像自己说的那样是飞行员，即使他们是飞行员，其飞行执照也是在飞机发明之前获得的。与此形成对照的是，通过对象检测，可以确定他们是否知道如何驾驶飞机，如果他们知道，就让他们驾驶——简单、直接、安全。

4.2　兼容性设计

浏览器每个重要的新版本都会给页面设计者带来兼容性问题。旧脚本很少在新版本中崩溃(出色的脚本很少在新版本中出错)。一些访问者没有升级到最新最强大的浏览器版本，或没有使用设计者的浏览器品牌配置，而设计者没有考虑到这些访问者，此时问题就会集中在吸引设计者的新功能上。

即使仅仅满足最低公用标准的要求，也会使开发时间加倍，因为需要扩展测试矩阵，确保页面在所有操作系统的所有版本中都工作良好；还需要确定页面中使用的脚本功能对于每个使用者的重要性。假如希望一些功能只在新版的浏览器中有效，就必须强制定义一个访问页面的最低浏览器配置，更低级的浏览器只能得到站点数据的简单显示。

另一个可能性就是使站点的一部分能供大多数浏览器(如果不是全部浏览器)访问，并将脚本限制在非脚本浏览器的用户也能访问的某些偶然改进上。应用程序到达导航流中的某个位置时，用户就需要功能更强的浏览器来查看非常不错的内容。这类设计做了精心的规划，它使所有用户都能到达站点的这个位置，接着只有使用某些浏览器(例如符合 W3C DOM 标准的浏览器)的用户才能继续访问应用程序。

理想的页面可以在任何浏览器上显示有用的内容，但页面的脚本能改进页面访问者的体验——其方式也许是提供更有效的站点导航，或者和页面的内容互动，这当然是值得追求的目标。但是，即使只在某些页面上达到如此境界，也会减少为功能不同的浏览器定义完全不同的、难以维护的路径的必要性。

无论浏览器兼容性策略是什么，随着时间的推移，问题都会越来越小。在过去几年中，Web 标准已经极大地固化了，浏览器厂商朝着完全支持这些标准迈出了重要的步伐。

4.2.1　处理 beta 版浏览器

设计了一个制作精良的 Web 页面或站点后，用户也许会担心，当浏览器的预发行版(或 beta 版)发布时，可能会在这个页面中出现脚本错误或其他兼容性问题。其实不必对预发行版浏览器中出现的错误大惊小怪，假如代码编写得很好，它就应该能在任何新浏览器版本中良好运行；假如代码不能正确工作，就说明浏览器存在错误，把该错误(最好附带一个简化的测试脚本)报告给浏览器厂商。

“这是一个 beta 版错误”规则有一个例外，它出现在从 Netscape Navigator 4 到 Mozilla 引擎(最初发布为 Netscape Navigator 6)的转换中。故意删除 NetscapeNavigator 4 的一个专用功能

更多资源请访问稀酷客(www.ckook.com)

(<layer>标记和相应的脚本对象)将导致许多 Netscape Navigator 4 脚本在 Mozl beta 版(和最后的发行版)上崩溃。脚本编写人员把该问题报告给新浏览器的开发人员(Mozilla)时，就已经知道策略有变化，并设计了新的实现方案。浏览器很少如此迅速地删除一个广泛使用的功能，但这确实有可能发生。更强大的 Web 标准尽量减少了再次发生这种情况的可能性。

用户通常很难不被浏览器厂商推出的新发行版所吸引，但是预发行版并非发行版，使用预发行版浏览器访问页面时，应该知道，浏览器中可能存在错误。代码在预发行版中不能运行不是自己的过失，不需要为此失眠。只是一定要联系浏览器厂商，看看这个问题是否出现在最后的发行版中，或者报告这个错误，以确保在发行版中不再有相同的问题。

4.2.2　参考章节中的兼容性等级

自从 Navigator 2 以来，随着脚本浏览器版本数量的增加，需要预先知道所设计的最低公用标准是否支持某种语言或对象模型、特性、方法或事件处理程序，这一点很重要。因此，本书包含常见的兼容性等级，例如：

兼容性： WinIE5+，MacIE5+，NN4+，Moz+，Safari+，Opera+，Chrome+

浏览器版本号后面的加号表示，语言特性首先在该版本的浏览器中实现，其后续版本继续支持该特性，减号表示浏览器不支持该特性。测试了兼容性的浏览器包括 Windows 和 Macintosh 的 Internet Explorer、Netscape Navigator、Mozilla(包括所有基于 Mozilla 引擎的浏览器)Apple Safari、Opera 和 Google Chrome。另外建议打印附录 A 中的 JavaScript 和 Browser 对象快速参考文件，此文件在配书光盘中，格式为 PDF。这一快速参考清楚地给出了每个对象的特性、方法、事件处理程序，以及每个语言项在哪个浏览器版本中得到支持的要点。此打印稿是宝贵的日常参考资源。

下面阐明“所有基于 Mozilla 引擎的浏览器”的含义。不久前，Mozilla 还主要指 Netscape，但那些日子已经远去了。现在，本书的兼容性图表中有多个基于 Mozilla 的浏览器属于 Moz+ 这一类，包括 Netscape、Firefox、Camino、SeaMonkey、Flock 和其他浏览器。

这里使用 Mozilla 表示 Gecko (née NGLayout)布局引擎。各浏览器品牌的版本号系统与其下的 Mozilla 引擎版本不同步，所以难以确定何种浏览器支持何种功能。表 4-1 给出了各浏览器品牌和版本与 Mozilla 引擎的版本号系统之间的对应关系。

表 4-1　各浏览器品牌和版本与 Mozilla 引擎版本号之间的对应关系

Mozilla	Netscape	Firefox	Camino
m18	6.0		
0.9.2	6.1		
0.9.4	6.2		
1.0.1	7.0		
1.2b		0.1	
1.3a		0.5	

更多资源请访问稀酷客(www.ckook.com)

(续表)

Mozilla	Netscape	Firefox	Camino
1.4	7.1		
1.5		0.7	
1.7		1.0	
1.7.2	7.2		
1.7.5	8.0–8.1		
1.8		1.5	1.0
1.8.1	9.0	2.0	1.6.5
1.9		3.0	2.0
1.9.1		3.5	
1.9.2		3.6	
1.9.3		3.7	
2.0		4.0	

可以看出，Netscape 6.0 和 6.2 基于低于 1 的 Mozilla 版本，这些版本现在极少使用，因此重点是 Moz1 和以后的版本。于是，兼容性图表将 Moz1 作为基本功能集。

总之，在兼容性图表中看到 Moz+时，它最终转化为 Netscape 7 或更新版本、Firefox 1 或更新版本、Camino 1 或更新版本，以及目前使用的最流行的基于 Mozilla 的浏览器。

4.3　资深程序员的语言基础

在本部分，富有经验的程序员可以读到 JavaScript 核心语言的重点，但脚本编写经验有限或完全没有经验的程序员也许不能完全理解这些内容。下面快速浏览 JavaScript 核心语言的基本问题：

- **JavaScript 是一种脚本语言。**该语言主要用于现有的主机环境中(例如，Web 浏览器)，在该环境中，对象的特性和行为可以用该语言编写的语句来控制。脚本在主机环境内执行，主机环境通过在其上运行的语句来控制外部环境对象(如果有的话)的寻址。由于安全和保密的原因，Web 浏览器几乎很少或根本不通过 JavaScript 直接访问浏览器首选项、操作系统或浏览器范围之外的其他程序。这个原则有一个例外，就是现代浏览器允许通过可信机制进行更深层次的客户访问(经过用户的允许)，如签名脚本(Mozilla)或可信的 ActiveX 控件(Microsoft)。
- **JavaScript 是基于对象的。**虽然 JavaScript 和 Java 语言的语法有许多类似之处，但 JavaScript 不像 Java 那样严格地面向对象。该核心语言包括几个生成工作对象的内置静态对象。要创建对象，需要通过 new 操作符给任何内置对象调用构造函数。例如，下列表达式生成一个 String 对象，并返回该对象的引用：

更多资源请访问稀酷客(www.ckook.com)

```
new String("Hello");
```

表 4-2 列出了脚本开发人员使用的内置对象。

表 4-2　JavaScript 的内置对象

核心对象	错误对象	XML 对象	JSON 对象
Array[1]	Error[2]	Namespace[4]	JSON[5]
Boolean	EvalError[2]	QName[4]	
Date	RangeError[2]	XML[4]	
Function[1]	ReferenceError[2]	XMLList[4]	
Global	SyntaxError[2]		
Math	TypeError[2]		
Number[1]	URIError[2]		
Object[1]			
RegExp[3]			
String[1]			

(1) 虽然在 ECMA Level 1 定义，但在 NN3 和 IE3/J2 中首次可用；
(2) 在 ECMA Level 3 中定义，在 Mozl 中实现；
(3) 在 ECMA Level 3 中定义，在 NN4 和 IE6 中完全实现；
(4) 在 E4X 中定义，在 Mozilla 1.8.1 (Firefox 2.0)中实现。
(5) 在 ECMA Level 5 定义，在 Mozilla 1.9.1(Firefox 3.5)中实现。

- **JavaScript 是弱类型语言**。变量、数组和函数返回值没有定义为特定的数据类型。事实上，变量在初始化后，其值在后续的脚本语句中可以是其他数据类型(这种做法显然欠佳，但这是可能的)。同样，数组可以包含多种类型的值。内置的数据类型只有几个：
  - Boolean(true 或 false)
  - Null
  - 数字(双精度 64 位格式 IEEE 734 值)
  - 对象(包括 Array 对象)
  - 字符串
  - 未定义
  - XML(在 E4X 中)
- **主机环境定义了全局作用域**：Web 浏览器通常定义一个浏览器窗口或框架，作为脚本语句的全局环境。当卸载文档时，该文档中定义的所有全局变量都被销毁。
- **JavaScript 变量可以有全局或局部作用域**：全局变量在所有函数外部的 var 语句中初始化，在 Web 浏览器中，该语句一般在加载文档时执行。文档中的所有语句都可以读写这个全局变量。局部变量也在函数内部用 var 操作符初始化，只有该函数内的语句才能访问局部变量。

更多资源请访问稀酷客(www.ckook.com)

- **脚本有时会访问 JavaScript 静态对象的特性和方法**：一些静态对象允许直接访问它们的特性或方法，例如，Math 对象的所有特性都用作常数值(例如，Math.PI)。
- **可以随意把特性或方法加入到工作对象中**：要将属性添加到对象中，只需把任何类型的值赋给它即可。例如，要把 author 特性添加到字符串对象 myText 中，可以使用如下语句：

```
myText.author = "Jane";
```

把函数引用赋给对象的一个特性，就可以给对象增加一个新方法：

```
// function definition
function doSpecial(arg1)
{
   // statements
}
// assign function reference to method name
myObj.handleSpecial = doSpecial;
...
// invoke method
myObj.handleSpecial(argValue);
```

在函数定义内，this 关键字表示拥有该方法的对象。

- **JavaScript 对象使用基于原型的继承**：所有对象的构造函数都能创建工作对象，它们的特性和方法继承了为该对象原型定义的特性和方法。脚本可以添加和删除与静态对象原型相关的自定义特性和方法，所以新的工作对象会继承原型的当前状态。脚本可以在工作对象中自由地重写原型特性值，或者把不同的函数赋给原型方法，而不影响静态对象原型。但是假如继承的特性或方法在当前工作对象中未经修改，静态对象原型的任何变化都会反映在工作对象中(其机制是对象特性的引用沿着原型继承链查找匹配特性名称的一个特性)。
- **JavaScript 包括许多操作符**：大多数在其他语言中使用的操作符，这里都有。
- **JavaScript 提供了典型的控制结构**：JavaScript 的所有版本都提供 if、if-else、for 和 while 结构，JavaScript 1.2 (NN4+、IE4+和所有现代的主流浏览器)还添加了 do-while 和 switch 结构，迭代结构还提供了 break 和 continue 语句来更改控制结构的执行。
- **JavaScript 函数可能返回或不返回值**：只有一种 JavaScript 函数，并且只有当函数包括 return 关键字后跟要返回的值时，函数才能返回一个值，返回值可以是任何数据类型。
- **JavaScript 函数不能重载**：JavaScript 函数接受 0 个或多个实参，不管为函数定义了多少个形参变量都是如此。所有实参都自动赋给 arguments 数组，它是函数对象的一个属性。另外，形参变量的数据类型没有预定义。
- **值可以按引用传递，也可以按值传递**：传递给函数的对象实际上是对象的一个引用，它提供了对象特性和方法的完全读/写访问。但其他类型的值(包括对象特性)是通过值传递的，不包含到原始对象的引用链。因此，触发 onChange 事件时，下面这段没有实

更多资源请访问稀酷客(www.ckook.com)

际意义的程序将清空文本框：

```
function emptyMe(arg1)
{
   arg1.value = "";
}
...
<input type="text" value="Howdy" onchange="emptyMe(this)">
```

但在下列版本中，文本框没有任何改变：

```
function emptyMe(arg1)
{
   arg1 = "";
}
...
<input type="text" value="Howdy" onchange="emptyMe(this.value)">
```

局部变量(arg1)从 Howdy 变为空字符串。

注意：

上例中的特性分配事件处理技术有意进行了简化，以使代码非常简短。通常最好使用更新的方式，即用 addEventListener()(NN6+/Moz/W3C)或 attachEvent()(IE5+)方法来绑定事件。第 32 章将详细说明现代的跨浏览器事件处理技术。

- **错误捕获技术依赖 JavaScript 的版本：**在 NN2 或 IE3 中没有错误捕获技术，NN3、NN4 和 IE4 中的错误捕获技术在 Web 浏览器对象模型中是事件驱动的，在 IE5+、Mozilla、Safari 和其他最新浏览器中实现的 JavaScript 支持 try-catch 和 throw 语句，也支持不依赖主机环境的内置错误对象。
- **内存管理不受脚本的控制：**主机环境管理内存分配，包括垃圾收集。不同浏览器会采用不同方式来处理内存。
- **空白字符(不是行终结符)无关紧要：**空格和制表符可将词汇单元分开(例如关键字、标识符等)。
- **行终结符通常作为语句分隔符：**除非常少有的结构外，JavaScript 解析器只要遇到一个或多个行终结符(例如回车或换行)，就自动插入分号分隔符。在源代码的同一个物理行中，两个语句之间需要有一个分号。此外，字符串字面量在其源代码中可以没有回车(但转义换行符\n 可以是字符串的一部分)。

Evaluator Sr.

Evaluator Sr.工具可以在后续章节中帮助学习 JavaScript 核心知识和 DOM 术语。它提供了一个交互的工作平台，可以进行表达式计算和对象检查(第 8 章将介绍其简化版本 Evaluator Jr.)。

图 4-1 显示了该页面的顶部，这个完整版本与第 8 章的 Jr.版本有两个不同的重要特性。

更多资源请访问稀酷客(www.ckook.com)

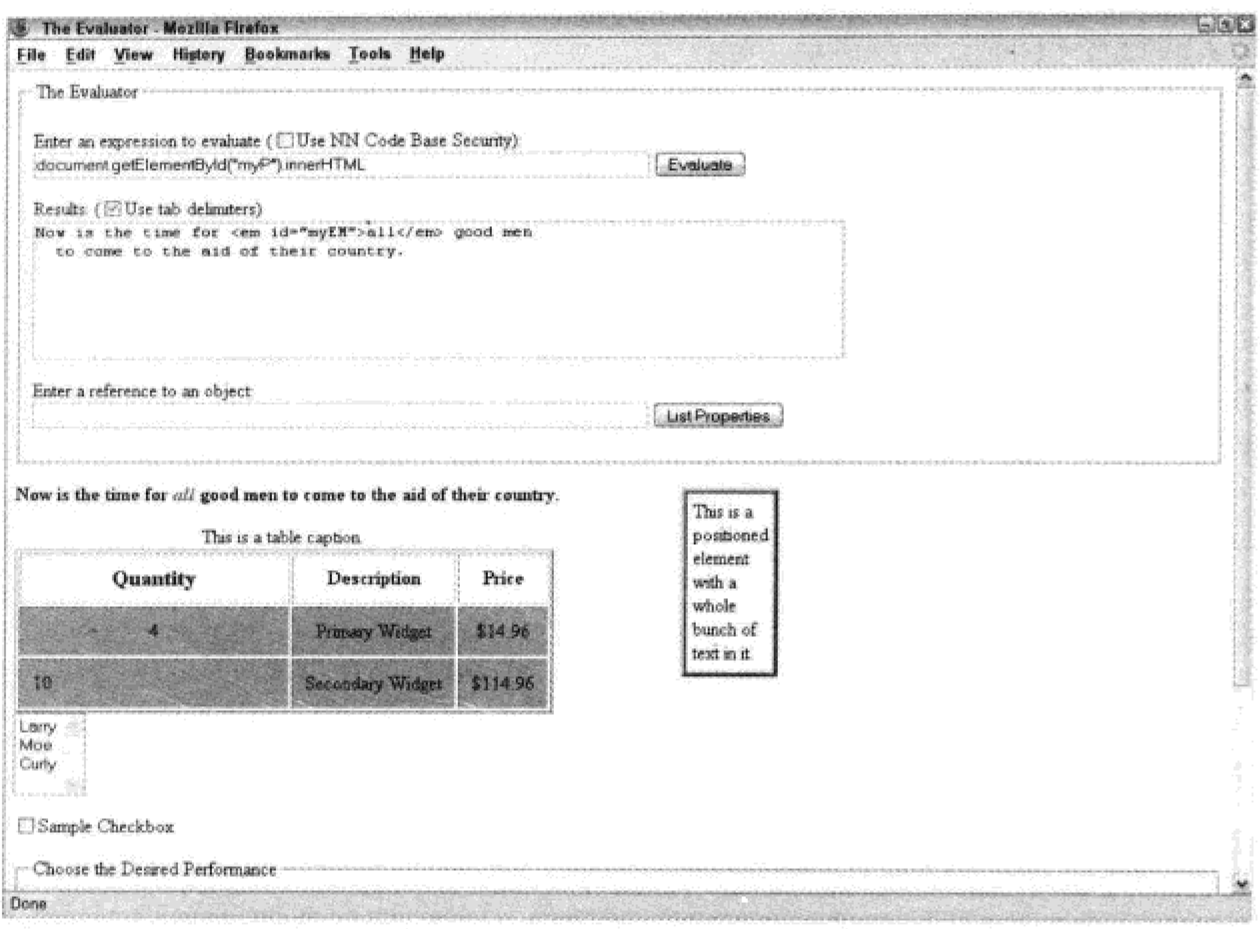


图 4-1 Evaluator Sr.

要试用一些 Mozilla 的安全功能，首先应给浏览器打开 Code Base Principles(光盘中的第 49 章)，并选中 Use Code Base Security 复选框(只有 Netscape Navigator 4+或者 Moz 有)。其次，页面有几个预先安装的 HTML 元素，它们可用于搜索 DOM 特性和方法。与低版本一样，这个版本也初始化了一套 26 个单字母全局变量(a～z)，用户可以对它们赋值，用在扩展求值序列中。

可以从配书光盘中把文件 evaluator.html 和 evaluator.js 复制到本地硬盘上，并在所有的测试浏览器中为这个文件设置一个书签，还可以在页面的底部随意增加自己的元素来研究其他对象。配书光盘中的第 48 章把嵌入项目中的 Evaluator 作为调试工具，第 48 章将学习它的更多内置功能。

更多资源请访问稀酷客(www.ckook.com)

第 II 部分

JavaScript 教程

本部分内容

第 5 章　第一个 JavaScript 脚本
第 6 章　浏览器对象和文档对象
第 7 章　脚本和 HTML 文档
第 8 章　程序设计基础(一)
第 9 章　程序设计基础(二)
第 10 章　window 和 document 对象
第 11 章　表单和表单元素
第 12 章　String、Math 和 Date 对象
第 13 章　编写框架和多窗口脚本
第 14 章　图像和动态 HTML

PDG
更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第 5 章

第一个 JavaScript 脚本

> **本章包含哪些内容？**
> 创建一个包含 HTML 页面、JavaScript 脚本和 CSS 样式表的简单应用程序

本章将编写一个简单脚本，其结果可在符合 JavaScript 标准的浏览器中查看。

教授口语的两种常见方式是(a)先上语法课程，再进行对话；(b)先进行对话，再上语法课程。

我们喜欢(b)，它比较有趣(寓教于乐)，可以立即看到学习效果，也更接近母语学习方式。语法总是要了解的，但先在屏幕上显示一些内容是一个良好的开端。

不必记住本章讨论的每个命令和语法细节，而应放松下来，看看 HTML 标记和 JavaScript 语句如何在浏览器上显示内容。如果喜欢这个过程，就会更快吸收细节内容。

本章会解释每行代码，但第一个教程使用的所有方法都会在本书的后面详细说明。现在只需输入所提供的脚本，看看它们如何完成期望的工作即可。

5.1 第一个脚本的功能

第一个教程脚本会把当前的日期和时间插入网页(这是第 3 章中“Hello World”脚本的改进版本)。首先创建一个仅显示静态内容的 HTML 页面，接着添加 JavaScript，使其变成活动页面，最后添加一些样式，使页面变得生动有趣。

注意这个序列——建立 HTML 结构，再给所显示的内容添加动态的 JavaScript 和 CSS——不仅在学习 JavaScript 的过程中很有用，而且是日常建立实际网站的一个非常好的模型。它是更长的开发序列的一部分，更长的开发序列还包括信息设计、图形设计、线框和概念验证、HTML 标记、服务器端脚本编程、CSS 样式和 JavaScript。尽管直接开始编写脚本是很吸引人的，但就像在烘烤蛋糕之前要先撒上酥皮一样，JavaScript 和 CSS 都需要了解其环境——它所操作的页面结构。另外，按照这个序列来工作，将有助于遵循“渐进增强”原则，即每个人都可以获得基本页面，而使用功能更强的用户代理(例如浏览器)就可以得到更多的内容。从 HTML 页面

更多资源请访问稀酷客(www.ckook.com)

开始，将可以确保即使不使用 JavaScript，页面也是完整的、有意义的。

目前大多数重要的 Web 制作过程都包含服务器端脚本语言和数据库，例如 PHP 和 MySQL，用于把基本的动态内容提供为 HTML 标记。服务器端脚本的编程超出了本书的讨论范围(这会时不时地提起)，但我们会显示一个完全静态的 HTML 页面。

5.2 输入第一个脚本

启动文本编辑器和浏览器。还可以启动标准的文件管理工具，以监控要在计算机的文件夹中创建的文件。

本书的例子都是脱机工作的，所有操作都在本地的计算机上执行，不需要连接互联网。假如浏览器支持到 Internet 服务提供商(ISP，Internet Service Provider)上的拨号或自动拨号，就可以取消或退出拨号操作。假如浏览器的 Stop 按钮处于激活状态，则单击它停止任何网络搜索。用户也许会收到对话框信息或页面，它提示浏览器主页的 URL(除非改变设置，否则它通常是浏览器发布者的主页)不可用。此时需要打开浏览器，但不需要连接到 ISP 上。假如在办公室或学校通过局域网、电缆调制解调器或 DSL 自动连接到互联网，这种方法同样奏效。然而，目前不需要连接网络。

5.2.1 第一步：HTML 文档

图 5-1 显示了第一个任务：带有标题和文本段的简单页面。最初的页面就是这个样子，但不同浏览器默认显示的内容可能略有区别(换言之，页面的样式可能不同)。

在一个新的文本文件中输入如程序清单 5-1 所示的 HTML 标记，并保存为 date-time.html(使用 UTF-8 字符编码)：

程序清单 5-1　"Date & Time": date-time.html 的 HTML 标记

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date &amp; Time</title>
   </head>
   <body>
      <h1>Date &amp; Time</h1>
      <p>It is currently <span id="output">now</span>.</p>
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

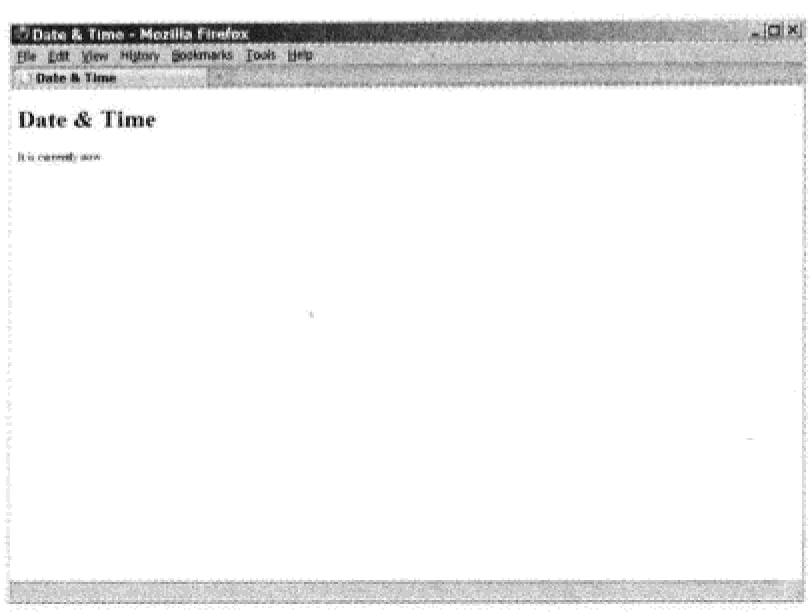


图 5-1　第一步：静态的 HTML 页面

注意：

本书始终给 HTML 文件使用 UTF-8 字符编码。字符编码告诉浏览器如何显示文本字符，UTF-8 的 Unicode 标准是一个优秀的系统，它允许在 HTML 文件中包含几乎所有语言。它在网站制作过程中工作得很好。

从一开始就养成用 UTF-8 编码保存文件的习惯。许多文本编辑器都允许在 Save As...对话框中选择编码，也可以给未来的文件设置默认编码，这样就不必每次都指定它了。如果文本编辑器未提供该选项，从现在开始就一直使用它，但应开始寻找另一个提供了该选项的编辑器。

现在，在浏览器中打开文档。为此，可在文件管理器屏幕上双击文件，或在浏览器的 File 菜单中选择 Open 或 Open File。

如果按照上面所示输入了所有代码行，浏览器窗口中的文档就应如图 5-1 所示(因计算机的操作系统和浏览器版本的不同，可能稍有差异)。

下面解释文档中的内容，以理解标记的一些细节。

1. DOCTYPE

```
<!DOCTYPE html>
```

DOCTYPE 告诉浏览器如何显示文档。使用哪个 DOCTYPE 会极大地影响 CSS 样式和标记的有效性验证。本书使用的 DOCTYPE 用于 HTML5，但可以使用 HTML 4.01-Strict 的 DOCTYPE 替代它，以执行有效性验证：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

或者使用 XHTML 1.0-Strict 的 DOCTYPE(并对标记样式进行其他几处修改)来替代：

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

注意：

如果要遵循 XHTML 样式的约定，标记和特性名就需要全部使用小写字母。本书的程序清

更多资源请访问稀酷客(www.ckook.com)

单都是 HTML，可以接受大小写标记和特性，但这里全部使用小写字母，以同时遵循两个标准。

XHTML 还需要对这些例子中使用的 HTML 进行几处修改。

2. html

接下来的标记是：

```
<html>
...
</html>
```

整个页面都放在 html 元素中，该元素声明了标记的类型。html 元素只有两个子元素：head 和 body。

3. head

```
<head>
...
</head>
```

head 元素可以包含各种子元素，来传递文档的信息，但它们一般不显示为文档内容。head 至少要有两个子元素：定义字符集的 meta 元素和 title 元素：

```
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Date &amp; Time</title>
```

meta 标记是一个冗长的标题，目前只需按照上述内容输入它即可。HTML5 可以接受较简短的版本：

```
<meta charset="UTF-8">
```

但是，较长的版本适用于所有 3 种标记类型(HTML 4.01、HTML5 和 XHTML1.0)。注意 meta 标记是空的，没有结束标记</meta>。它传递的所有信息都放在其特性中，这里是 http-equiv 和 content。

另一方面，title 需要结束标记，它们把内容封装起来。标题“Date & Time”使用 HTML 实体&(表示宏符号&)输入为 Date &Time。如果其中的 4 个字符出现在页面的内容中——特性值和用标记封装起来的文本，它们就必须总是转换为 HTML 实体，才能避免使解析引擎迷惑：

```
< &lt;
> &gt;
& &amp;
" &quot;
```

最后一个是 HTML 标记中使用的直双引号，不是排版文本中常见的弯双引号。

4. body

```
<body>
...
</body>
```


PDG

更多资源请访问稀酷客(www.ckook.com)

body 元素包含文档的内容——其结构、文本、图像、多媒体对象等。

5. headline

```
<h1>Date &amp; Time</h1>
```

我们使用标题标记 h1 到 h6 创建文档中的大纲结构。文档页面通常只有一个 h1 来重申页面的标题，从 h2 开始的其他标题则嵌套在 h1 中。

以前，标题标记只能使用其默认外观，而现在，可以像在嵌套的大纲结构中使用数字一样使用它们，所有文本的外观则通过样式表来控制。

6. paragraph

```
<p>It is currently <span id="output">now</span>.</p>
```

最后，这是 JavaScript 要在其中插入当前日期和时间的文本段。JavaScript 使用 span 元素的 ID 属性(output)来定位。这样脚本就可以用实际的日期和时间来替代 span 的内容。

这是静态的 HTML 页面。在 JavaScript 中插入日期和时间后，即使使用不支持 JavaScript 的用户代理(例如搜索引擎)或关闭了 JavaScript 支持的用户代理，也可以看到这里显示的内容。页面目前的内容并不多，但至少它不会崩溃。对于公共网站，脚本应提高页面的价值，而非完成重要任务。

5.2.2　第二步：添加 JavaScript

接着给页面创建 JavaScript。现在应该养成一些良好的 JavaScript 习惯，这对于编写脚本非常重要。首先，JavaScript 是区分大小写的，因此，必须用正确的大小写字母输入脚本中的每个字。如果某行 JavaScript 不能奏效，首先应看看是否有大小写错误。总是比较自己输入的代码和本书中的程序清单，以及后面讨论的各个词条。

其次，注意每 JavaScript 个语句都以分号结束。从技术角度讲，这些尾部的分号——可以看成是句子尾部的句点——在 JavaScript 中是可选的，但强烈建议使用它们，以免脚本含糊不清。如果有朝一日研究其他编程语言，例如 PHP、Java 或 C++，就会发现这些语言也需要分号。本书的所有 JavaScript 程序清单都使用分号。

在编辑器中创建第二个文本文件 date-time.js，并输入如程序清单 5-2 所示的脚本。

注意：
JavaScript 注释文本放在双斜杠(//)的后面，且位于当前行上，或者放在斜杠-星号(/*)和星号-斜杠(*/)之间。JavaScript 解释器会忽略掉注释，添加注释仅为了便于用户理解代码中的操作。因此，不必像这里的程序清单那样输入注释，但建议养成给代码添加注释的习惯。其他程序员和你自己都会在将来从中受益。

程序清单 5-2　"Date & Time" : date-time.js 的 JavaScript 代码

```
// tell the browser to run this script when the page has finished loading
window.onload = insertDateTime;
```

更多资源请访问稀酷客(www.ckook.com)

```
// insert the date & time
function insertDateTime()
{
      // ensure a DOM-aware user agent
      if (!document.getElementById) return;
      if (!document.createTextNode) return;

   // create a date-time object
   var oNow = new Date();

   // get the current date & time as a string
   var sDateTime = oNow.toLocaleString();

   // point to the target element where we want to insert the date & time
   var oTarget = document.getElementById('output');
      // make sure the target is found
      if (!oTarget) return;

   // delete everything inside the target
   while (oTarget.firstChild)
   {
      oTarget.removeChild(oTarget.firstChild);
   }

   // use the date-time string to create a new text node for the page
   var oNewText = document.createTextNode(sDateTime);

   // insert the new text into the span
   oTarget.appendChild(oNewText);
}
```

为了把这个 JavaScript 文件连接到 HTML 文件上，需要给 HTML head 添加一个 script 元素。在文本编辑器中打开 date-time.html，添加如下突出显示的代码：

```
...
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date &amp; Time</title>
      <script type="text/javascript" src="date-time.js"></script>
   </head>
...
```

script 标记标识了所链接的 JavaScript 文件的 MIME 类型，并为其指定名称(和路径，以防该文件不在 HTML 文件所在的文件夹中)。这会让浏览器在建立文档页面时读取 JavaScript 文件。浏览器中的 JavaScript 解释器会验证并编译脚本，并立即运行它，甚至在读取 HTML 文档的其余内容之前运行。

下面逐行解释 JavaScript 代码，以便读者理解其作用。

更多资源请访问稀酷客(www.ckook.com)

1. 触发事件

问题是：脚本要把日期和事件插入页面的段落中，但浏览器在读取和运行 JavaScript 代码之前，并没有读取 HTML 体。在 JavaScript 开始运行时，要插入新内容的段落甚至在浏览器的内存中不存在。

解决方法只是告诉 JavaScript，在文档完全读入内存中之前，不要执行插入操作。为此使用了 onload 事件。页面的元素——在 DOM 中称为对象——可以感知许多事件，例如鼠标单击或按下一个键。文档加载到浏览器中后，就会触发 window 对象的 onload 事件。

```
// tell the browser to run this script when the page has finished loading
window.onload = insertDateTime;
// insert the date & time
function insertDateTime()
{
  ...
}
```

这里告诉 JavaScript，在文档加载完毕后调用 insertDateTime()函数。日期和时间插入例程需要在文档加载完毕后进行，所以会延迟到浏览器内存中包含了目标段落之后。

2. 确保一个安全的环境

文档加载到浏览器中后，就会触发 window.onload 事件，调用 insertDateTime()函数。脚本首先要确保它由使用相同语言的现代浏览器来解释：

```
// ensure a DOM-aware user agent
if (!document.getElementById) return;
if (!document.createTextNode) return;
```

这个函数中使用的 getElementById()和 createTextNode()是两个某些旧 JavaScript 解释器不能理解的 DOM 方法。感叹号(!)表示“not”，return 告诉 JavaScript 停止运行当前的函数，所以第一个语句表示：“如果不知道 document 对象的 getElementById()方法，就退出函数。”如果不包含这些保护措施，老式浏览器或其 JavaScript 解释器就会在运行其后的代码时崩溃。

可能只是程序较小的崩溃，例如 JavaScript 解释器停止运行(只显示普通的 HTML 页面，根本没有 JavaScript 功能)，也可以非常大，例如整个浏览器停止运行，需要重启。这两个事件都不受欢迎。对于许多页面，我们都编写 JavaScript 来执行几个任务，其中一些任务可以由老式浏览器执行，但其他任务不行。我们希望让浏览器绕过它不能处理的部分，只运行它能运行的部分。防止旧 JavaScript 版本因遇到现代方法而崩溃是专业程序员的标志。

我们赞同“降低功能”和“渐进增强”原则，即对于不能利用特定功能的浏览器，会降低要求，并让其做到最好。与 DOM 交互操作的脚本是使不支持 DOM 的老式解释器避开与的 DOM 交互操作。

检查旧 JavaScript 解释器的老办法是浏览器检测，即检查 navigator 对象报告的浏览器类型。但是，这种自我声明的身份可能靠不住，JavaScript 解释器知道自己可以执行哪些方法才是可信的，这是目前使用的基本安全特性。

更多资源请访问稀酷客(www.ckook.com)

3. 生成日期和时间

要插入这个文档中的文本是当前的日期和时间(取决于用户计算机上的时钟)。JavaScript 很容易完成这个插入操作：创建一个 Date 对象，并使用它的一个方法生成字符串：

```
// create a date-time object
var oNow = new Date();
// get the current date & time as a string
var sDateTime = oNow.toLocaleString();
```

这些代码把 oNow 变量设置为一个新的 Date 对象。之后使用其 toLocaleString()方法输出一个文本字符串，它包含：

```
Thursday, August 20, 2009 4:16:05 PM
```

日期和时间字符串保存在 sDateTime 变量中，该变量稍后会输出到页面上。

toLocaleString()方法的优点在于，它输出的日期和时间根据用户的设置和用户的时区进行格式化。如果两个人在地球的两端同时运行这些代码，就会得到两个不同的日期和时间，它们使用不同的格式和语言进行表达。后续章节将详细介绍 Date 对象，尤其是第 12 章和第 17 章。

注意：
在这个脚本示例中，oNow 是一个 Date 对象，sDateTime 是一个文本字符串。在无数种编程风格中，一种是所谓的 Systems Hungarian 表示法，其中每个变量名以其类型的指示开头，这里 o 表示对象，s 表示字符串，i 表示整数等。像这样给变量命名完全是可选的，由程序员来确定是否使用它。JavaScript 并不介意变量名是什么，只要变量名没有使用关键字或非法字符即可。JavaScript 是一种“动态的类型化”语言，可以用数值、字符串、布尔值和对象引用值随意设置同一个变量。一些程序员喜欢在每个变量中仅保存一种类型的值，以便于代码的管理和调试。使用 Hungarian 表示法使这些变量类型更容易记忆。

4. 确定目标

下一步是把日期和时间插入文档中需要的位置。为此需要执行以下操作：

(1) 指向目标 span 元素。

(2) 删除该元素中已有的文本。

(3) 在元素中插入新文本。

为了指向 output span，脚本使用业界标准的方式来指向拥有 ID 特性的 HTML 元素：

```
// point to the target element where we want to insert the date & time
var oTarget = document.getElementById('output');
```

第一个语句在 HTML 页面中通过 ID (output)定位目标元素，并把该对象的引用存储在 oTarget 变量中。HTML 规范指出，每个元素 ID 在当前页面中必须是唯一的。即使不小心在页面上多次使用相同的 ID，getElementById()方法也仅返回第一个 ID 实例。所以，这个方法仅定位一个元素——在这个例子中，就是 HTML 中标记的 span 元素：

```
<p>It is currently <span id="output">now</span>.</p>
```


更多资源请访问稀酷客(www.ckook.com)

注意调用 getElementById()时，要指定元素的 ID (output)，而不是其标记名(span)。在 HTML 标记中，可以把特性 id="output"从 span 移动到页面中的任何其他元素上，脚本将尝试操作该元素。这是非常棒的。它表示，即使标记和脚本必须就目标元素的 ID 达成一致，但无论在 HTML 中使用了什么标记，脚本都是可以执行的。一段时间后，这种灵活性有助于生成一些真正通用的 JavaScript 函数。

如果遗漏或拼错了标记中的 ID，getElementById()将返回空值。为避免出现这种情况，执行如下测试：

```
// make sure the target is found
if (!oTarget) return;
```

与前面的 if 测试类似，这个测试表示："如果 oTarget 变量不存在或值为空，就从这个函数中返回。"添加这个脚本的现状核实有助于使脚本刀枪不入。在页面上添加数十个元素或者处理由许多人和过程生成的页面时，养成这样的开发习惯是非常有益的。

这个测试可用于警告用户出现了一个严重错误，例如：

```
if (!oTarget) return alert("Warning: output element not found");
```

但在大多数情况下，一般的网站访问者不能处理这样的问题，所以脚本最好能禁止出现该问题，或者执行其他处理，例如通知 Web 管理员。

5. 删除原来的内容

HTML 文档预先填充了一些文本，后面将替换这些文本：

```
<p>It is currently <span id="output">now</span>.</p>
```

我们要删除中 span 的文本，再插入新内容。在表示页面内容的 DOM 中，span 是其所有子元素的父元素。span 现在只有一个子元素——单词"now"，但我们希望，即使 HTML 以后改为在目标元素中包含其他标记，这个脚本也能成功运行。

图 5-2 显示了从 DOM 角度看到的段落。p 元素有 3 个子元素：两个文本节点和 span。span 只有一个子节点——包含值 now 的文本节点。

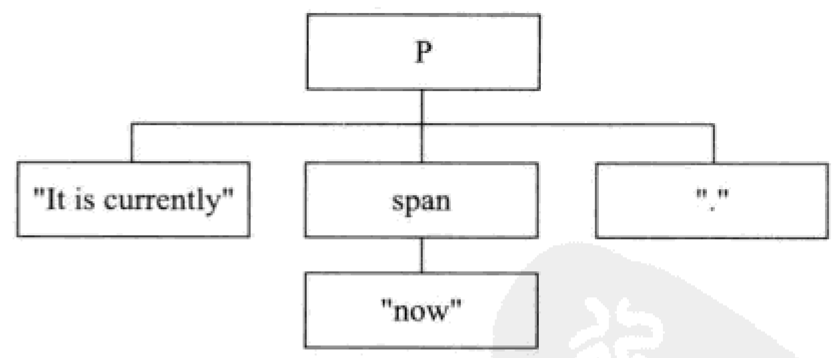


图 5-2 从 DOM 的角度来看插入之前的日期和时间段落

为了删除 span 的内容，执行一个简短循环，只要还有要删除的元素，该循环就一次删除一个子元素：

```
// delete everything inside the target
while (oTarget.firstChild)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    oTarget.removeChild(oTarget.firstChild);
}
```

对于当前的标记，这个循环只执行一次。只要圆括号中的表达式为 true，while 循环就会继续执行。该循环开始时，会计算 oTarget.firstChild 属性：oTarget 对象的第一个子节点存在吗？答案是肯定的(true)，因为它有文本节点 now。在循环中，removeChild()方法删除了第一个子节点。接着 JavaScript 循环再次询问相同的问题：oTarget.firstChild 存在吗？答案是否定的(false)，所以 JavaScript 停止处理 while 循环，而是执行闭合花括号后面的语句。

6. 插入日期和时间

最后在 DOM 中创建一个包含所需文本的新节点，并插入 span：

```
// use the date-time string to create a new text node for the page
var oNewText = document.createTextNode(sDateTime);

// insert the new text into the span
oTarget.appendChild(oNewText);
```

sDateTime 是从几个语句前面的 Date 对象中生成的字符串。这里调用与 document 对象关联的一个方法，该方法会创建一个新的文本节点，并把日期和时间字符串作为其值。此时，该文本节点还不是可见文档内容的一部分，它在等待中。使用目标元素的 appendChild()方法就可以插入它。

段落结构现在如图 5-3 所示。其结构形式是相同的，但 span 现在有一个新的子节点。

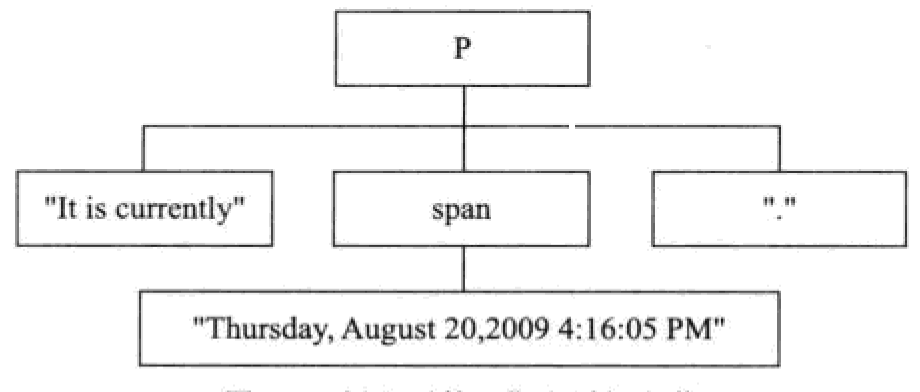


图 5-3 插入后的日期和时间段落

7. 调试

用 script 标记保存了 JavaScript 文件和 HTML 文件后，就应能在浏览器中加载 HTML 页面，看到 now 已被当前的日期和时间所取代。

如果还看不到，应花点时间找出问题并更正。在这个过程中，将学习 JavaScript 的重要细节，使将来的编程更快捷、更流畅。

校对 HTML 标记和 JavaScript 代码，确保它们匹配前面的程序清单。标记中的两个要点是 script 标记及其指向 JavaScript 文件的特性，以及 span 标记及 JavaScript 寻找的 ID。在 JavaScript 文件中，确保每个语句都拼写正确，包括空格、标点符号和大小写字母。

如果对 HTML 没有把握，在 W3C HTML (http://validator.w3.org/)上验证它总是不错的。选择 Validate by File Upload，从计算机中上传 HTML 文件，或选择 Validate by Direct Input，把 HTML 标记从文本编辑器中复制并粘贴到 Validator 中。在撰写本书时，本书中使用 HTML5

更多资源请访问稀酷客(www.ckook.com)

DOCTYPE 的所有示例都收到一个警告："使用试验性的特性：HTML5 一致性检查器"。这是一个提示性消息，不是一个错误。我们的目标是在 Validator 结果页面的顶部得到绿旗，它表示"这个文档成功通过了 HTML5 检查！"

检查浏览器的错误控制台，看看它是否报告了错误。如果是，它还可能指出脚本文件中错误的位置。

更多步骤可参阅第 48 章。

5.2.3　第三步：添加样式

最后，为了说明如何把典型网页的第三个组件关联进来，下面添加一个简单的样式表。样式表告诉浏览器页面应如何显示——布局、字体选择、背景色等。

在编辑器中创建一个新的文本文件，输入如程序清单 5-3 所示的样式表脚本，保存为 date-time.css：

程序清单 5-3　"Date & Time" : date-time.css 的 CSS 代码

```
@charset "utf-8";

*
{
   margin: 0;
   font-family: sans-serif;
}
h1
{
   font-size: 10em;
   color: #DDF;
}
p
{
   position: absolute;
   top: 2.5em;
   left: 1.5em;
   font-size: 2em;
   font-weight: bold;
   color: #338;
}
#output
{
   font-style: italic;
   color: #C33;
}
```

为了把这个样式表链接到 HTML 文件上，给 HTML head 添加一个 link 元素。在文本编辑器中打开 date-time.html，添加下面一行：

```
...
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>Date &amp; Time</title>
        <link href="date-time.css" rel="stylesheet" type="text/css" media="all">
        <script type="text/javascript" src="date-time.js"></script>
    </head>
...
```

保存样式表，保存修改过的 HTML 文件，在浏览器中重新加载页面，它应如图 5-4 所示。

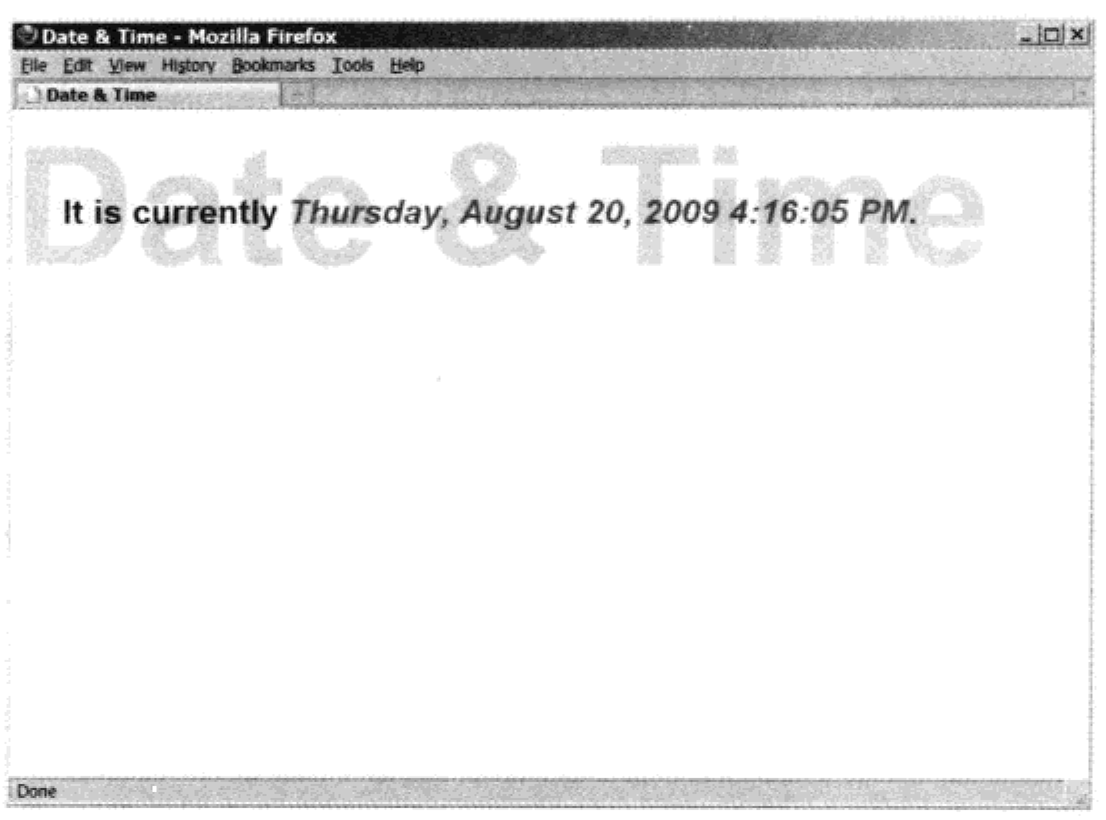


图 5-4 第三步：样式化的页面

简言之，样式表是应用于页面元素的一系列显示规则。基本语法如下：

```
selector { property: value; [...] }
```

上述样式表开始于一个字符编码指令：

```
@charset "utf-8";
```

注意它匹配 HTML 文件的字符编码。

*(通配符)选择器应用于页面上的所有元素。这里它指定默认页边距为 0，并设置了所有文本的字体系列。

h_1 选择器把标题的字号设置为基本字号的 10 倍(1em = 100%的基本字号，em 是西文排版行长单位)，字体颜色使用 DDF 的 RGB(red-green-blue，红-绿-蓝)16 进制代码设置为淡蓝色。

p 选择器应用于页面上的所有段落(这里只有一个段落)。Position、top 和 left 属性可以把段落定位在标题顶部。字体粗细设置为黑体，确保它在淡色的标题上可以读出来，文本颜色则用 RGB #338 设置为深蓝色。

#output 选择器应用于 ID 为 output 的元素。这里把字体样式改为斜体，文本颜色用 RGB #C33 改为深红色。这个元素还继承了其父元素的属性，包括粗体字体。

有关样式表(CSS)的更多信息，请访问 www.w3.org/Style/CSS/。

更多资源请访问稀酷客(www.ckook.com)

5.3　进行改动

这个脚本正常工作后，就对它进行一些改动。把 output ID 从一个元素移到另一个元素。修改要插入的文本。修改标记、脚本和样式表，获得自己感到满意的效果。在每次修改后，都要保存文件，在浏览器中再次加载页面，检查修改的效果。在学习过程中不要怕犯错误。总是可以撤消有问题的修改。反复尝试是最好的学习方式！

5.4　习题

(1) 对程序清单 5-1 中的 HTML 标记进行一次剪切-粘贴操作，让 JavaScript 用日期和时间替代整个段落，而不仅仅替代 span。提示：编辑后，该段落应包含文本和一个 span 元素。

(2) 对标记进行了上述修改后，考虑程序清单 5-2 的 JavaScript 中的 while 循环。写出该循环在删除段落内容时执行的每一步。

(3) 对于题(1)进行的 HTML 编辑，确定需要对程序清单 5-3 中的样式表进行哪些修改，才能给段落赋予与 span 以前相同的字体和颜色特性。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第6章

浏览器对象和文档对象

本章将介绍 JavaScript 的几个实际应用，并开始了解支持 JavaScript 的浏览器如何将熟悉的 HTML 元素转换为脚本控制的对象。本教程将介绍应用于现代浏览器的概念和术语，重点是标准的兼容性，以便你使用当今和未来的浏览器。学习这些内容应使用如下浏览器：Internet Explorer 5 或更高版本(Windows 或 Macintosh)、基于 Mozilla 的浏览器(Firefox、Netscape 7 或更高版本，或 Camino)、Apple Safari、Opera 7 或更高版本。

本章包含哪些内容？
客户端脚本的作用
载入文档时会发生什么
浏览器如何创建对象
脚本如何引用对象
如何区分对象

6.1 脚本运行初步

用 HTML 编写过网页的用户知道，在浏览器中查看内容时，HTML 标记会影响内容在页面上的呈现方式。浏览器在载入页面时，将尖括号中的标记识别为格式化指令。文档中的指令自上而下地读取，而 HTML 文档中定义的元素按照它们在文档源代码中的顺序显示在屏幕上。访问者每次将页面载入到浏览器中时，页面编写人员只预先进行少量的一次性工作(把标记添加到文本内容中)，而浏览器要进行大量的处理工作。

假设页面上的一个元素是表单中的文本输入域。用户在该文本域中输入一些文本，然后单击 Submit 按钮，将该信息发送给 Web 服务器。如果该信息是一个 Internet 电子邮件地址，如何确保用户在地址中包括了@符号？

一种方法是在用户单击 Submit 按钮，表单数据传送到服务器之后，Web 服务器就运行脚本(例如用 PHP、ASP、ColdFusion 或公共网关接口(CGI)语言编写的程序)，检查提交的表单数据。如果用户遗漏或忘记了@符号，服务器端程序就将页面发回浏览器——但这次指示用户要在地址中包含@符号。这个交换过程没什么问题，但它意味着，因为发现地址中没有包含这个关键符号，所以延迟了响应用户的时间。另外，Web 服务器必须使用其部分资源来执行验证，并将结果返回给访问者。如果 Web 站点十分繁忙，服务器可能要执行数百次此类验证，再次延长了响应用户的时间。

更多资源请访问稀酷客(www.ckook.com)

现在假设包括该文本输入域的文档有某种内置的智能，它确保在向服务器提交数据之前，文本输入域包含了@符号。这个功能必须以某种方式嵌入文档，并与文档的内容一起下载，才能在调用时执行它。浏览器还必须知道如何运行该嵌入的程序。用户必须执行某个操作来启动该功能，例如在用户单击 Submit 按钮时启动它。如果该功能在浏览器内部运行，并检测到缺少@符号，就应显示一条警告消息，通知用户出现了问题。此程序还应能决定是继续提交数据，还是等待用户在域中输入有效的电子邮件地址。

这种提交前验证数据项的功能只是 JavaScript 为 HTML 文档添加智能的可行功能之一。从此示例可以看出，脚本必须知道如何查看在文本域中输入的内容，如何继续提交和中止提交。支持 JavaScript 的浏览器可以将文本域等元素视为对象。JavaScript 脚本能控制对象的动作和行为，大多数动作和行为可在浏览器窗口的屏幕中看到。

6.2　使用 JavaScript 的场合

有了这么多面向 Web 的开发工具和语言，就应将客户端 JavaScript 用于最适合它们的任务。面对 Web 应用程序的任务时，可以用客户端的 JavaScript 解决以下问题：

- **数据项的验证**：假如需要填写表单字段，以便在服务器上处理，可以用客户端脚本预先验证用户输入的数据。
- **无服务器的 CGI**：这个术语表示，如果没有 JavaScript，该过程会用作服务器上的 CGI。由于程序和用户之间需要交互，性能会降低。这些过程包括少量数据集合查询、图像修改、在其他框架和窗口上根据用户输入生成 HTML。
- **动态 HTML 交互**：使用 DHTML 能精确地定位页面上的元素，而不必为其编程。但假如想让页面的内容动起来，就需要编程了。
- **CGI 原型**：有时希望 CGI 程序位于应用程序的根部，因为这样可以降低浏览器品牌和版本之间的不兼容性。在客户端的 JavaScript 中构建 CGI 的原型是比较方便的，在服务器端脚本中实现应用程序之前，采用这种方式可以完善用户界面。
- **减轻繁忙服务器的负担**：如果 Web 站点的流量很高，则把频繁使用的 CGI 程序转换为客户端 JavaScript 脚本是很有用的。页面下载完毕后，服务器就可以给其他访问者提供服务。这不仅可减轻服务器的负担，用户也可以从嵌入页面的应用程序中获得更快的响应速度。
- **给静态的页面增加活力**：HTML 本身是很单调的。给文本加入闪烁特效也没有多大帮助；动态 GIF 图像经常分散而不是集中用户对站点的注意力。但是，若给页面添加一些交互效果，就可能吸引用户，用户还可能将其推荐给朋友，或者再次访问该站点。
- **创建“智能 Web 页面”**：充分发挥想象力，可以想出有趣的新办法，让网页看起来很“聪明”。比如，在 Intelligent“Updated”Flags(配书光盘的第 57 章)应用程序中，HTML 页面可以记住访问者上次何时访问页面(没有服务器 CGI 或数据库)，并将上次访问后更新的内容(不管页面已经更新了多少次)标记出来。这种巧妙善思的 Web 页面很好地展现了 JavaScript 的魔力。

更多资源请访问稀酷客(www.ckook.com)

注意：

用于公共访问的网页和应用程序不应仅依赖 JavaScript 来提供重要功能。要保证关闭了 JavaScript 的访问者和不支持 JavaScript 的用户代理(例如搜索引擎和移动设备)可以访问基本数据和网站功能。对于使用支持 JavaScript 的浏览器的大多数访问者来说，编写脚本是为了增强其体验，且网站不会在其他访问者访问时崩溃。

6.3　文档对象模型

在正式编写脚本之前，要对所编程的对象有很好的认识。脚本浏览器做了许多工作来创建软件对象，它们一般是在浏览器窗口的 HTML 页面中显示的可见对象，可见对象包括表单控件(文本框和按钮)和图像。然而，页面上也有一些不可见的对象，它们对于生成页面内容的 HTML 标记是很有意义的，比如段落对象或框架集中的框架。

为了帮助脚本控制这些对象，并帮助页面设计者控制页面上潜在的大量对象所造成的混乱，浏览器厂商定义了文档对象模型(DOM)。在 DOM 中，模型就是页面上对象的组织方式。

由于后续的浏览器版本和品牌之间缺乏兼容性，浏览器 DOM 的发展给脚本开发者带来了许多混乱和惊慌。幸好，在 W3C 发布的正式规范上固化了 DOM。当今的浏览器继续支持早期 DOM 的一些老方式，因为 Web 上有如此之多的脚本代码要依靠这些老方式来工作(参见第 11 章)。但由于今天的绝大多数浏览器都支持基本的 W3C DOM 语法和术语，因此脚本编写人员应尽可能与标准兼容。

6.3.1　HTML 结构和 DOM

HTML 标记的一个重要趋势是只将标记用于定义文档的结构和文档中各个内容的上下文。只用 HTML 标记来改变文本块外观的日子一去不复返了，不再能将一行文本放入<h1>标记，以浏览器自动应用的字号和粗体来显示。<hl>元素在文档结构中具有特殊的含义：一级标题。在目前的 HTML 中，如果希望用特殊的样式显示单行文本，该文本应放在简单的段落(<p>)标记中，其外观由层叠样式表(CSS)规则来控制。现在甚至不赞成用<b>和<i>标记给一段文本指定粗体和斜体样式，而将文本放入上下文标记(如<em>元素以表示强调)，并给文档中要强调的文本定义希望使用的 CSS 样式。

对 HTML 标记应用严格的结构化设计，文档就会根据其元素的嵌套关系生成完好定义的层次结构。例如，格式正确的 HTML 文档至少包含以下元素：

- 文档类型声明
- HTML
- Head
  - 字符编码
  - Title
- Body
  - 块级内容

这个空文档的 HTML 标记如下所示：

更多资源请访问稀酷客(www.ckook.com)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
      <title></title>
   </head>
   <body>
      <p></p>
   </body>
</html>
```

HTML 文档可以看作家谱，其中每个子元素只有一个父元素，如图 6-1 所示。HTML 文档必须总是包含两个子元素 DOCTYPE 和 html。

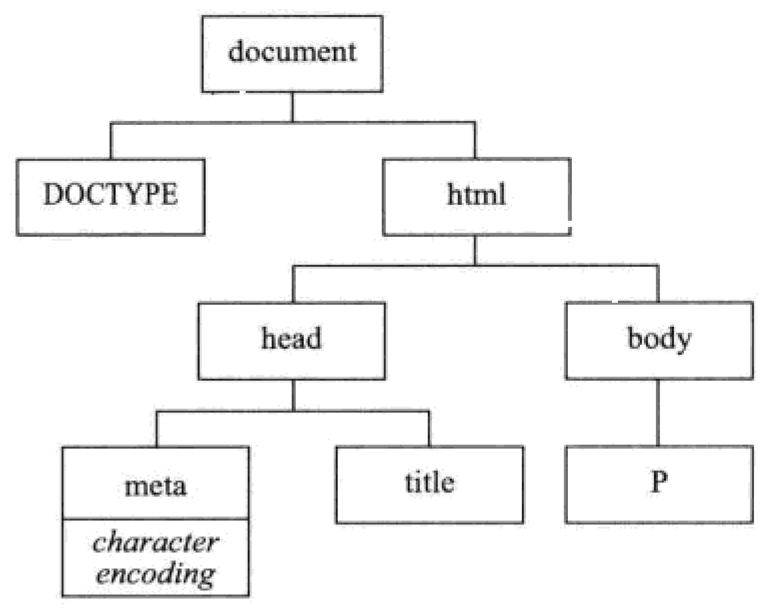


图 6-1　空 HTML 4.01 文档的元素层次结构

DOCTYPE 告诉浏览器使用哪个版本的标记，这对标记的验证和样式表规则的解释都非常重要。html 有两个必选的子元素 head 和 body。head 至少要包含一个 meta 元素，它指定字符编码和 title，head 还可以包含其他子元素，例如脚本和样式表链接。head 中的元素一般不在文档页面中显示为对象，而是告诉浏览器如何配置文档，且可以从根本上影响其外观和行为。body 包含我们通常视为页面内容的文本、图像和其他元素，它至少要包含一个块级元素，例如标题、段落或分部。

尽管本书给 HTML5 使用更简单的 DOCTYPE 标记：

```
<!DOCTYPE html>
```

但我们仍遵循 HTML 4.01 的标记规则，这样 HTML 标记就可以在最大程度上验证为 HTML 4.01 和 HTML5，为读者选择使用的标记提供有效的例子。

6.3.2　浏览器窗口中的 DOM

顾名思义，正式的 DOM 主要关注 HTML 文档及其中的内容。但实际上，脚本编写人员通常需要控制文档所在的环境：窗口。window 对象是浏览器脚本所处理的层次结构的顶层。在

更多资源请访问稀酷客(www.ckook.com)

现代浏览器中，对象模型的基本结构(假设是空白 HTML 文档)如图 6-2 所示。

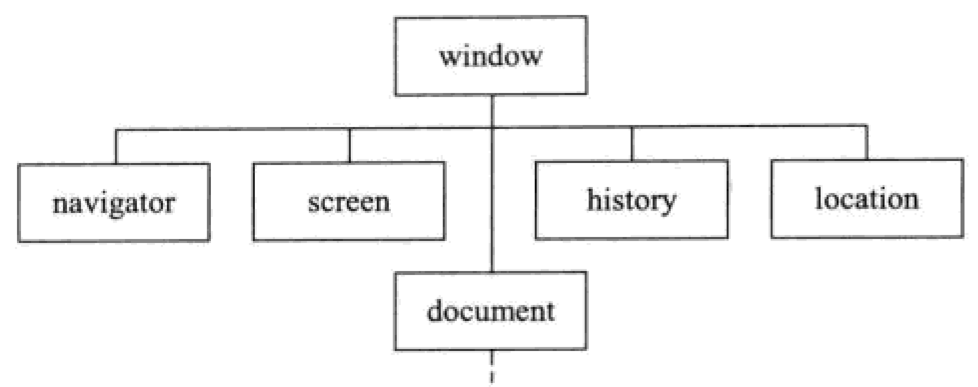


图 6-2　所有现代浏览器的基本对象模型

为了说明顶层对象之间的关系，下面描述了它们的作用：

- **window 对象**：在层次结构的顶部是 window 对象。这个对象代表显示 HTML 文档的浏览器窗口的内容区域。在多框架环境下，每个框架也是一个窗口。所有文档动作都是在窗口内发生的，所以窗口是对象层次结构中最外部的元素，它包含文档。
- **navigator 对象**：脚本从这个对象开始访问浏览器程序，主要是读取浏览器的品牌和版本。此对象为只读，以禁止流氓脚本对浏览器执行不当操作。但如后面所述，不能依赖 navigator 对象获得当前浏览器的实际品牌、型号和版本。
- **screen 对象**：这是另一个只读对象，它给脚本提供了运行浏览器的物理环境信息。例如，此对象显示了监视器的高度和宽度(像素值)。
- **history 对象**：尽管浏览器保留了其最近浏览历史的内部细节(如单击后退按钮后显示的内容列表)，脚本却无权访问这些细节。这一对象可以帮助脚本模拟后退或前进按钮的单击。
- **location 对象**：此对象主要用于将另一个页面载入到当前的窗口或框架中。窗口的 URL 信息在严格控制的环境下才可用，这样脚本就不能跟踪对其他 Web 站点的访问。
- **document 对象**：每个载入窗口的 HTML 文档都会变成一个 document 对象。document 对象包含脚本中的内容。除了每个 HTML 文档中都有的 html、head 和 body 元素对象外，文档中元素对象层次的具体标记和结构取决于文档中的内容。

6.4　文档的载入

编程语言(比如 JavaScript)是人们头脑中对程序运行方式的想象和计算机内部实际工作之间的一种方便媒介。在计算机内部，程序清单中的每个词都影响着位(计算机二进制的 1 和 0)在 RAM 存储器之间的转换和移动。语言和对象模型在计算机内部(对于 JavaScript 和 DOM，语言和对象模型在计算机的浏览器区域内部)，所以程序员能方便地查看程序的运行情况，并查看其结果。这个关系就好像从 A 处开车到 B 处，并不需要知道内燃机、转向联动装置以及其他内部零件的工作原理，只需要控制高级对象，诸如点火钥匙、变速杆、油门、刹车和方向盘，就可以到达目的地。

当然，编程和开一辆自动换挡的车不是一回事，脚本编程甚至还需要完成打开车盖，检查

更多资源请访问稀酷客(www.ckook.com)

液压传动装置或换油等工作。因此，现在打开车盖，看看在页面载入浏览器时，文档的对象模型中都发生了什么事情。

6.4.1 简单文档

图 6-3 给出了稍后添加的文档中的 HTML 及对应的对象模型。图中只显示了 document 对象部分：即使是空文档，也总是包含 window 对象及其他顶层对象(包括 document 对象)。载入此页面时，浏览器在其内存中保存由文档中的 HTML 标记生成的对象映射。Body 必须包含块级内容，但现在还没有添加它们。

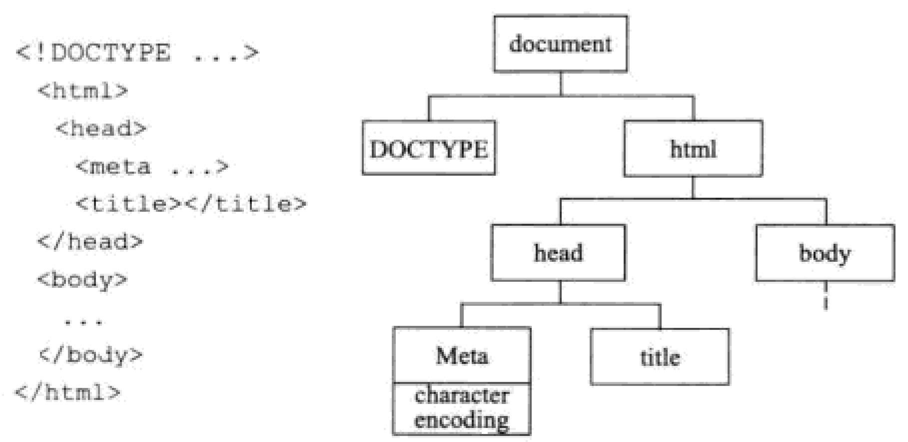


图 6-3 空文档的对象映射

6.4.2 添加段落元素

现在把 HTML 文件改为包含一个空段落元素，并重新载入文档。图 6-4 给出了 HTML 和浏览器构造的对象映射的变化(显示为粗体)。这个段落中没有内容，<p>标记只是告诉浏览器创建该 p 元素对象。还要注意，在当前映射的对象层次结构中，p 元素对象包含在 body 元素对象中。换句话说，p 元素对象是 body 元素对象的子元素。对象层次结构与 HTML 标记包含层次结构相匹配。

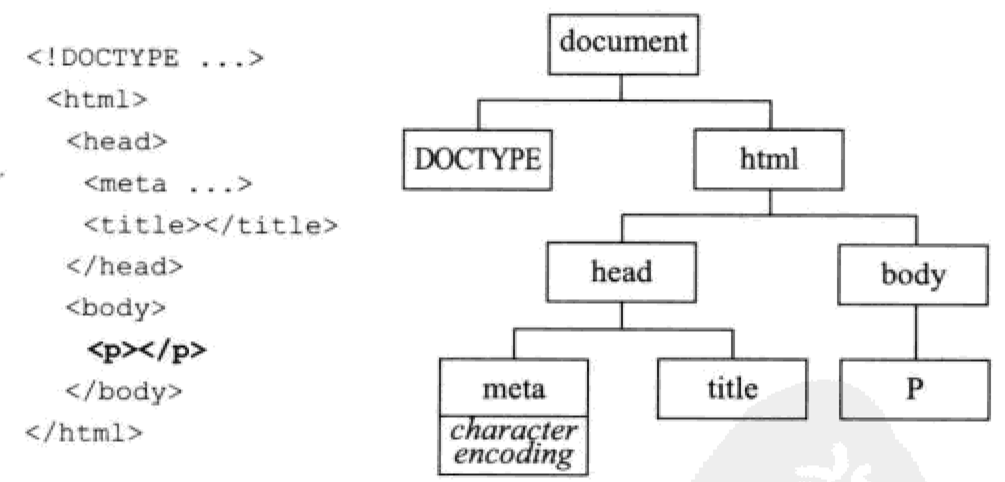


图 6-4 添加空的段落元素

6.4.3 添加段落文本

再次修改 HTML 文件，这次在段落元素的开始和结束标记之间插入段落文本，如图 6-5 所示，然后重新载入该文件。在标记间插入的连续文本是 DOM 中的一种特殊对象，称为文本节点。文本节点总是有一个容器元素，也就是说，文本节点是其父元素 p 的子元素。现在文档对

更多资源请访问稀酷客(www.ckook.com)

象树有一个分支包含了几个层级：document→html→body→p→文本节点。

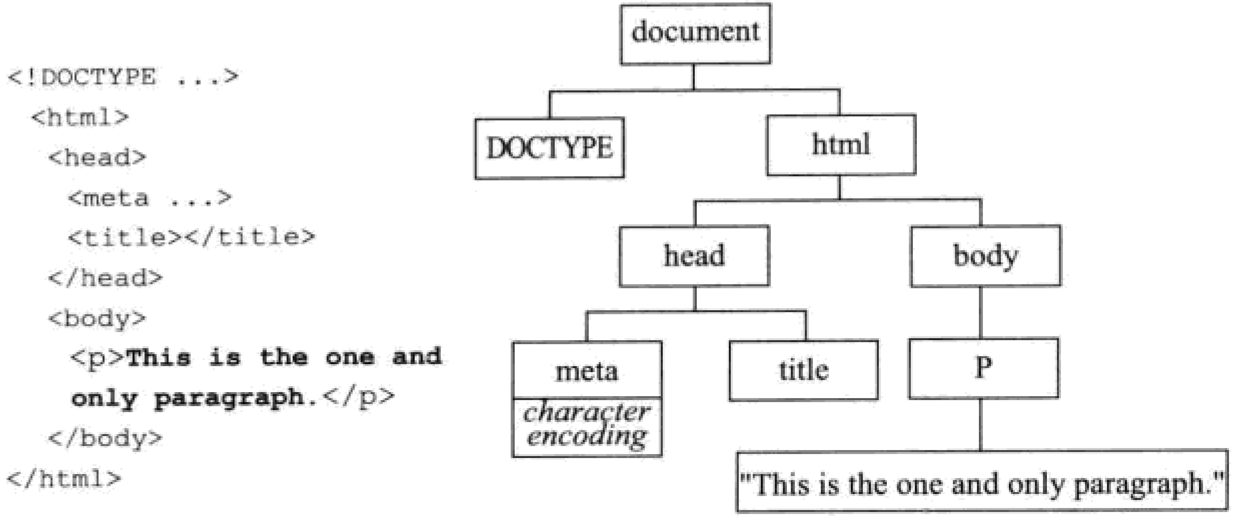


图 6-5　给 P 元素对象添加文本节点

6.4.4 生成新元素

对文件的最后一个修改是将部分段落文本包含在一个<em>标记中，表示要强调这些文本。这个插入操作对 p 元素对象的层次结构有很大的影响，如图 6-6 所示。p 元素从只有一个(文本节点)子元素变成有三个子元素：两个文本节点，和一个在它们之间的元素。在 W3C DOM 中，文本节点不能有任何子元素，因此不能包含元素对象。em 元素中的文本节点不再是 p 元素的子元素，而是 em 元素的子元素。该文本节点现在是 p 元素对象的孙子元素。

理解了如何在内存中创建对象来响应 HTML 标记后，下一步是弄清楚脚本如何与这些对象通信。毕竟，脚本编程主要是控制这些对象。

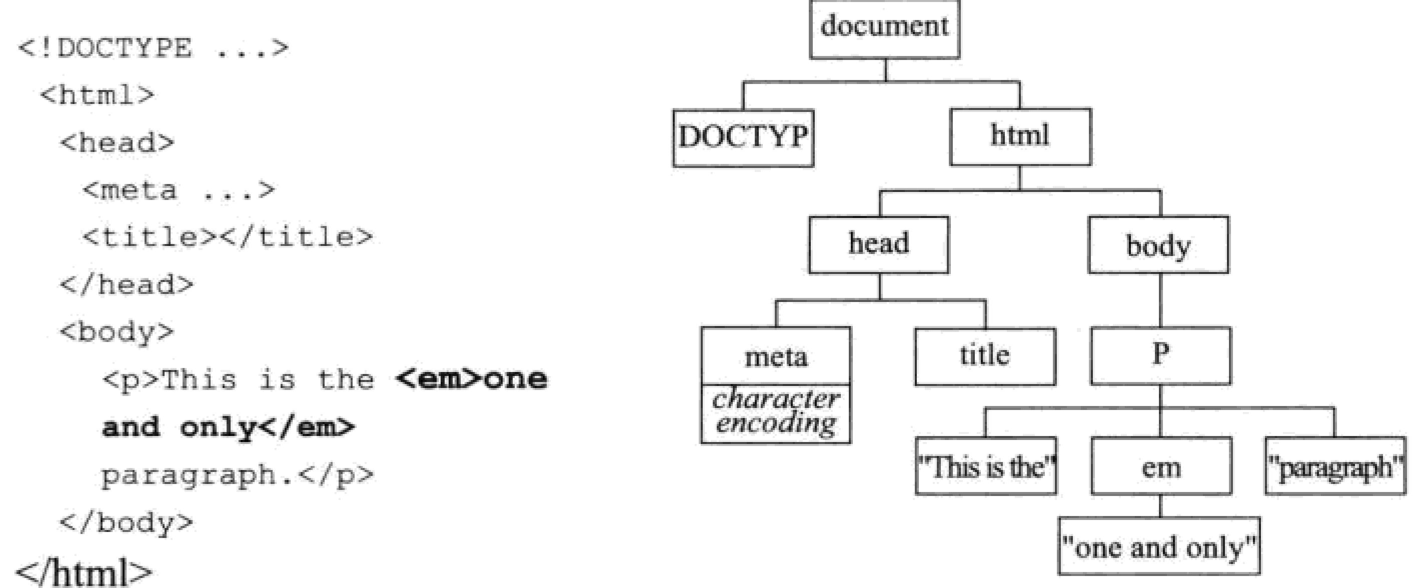


图 6-6　在文本节点中插入元素

6.5 对象引用

将文档载入浏览器后，它的所有对象就按照浏览器文档对象模型指定的层次结构安全地存放在内存里。脚本要控制其中的一个对象，就必须有一种方法与该对象通信，并找出与其相关的信息，比如，“嘿，文本字段，用户输入什么了？”为了让脚本与对象通信，需要一种指向

更多资源请访问稀酷客(www.ckook.com)

该对象的方法，这就是脚本中的对象引用。

6.5.1　对象命名

给对象创建脚本引用时，最有用的方式是给 HTML 中每个可编写脚本的对象指定名称。在 W3C DOM(以及当前的 HTML 规范)中，要使用 id 特性给元素指定名称。这个特性是可选的，如果打算使用脚本来访问页面中的元素，最方便的方式是，在 HTML 代码中，直接通过元素的 id 特性指定其名称。下面的示例是给一些标记添加 id 特性：

```
<p id="firstParagraph">

<img id="logo-1" src="images/logo.jpg" alt="Corporate Logo">

<div class="draggable" id="puzzle_piece">
```

对象 ID(也叫做标识符)有如下规则：

- 不能包含空格。
- 不能包含标点符号，但下划线、连字符、句点和冒号除外。
- 赋予 id 特性的值必须放在引号内。
- 不能以数字开头。
- 同一个文档不能包含相同的名称。

指定 ID 可以看作给每个参会者指定名签。要找到某个姓名已知的参会者，可在入口处等待，查看每个名签，直至找到该姓名，也可以在参会者中无目的地搜寻，以期找到该姓名。但最好能设法很快按姓名确定参会者，如在公共广播系统向整个人群广播该名字。

6.5.2　引用特定对象

W3C DOM 允许即时访问文档中已命名的元素。编写浏览器脚本的常用语法是：

```
window.document.getElementById("elementID")
```

使用要引用的元素的 ID 替换 elementID。例如，如果要引用 ID 为 firstParagraph 的段落元素，引用代码为：

```
window.document.getElementById("firstParagraph")
```

JavaScript 允许省略 window 引用以缩短这个语句，所以我们常常使用如下简短格式：

```
document.getElementById("firstParagraph")
```

注意，JavaScript 区分大小写。命令中的三个大写字母一定要大写，结尾的 d 要小写，而 ID 必须大写。

getElementById()命令属于 document 对象，这意味着这一即时搜索操作将在整个文档的所有元素中查找匹配的 ID。JavaScript 为此使用了圆点(传统的句点符号)，即圆点左侧的项(这里是 document 对象)把圆点右侧的项(这里是 getElementById())作为一个资源，可以根据需要调用它。每个对象都有一组这样的资源，稍后介绍(在附录 A 中汇总)。

更多资源请访问稀酷客(www.ckook.com)

id 与 name 特性

在 HTML 4.0 规范引入 id 特性之前，脚本可以访问少数支持 name 特性的元素。支持 name 特性的元素主要与表单、图像和框架有关。第 11 章将介绍 name 特性在表单中的工作方式。事实上，大多数浏览器仍然要求表单和表单控件(文本域、按钮和选择列表)有 name 特性，以便将其数据提交给服务器。可将相同的标识符赋予元素的 id 和 name 特性，实际上在 XHTML 中，建议给一个元素的 id 和 name 特性赋予相同的值。例如：

```
<input type="text" id="firstname" name="firstname" />
```

6.6　节点术语

W3C DOM 术语使用比喻，来帮助程序员了解文档及其内容的包含层次结构。在学习初期应该掌握的一个概念是节点；另一个概念是文档中对象之间的关系。

6.6.1　节点

英语词典包含节点的许多定义，但与 W3C DOM 最接近的定义是指树枝上的一个结节或隆起。树枝上的此类结节通常会产生两种事物：树叶或另一个树枝。树叶是一个终端，因为树枝不会从树叶上长出来，树枝类结节会长出新的树枝，新树枝可以有结节，从这些结节中可以长出树叶或树枝。W3C DOM 在定义 HTML 文档的结构时，也定义了一个节点结构(也称为节点树)，其树枝和树叶的位置完全取决于 HTML 元素和文本内容。

在 W3C DOM 中，基本的构建块是普通的节点。但在 HTML 文档中，需要处理某些特殊的节点。脚本最常处理的两类节点是元素节点和文本节点，元素节点对应于 HTML 元素，文本节点对应于元素的开始和结束标记间的文本。前面在编写 HTML 的过程中已使用了元素和文本节点，只是没有提及它们而已。

再看看前面编写的简单文档及其包含层次结构图，如图 6-7 所示。代表 HTML 元素(html、head、body、p 和 em)的框是元素节点，包含文档中实际文本的三个框是文本节点。前面在一个长文本节点(参见图 6-5)中插入 em 元素(参见图 6-6)后，长文本节点就分为了三块。相对于包含元素 p，两个文本节点在层次结构中的位置保持不变，新元素 em 插入到两个文本节点之间，将第三个文本节点从 p 元素下移了一级。

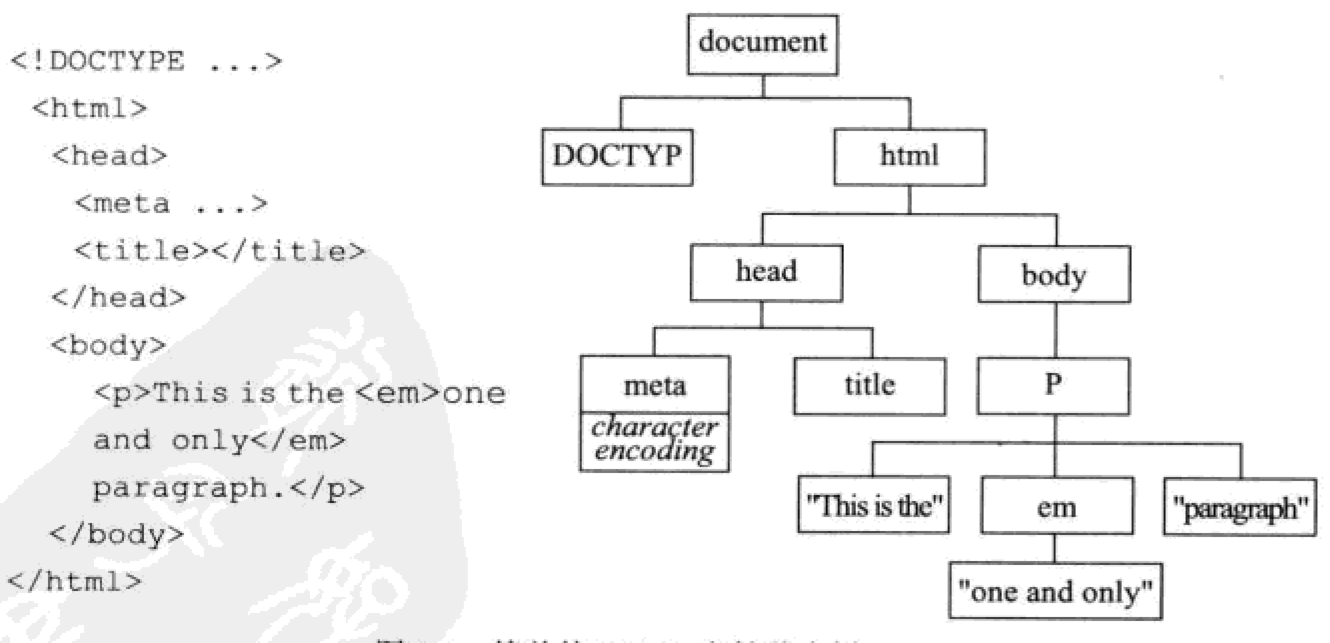


图 6-7　简单的 HTML 文档节点树

更多资源请访问稀酷客(www.ckook.com)

6.6.2　父子节点

查看图 6-7 中的 p 元素及其内容，可以看到该元素有三个子节点。第一个和最后一个都是文本节点类型，而中间一个是元素节点。当元素包含多个子节点时，子节点的顺序完全取决于 HTML 源代码的顺序。因此，p 元素的第一个子节点是包含文本“This is the”的文本节点。em 元素的子文本节点是其唯一的后代。

元素节点的子节点并非总是文本节点，树枝也并非总是以文本节点结束。在图 6-7 中，html 元素有两个子节点，它们都是元素节点；body 元素有一个子节点 p 元素。HTML 中的标记表示元素节点，无论其是否有子节点。与此形成对照的是，文本节点永远不能包含其他节点，它是一种位于终端的叶节点。

注意，子节点总是包含在一个元素节点中。该容器是其子节点的父(parent)节点。例如，em 元素节点有一个子节点(文本节点)和一个父节点(p 元素节点)。许多 W3C DOM 术语(参见第 25 章)都有助于脚本在需要时从文档层次结构的任意位置开始，获得对相关节点的引用。例如，如果动态 HTML 脚本要修改图 6-7 中 em 元素的文本，通常先用 document.getElementById()命令(假设给 em 元素赋予了 ID)获得对 em 元素的引用，然后修改该元素的子节点。

节点树顶部的 document 对象本身就是一个节点，称为文档节点，它在树中的位置很特殊。每个载入的 HTML 文档都包含一个文档节点，该节点是脚本开发者通向文档其余节点之门。所以引用元素节点的语法 document.getElement ById()以 document 对象的引用开头。

6.7　对象的定义

HTML 标记在源代码中定义对象，这样浏览器在载入页面时，会在内存中给该对象分配一块区域。但对象并不只是内存中存储的一串数字，它要复杂得多，其作用是代表某个“东西”。在浏览器及其 DOM 中，对象常常代表元素，比如输入文本域、表格元素或整个 HTML 文档。在 DOM 外部，对象也可以代表抽象的实体，比如日历程序的约会项，或者画图程序中的一个图层。浏览器脚本常操作 DOM 对象和用户自己设计的抽象对象。

在某种程度上，每个 DOM 对象都是唯一的，但有时两个或多个对象在浏览器中看起来是一样的。对象定义的三个非常重要的方面是：它像什么、表现出的行为以及脚本如何控制它，分别称为属性、方法和事件(也称为处理程序)。它们在将来的 DOM 脚本中起重要的作用。附录 A 中的快速参考针对各种浏览器实现的对象模型中的每个对象，汇总了每个对象的属性、方法和事件。

6.7.1　属性

任何物理对象都有一些定义它的特征，比如，硬币有形状、直径、厚度、颜色、重量、其两面都有浮雕的图像，以及无数将它与羽毛区别开的特征，这些特征就叫做属性。每个属性有一个对应的值(值可以是空或 null)。比如，硬币的 shape 属性可以是 circle，这是一个文本值；而它的 denomination(面值)属性就是一个数值。

只要为脚本浏览器编写过 HTML，就设置过对象属性，但不必编写 JavaScript 代码。设置

更多资源请访问稀酷客(www.ckook.com)

HTML 元素对象的初始属性时，最常见的方法是使用标记特性。比如，下面的 HTML 标记定义了一个 input 元素对象，它指定了 4 个属性值：

```
<input type="button" id="clicker" name="clicker" value="Hit Me...">
```

根据 JavaScript 的术语，type 属性的值是 button，id 和 name 属性的值都是 clicker，而 value 属性是显示在按钮上的文字 Hit Me...。实际上，按钮输入元素的属性不止于此，但不必为每个对象设置每个属性。多数属性都有默认值，假如 HTML 或脚本没有特别要求，它们就自动设置为默认值。

在载入了文档，用户与页面交互后，一些属性的内容可能会发生变化。考虑下面的文本输入标记：

```
<input type="text" id="donation" name="donation" value="$0.00">
```

这个对象的 id 和 name 属性都是 donation。载入这个页面时，value 属性设置的文本显示文本域中，这是指定 value 属性时 HTML 文本域的自动行为。但是，假如用户在文本域中输入其他文本，value 属性就会改变——不在 HTML 中改变，而是在浏览器维护的对象模型的内存副本中改变。因此，假如脚本访问文本域的 value 属性，浏览器将返回该属性的当前设置，它不一定是 HTML 指定的。

为了访问对象的属性，需要使用和前面对象相同的圆点定位方式。属性是从属于对象的资源，所以属性引用就是对象引用后加属性名。因此，对于刚才的 button 和 text 对象标记，各种属性的引用是：

```
document.getElementById("clicker").name
document.getElementById("clicker").value
document.getElementById("entry").value
```

提示：

如果 JavaScript 尝试引用不存在的元素的属性，就会失败或“终止”。因此引用这些属性的较安全方式是先测试一下对象是否存在：

```
var oClicker = document.getElementById("clicker");
   if (oClicker) var sName = oClicker.name;
```

或

```
var oClicker = document.getElementById("clicker");
   if (!oClicker) return;
var sName = oClicker.name; ■
```

引用的 window 部分哪里去了？其实，一个窗口只能包含一个文档，所以文档中对象的引用可以省略 window 部分，而从 document 开始；但不能从引用中省略 document 对象。

6.7.2 方法

如果说属性是描述对象的形容词，方法就是描述对象的动词。方法是和对象相关的动作。方法或者对对象起作用，或者和对象一起对文档或脚本的其他部分施加影响。方法是一种指令，

更多资源请访问稀酷客(www.ckook.com)

其行为和具体的对象相关。

Internet Explorer 引用

在 W3C DOM 推出之前，Microsoft 创建了通过元素的 id 特性来引用元素对象的方法。在仅为 IE4+编写的代码中，多处采用了这种语法。该语法使用 document.all 结构。这一结构有几种不同的用法，但最常用的方式是在圆点的后面加上元素的 ID。例如，如果段落元素的 ID 是 myParagraph，则仅用于 IE 的引用语法为：

```
document.all.myParagraph
```

也可省略引用的前导部分，直接引用元素的 ID：

```
myParagraph
```

注意，W3C DOM 标准不支持这种语法。IE5+实现了这种 IE 专用语法和 W3C DOM 引用语法。另外，document.all.elementId 语法不允许使用某些有效的 HTML ID 字符，例如连字符、冒号和句点，因为 JavaScript 会把它们解释为它自己的语法。

只有熟悉 document.all，才能理解上述代码，但在自己的代码中应使用 W3C DOM 方法，例如 getElementBy()，以兼容现代浏览器。

注意：

document.all 语法不允许在 ID 中使用连字符、冒号和句点，尽管这些是有效的 HTML ID：

```
my-paragraph
my.paragraph
my:paragraph
```

但所得到的 document.all 表达式会使 JavaScript 得到错误结果甚至崩溃：

```
document.all.my-paragraph
document.all.my.paragraph
document.all.my:paragraph
```

在现代标准中，getElementBy Id()比较安全，因为 ID 总是包含在引号中。

一个对象可以有任意多个方法(包括什么都不做的方法)。为了让方法执行操作(通常称为调用方法)，JavaScript 语句必须通过其对象包括该方法的引用：在方法名后加上一对括号，如下所示：

```
var oForm = document.getElementById("orderForm");
oForm.submit();

var oEntry = document.getElementById("entry");
oEntry.focus();
```

第一个方法将表单 orderForm 发送到服务器上，第二个方法使文本域 entry 获得焦点。

有时为了完成任务，还要给方法发送一些附加的信息，给方法发送的信息称为形参或实参。例如，document.getElementById()方法需要一个形参：要寻址的元素对象的标识符。这个方法的

更多资源请访问稀酷客(www.ckook.com)

形参值必须是纯文本，并且放到引号中。

一些方法需要多个形参，此时，形参由逗号隔开。比如，现代浏览器支持的一个 window 对象方法可以把窗口移动到屏幕的某个坐标点上，坐标点由两个数字定义，这两个数字表示距离屏幕左边缘和上边缘的像素值。要将浏览器窗口移到距离屏幕左边缘 50 像素，距离屏幕上边缘 100 像素的地方，其方法是：

```
window.moveTo(50,100)
```

在学习 JavaScript 和文档对象的更多内容时，请注意每个对象定义的各个方法，它们有助于了解对象在脚本的控制下能执行什么操作。

6.7.3　事件

DOM 对象的最后一个特征是事件。事件是文档中执行的动作，通常是用户活动的结果。触发事件的用户活动通常是单击按钮，或在文本域中键入字符。某些事件并不明显，比如将文档载入浏览器窗口，或者图像载入时发生网络错误。

文档中的每个 DOM 对象几乎都接收某种类型的事件，请参见附录 A。脚本编写者的任务就是编写代码，告诉元素对象，只要元素接收到某种事件，就执行一个动作。这个动作就是执行其他一些 JavaScript 代码。

学习事件的一种简单方式，就是对 HTML 元素应用事件。程序清单 6-1 显示了一个非常简单的文档，它显示一个按钮，脚本则把一个事件处理程序应用于该按钮。事件的名称包含事件的类型(例如 click)及其前缀 on(表示“接收到 click 事件…”：onclick)。

程序清单 6-1　具有事件处理程序的简单按钮

HTML: jsb-06-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>A Simple Button with an Event Handler</title>
      <script type="text/javascript" src="jsb-06-01.js"></script>
   </head>
   <body>
      <form action="">
         <div>
            <button id="clicker">Click me</button>
         </div>
      </form>
   </body>
</html>
```

JavaScript: jsb-06-01.js

```
// tell the browser to run this script when the page has finished loading
window.onload = applyBehavior;

// apply behavior to the button
```


更多资源请访问稀酷客(www.ckook.com)

```
function applyBehavior()
{
        // ensure a DOM-aware user agent
        if (document.getElementById)
        {
            // point to the button
            var oButton = document.getElementById('clicker');

                // if it exists, apply behavior
                if (oButton)
                {
                oButton.onclick = behave;
            }
        }
}

// what to do when the button is clicked
function behave(evt)
{
alert('Ouch! ');
}
```

浏览器载入 HTML 页面时，也载入了链接的 JavaScript 文件，编译并开始执行其指令。在这个例子中，它执行的唯一指令是 window.onload 语句，因为其他语句都在函数中，在按名称调用这些函数之前，它们是不会执行的。仅当 HTML 页面载入完毕(其 DOM 也构建完成)，applyBehavior()函数才会运行。如果所运行的 JavaScript 解释器知道 DOM 方法 getElementById()，该函数就使用它查找 ID 为 clicker 的元素。如果找到，就把 onclick 行为应用于按钮，告诉它被单击时执行 behave()函数。用户单击按钮时，behave()函数就插入一个警告(弹出式对话框)，显示“Ouch!”。

交叉引用：
在第 25 章和第 32 章，可以学到把脚本语句连接到事件上的其他方式。

6.8　习题

1．下面哪些应用程序最适合编写为客户端的 JavaScript？请说明原因。

a．产品目录页面，访问者可以用 5 种不同的颜色查看产品

b．显示当前页面访问总数的计数器

c．聊天室

d．图形化的计算器，将华氏温度转换为摄氏温度

e．以上都是

f.　以上都不是

2．以下哪些元素 ID 在 HTML 中是合法的？对于每个不合法的 ID，解释不合法的原因。

a．lastName

b．company_name

更多资源请访问稀酷客(www.ckook.com)

c. LstLineAddress

d. zip code

e. today's_date

f. now:you-hear.this

3. 参照图 6-7，绘制浏览器在其内存中为以下 HTML 创建的对象模型包含层次图。使用 W3C DOM 语法写出对第一个段落元素的脚本引用。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Search Form</title>
</head>
<body>
<p id="logoPar"><img src="images/logo.jpg" height="90" width="300"
     alt="Logo"></p>
<div id="searchForm">
<form action="cgi-bin/search.pl" method="post">
<div>
<label for="searchText">Search for:</label>
<input type="text" id="searchText" name="searchText">
<input type="submit" value="Search">
</div>
</form>
</div>
</body>
</html>
```

4. 描述文本节点与元素节点的至少两个共同特征，描述区分文本节点与元素节点的至少两个特征。

5. 给按钮输入元素 Hi 编写 HTML 标记和 JavaScript 代码，其可见标签是 Howdy，单击该按钮时，会显示一个内容为“Hello to you，too!”的警告对话框。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第 7 章

脚本和 HTML 文档

第 4 章介绍了把 JavaScript 和 HTML 文档连接起来的许多基础知识。本章将讨论脚本如何链接到 HTML 文档中，以及脚本语句的构成；还将论述载入文档或响应用户的动作时，如何运行脚本语句。最后学习如何找出脚本的错误信息。

本章包含哪些内容？
在 HTML 文档中放置脚本
JavaScript 语句
运行脚本
查看脚本错误

7.1 把脚本连接到文档上

使用 script 元素可以告诉浏览器把一段文本看作脚本，而不是 HTML。无论脚本是链接到 HTML 文档上的外部文件，还是直接嵌入页面中，脚本都放在<script>...</script>标记对中。

不同的浏览器可以给<script>标记设置不同的特性，来管理脚本。type 特性告诉浏览器，标记中的代码应看作 JavaScript。其他一些浏览器接受其他语言(比如，在 Internet Explorer 的 Windows 版本中的 Microsoft VBScript)。所有现代的脚本浏览器都能接受下面的设置：

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

一定要包含脚本的尾标记。把 JavaScript 程序代码行放在 src 特性指定的外部文件中：

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

或者放在两个标记之间：

```
<script type="text/javascript">
   one or more lines of JavaScript code here
</script>
```

假如忘了尾标记，脚本就不能正确运行，页面中其他地方的 HTML 也会显得很奇怪。

旧的语言属性

另一个<script>标记特性 language 过去用于指定所封装的代码使用的脚本语言。脚本开发人员可以使用该特性指定语言的版本。例如，如果脚本代码需要仅用于第 4 版浏览器(它实现了

更多资源请访问稀酷客(www.ckook.com)

JavaScript 1.2 版本)的 JavaScript 语法，<script>标记就如下所示：

```
<script language="JavaScript1.2">...</script>
```

Language 特性从来没有被纳入 HTML 4.0 规范，现在已废弃。如果 W3C 验证是开发重点之一，则注意该特性在严格的 HTML4.01 或 XHTML1.0 版本中无效。较旧浏览器无法识别 type 特性，所以会自动将脚本语言设置为默认的 JavaScript。尽可能只使用 type 特性。

7.1.1　script 标记的位置

这些 script 标记放在文档中的什么位置？可放在文档中任何需要它们的地方。在大多数情况下，可将 script 标记放在<head>...</head>标记对中；但有时要放在<body>...</body>段中的特殊位置。

下面的 3 个程序清单演示了<script>标记可以放在 HTML 文档中的什么位置。这些程序清单只显示了 HTML 文档的框架。本章后面将说明看到为什么需要根据编写脚本的需求，将脚本放在页面的不同位置。每个例子都显示了两个 script 标记，一个用于链接外部脚本，另一个用于内部嵌入的脚本，这只是为了说明如何编写它们。实际上，应总是只链接外部脚本。

程序清单 7-1 指出，<script>标记对通常位于文档的<head>标记段中。放在 head 中的标记一般会影响页面中的非内容设置——所谓的 HTML 指令元素，例如<meta>标记和文档题目，这里也适合放置响应页面载入或用户动作的脚本。

程序清单 7-1　head 段中的脚本

```
<html>
   <head>
      <title>A Document</title>
      <script type="text/javascript" src="example.js"></script>
      <script type="text/javascript">
         //script statement(s) here
         ...
      </script>
   </head>
   <body>
   </body>
</html>
```

假如希望在页面载入时运行脚本，以便生成页面的内容，脚本就应放在文档的<body>部分，如程序清单 7-2 所示。

程序清单 7-2　body 段中的脚本

```
<html>
   <head>
      <title>A Document</title>
   </head>
   <body>
      <script type="text/javascript" src="example.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <script type="text/javascript">
            //script statement(s) here
            ...
        </script>
    </body>
</html>
```

在一个文档中，<script>标记对的数量不限。比如，程序清单 7-3 的 head 和 body 部分都有脚本。这个文档需要 body 段的脚本，也许是要在页面载入时创建一些动态内容。该文档也包括一个按钮，这个按钮需要稍后运行的、存储在 head 段中非脚本。

程序清单 7-3　head 和 body 段中的脚本

```
<html>
    <head>
        <title>A Document</title>
        <script type="text/javascript" src="example.js"></script>
        <script type="text/javascript">
            //script statement(s) here
            ...
        </script>
    </head>
    <body>
        <script type="text/javascript" src="example.js"></script>
        <script type="text/javascript">
            //script statement(s) here
            ...
        </script>
    </body>
</html>
```

7.1.2　非 JavaScript 的浏览器和 XHTML

只有包含 JavaScript 的浏览器才知道把<script>...</script>标记对中的代码行解释为脚本语句，而不是在浏览器中显示的 HTML 文本。这意味着不支持 JavaScript 的浏览器或手机中的简化浏览器不仅会忽略<script>标记，而且会将 JavaScript 代码处理为页面内容。页面上的结果就会变得一团糟。

另一方面，非 JavaScript 浏览器不会执行外部链接的脚本。这是链接外部脚本优于内嵌脚本的一个方面。只有把 JavaScript 代码嵌入 HTML 文档，才会出问题。

把脚本行放在 HTML 注释符号中，如程序清单 7-4 所示，老式的非 JavaScript 浏览器就不会显示脚本行。大多数不支持脚本的浏览器会忽略<!--和-->注释标记之间的内容，而脚本浏览器忽略<script>标记对里的注释符号。

程序清单 7-4　对大多数旧浏览器隐藏脚本

```
<script type="text/javascript">
<!--
    //script statement(s) here
```


更多资源请访问稀酷客(www.ckook.com)

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

下面解释一下尾标记</script>之前的较为奇异的结构。两个斜杠是 JavaScript 的注释符，这个符号是必需的，否则 JavaScript 就会试图解释 HTML 尾注释符号(-->)。因而，斜杠告诉 JavaScript 跳过这一行；而非脚本浏览器只把斜杠字符作为应忽略的整个 HTML 注释的一部分。

虽然不再需要采用这种方式对老式浏览器隐藏 JavaScript，但有时需要另一种隐藏技术。XML 常用于给浏览器提供内容，这个工作也常使用 XHTML 来完成。在 XML 中，所有特殊字符都必须包含在 CDATA 段中，否则文件的解析就可能不正确；至少文件的验证会失败。其解决方法仍是把脚本封装起来，但这次应如程序清单 7-5 所示。注意尾标记</script>前的注释又一次对 JavaScript 隐藏了 CDATA 段的尾标记。

程序清单 7-5　对 XML 解析器隐藏脚本

```
<script type="text/javascript">
<![CDATA[
   //script statement(s) here
   ...
//]]>
</script>
```

尽管这些技术常常称为脚本隐藏，但它们并没有对人们隐藏脚本。所有客户端 JavaScript 脚本是 HTML 文档的一部分，和其他构成页面的内容一起下载到浏览器上。查看 JavaScript 源代码像查看 HTML 文档源代码一样容易。不要误以为可以把脚本完全隐藏起来。一些开发人员删除了脚本中的所有回车换行符，并使用无意义的变量名和函数名，来混淆其脚本，但只能减慢(但不能阻止)好奇的访问者阅读和理解代码。

既然不可能对公众真正隐藏 JavaScript 编程代码，那就炫耀它吧：给自己的得意之作加上签名，在脚本注释中包含版权或作者的“知识共享”声明，并鼓励喜欢该脚本的人与自己探讨更深入的内容。

7.2 JavaScript 语句

外部链接文件中的每行代码或<script>...</script>标记对之间的每行代码都是 JavaScript 语句(HTML 注释标记除外)。为了迁就有经验的程序员的习惯，JavaScript 允许在每条语句后加一个分号(相当于句子尾部的句号)。这个分号是可选的，但强烈建议总是使用它，以避免歧义。JavaScript 看到语句尾部的回车，就知道语句结束了。也许在将来分号是必须的，所以最好现在就养成使用分号的习惯。

脚本中的语句必然用于达到某种目的，因此，每个语句都会执行某个与脚本相关的操作。语句能完成的操作有：

- 定义或初始化变量
- 给属性或变量赋值

更多资源请访问稀酷客(www.ckook.com)

- 改变属性或变量的值
- 定义或调用对象的方法
- 定义或调用函数例程
- 作决策

学习完本章的内容后，就明白这些是什么意思了。注意，每个语句都对脚本有影响，唯一不执行任何动作的语句是注释。在脚本中包含单行注释的最常用方法是使用一对斜杠(中间没有空格)：

```
// this is a one-line comment

var a = b;    // this comment shares a line with an active statement
```

多行注释可以包含在斜杠-星号中：

```
/*
   Any number of lines are comments if they are
   bracketed by slash-asterisks
*/
```

在脚本中加入注释对自己和其他读者很有用。注释常以清晰明了的语言解释语句的功能，包含注释的目的是使用户在 6 个月之后还能想起脚本的工作原理，或者帮助另一个需要阅读这些代码的开发人员理解其作用。

7.3　脚本语句的执行时间

理解了脚本放在文档中的何处后，就要了解它们何时运行。根据脚本要完成的功能，脚本运行的时间有以下 4 种选择：

- 文档载入时
- 文档载入后
- 响应用户动作时
- 其他脚本语句调用时

决定性因素是脚本语句在文档中的位置。

7.3.1　文档载入时即刻执行

程序清单 7-5 是第 5 章中第一个脚本的变体。在此版本中，脚本在页面载入时，将当前日期和时间写到页面上。document.write()方法是在页面载入过程中，使动态内容呈现在页面上的常见方式。这类在页面载入时运行的语句称为立即语句(immediate statements)。使用 document.write()编写脚本时必须谨慎从事。一旦页面载入完毕，后续的 document.write()语句就会创建一个新页面，改写以前仔细编写的所有内容，如程序清单 7-6 所示。

程序清单 7-6　包含立即执行的脚本语句的 HTML 页面

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date &amp; Time: Immediate Execution</title>
   </head>
   <body>
      <h1>Date &amp; Time: Immediate Execution</h1>
      <p>It is currently <span id="output">
      <script type="text/javascript">
         <!-- hide from old browsers
         var oNow = new Date();
         document.write(oNow.toLocaleString());
         // end script hiding -->
      </script>
      </span>.</p>
   </body>
</html>
```

近年来，document.write()的使用引来一些争议。至少，它会给文档引入糟糕的结构，把脚本与 HTML 混合在一起。良好的结构应将样式(样式表)和行为(JavaScript)与 HTML 标记清晰地分开。document.write()带来的一个问题是，使用更多的 XML 给浏览器提供内容。XML 文档必须是格式良好的。如果 document.write()语句结束了一个在其外部开始的元素，或者开始了一个新元素，将无法载入 XML 文档，因为浏览器会发现其格式有误。

document.write()带来的另一个问题是以 DOM 为中心，尤其是附有 XHTML 文档(它们会用作 XML)。使用 document.write()意味着，内容没有包含在 DOM 中。最重要的缺陷是，不能继续用 JavaScript 程序处理这些内容，限制了对访问者在网页上执行的动作的动态响应。这是一个有趣的难题，因为在载入页面时，我们可能常常根据浏览器环境，选用 document.write()来提供动态内容，如本书的几个例子所示。

7.3.2　延时脚本

执行脚本的另外三种方式统称为延时脚本。为了演示这些延时脚本，首先需要简单介绍一下第 9 章深入讨论的一个概念：函数。函数定义了一个脚本语句块，它们在载入浏览器之后的某个时刻执行。显然，函数在<script>标记内是可见的，因为每个函数的定义都以 function 后跟函数名(和括号)开头。将函数载入浏览器后(一般在 head 段中，以便能早些载入)，只要调用它就可运行。

1. 页面载入后运行

函数在页面载入后立即运行。如果尝试在页面元素出现在 DOM 上之前操作它们，使用业界标准 DOM 方法的脚本就会失败，所以要求浏览器仅在页面完成载入后运行脚本。多数事件处理程序由用户的动作(比如单击一个按钮)触发，而 window 的事件处理程序属性 onload 在页面的所有组成部分(包括图片、Java Applet 和嵌入的多媒体)载入浏览器后触发。

连接 onload 事件处理程序和函数有两种跨浏览器的方式：使用对象事件属性，或者使用 HTML 事件特性。对象事件属性如程序清单 7-6 所示。事件处理程序的指定以即时模式进行，

更多资源请访问稀酷客(www.ckook.com)

它把需要的函数调用连接到 window 对象的 load 事件上。在页面显示过程的这个早期阶段，只有 window 对象存在于 DOM 中。

在老式的 Web 开发过程中，window.onload 事件的指定在 HTML 中完成，它利用<body>元素来表示窗口。因此可在<body>标记中包含 onload 事件特性，如程序清单 7-7 所示。在第 6 章中，事件处理程序能直接运行脚本语句。但假如事件处理程序必须运行几个脚本语句，通常把这些语句放在一个函数定义中，然后让事件处理程序调用这个函数。如程序清单 7-7 所示，页面载入完毕后，onload 事件处理程序就会触发 done()函数，该函数(此例中已简化了)显示一个警告对话框。

程序清单 7-7　在 onload 事件处理程序中运行脚本

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>An old-school HTML-based onload script</title>
      <script type="text/javascript">
         function done()
         {
            alert("The page has finished loading.");
         }
      </script>
   </head>
   <body onload="done()">
      <h1>An old-school HTML-based onload script</h1>
      <p>Here is some body text.</p>
   </body>
</html>
```

现在不要理会程序清单 7-7 的大括号和其他怪异之处，而应注意文档的结构和流程。在载入页面的整个过程中，没有运行任何脚本语句，页面只是将 done ()函数载入内存，以便随时运行它。载入文档后，浏览器触发 onload 事件处理程序，它会运行 done()函数，然后用户就会看到警告对话框。

HTML 事件特性方法可以追溯到最早的 JavaScript 浏览器，而现在，一般是将 HTML 标记与样式(样式表)和行为(脚本)分开。脚本开发人员现在使用的是功能相当的对象事件处理属性。为了在<body>标记外部使用 onload 特性，可将所需的 JavaScript 函数作为属性赋给对象的事件，如：

```
window.onload = done;
```

这类语句通常放在文档 head 部分的脚本结尾处。还要注意，此语句的右侧只有函数名，没有引号或括号。把事件处理程序指定为 HTML 特性，将更便于理解，因此本教程中的大多数示例都采用该方法。不过这需要指定属性版本，因为用户将看到使用该格式的大量实际代码。

2. 由用户运行

为响应用户动作而执行脚本，非常类似于前面在文档载入后马上运行的延时脚本例子。脚

更多资源请访问稀酷客(www.ckook.com)

本函数一般在 head 段中定义，由表单元素中的事件处理程序在函数运行时调用。程序清单 7-8 中的脚本在用户单击按钮时运行。

程序清单 7-8 通过用户操作来运行脚本

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>An onclick script</title>
      <script type="text/javascript">
         function alertUser()
         {
            alert("Ouch!");
         }
      </script>
   </head>
   <body>
      Here is some body text.
      <form>
         <input type="text" name="entry">
         <input type="button" name="oneButton"
                value="Press Me!" onclick="alertUser()">
      </form>
   </body>
</html>
```

如程序清单 7-8 所示，不是每个对象都需要定义事件处理程序，只有需要脚本的对象才定义事件处理程序。只有用户单击按钮后，才会执行程序清单 7-8 中的脚本语句，alertUser()函数在页面载入时定义，只要页面没有完全载入浏览器，它就会等待。即使它从来没有执行过，也没有问题。

3. 由另一个函数调用

执行脚本语句的最后一种情形也与函数有关。在这种情形中，函数由另一个脚本语句调用。在学习函数之前，应阅读第 8 章。因此，这个例子在下一章介绍。

7.4 查找脚本错误

在早期浏览器的 JavaScript 中，脚本错误在明确打开的对话框中显示出来。这些对话框肯定对脚本调试人员很有用。然而，假如错误警告对话框呈现在非技术用户面前，不仅会使程序停止，而且会引起惊慌。为了防止这些对话框惊扰一般用户，浏览器厂商尽力在浏览器窗口中减少错误的视觉影响。不过脚本开发人员常常容易忽略脚本中的错误，因为错误不是那么明显。

不仅每个浏览器显示错误消息的方式都不同，而且各个版本的显示也不尽相同。在每种主流浏览器中显示错误的方法参见配书光盘上第 48 章的“Debugging Scripts”一节。在测试代码之前

更多资源请访问稀酷客(www.ckook.com)

需要阅读这节内容，这里则包含一个显示在 IE5+中的错误对话框(如图 7-1 所示)和显示在 Mozilla 1.4+中的错误对话框(如图 7-2 所示)。注意在这些浏览器和其他浏览器中，这些脚本错误对话框在默认情况下未必会显示出来，所以必须在载入页面时和运行每个脚本后监控状态栏。

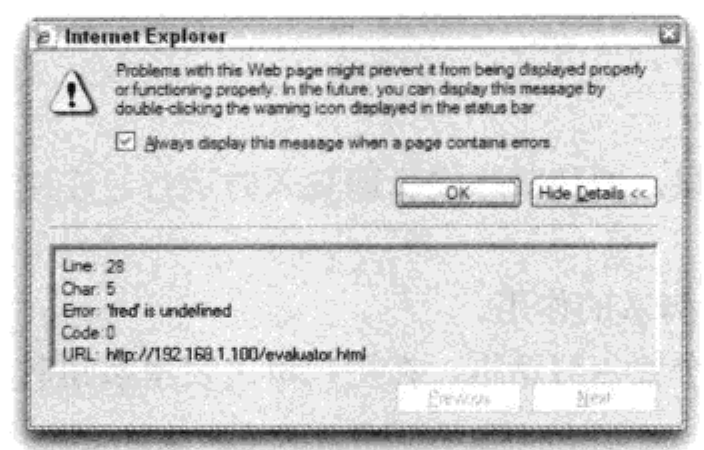


图 7-1　展开的 IE 错误对话框

图 7-2　Mozilla 1.4 JavaScript 控制台窗口

理解错误消息和处理它们是一个很宽泛的主题，第 48 章有详细说明。在本章中，可以使用错误消息来检查本书程序清单中的脚本是否有输入错误。

7.5　脚本和编程

“编写脚本”比“编程”听起来更容易、更友好。在许多方面确实如此。这里有一个类比：一些航模爱好者从头开始制作飞机模型，另一些人则从商业零件开始。“从头开始”的爱好者按照很详细的计划切割木头和金属，再拼接成飞机模型。而从商业零件开始的人先购买预制的零件，然后拼接为成品。做完之后，根本看不出哪个飞机模型是从头开始制作的，哪个是从一盒子零件拼接而来。最后，每个制造者在制作飞机模型的过程中都运用了许多相同的技术，而且都对自己的成果很满意。

由于实现了文档对象模型(DOM)，浏览器就能给脚本开发人员提供很多预制部件，以便进行处理。而如果没有浏览器，就必须由一位优秀的程序员从头开始开发应用程序，提供内容以及用户交互。最后，每个开发人员的应用程序看起来都很专业。

然而，除了文档对象模型之外，实际编程已逐步渗入脚本编程，因为脚本(和程序)处理的不仅仅是对象。本章前面提到，JavaScript 脚本的每个语句都执行某个操作，而这个“操作”涉及某种类型的数据。数据是与对象有关的信息，或者是由脚本在每个语句之间传递的另一些信息。

数据有很多形式。在 JavaScript 中，数据的常见形式有数字、文本(称为字符串)、对象(派生于对象模型，或使用脚本创建的对象)以及 true 和 false(称作 Boolean 值)。

每种编程或脚本语言都给每类数据提供了大量的结构和限制。在 JavaScript 中，处理数据所需的知识比 Java、C++等语言少得多。而在 JavaScript 中学到的数据知识可以马上应用于以后其他编程语言的学习，因此学习脚本编程付出的努力不会白费。

脚本说到底还是编程，所以需要对基础编程概念有基本的了解，才能成为优秀的 JavaScript 编程员。下面的两章先不考虑 DOM，而主要讨论在 JavaScript 和未来编程工作中大有帮助的编程原则。

更多资源请访问稀酷客(www.ckook.com)

7.6 习题

1. 为下面的脚本语句编写完整的脚本标记对。

```
document.write("Hello, world.");
```

2. 建立一个 HTML 文档，使其包括上一题的结果，并使脚本在页面载入时运行。在浏览器中打开文档，测试其结果。
3. 在上题的结果中给脚本添加一句注释，来说明脚本的作用。
4. 创建一个 HTML 文档，在页面载入后显示一个警告对话框，在用户单击表单按钮时显示另一个警告对话框。
5. 仔细研究程序清单 7-9 中的文档，不要输入和载入文档，请预测：
   a. 页面没有添加其他样式的样子；
   b. 用户和页面如何交互；
   c. 脚本的作用。

然后将程序清单输入文本编辑器中，注意所有的大写字母和标点符号。在 upperMe 函数的=符号后不要回车，让语句自然换行，如程序清单 7-9 所示。在特性名/值对中可以使用回车，如第一个<input>标记所示。将文档保存为 HTML 文件，并将文件载入浏览器中，查看其运行效果。

程序清单 7-9 这个页面的工作方式

```
<!DOCTYPE html>
<html>
   <head>
   <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>Text Object Value</title>
      <script type="text/javascript">
         function upperMe()
         {
            document.getElementById("output").value =
            document.getElementById("input").value.toUpperCase();
         }
      </script>
   </head>

   <body>
      Enter lowercase letters for conversion to uppercase:<br>
      <form name="converter">
         <input type="text" name="input" id="input"
                value="sample" onchange="upperMe()" /><br />
         <input type="text" name="output" id="output" value="" />
      </form>
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

第 8 章

程序设计基础(一)

本章先暂时停止讨论 HTML 和文档，而开始学习编程基础知识，几乎每种脚本和编程语言都使用些基础知识。本章从变量、表达式、数据类型和操作符开始，没有编写过程序的读者会觉得，这些听起来令人畏惧。但只要有了一点编程经验，就会很快熟悉这些术语和概念。

本章包含哪些内容？
变量及其用法
学习计算表达式
转换数据类型
使用基本操作符

8.1 JavaScript 语言

本书学习的语言是 JavaScript，它也有其他的名字。Microsoft 把该语言命名为 JScript，省去了 ava，Microsoft 就不必从 Java 商标所有者 Sun Microsystems 处得到 Java 名称的许可了。

现在，ECMA(读音是 ECK-ma)标准协会管理这种语言的规范，文档 ECMA-262 提供了此语言的所有细节，是 ECMA 发布的第 262 个标准，它略晚于最早期的浏览器。JavaScript 和 JScript 都与 ECMA-262 兼容，一些早期的浏览器版本和 ECMA-262 稍有不同，其中最大的差异会在本书第III部分予以介绍。

8.2 处理信息

一般情况下，每个 JavaScript 语句都要处理信息——数据。数据可以是通过 JavaScript 语句显示在屏幕上的文本信息，也可以是表单中单选按钮的开/关设置。编程时的每条信息也称为值，在编程领域之外，值常常有许多含义。然而在编程领域，该术语的含义是受限制的。字符串是值，数字是值，复选框的设置(不管是开还是关)也是值。

在 JavaScript 中，值的类型有几种。表 8-1 列出了 JavaScript 的正式数据类型，以及取值的例子和说明。

更多资源请访问稀酷客(www.ckook.com)

表 8-1　JavaScript 值(数据)类型

类　型	示　例	说　明
String	"Howdy"	引号内的一系列字符
Number	4.5	不在引号内的数字
Boolean	true	逻辑真或假
Null	null	不包含任何内容，但仍是一个值
Object		通过属性和方法定义的软件对象(数组也是对象)
Function		函数定义

语言包含这几种数据类型就简化了编程任务，若编程任务涉及与其他语言不兼容的数字类型(整数、实数或浮点数)，简化就更明显。在本书后面的一些语法定义和对象部分特别说明了支持占位符的值类型。在需要字符串时，可将文本放在引号内。

然而有时这些值类型会阻碍脚本的顺利执行。比如，假如用户在表单的文本输入域中输入数字，浏览器就把该数字存储为字符串值类型。假如脚本要对那个数字执行一些算术操作，就必须将字符串转换为数字，再用该值进行数学运算。本章的后面将列举相关示例。

8.3　变量

与在程序里处理数据相比，在厨房里按照食谱烹调食物的一个优点是，在厨房里，可以遵照食谱的步骤处理具体的食材：胡萝卜、牛奶或大马哈鱼片，而计算机要按照一列指令来处理数据。尽管数据表示实际的东西，诸如在表单的输入域中输入的文字，但这个值进入程序后，用户就不再与之接触了。

实际上，程序只是处理计算机内存中的一些位(开和关状态)。更确切地讲，在浏览器软件独占的计算机内存中，增强 JavaScript 网页中的数据仅占据了部分内存。过去，程序员必须知道某个值在内存(RAM)中的地址才能提取它的副本来进行(例如)加法计算。虽然程序的内部有些复杂，但像 JavaScript 这样的编程语言隐藏了这些复杂性。

在脚本中处理数据时，最简便的方法是先把数据分配给变量。变量可以理解为存放信息的筐，变量存放信息的时间取决于诸多因素，但是，一旦 Web 页面清除了窗口(或框架)，就会删除窗口内的所有变量。

8.3.1　创建变量

在 JavaScript 中创建变量有两种方法，其中一种方法适合所有情形：使用 var 关键字，并在它后面加上变量名。声明新变量 myAge 的 JavaScript 语句为：

```
var myAge;
```

这个语句告诉浏览器，可在后面使用这个变量来存放信息或者修改这个变量内的数据。

要给变量赋值，可以使用赋值操作符，目前最常用的赋值操作符是等号。例如，如果在声明 myAge 变量的同时给它赋值(这种组合方法称为变量初始化)，可在包含关键字 var 的语句中

更多资源请访问稀酷客(www.ckook.com)

使用赋值操作符：

```
var myAge = 45;
```

假如在一条语句中声明变量，在下一条语句中给它赋值，语句序列就是：

```
var myAge;
myAge = 45;
```

var 关键字用于声明或初始化变量——文档中的任何变量都只需要使用一次 var 关键字。

JavaScript 变量可存放任何类型的值。在声明变量时，不必定义变量包含的值的类型，这与其他许多语言都不同。实际上，变量值的类型在程序的执行过程中可能会变化(这种灵活性会使有经验的程序员发狂，因为他们习惯于同时给变量指定数据类型和值)。

8.3.2　变量的命名

给变量选择名称时要十分小心，一些脚本可能使用模糊的变量名，比如使用单个字母作为变量名。除了几个特殊情况常常使用字母作为变量名之外(比如，第 9 章使用 i 作为循环中的计数变量)，应使用能描述变量内容的名字，这有助于在较长的语句系列或跳转处(特别是在复杂的脚本中)跟踪数据的状态。

变量命名的一些限制有助于养成好的编程习惯。首先，不能使用任何保留的关键字作为变量名，这包括 JavaScript 当前使用的所有关键字和留给未来版本的关键字。然而 JavaScript 的设计者并不能预见该语言将来使用的所有关键字，如果使用在保留关键字的当前列表中的单词(见配书光盘上的附录 C)，就可能与将来的版本冲突。

更复杂的情形是，变量名不能包含空格字符。因此，变量名仅包含单个字是比较合适的。假如变量用多字描述更为贴切，就可以使用两个约定将多字组合成一个字。一个约定是在字之间加入下划线；另一个约定是第一个字的首字母小写，而后续每个字的首字母大写，这称为 CamelCase 或 interCap 形式。下面的两个例子都是有效的变量名：

```
my_age
myAge
```

第二种形式更好，它更便于输入和阅读。实际上，单字变量名与将来版本的单字关键字存在潜在的冲突，最好使用多字组合的变量名。保留字列表一般很少有多字组合的关键字。

变量名还有两个重要的限制：除了下划线之外不能使用任何标点符号，变量的首字符不能是数字。这些限制类似于第 6 章中 HTML 元素标识符的限制。

8.4　表达式和求值

与值以及变量密切相关的另一个概念是表达式求值，它是学习编程的最重要的概念。

在日常语言中我们会用到表达式，例如，The Beverly Hillbillies 的主题歌是

Then one day he was shootin" at some food
And up through the ground came a-bubblin' crude

更多资源请访问稀酷客(www.ckook.com)

Oil, that is. Black gold. Texas tea.

歌曲的结尾处有 4 个不同的引用(crude、oil、black gold 以及 Texas tea)，它们都表示石油，都是石油的表达式。只要说出其中任何一个，其他人就会明白它的意思。其实，这些表达式只有一个意思：石油。

在编程中，变量总是表示它的内容或值，比如给变量赋值：

```
var myAge = 45;
```

只要在语句中使用变量，它的值(45)就自动应用在该语句调用的操作中。例如，假如甲比乙小 15 岁，乙就可以根据自己的年龄，也就是 myAge 的值，给表示甲年龄的变量赋值：

```
var yourAge = myAge - 15;
```

脚本下次使用变量 yourAge 时，它的值为 30。假如脚本在后面修改了 myAge 的值，这个变化与 yourAge 变量没有关系，因为给 yourAge 变量赋值时，myAge 的值为 45。

8.4.1 脚本中的表达式

在前面章节列举的几个脚本例子中，提到了表达式的计算非常方便，但那时读者也许还不知道表达式。查看程序清单 5-6，这个脚本会在加载页面时将动态文本显示在页面上。第二个 document.write()语句如下：

```
document.write(" of " + navigator.appName + ".");
```

测试 JavaScript 的表达式求值

如图 8-1 所示，可以利用 The Evaluator Jr.来测试 JavaScript 表达式的求值方法，这个 HTML 页面在配书光盘中(第 4 章介绍了其 Senior 版本)。在顶端的文本框中输入 JavaScript 表达式，然后按下 Enter/Return 或单击 Evaluate 按钮进行测试。

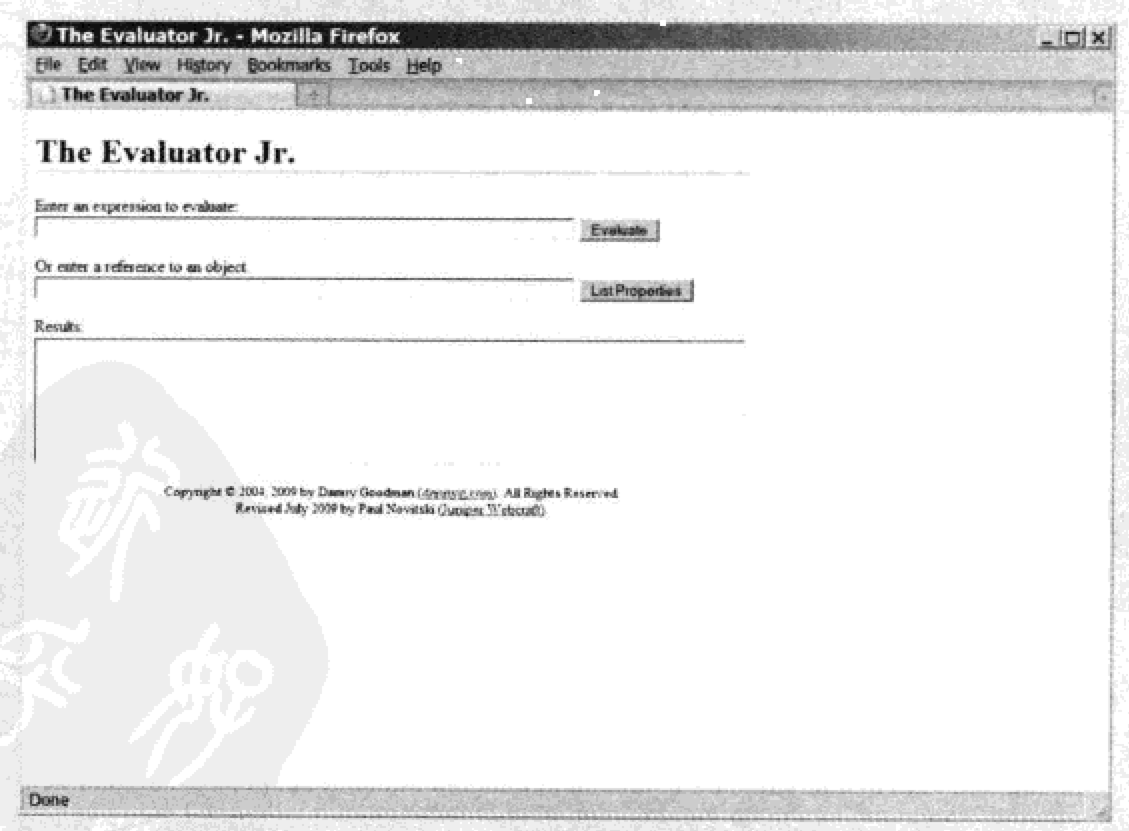


图 8-1 The Evaluator Jr.测试表达式的求值

更多资源请访问稀酷客(www.ckook.com)

The Evaluator Jr.预定义了 26 个变量(其变量名是小写字母 a～z)，因此，可给这些变量赋值、测试比较操作符甚至执行数学运算。使用本章前面的 age 变量例子，在顶部的文本框中输入下列语句，并在 Results 域中观察每个表达式的结果。在输入时必须注意大小写。在 The Evaluator 中，尾部的分号是可选的。

```
a = 45;
a;
b = a - 15;
b;
a - b;
a > b;
```

如果要重新开始，可以单击 Reload/Refresh 按钮。

document.write()方法(注意，JavaScript 使用“方法”来表示“命令”)需要一个放在括号中的参数：在 Web 页面中显示的文本串。这里的参数是把三个不同字符串连接起来的表达式：

```
" of "
navigator.appName
"."
```

加号是 JavaScript 连接字符串的一种方式。JavaScript 必须执行一些快速计算，才会显示该行。第一个计算是求 navigator.appName 属性的值，得到浏览器名称的字符串。之后，JavaScript 在最后的计算中连接三个字符串，所得的字符串最后显示在 Web 页面上。

8.4.2　表达式和变量

表达式的求值非常灵活，下面将展示 document.write()语句的另一个求值路径。这个例子不把字符串作为 document.write()方法的直接参数，而是先在变量中把字符串组合在一起，然后把该变量应用在 document.write()方法中。下面就是该例子的代码，其中还声明了一个新变量，并给它赋值：

```
var textToWrite = " of " + navigator.appName + ".";
document.write(textToWrite);
```

因为变量 textToWrite 计算为组合在一起的字符串，所以这个方法是有效的。document.write()方法接受该字符串值，并执行显示操作。阅读脚本或试图改正错误时，要特别注意每个表达式(变量、语句、对象属性)的计算方式。当脚本需要某种类型的值时，如果不明白表达式是如何求值的，那么在学习 JavaScript 或其他语言的过程中，就会常常感到迷惑不解。

8.5　数据类型转换

前面提到过，假如在运算过程中，某个组成部分的数据类型不正确，表达式就会在一些脚本运算中失败。JavaScript 会尽力执行内部转换来解决这些问题，但是它不可能完全了解程序员

更多资源请访问稀酷客(www.ckook.com)

的意图。假如程序员的想法不同于 JavaScript 处理数值的方式，就得不到想要的结果。

一个例子是对采用文本字符串形式的数字执行相加操作。在一个简单的算术表达式中，对两个数字相加，可得到预期的结果：

```
3 + 3                    // result = 6
```

但假如其中一个数字是字符串，JavaScript 就会将另一个值转换为字符串——将这个加号操作从算术相加变成字符串连接。因此，表达式如下：

```
3 + "3"                  // result = "33"
```

第二个操作数的“字符串性质”决定了表达式的运算结果，第一个值自动转换为字符串，结果是这两个字符串的连接，用户可在 Evaluator Jr.中测试一下。

假如在语句中加入另一个数字，看看会有什么结果：

```
3 + 3 + "3"              // result = "63"
```

看起来完全不合乎逻辑，但结果是合理的。表达式按从左到右的顺序进行计算，第一个加号作用于两个数字，得到 6。但是 6 与“3”相加时，JavaScript 使“3”的“字符串性质”起作用，把 6 转换为一个字符串，两个字符串连接起来就得到“63”。

我们对数据类型的关注主要是类似这样的算术运算，有一些对象方法也需要一个或多个特殊数据类型的参数。JavaScript 提供了转换数据类型的多种方法，在此先介绍两种最常用的数据转换：字符串转换为数值；数值转换为字符串。

8.5.1　将字符串转换成数值

如上一节所述，假如把数字存储为字符串，例如在表单文本域中输入的数字，脚本就很难把这个数字应用到数学运算中。JavaScript 语言提供了两个内置函数 parseInt()和 parseFloat()，将表示数字的字符串表示转换为真正的数字。

在 JavaScript 中，整数和浮点数是有区别的：整数总是整的数值，没有小数点或小数点后没有数值；但浮点数在小数点的后面有数值。大体上，JavaScript 的数学运算不区分整数和浮点数：数值就是数值。只有方法的参数必须为整数时(因为它不能处理小数)，才需要考虑这种区别。比如，窗口的 scroll ()方法的参数需要滚动窗口的水平和垂直像素的整数值，因为不能在屏幕上以小数像素值滚动窗口。

为了使用这些转换函数，需要将要转换的字符串作为参数传给函数，比如，把两个不同的字符串传送给 parseInt()函数，其结果如下：

```
parseInt("42")           // result = 42
parseInt("42.33")        // result = 42
```

第二个表达式将浮点数的字符串表示传给函数，但函数返回的值还是整数。这里不进行四舍五入(但必要时，可使用其他数学函数实现这个功能)，而是舍弃小数点和它后面的所有数字。

如果可能，parseFloat()函数就返回一个整数，否则它返回浮点数，如下所示：

```
parseFloat("42")         // result = 42
```


更多资源请访问稀酷客(www.ckook.com)

```
parseFloat("42.33")       // result = 42.33
```

这两个转换函数会计算出其结果，所以，只要需要把字符串值转换为数字，就可以插入这些函数。例如，修改前面三个值中有一个是字符串的例子，表达式就可得到预期的结果：

```
3 + 3 + parseInt("3")     // result = 9
```

8.5.2 将数字转换成字符串

将数字转换为字符串的情况没有字符串转换为数字的情况多，如上一节所述，当表达式含有多个数据类型时，JavaScript 会把数字转换为字符串。即使这样，也应在代码中明确地执行数据类型转换，以避免潜在的问题。将数字转换为字符串的最简单方法是，在加法运算中利用 JavaScript 中自动转换为字符串的功能，即在数字中加入空字符串，就可以把数字转换为字符串：

```
("" + 2500)          // result = "2500"
("" + 2500).length   // result = 4
```

在第二个例子中，表达式求值功能在起作用，括号将数字强制转换为字符串。字符串是一个 JavaScript 对象，它具有一些相关的属性，其中之一是 length 属性，它返回字符串的字符个数。因此字符串“2500”的长度是 4。注意 length 值是一个数字，不是字符串。

8.6 操作符

表达式会使用许多操作符。前面将等号(=)用作赋值操作符给变量赋值。上面的字符串例子使用加号(+)连接两个字符串。操作符一般对两个值(操作符两边的值称为操作数)执行某种计算(操作)或比较，来得到第三个值。本章简要描述了两类操作符：算术和比较操作符。第 22 章将探讨更多操作符，一旦理解了本章的基础内容，其他操作符就很容易掌握。

8.6.1 算术操作符

在算术操作符中探讨文本串有点奇怪，但是前面介绍了一个或多个操作数是字符串时加号(+)操作符的特殊用法。加号告诉 JavaScript，把操作符两边的字符串连接或者结合起来。字符串连接操作符不区分字和空格，因此程序员必须保证，要连接起来的两个或多个字符串把适当的字间隔符作为字符串的一部分，有时这意味着增加一个空格：

```
firstName = "John";
lastName = "Doe";
fullName = firstName + " " + lastName;
```

JavaScript 还使用加号做算术加法。只有两个操作数都是数字，JavaScript 才把表达式处理为算术加法，而不是字符串连接。JavaScript 包含标准的算术加法、减法、乘法和除法(+、−、*、/)。

更多资源请访问稀酷客(www.ckook.com)

8.6.2　比较操作符

另一类操作符可以比较脚本中的值，例如比较两个值是否相等。这类比较操作返回 Boolean 值：true 或者 false。表 8-2 列出了比较操作符。测试两项是否相等时，应使用两个等号，这与单等号的赋值操作符不同。

表 8-2　JavaScript 比较操作符

符　　号	说　　明	符　　号	说　　明
= =	等于	>=	大于等于
! =	不等于	<	小于
>	大于	<=	小于等于

比较操作符最常用于运行时需要做出判断的脚本结构。厨子在厨房中总要做出决策：假如调味汁太淡，就加入一些面粉。第 22 章将介绍比较操作符的用法。

8.7　习题

1. 下面哪些变量的声明和初始化是有效的？为什么？假如无效，应该如何修改？

```
a. my_name = "Cind";
b. var how many = 25;
c. var zipCode = document.getElementById("zip").value
d. var 1address = document.("address1").value;
```

2. 假设下面的语句按顺序执行，每个语句都依赖前一个语句的执行结果。对下面的每个语句，写出在 JavaScript 中执行语句后表达式 someVal 的值。

```
var someVal = 2;
someVal = someVal + 2;
someVal = someVal * 10;
someVal = someVal + "20";
someVal = "Robert";
```

3. 说出将字符串转换为数字的两个 JavaScript 函数，如何把要转换为数字的字符串传给这个函数？
4. 输入并加载程序清单 8-1 中的 HTML 页面和脚本，在顶部的两个文本域中输入一个 3 位数字，并单击 Add 按钮。分析代码并找出脚本的错误。如何修改脚本，使输出域显示正确的求和结果？

程序清单 8-1　这个页面有何错误

HTML：jsb-08-01.html

```
<!DOCTYPE html>
<html>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Making Sums</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb-08-01.js"></script>
    </head>
    <body>
        <h1>Making Sums</h1>
        <form action="addit.php">
            <p><input type="text" id="inputA" name="inputA" value="0"></p>
            <p><input type="text" id="inputB" name="inputB" value="0"></p>
            <p><input type="button" id="add" name="add" value="Add"></p>
            <p><input type="text" id="output" name="output"></p>
        </form>
    </body>
</html>
```

JavaScript: jsb-08-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to critical elements
        var oInputA = document.getElementById('inputA');
        var oInputB = document.getElementById('inputB');
        var oButton = document.getElementById('add');
        var oOutput = document.getElementById('output');

            // if they all exist...
            if (oInputA && oInputB && oButton && oOutput)
            {
                // apply behaviors
                addEvent(oButton, 'click', addIt);
            }
    }
}
// add two input numbers & display result
function addIt()
{
    var value1 = document.getElementById("inputA").value;
    var value2 = document.getElementById("inputB").value;

    document.getElementById("output").value = value1 + value2;
}
```

5. 在 JavaScript 中，术语“连接”的含义是什么？

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第 9 章

程序设计基础(二)

本章包含哪些内容？

- 控制结构如何做决策
- 如何定义函数
- 在何处有效地初始化变量
- 大括号的含义
- 数据数组的基础知识

本章将继续学习编程基础中更有趣的内容。比如在程序中如何做决策，如何重复执行一些语句，以及如何使用 JavaScript 语言中最强大的信息存储器：数组。

9.1 决策和循环

每人每天都会做很多决策，但大多都没有意识到这一点。下面看看顾客从进入一家超市到结账离去这段时间所做的一些决策。

顾客一进入商店，就要做出一个决策：根据准备采购物品的数量和大小，是拿一个手提筐还是从商店门边推一个手推车？这个选择很重要，会对以后的购物产生影响，因为如果要购买一个非常重的特价物品，手提筐就无法装下。

接着顾客来到食品区，走到一个货架前，拿起货架上的食品和自己的购物单相比较。假如这个货架上有需要的物品，顾客就会转到这个货架开始寻找需要的东西，否则顾客将走过这个货架，到下一处购物。

随后顾客来到农产品区，准备买一些西红柿。在西红柿货架前，开始逐一审视，挑选一个，感觉是否结实，检查一下是否新鲜，看看有没有脏东西或虫眼等。放下这个，拿起那个，不断重复这个过程，直到找到自己感到满意的西红柿为止。最后来到收银处，店员会问“要纸袋还是塑料袋？”这是又一个要做出的决策。这个决策会影响顾客把食品从汽车拿到厨房的方式，还会影响回收习惯。

在购物过程中进行的决策和重复在 JavaScript 程序中也会遇到。假如理解了现实生活中的这些过程，就可以研究完成这些过程的 JavaScript 步骤和语法了。

9.2 控制结构

在编程术语中，决策和循环语句称为控制结构，控制结构根据简单的决策以及其他因素来

PDG

更多资源请访问稀酷客(www.ckook.com)

确定脚本语句序列的执行流程。

控制结构的一个重要部分是条件。有了某些条件(比如，天气不错，是晚上，是否去观看足球比赛)，我们就可以走不同的上班路线。同样，如果存在某个条件，程序就执行另一个操作。每个条件都是值为 true 或 false 的表达式——这是第 8 章提到的 Boolean 数据类型，这种用作条件的表达式通常包含比较操作符。在现实生活中也是这样：假如室外温度低于 0 度，出门就要穿上一件外套。在编程中，主要比较的是数值和字符串。

JavaScript 为不同的编程环境提供了几种控制结构。最常用的三个控制结构是 if 结构、if...else 结构和 for 循环。

第 21 章将详细介绍其他常用的控制结构。本章只学习上述 3 种常用结构。

9.2.1 if 结构

最简单的程序决策是：假如某个条件为真，就执行某个程序分支或路径。该结构的正式语法如下，斜体部分在实际的脚本中用相应的表达式和语句替换。

```
if (condition)
{
   statement[s] if true
}
```

不必考虑大括号，只学习基本结构即可。关键字 if 是必需的，括号内是一个运算结果为 Boolean 值(true/false)的表达式，这是程序运行到此处要检测的条件。假如条件为 true，就执行大括号内的一条语句或多条语句，然后继续执行括号后面的语句。假如条件为 false，则跳过大括号内的语句，继续执行括号后面的语句。

下面的例子假设在脚本的前面给变量 myAge 赋了值(如何赋值对本例来说不重要)，条件表达式比较 myAge 与 13 的大小：

```
if (myAge < 13)
{
   alert("You are not yet a teenager.");
}
```

在这个例子中，myAge 的数据类型只有是数字，才能得到正确的结果(通过小于比较操作符“＜”进行比较)。对于小于 13 的所有 myAge 实例，都会执行大括号内的语句，并显示警告信息。用户关闭这个警告对话框后，脚本继续执行 if 结构后的语句。

9.2.2 if ... else 结构

前述的 if 结构很简单，但其他程序决策比较复杂。有时可以根据给定的条件使程序选择两个分支之一，而不是指定一条程序运行路径。这是一个很好、很重要的区别。在一般的 if 结构里，当条件为 false 时不进行任何处理。但假如必须选择两个分支之一，就需要使用 if ... else 结构。if... else 结构的语法如下所示：

```
if (condition)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    statement[s] if true
}
else
{
    statement[s] if false
}
```

if 结构的所有条件都适用于此结构，唯一的区别是使用了 else 关键字，它提供了条件为 false 时的执行路径。

例如，下面的 if ... else 结构确定某年的二月有多少天。为了简化这个例子，条件仅仅测试该年能否被 4 整除(正确的测试应该还包括处理 100 的整数倍年，在此忽略了它)。取模操作符%(详见第 22 章)的运算结果是两个值相除的余数。余数为 0，就意味着第一个值能被第二个值整除。

```
var febDays;
var theYear = 2010;
   if (theYear % 4 == 0)
{
      febDays = 29;
}
else
{
      febDays = 28;
}
```

这个例子的要点是在 if...else 结构的结尾处，febDays 变量设置为 28 或 29，没有其他的可能值。如果年数能被 4 整除，就执行第一个分支语句，否则运行第二个语句，然后继续执行 if...else 结构后面的语句。

9.3　重复循环

现实生活中的重复循环通常意味着一系列步骤的重复，满足某个条件后跳出循环。本章前面的买西红柿就是这种情形，同样，在一条拥挤的街道上寻找一个空停车位也是如此。

脚本中的循环结构会重复执行一些语句，直到满足特定的条件为止。比如，JavaScript 数据验证例程会检查用户输入到表单文本域的每个字符，以确保每个字符都是数字。或者把一个数据集存储在一个列表中，检查输入的值是否在列表中。如果满足条件，脚本就跳出循环，继续执行循环结构后面的语句。

JavaScript 中最常用的循环结构是 for 循环，其名称来源于该循环结构开头的关键字。for 循环可以设置循环的次数，其正式语法结构如下：

```
for ([initial expression]; [condition]; [update expression])
{
    statement[s] inside loop
}
```


更多资源请访问稀酷客(www.ckook.com)

方括号的项是可选的。不过，在熟练掌握 for 循环之前，建议在构造循环时使用所有的三个可选项。initial expression 部分常用于设置计数变量的初值，condition 部分(与 if 结构中的条件一样)定义了停止循环的条件，最后，update expression 语句在每次循环结构中的所有语句都执行完毕后才执行。

循环一般先初始化计数变量 i；每迭代一次循环，就给 i 加上 1；i 超过某个最大值时，循环终止，如下所示：

```
for (var i = startValue; i <= maxValue; i++)
{
   statement[s] inside loop
}
```

占位符 startValue 和 maxValue 可以是任何数值，包括数字或包含数字的变量。在 update expression 中有一个以前尚未介绍过的操作符++，每次在循环的最后执行 update expression 时，++操作符就将 i 值加 1。假如 startValue 是 1，则第一次循环时，i 值为 1，第二次为 2，依此类推。因此，如果 maxValue 为 10，循环就重复 10 次(即只要 i 小于等于 10)。循环内的语句在执行时一般会使用计数变量的值。本章后面会说明该变量如何对循环的语句发挥作用，同时介绍如何提前终止循环，以及为什么需要在脚本中提前终止循环。

9.4　函数

第 7 章简单介绍了 JavaScript 函数。函数是一组延迟动作集的定义，由事件处理程序或者脚本中其他地方的语句调用。设计良好的函数应尽可能在其他文档中重用，它们是可重用的构造块。

假如用户有编程经验，就可以看出 JavaScript 函数和其他语言的子例程有相似之处。一些语言区分过程(执行动作)和函数(执行动作并返回值)，而 JavaScript 只有一种例程。函数能给调用它的语句返回一个值，但这不是必须的。当函数返回值时，调用语句把函数调用处理为表达式——在函数调用处插入返回值。稍后将列举一些例子。

函数的正式语法结构如下：

```
function functionName ( [parameter1]...[,parameterN] )
{
   statement[s]
}
```

函数命名的限制与 HTML 元素和变量一样，应该设计足以表示函数功能的名称，最好使用 CamelCase 或 interCap 格式(大小写相间)的多字名称，这种名称以动词开头，因为函数完成一定的动作，即使它们只是获取或设置一个值。

在开始创建函数前，应尽可能使每个函数只实现一个功能，编写长达数百行的函数是完全可能的，但这样的函数常常很难维护和调试。一个长函数可以分解为多个更简单明了的小函数。

更多资源请访问稀酷客(www.ckook.com)

9.4.1　函数的参数

第 4 章介绍了事件处理程序如何按名称来调用函数。任意一个函数调用，包括来自另一个 JavaScript 语句的函数调用，其调用形式都是一样的：函数名加括号。

函数也可以从调用语句中接收参数值。程序清单 9-1 显示了一个简单的脚本，其中一个语句在调用函数的同时把文本数据传给函数。

程序清单 9-1　从事件处理程序中调用函数

HTML: jsb-09-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Passing a Parameter to a Function</title>
      <script type="text/javascript" src="jsb-09-01.js"></script>
   </head>
   <body>
      <h1>Passing a Parameter to a Function</h1>
   </body>
</html>
```

JavaScript: jsb-09-01.js

```
// display a personalized greeting
function greeting(sName)
{
   alert("Hello, " + sName + "!");
}

greeting("Chris");
```

参数提供一种通过函数调用将值从一个语句传递给另一个语句的机制。如果函数没有参数，则在函数定义和函数调用中，函数名后跟一对空括号(如第 4 章的程序清单 4-2 所示)。

函数收到参数后，就将这个参数值赋给函数内指定的变量。分析下面的脚本段：

```
function sayHelloFirst(a, b, c)
{
   alert("Say hello, " + a);
}
sayHelloFirst("Gracie", "George", "Harry");
sayHelloFirst("Larry", "Moe", "Curly");
```

在脚本中定义函数后，下一个语句就调用该函数，并传送 3 个字符串作为参数。函数自动将这些字符串赋给变量 a、b 和 c。因此，在函数中的 alert()语句运行之前，a 的值为 Gracie，b 的值为 George，c 的值为 Harry。在 alert()语句中，只用到了 a 的值，且显示：

```
Say hello, Gracie
```


更多资源请访问稀酷客(www.ckook.com)

用户关闭第一个警告框后，就调用这个函数，但这次会把不同的值传递给函数，并赋给 a、b 和 c。对话框显示：

```
Say hello, Larry
```

与脚本中定义的其他变量不同，函数参数不使用 var 关键字来初始化，调用函数时它们会自动初始化。

9.4.2　变量的作用域

在函数内部和外部定义的变量是有区别的。函数外定义的变量称为全局变量；而在函数内部定义的变量称为局部变量。

JavaScript 的全局变量与大多数其他语言的全局变量有一个微小的差别。在 JavaScript 脚本中，全局变量的作用域是当前载入浏览器窗口或框架的文档。因此，初始化全局变量后，页面中所有的脚本语句(包括函数内的语句)就可通过变量名直接访问其值，语句可以在脚本的任何地方提取和修改全局变量。在编程术语中，此类变量拥有全局作用域，因为页面上的所有语句都能访问它。

重要的是，一旦卸载页面，在页面中定义的所有全局变量都会从内存中永久清除。假如某个值需要在页面卸载后继续使用，则必须用其他的技术来保存这个值(比如，将它存储为框架集文档中的全局变量，详见第 27 章；或将它存储在 cookie 中，详见第 29 章)。在初始化全局变量时，var 关键字通常是可选的，最好总是在初始化变量时使用它，以防 JavaScript 语言将来发生变化。

与全局变量相反，局部变量在函数内部定义。前面介绍了如何在函数中定义参数变量(初始化时不使用 var 关键字)，还可以用 var 关键字定义其他变量(定义局部变量时必须使用 var 关键字，否则它们就会被视为全局变量)。局部变量的作用域是函数内部的所有语句，其他函数或函数外部的语句不能访问局部变量。

可在文档内部重复使用局部变量名。但对多数变量而言，最好不要重复使用变量名，因为这会导致混乱和很难检测的错误。有时重用某些变量名是十分方便的，比如 for 循环计数变量。因为在 for 循环的开头，循环计数变量总会重新初始化，所以这是很安全的。然而在嵌套的 for 循环结构中，必须指定不同的循环计数变量。

为了演示全局变量和局部变量的结构和行为，并说明为什么在脚本中重用大多数变量名会引起混乱，程序清单 9-2 定义了两个全局变量和一个局部变量。这里特意采用了不当的做法：局部变量和全局变量同名。

程序清单 9-2　全局和局部变量的作用域

HTML: jsb-09-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Global and Local Variable Scope</title>
      <script type="text/javascript" src="jsb-09-02.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
    </head>
    <body>
        <h1>Global and Local Variable Scope</h1>
    </body>
</html>
```

JavaScript: jsb-09-02.js

```
var Boy = "Charlie Brown"; // global
var Dog = "Snoopy";        // global

// using improper design to demonstrate a point
function demo()
{
    // local variable with the same name as a global
    var Dog = "Gromit";

    alert(Dog + " does not belong to " + Boy + ".");
}

// use global variables
alert(Dog + " belongs to " + Boy + ".");

// use global & local variables
demo();
```

页面加载时，脚本初始化了两个全局变量 Boy 和 Dog，并在内存中定义了 demo()函数。在该函数内部，把一个局部变量初始化为与全局变量 Dog 同名。在 JavaScript 中，这样的局部变量初始化会为函数中的所有语句覆盖全局变量(但是注意，假如在局部变量的初始化中省略了 var 关键字，函数中的变量 Dog 就是全局变量，例如 Gromit)。

脚本会显示两个警告对话框。第一个对话框显示连接起来的两个全局变量：

```
Snoopy belongs to Charlie Brown.
```

脚本最后一行调用函数 demo()，该函数会显示第二个对话框，它使用局部变量 Dog 替代全局变量 Dog：

```
Gromit does not belong to Charlie Brown.
```

9.5　大括号

大括号({})在写作时很少用到，但在 JavaScript(和其他语言)中常常要用到它。大括号把一组语句括起来，以帮助阅读者理解脚本，并帮助浏览器确定哪些语句是属于一组的。大括号必须成对使用。

大括号常常在函数定义和控制结构中使用。在程序清单 9-2 的函数中，大括号把构成函数的 3 个语句(包括注释行)括起来，右大括号告诉浏览器，下一个语句不属于函数。

大括号的物理位置无关紧要(代码中的缩排方式也不重要)，下面的函数对脚本浏览器来说是相同的：

更多资源请访问稀酷客(www.ckook.com)

```
function sayHiToFirst(a, b, c)
{
   alert("Say hello, " + a);
}
function sayHiToFirst(a, b, c) {
   alert("Say hello, " + a);
}

function sayHiToFirst(a, b, c) {alert("Say hello, " + a);}
```

本书使用第一个例子的样式，因为它能使复杂的长脚本更容易阅读，特别是有多层嵌套的控制结构的脚本。但脚本样式是高度个性化的，每个程序员都有自己的脚本样式——甚至是本书的合作作者也是如此！

9.6 数组

数组是最常用的 JavaScript 数据结构之一，基本数组的结构可以看成单列的电子表格，每行包含不同的数据，且每行都编了号。每行的编号严格按照数字顺序，第一行的编号为 0(计算机常常从 0 开始计数)，这个行号称为下标。为了访问数组中的元素，需要数组名和这个元素的下标。因为下标从 0 开始，所以数组元素的个数(由数组的 length 属性确定)总是比数组的最大下标值大 1。JavaScript 允许建立含有多列的数组，这是更高级的数组概念(详见第 18 章)。本章只讨论单列基本数组。

JavaScript 数组中的数据元素可以是任何数据类型，包括对象。同一个 JavaScript 数组的不同行可以包含不同的数据类型，这与其他一些编程语言不同。

9.6.1 创建数组

数组存储在变量中，因此创建数组时，应该将数组对象赋予一个变量(数组是 JavaScript 核心语言的对象，不属于文档对象模型(DOM))。在创建数组的 JavaScript 函数 Array()前面加上专用关键字 new，可以在内存中为数组分配空间。Array()函数的一个可选参数可以在创建数组时指定数组的元素个数，JavaScript 在这方面非常宽容，允许随时改变数组的大小，所以在创建新数组时，可以省略这个参数。

为了演示数组创建过程，下面创建一个数组，它存储了 50 个美国州名和一个哥伦比亚特区(总共 51 个)。首先创建数组，把它赋予一个变量，该变量名便于我们回忆起其中的数据集合：

```
var USStates = new Array(51);
```

现在，USStates 数组在内存中相当于一个包含 51 行但没有数据的表。要填充这些行，必须给每一行指定数据。为了表示数组的每一行，需要把行的下标放在数组名后的方括号中。数组的第一行表示为：

```
USStates[0]
```

为了把第一个州名的字符串赋给这一行，可以使用简单的赋值操作符：

更多资源请访问稀酷客(www.ckook.com)

```
USStates[0] = "Alabama";
```

为了填充剩余的行，给每行使用一个语句：

```
USStates[0] = "Alabama";
USStates[1] = "Alaska";
USStates[2] = "Arizona";
USStates[3] = "Arkansas";
...
USStates[50] = "Wyoming";
```

注意，第 51 个数组元素的下标是 50，因为下标从 0 开始，而不是从 1 开始。

因此，假如要包括一张信息表，使脚本不用访问服务器即可查询信息，应该在脚本中创建数组，来包含这些数据。把脚本加载到浏览器中，并运行语句，数组就会创建好。尽管文档中有许多创建数据集的语句，但给数组下载的数据量非常小，不会显著影响页面的加载，甚至对拨号用户也是如此。第 18 章将介绍创建数组的一些语法快捷方式，以减少源代码。

9.6.2　访问数组的数据

数组下标是访问数组元素的关键，通过数组名和方括号中的下标就可得到数组中指定位置的内容。比如，建立 USStates 数组后，脚本可以用下面的语句显示含有 Alaska 名字的对话框：

```
alert("The largest state is " + USStates[1] + ".");
```

可以通过下标从数组元素中提取数据，也可以通过下标给数组中的任何元素重新赋值，来改变元素的值。

9.6.3　关联数组

现在解释一下为什么数值下标在 JavaScript 中比较好用。为便于演示，这里生成另一个与 USStates 数组关联的数组。这个新数组仍有 51 个元素，并包含与 USStates 行相对应的州的成立年份。其数组结构如下所示：

```
var stateEntered = new Array(51);
stateEntered [0] = 1819;
stateEntered [1] = 1959;
stateEntered [2] = 1912;
stateEntered [3] = 1836;
...
stateEntered [50] = 1890;
```

于是在浏览器内存中有两个表，如图 9-1 所示。可以建立更多的关联数组，比如邮政编码和首府等。要点是每个表中下标为 0 的元素都对应 USStates 数组中的第一个州 Alabama。

更多资源请访问稀酷客(www.ckook.com)

USStates		stateEntered
"Alabama"	[0]	1819
"Alaska"	[1]	1959
"Arizona"	[2]	1912
"Arkansas"	[3]	1836
⋮	⋮	⋮
"Wyoming"	[50]	1890

图 9-1　两个相关的数据表

假如一个 Web 页面包括这些数据表，并允许查询某个州加入联邦的日期，页面就需要查询 USStates 的所有项，以找到对应于用户输入的项的下标。然后，将下标用在 stateEntered 数组中，来寻找对应的年份。

在这个例子中，页面包括一个文本输入域，用户可在其中输入要查询的州名。但这种方法是很危险的，除非脚本执行一些错误检查，以防用户出错。不能假设用户总是能输入有效的州名，而应从容地处理用户出错的情形(不要假设用户在网站页面上的输入是有效的)。文本域或可单击按钮的事件处理程序会调用查询州名的函数，提取对应的年份，并在警告对话框中显示该信息。该函数如下：

```
function getStateDate()
{
   var selectedState = document.getElementById("entry").value;
   for (var i = 0; i < USStates.length; i++)
   {
       if (USStates[i] == selectedState)
       {
          break;
       }
   }

   if (i < USStates.length)
   {
      alert(selectedState + " entered the Union in " + stateEntered[i] + ".");
   }
   else
   {
      alert("Sorry, '" + selectedState + "' isn't a US state.");
   }
}
```

函数的第一个语句提取文本框的值，并赋给变量 selectedState。这主要是为了便于以后在脚本中使用简短的变量名。实际上，这个值在 for 循环中使用，这个脚本就更加高效，因为浏览器不用在每次循环时都计算文本域的长引用。

这个函数的关键在于 for 循环，这里把循环计数变量的自动递增与赋予两个数组的下标值合并在一起。循环的定义指定，计数变量 i 初始化为 0。只要 i 的值小于 USStates 数组的长度，循环就继续进行。注意，数组的长度比最后一个元素的下标值大 1。因此，当 i 等于 50 时，循环运行最后一次。此时的 i 比数组长度 51 小 1，且等于最后一个元素的下标值。循环运行一次

更多资源请访问稀酷客(www.ckook.com)

后，计数器就加 1 (i++)。

嵌套在 for 循环中的是一个 if 结构，其条件是比较数组的元素值与用户输入的值。每循环一次，这个条件都测试不同的数组行，从数组的第 0 行开始。换言之，在找到匹配的值之前，该 if 结构可能要执行若干次，但是每次 i 的值都比前一次大 1。

等于比较操作符(==)在比较字符串值时非常严格，每个字母的大小写也必须相同。在前面的例子中，用户输入的州名必须完全匹配数组 USStates 中的项，才能匹配成功。第 12 章将学习一些辅助方法，以便在比较字符串时，忽略字母的大小写。

找到一个匹配时，就执行 if 结构中的语句。break 语句用于在程序需要时终止循环过程。这个例子找到一个匹配的州名时，就应当立即结束 for 循环。当 for 循环终止时，i 计数器的值等于 USStates 数组中含有所输入州名的元素下标，再使用该下标寻找另一个数组中的对应项。尽管计数变量 i 在 for 循环中初始化，但它可用于函数中初始化后的所有语句，所以可以在最后一个语句中用它来提取 stateEntered 数组的元素，并将结果显示在一个警报对话框中。

如果所输入的州名不匹配 USStates 数组中的任何值，计数变量 i 就会递增到数组的长度 51，此时 for 循环结束，i 等于 51。这样就很容易看出是否成功找到了匹配的州名，如果没有找到匹配，就显示一个错误消息。

for 循环和数组下标的这种应用在 JavaScript 中很常见。请仔细研究这些代码，并理解其工作方式。这种遍历数组的方式不仅在上述代码创建的数组中扮演重要角色，而且在浏览器为文档对象模型创建的数组里发挥重要作用。

9.6.4 数组中的 document 对象

查看附录 A 中的 document 对象部分，会发现一些对象属性的后面带有方括号。这些方括号与前述数组下标使用的方括号是一样的。因为载入文档时，浏览器会为文档中类似的对象创建数组。比如，如果页面包括两个<form>标记对，在文档中就会出现两组<form>标记。浏览器为文档使用一个 form 对象数组，对这些表单的引用如下：

```
document.forms[0]
document.forms[1]
```

也可以使用 DOM 方法 getElementsByTagName()访问这个数组：

```
var aForms = document.getElementsByTagName('form');
```

再使用如下代码引用表单数组项：

```
aForms[0]
aForms[1]
```

document 对象的下标值是根据对象的载入顺序分配的，form 对象的顺序由文档中<form>标记的顺序确定。这种带下标的数组语法是引用表单的另一种方法。引用表单仍可以使用表单的标识符(id 特性)，但应尽可能使用对象名，因为即使改变了对象在 HTML 文档里的物理顺序，对象名的引用不用修改仍可工作。假如页面只包含一个表单，则可以交替使用这两种引用方式。下面是对表单 elements 数组的 length 属性的等价引用(elements 数组包含表单中的所有表单控件)：

更多资源请访问稀酷客(www.ckook.com)

```
document.getElementById("entryForm").elements.length
document.forms[0].elements.length
document.getElementsByTagName('form')[0].elements.length
```

本书的例子常常在简单的文档中对简单的表单使用数组类型引用。但是在商业页面中，应总是使用名称引用。

9.7 习题

1. 使用函数、事件处理程序和控制结构知识以及本章的脚本完成一个页面，这个页面中的查询表包括美国的所有州名和州加入联邦的年份。如果没有建州年份的参考书，就给每一项使用另一个年份，从 1800 年开始。在该页中，为州创建一个文本输入域和一个触发数组查询的按钮。

2. 下面的函数定义存在问题吗？假如有，如何修改？

```
function format(ohmage)
{
   var result;
   if ohmage >= 1e6
   {
      ohmage = ohmage / 1e6;
      result = ohmage + " Mohms";
   }
   else
   {
      if (ohmage >= 1e3)
         ohmage = ohmage / 1e3;
         result = ohmage + " Kohms";
      else
         result = ohmage + " ohms";
}
alert(result);
```

3. 根据在超市里买西红柿的体验，使用对象和属性语法编写一个 for 循环。

4. 修改程序清单 9-2，使其不在函数内重用 Dog 变量名。

5. 表 9-1 给出太阳系中几个行星的数据，创建一个 Web 页面，允许用户输入行星名，在用户单击按钮时，在警报对话框(作为额外奖励)或页面的其他域中显示与太阳的距离和直径。

表 9-1　太阳系中几个行星的数据

行　　星	与太阳的距离	直　　径
水星	3 600 万英里	3 100 英里
金星	6 700 万英里	7 700 英里
地球	9 300 万英里	7 920 英里
火星	14 100 百万英里	4 200 英里

更多资源请访问稀酷客(www.ckook.com)

第10章

window 和 document 对象

学习了编程基础知识之后，就很容易说明在文档中如何为对象编写脚本。从本章开始介绍文档对象模型(DOM)，并深入讨论在很多文档中使用的对象。

本章包含哪些内容？

- Window 对象的作用
- 访问 window 对象的重要属性和方法
- 文档加载后触发脚本动作
- Location 和 navigator 对象的作用
- 创建 document 对象
- 访问 document 对象的重要属性和方法

10.1 顶层对象

初学者应研究图 10-1 中顶层对象的层次结构。本章重点讨论脚本编程中常见的顶层对象：window、location、navigator 和 document。通过学习本章的内容，不仅可以掌握基本知识，还可以完成简单的任务，而且为本书第Ⅳ部分深入了解每个对象及其属性、方法和事件处理程序做准备。在此只介绍对象的基本属性、方法和事件，第Ⅲ部分的例子假定读者掌握了第Ⅱ部分介绍的编程基础知识。

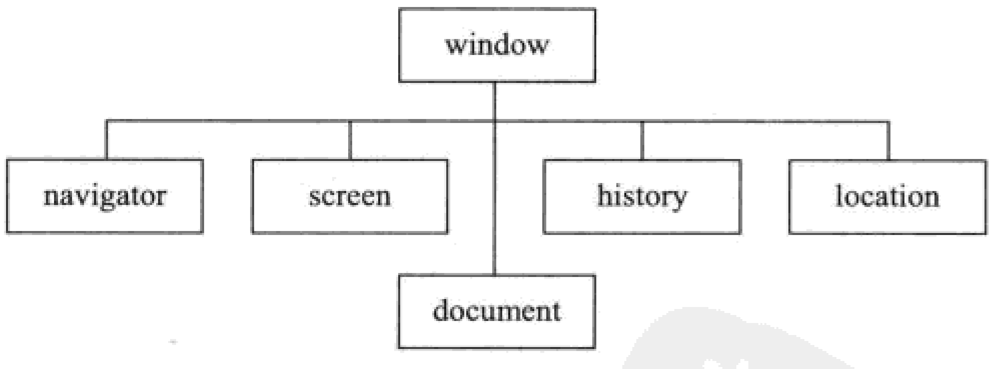


图 10-1　所有脚本浏览器的顶层浏览器对象模型

10.2 window 对象

对象层次结构的顶层是 window 对象。这个对象处于对象链的顶端，因为它是在 Web 浏览器中查看所有内容的主要容器。只要打开浏览器窗口，即便窗口中没有加载文档，window 对象也在内存的当前模型中定义好了。

更多资源请访问稀酷客(www.ckook.com)

除了文档所在窗口的内容外，窗口的影响范围还包括窗口的尺寸和包围内容区域的所有“元素”。滚动条、工具栏、状态栏和(非 Macintosh)菜单栏所在的区域叫做窗口的窗框。不是每个浏览器都可完全控制浏览器主窗口的窗框，但可以便捷地创建其他窗口，随意地设置其大小，指定要显示的窗框元素。

框架在第 13 章讨论，每个框架都可看做一个 window 对象，因为每个框架都可以存放不同的文档。脚本在其中一个文档上运行时，它把拥有这个文档的框架作为对象层次结构图中的 window 对象。

window 对象能使 DOM 方便地关联一个方法，来显示模式对话框，调整浏览器窗口底部状态栏的文本。window 对象方法允许创建显示在屏幕上的独立窗口。学习了为 window 对象定义的属性、方法和事件(详见第 27 章)后，就应清楚地了解它们和 window 对象相关联的原因：这些属性和方法用来显示 window 对象的范围以及浏览器窗口的范围。

注意：

在现代浏览器中，用户可以选择在浏览器的新选项卡中打开网页，或者在新的浏览器窗口中打开网页，新浏览器窗口是桌面上浏览器程序的独立副本。JavaScript 不对这个选择进行任何控制，而把选择权留给用户。浏览器窗口和浏览器选项卡对 JavaScript 而言是相同的 window 对象，试图控制窗口的大小和工具栏的外观，会令用户使其浏览器遵循个性化使用标准的尝试失败或受挫。也就是说，如果不希望干扰网站访问者，就不要破坏他们的设置。相信用户能管理好其浏览器。重点考虑页面的内容和样式，而不是窗口的窗框。

10.2.1　访问窗口的属性和方法

可以采用多种方法在脚本中引用 window 对象的属性和方法，这取决于设计的想法和风格，而不是具体的句法要求。编写这种引用最符合逻辑、最通用的方法是在引用中包含 window 对象：

```
window.propertyName
window.methodName([parameters])
```

在脚本引用指向包含文档的窗口时，window 对象有一个同义词 self。此时，引用语法是：

```
self.propertyName
self.methodName([parameters])
```

这些简单的引用对象名可以互换使用，但在涉及多框架和窗口的更复杂脚本中，使用 self 较合适。self 清楚地表示存放脚本文档的当前窗口，使所有用户更易理解脚本。

第 6 章提到，因为 window 对象在脚本运行时一直存在，所以对窗口内任何对象的引用都可以忽略它。因此，下面的语法假定属性和方法属于当前窗口：

```
propertyName
methodName([parameters])
```

实际上，如果忽略 window 对象的引用，有些方法就会更便于理解，这两种引用方式都能很好地运行这些方法。

PDG
更多资源请访问稀酷客(www.ckook.com)

10.2.2 创建窗口

脚本并不创建主浏览器窗口，用户在创建主窗口时，需要启动浏览器，或在浏览器菜单中打开一个 URL 或文件(如果窗口没有打开)。但是在主窗口打开后(并且该窗口包含的文档脚本需要打开子窗口)，脚本就能生成大量的子窗口。

生成新窗口的方法是 window.open()，这个方法至多包含三个参数，来定义窗口属性，比如待加载文档的 URL、HTML 标记中 target 属性引用目标的名称和外观(暂时的大小和窗框)等。这里不讨论参数(详见第 27 章)，但介绍一个涉及 window.open()方法的重要概念。

下面的语句打开一个具有指定大小的新窗口，这个窗口包含的 HTML 文档与当前页面处于同一个服务器目录下：

```
var subWindow = window.open("define.html","def","height=200,width=300");
```

这个语句的要点是，它是一个赋值语句，值赋给了变量 subWindow。这个值是什么呢？运行 window.open()方法时，不仅根据指定的参数打开了新窗口，而且计算出新窗口的引用。在编程术语中，这个方法返回一个值(这里是一个普通的对象引用)，并赋给变量 subWindow。

现在脚本可以使用这个变量作为第二个窗口的有效引用。如果需要访问它的一个属性或方法，则必须将把这个引用作为整个引用的一部分。例如，为了在主窗口中使用窗口关闭子窗口，应使用该子窗口引用的 close()方法：

```
subWindow.close();
```

如果在主窗口的脚本中使用 window.close()、self.close()或 close()方法，就会关闭主窗口(在用户确认之后)，而不是子窗口。于是，为了定位另一个窗口，就必须把那个窗口的引用作为整个引用的一部分。这会对代码产生一定影响。因为用户希望，只要主文档载入浏览器，包含子窗口引用的变量就有效。为此，这个变量必须初始化为全局变量，而不是在函数内部定义(虽然可在函数内设置它的值)。这样，可以用一个函数打开窗口，而用另一个函数关闭它。

程序清单 10-1 是一个包含两个按钮的页面，一个按钮用于打开一个空的新窗口，另一个按钮用于从主窗口中关闭该窗口。为了观看这个例子的效果，最好将主浏览器窗口缩小。这样，生成新窗口后，如果主窗口在前面，应重定位主窗口，使较小的新窗口可见(如果一个窗口完全隐藏在另一个窗口后面，可以使用浏览器的 Window 菜单来选择隐藏的窗口)。程序清单 10-1 的要点是，newWindow 变量定义为全局变量，因此 makeNewWindow ()和 closeNewWindow()函数都可以访问它。如果没有变量声明时赋值，则其初始值为 null。在条件中，null 值等同于 false，而任何非 0 值等同于 true。因此，在 closeNewWindow()函数中，调用子窗口的 close ()方法前，需要测试是否创建了该窗口，然后函数将 newWindow 变量设置为 null，以清空窗口，这样再次单击 Close 按钮时，就不会关闭不存在的窗口。

注意：

本章和本书其他许多地方使用的一个属性赋值事件处理技术是 addEvent()，这是一个跨浏览器的事件处理程序，详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，该文件位于配书光盘中，这个文件夹可以在所有章节的脚本中访问。

更多资源请访问稀酷客(www.ckook.com)

程序清单 10-1　window 对象的引用

HTML: jsb-10-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Window Open &amp; Close</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-10-01.js"></script>
   </head>
   <body>
      <h1>Window Open &amp; Close</h1>
      <form>
         <p>
            <input type="button" id="create-window" value="Create New Window">
            <input type="button" id="close-window" value="Close New Window">
         </p>
      </form>
      </body>
</html>
```

JavaScript: jsb-10-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to critical elements
      var oButtonCreate = document.getElementById('create-window');
      var oButtonClose = document.getElementById('close-window');

         // if they all exist...
         if (oButtonCreate && oButtonClose)
         {
            // apply behaviors
            oButtonCreate.onclick = makeNewWindow;
            oButtonClose.onclick = closeNewWindow;
         }
   }
}

var newWindow;

function makeNewWindow()
{
   newWindow = window.open("","","height=300,width=300");
}
```


更多资源请访问稀酷客(www.ckook.com)

```
function closeNewWindow()
{
    if (newWindow)
    {
        newWindow.close();
        newWindow = null;
    }
}
```

10.3　window 对象的属性和方法

本节讨论 window 对象的三个方法，它们会显示各种类型的对话框，会直接影响用户交互。它们可用于所有脚本浏览器。第Ⅳ部分包含每个属性和方法的详细代码示例，在 The Evaluator Jr.(详见第 8 章)顶部的文本框中输入单语句脚本，就可以测试它们。

脚本编写新手遇到的第一个问题是如何自定义这些对话框的标题栏、大小和按钮标签。每个浏览器厂商都会说明如何给对话框加标签。因为多年来，总是有人试图将这些对话框用于达到不当的目的，浏览器厂商现在做了大量的工作，告诉用户对话框是如何从网页脚本中生成的。脚本编写者不能修改这些对话框的用户界面。

10.3.1　window.alert()方法

本书多次使用了 alert()方法。这个 window 方法会生成一个对话框，显示作为参数传送的文本(如图 10-2 所示)，OK 按钮(其标签不能改变)允许用户关闭这个警告对话框。

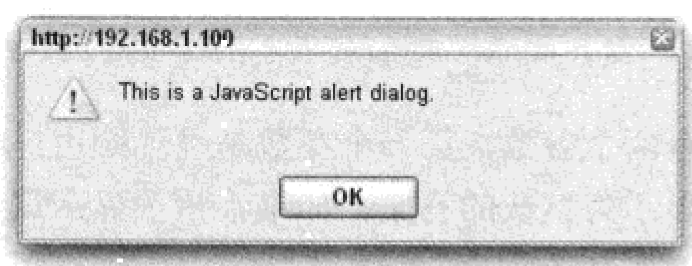


图 10-2　JavaScript 警告对话框

这三个生成对话框的 window 方法都没有引用 window 对象。从技术角度讲，alert()是 window 对象的一个方法，但在对话框和创建它的窗口之间并不存在特殊关系。在生产脚本中常使用快捷引用：

```
alert("This is a JavaScript alert dialog.");
```

10.3.2　window.confirm()方法

第二种风格的对话框有两个按钮(大多数平台上的大多数版本是 Cancel 和 OK)，称为确认对话框(见图 10-3)。更重要的是，这个方法有返回值：单击 OK 按钮的返回值为 true，单击 Cancel 按钮的返回值为 false，用这个对话框和它的返回值可以使用户决定脚本如何继续进行。

这个方法常常返回 Boolean 值，所以可以将该方法的计算值作为 if 或 if ... else 结构中的条件语句。例如，下面的代码段询问用户是否启动应用程序，如果是，则将 index.html 页面载入

更多资源请访问稀酷客(www.ckook.com)

浏览器。

```
if (confirm("Are you sure you want to start over?"))
{
    location.href = "index.html";
}
```

图 10-3　JavaScript 确认对话框 (IE7/WinXP 风格)

10.3.3　window.prompt()方法

window 对象的最后一种对话框是提示对话框(见图 10-4)，它显示了预置的信息，并提供一个文本域，供用户输入响应。这个对话框有两个按钮 Cancel 和 OK，允许用户在关闭这个对话框后执行完全不同的操作：取消整个操作或接收输入到对话框中的文本。

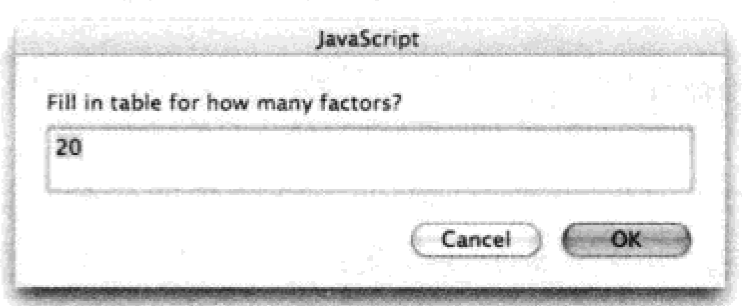


图 10-4　JavaScript 提示对话框(Safari 2 风格)

window.prompt()方法有两个参数，第一个参数是呈现给用户的提示信息；第二个参数是一个字符串，它在文本域中显示为默认的答案。如果不希望显示默认答案，就传入一个空字符串(不含任何空格的双引号)。

用户单击按钮时，这个方法就会返回一个值。不管用户在文本域中输入什么，单击 Cancel 按钮都返回 null 值，单击 OK 按钮则返回输入的字符串值。这个信息可在脚本的 if 和 if... else 结构条件中使用。在条件中，null 值等同于 false，空字符串也看做 false。因此，很容易检测用户是否在文本域中输入了字符，来简化条件测试，如下面的程序段所示：

```
var answer = prompt("What is your name?","");
if (answer)
{
   alert("Hello, " + answer + "!");
}
```

在这个例子中，只有用户在提示对话框中输入字符并单击了 OK 按钮，才会调用 alert()方法。

10.3.4　load 事件

window 对象会响应几个系统和用户事件，最常见的是页面加载完毕时触发的事件，这个

更多资源请访问稀酷客(www.ckook.com)

事件等待图像、Java applet 和插件程序的数据文件完全下载到浏览器中。脚本在加载页面时访问 document 对象的元素可能十分危险，因为如果还没有载入该对象(也许由于网络连接或服务器较慢)，就会产生脚本错误。使用 load 事件调用函数的优点在于，它确保所有 document 对象都存在于浏览器的 DOM 中。

可以通过几种方式将 load 事件处理程序应用于 window 对象，具体取决于浏览器和环境：

```
window.addEventListener('load', functionName, false);
window.attachEvent('onload', functionName);
```

其中 functionName 是页面下载完毕后要运行的函数(跨浏览器的事件添加函数参阅第 32 章)。可以多次调用 addEventListener 和 attachEvent，把多个函数添加到页面加载完毕后要执行的列表中。

还可以把该事件直接应用于元素：

```
window['onload'] = functionName;
window.onload = functionName;
```

但是，这种用法表示，页面加载完毕后，只执行一个函数，并替代已赋予 window 对象的其他事件处理程序。

在老式的网页中，有时窗口事件处理程序会应用于 HTML 中的 body 元素：

```
<body onload="functionName()">
```

即使以后要把<body>标记的特性与 document 对象的属性联系起来，放在该标记中的也是 window 对象的事件处理程序。目前，这种把 JavaScript 嵌入 HTML 中的方式是不好的用法，原因有几个：它不允许不能处理脚本的浏览器按正常方式失败，HTML 文件比较大，脚本和标记混合在一起，修改会比较耗时。分层是其解决方法。

交叉引用：
window.onload 事件详见第 27 章。

10.4 location 对象

有时，层次结构中的对象与窗口或按钮似乎代表不同的东西，location 对象就是如此，这个对象表示载入窗口的 URL。它和 document 对象(本章后面讨论)不同：文档是页面的内容，而位置是页面的 URL(Uniform Resource Locator 或地址)。

URL 包含许多内容，它定义了文件的地址和数据传输方法。URL 包括协议(如 http:)和主机名(如 www.example.com)，使用 location 对象的属性可以访问所有这些内容。脚本大多只使用该对象的一个属性：href，它定义了完整的 URL。

脚本导航到其他页面的基本方法是设置 location. href 属性：

```
location.href = "http://www.example.com/contact.html";
```

对于当前页面之外的页面，需要指定完整的 URL。指定相对的 URL(相对于当前载入的页

更多资源请访问稀酷客(www.ckook.com)

面)，一般可以导航到 Web 站点的一个页面，不必使用包含协议和主机信息的完整 URL。

```
location.href = "contact.html"; // relative URL
```

如果要加载的页面在另一个窗口或框架中，就必须在语句中包含窗口的引用。例如，用脚本打开一个新窗口，并将它的引用赋给变量 newWindow，将页面载入子窗口，则语句如下：

```
newWindow.location.href = "http://www.example.com";
```

10.5　navigator 对象

navigator 这个名称令人想起 Netscape Navigator 浏览器，但这个对象在所有脚本浏览器中都得到了支持。这些浏览器还实现了它的一些属性，每次请求页面时，这些属性都会提供浏览器发送给服务器的信息。因此，navigator.userAgent 属性返回一个包含浏览器和操作系统的诸多细节的字符串。例如，在 Windows XP 的 Internet Explorer 8 中运行脚本，会显示以下 navigator.userAgent 属性值：

```
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1)
```

在 Macintosh 的 Firefox 3.5.2 中运行相同的脚本，会显示以下 userAgent 详情：

```
Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10.4; en-US; rv:1.9.1.2)
    Gecko/20090729 Firefox/3.5.2
```

交叉引用：

有关 Navigator 对象及其属性值的详情，请参见光盘上的第 42 章。可惜它不能准确地报告用户代理的厂家、型号和版本。此对象曾经广泛用于根据不同的浏览器版本执行不同的脚本。第 25 章描述了检测浏览器功能的现代方式。

10.6　document 对象

document 对象包含页面的实际内容，其属性和方法通常会影响文档在窗口中的外观和内容。如第 5 章所示，所有符合 W3C DOM 标准的浏览器都允许脚本在文档加载后访问页面的文本内容，DOM 方法还允许脚本在页面加载后动态创建内容。document 对象的许多属性都是文档中其他对象的数组，它们提供了引用这些对象的额外方式(document.getElementById()方法除外)。

访问 document 对象的属性和方法很简单，如下所示：

```
[window.]document.propertyName
[window.]document.methodName([parameters])
```

脚本访问包含它的 document 对象时，可以添加或不添加 window 引用。要预览 IE 或基于 Mozilla 浏览器的 document 对象属性的长列表，可在 The Evaluator Jr.的对象文本框中输入 document，然后按回车键。对象的属性、当前值和值的类型就会显示在 Results 框中(在 Mozilla 中还有方法)。以下是 document 对象中最常用的属性和方法。

更多资源请访问稀酷客(www.ckook.com)

10.6.1　document.getElementById()方法

第 5 章学习引用元素对象的语法时，就遇到过 document.getElementById()方法。这个 W3C DOM 方法十分常用，也非常重要，必须熟悉其名称，注意大小写。

此方法的唯一参数是带引号的字符串，该字符串包含要引用的元素 ID。第 8 章(及光盘上的程序清单)中的 Evaluator Jr.页面有几个元素对象，其 ID 分别为 input、output 和 inspector。在顶部的文本框中输入此方法，并把每个 ID 作为其参数，如下所示：

```
document.getElementById("output")
```

该方法返回一个值，通常将该值保存在一个变量中，供随后的脚本语句使用，例如：

```
var oneTable = document.getElementById("salesResults");
```

在赋值语句后，变量就表示元素对象，可用于获取和设置该对象的属性，或者调用此类对象的方法。

10.6.2　document.getElementsByTagName()方法

可以使用 getElementsByTagName()方法方便地收集一组共享同一个标记名称的页面元素。例如，要获得页面上所有图像，可以调用：

```
var aImages = document.getElementsByTagName('img');
```

在包含图像的数组中根据位置引用一个图像是比较困难的——例如，要处理页面上的一个图像库，而页面在该图像库的前后都包含其他图像，就是比较难的。使用 getElementsByTagName()的一个优点是，它可以在页面的任何容器元素中收集某个集合：

HTML:

```
<div id="gallery">
   <h2>My Gallery</h2>
   <img src="bat.jpg" alt="bat">
   <img src="cat.jpg" alt="cat">
   <img src="hat.jpg" alt="hat">
</div>
```

JavaScript:

```
var oGallery = document.getElementById('gallery');
var aCollection = oGallery.getElementsByTagName('img');
```

这个例子中的 aCollection()数组包含 3 个图像对象(当然不包含 h2 元素)。根据第 9 章对数组的讨论，数组方括号中的下标号指向数组中的一个元素。要确定集合中的元素个数，可以使用：

```
aCollection.length
```

每个对象都可以根据它在数组中的偏移量来访问：

更多资源请访问稀酷客(www.ckook.com)

```
oImage = aImages[0];       // first image
oImage = aImages[1];       // second image
```

这个方法的其他用法有：收集列表中的所有项、文本块中的所有超链接或页面上的所有多媒体对象。

10.6.3 document.forms[ ]属性

在 DOM 推出之前，文档对象的 document.forms 属性是文档中所有 form 元素对象的数组。比较下面两个表达式：

```
var aForms = document.forms;
var aForms = document.getElementsByTagName('form');
```

一个重要的区别在于，document.forms 集合允许按名称直接引用某个表单，而不仅仅是按该表单在数组中的偏移量来引用。按下标号引用表单未必可行。动态网页可能根据上下文包含数量可变的表单，所以表单在页面上的位置可能随上下文而变化。例如，网站上每个网页顶部的 Search 表单可能在 Advanced Search 页面上隐藏，Join List 表单在 Contact 页面上可能隐藏，页面程序清单 workshoop 则可能填满了所选 workshoop 的注册表。

支持脚本编程的浏览器允许通过表单名称或 ID(即分配给<form>标记的 name 或 id 特性的标识符)更直接地引用表单：

```
<form id="formId" name="formName" ...>
```

第一种方式是使用 getElementById()方法：

```
document.getElementById("formId")
```

第二种方法使用数组语法，将表单名称或 ID 作为数组的字符串下标：

```
document.forms["formId"]
document.forms["formName"]
```

第三种按名称引用 form 对象的简短方式是把表单的名称附加为 document 对象的属性：

```
document.formName
```

但是，只有 name 特性忽略了几个在 HTML 元素名中合法、但在 JavaScript 对象名中非法的字符，例如连字符(-)、句点(.)和冒号(:)(详见第 6 章)，最后这种方法才能生效。

这几种方法会引用相同的对象。本书主要使用 DOM 方法 getElementById()和 getElementsByTagName()，但 document.forms 集合语法可追溯到最早的脚本浏览器，且仍可用于大多数现代浏览器。

10.6.4 document. images[ ]属性

文档利用特殊数组属性来跟踪表单，同样，document 对象也通过<img>标记保存插入文档的图像集合(数组)。通过 document.images 数组引用的图像可以用 img 元素名称中的数字或字符串下标来得到。与表单一样，name 属性值是用作字符串下标的标识符。

更多资源请访问稀酷客(www.ckook.com)

document 拥有 images 属性，这表明浏览器支持图像对象和相关的脚本编写技术，例如图像交换技术。因此可将此属性存在与否作为测试条件，在对图像执行任何脚本操作之前，确保浏览器支持图像对象。为此，将处理图像的语句放在 if 结构中，来验证该属性是否存在，如下所示：

```
if (document.images)
{
    // statements dealing with img objects
}
```

较旧的浏览器会跳过嵌套语句，以防向其用户显示错误消息。

在当今典型的复杂页面中，不同部分的图像往往用于完全不同的目的，把页面上的所有图像收集到一个数组中通常没有意义。此时，使用 DOM 方法 getElementsByTagName()会更加方便(参见上面的例子)。

10.6.5　document.createElement()和 document.createTextNode()方法

在 HTML 文档中添加新元素至少需要两个步骤：

(1) 为文档创建新元素。

(2) 在页面的树形结构中，将其插入到需要的位置。

document.createElement()方法可以在浏览器的内存中创建一个全新的元素对象。指定要创建的元素时，应将元素的标记名作为该方法的字符串参数：

```
var newElem = document.createElement("p");
```

还可给元素添加一些属性值，为此应在元素成为文档的一部分之前，向新建对象的属性赋值：

```
newElem.setAttribute("class", "intro");
```

可将这个元素插入文档中的任何位置，例如末尾：

```
document.body.appendChild(newElem);
```

下面 3 行代码在文档体的尾部生成一个段落：

```
<body>
...
<p class="intro"></p></body>
```

如第 6 章的对象层次结构图所示，元素对象通常需要在其首尾标记之间有文本内容。创建该文本的 W3C DOM 方式是利用 document.createTextNode()方法生成一个全新的文本节点，再用所需的文本填充该节点。例如：

```
var newText = document.createTextNode("Greetings to all.");
newElem.appendChild(newText);
```

得到：

```
<body>
```


更多资源请访问稀酷客(www.ckook.com)

```
...
<p class="intro">Greetings to all.</p></body>
```

可以看出，创建元素或文本节点的操作本身并不会影响文档节点树。必须调用各种插入或替换方法中的一个，将新文本节点放入其元素中，并将该元素插入文档，参见第 5 章。

10.6.6　document.write()方法

document.write ()方法是给文档写入内容的另一种方式。它的优点是可以临时应急，缺点如下：

- 只有页面第一次加载到浏览器中时，document.write()才能给一个网页添加内容。以后调用该方法都会替代整个页面。
- document.write()会将旧文本块插入文档，编写出草率的程序和错误的标记，将标记与脚本内容混合在一起，不利于开发的分层(内容与结构分开)。
- document.write()不能用于 XHTML，XHTML 这种文档类型不允许在第一次解析期间修改其内容。

因此建议在自己的代码中使用 document.write()，但理解其工作原理仍有助于阅读旧代码。

在页面载入过程中，可以在即时脚本中使用 document.write()创建页面内容，也可以在延时脚本中使用它创建本窗口或新窗口的内容。该方法需要一个字符串参数：要写到窗口或框架中的 HTML 内容。此类字符串参数可以是值为字符串的变量或表达式，写入的内容常常包括 HTML 标记。

在载入页面后，浏览器输出流会自动关闭。之后，任何一个操作当前页面的 document.write()方法都会立即清除当前页面(包括源文档的变量或其他值)，再打开一个新的输出流。因此，如果希望用脚本生成的 HTML 替换当前页面，就必须把 HTML 内容赋给一个变量，再使用 document.write()方法完成写操作。不必明确地清除文档，再打开一个新数据流，一个 document.write ()调用就可完成这些操作。

document.write()方法的最后一个要点涉及它的相关方法 document.close()。脚本向窗口(不管是本窗口或其他窗口)写完内容后，必须关闭输出流。所以在延时脚本的最后一个 document.write()方法后面，必须包含 document.close()方法，否则就不能显示图像和表单。并且，以后调用的任何 document.write()方法只会把内容追加到页面上，而不会清除现有内容，再写入新值。

为了演示 document.write()方法，下面提供同一个应用程序的两个版本。一个应用程序将内容写入包含脚本的文档，另一个则写入到一个独立的窗口。首先在文本编辑器中打开一个新文件，输入代码，以.html 文件扩展名保存，然后在浏览器中打开该文件。

程序清单 10-2 创建了一个按钮，用于为文档创建新的 HTML 内容，包括新文档标题的 HTML 标记和 <body>标记的颜色特性。程序清单中有一个不熟悉的操作符+=，它把其右侧的字符串追加到其左侧变量保存的字符串中。这个操作符能方便地把几个语句组合成一个长字符串。在 newContent 变量中组合好内容后，只使用一个 document.write()语句就可以把所有新内容写入到同一个文档中，完全清除程序清单 10-2 中的内容。然后调用 document.close()语句按正常方式关闭输出流。载入该文档并单击按钮时，注意浏览器标题栏中的文档标题会相应地改变。回到原始文档，再次单击该按钮时，动态写入的第二个页面的载入速度甚至比重载原始文档还要快。

更多资源请访问稀酷客(www.ckook.com)

程序清单 10-2　在当前窗口中使用 document.write()

HTML: jsb-10-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Writing to Same Doc</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-10-02.js"></script>
   </head>
   <body>
      <h1>Writing to Same Doc</h1>
      <form>
         <p>
            <input type="button" id="rewritePage" value="Replace Content">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-10-02.js

```
// replace the page with new markup
function reWrite()
{
 // assemble content for new window
 var newContent = '<!DOCTYPE html>';
 newContent += '<html>';
 newContent += '<head>';
 newContent += '<meta http-equiv="content-type"
    content="text/html;charset=utf-8">';
 newContent += '<title>A New Doc</title>';
 newContent += '<style type="text/css">';
 newContent += 'body { background-color: aqua; }';
 newContent += '</style>';
 newContent += '</head>';
 newContent += '<body>';
 newContent += '<h1>This document is brand new.</h1>';
 newContent += '<p>Click the Back button to see the original document.</p>';
 newContent += '</body>';
 newContent += '</html>';

 // write HTML to new window document
 document.write(newContent);
 document.close(); // close layout stream
}

// apply behaviors when document has loaded
function initialize()
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
       // point to the button
       var oButtonRewrite = document.getElementById('rewritePage');

           // if it exists...
           if (oButtonRewrite)
           {
              // apply event handler
              addEvent(oButtonRewrite, 'click', reWrite);
           }
    }
}

// initialize when the page has loaded
addEvent(window, 'load', initialize);
```

程序清单 10-3 的情况有点复杂，因为脚本生成了一个子窗口，脚本生成的整个文档都将写入该子窗口。

程序清单 10-3　在另一个窗口中使用 document.write()

HTML: jsb-10-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Writing to Subwindow</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-10-03.js"></script>
   </head>
   <body>
      <h1>Writing to Subwindow</h1>
      <form>
         <p>
            <input type="button" id="writeSubwindow" value="Write to Subwindow">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-10-03.js

```
var newWindow;

function makeNewWindow()
{
   newWindow = window.open("","","status,height=200,width=300");
}

function subWrite()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
    // make new window if someone has closed it
    if (!newWindow || newWindow.closed)
    {
        makeNewWindow();
    }

  // bring subwindow to front
  newWindow.focus();

  // assemble content for new window
  var newContent = '<!DOCTYPE html>';
  newContent += '<html>';
  newContent += '<head>';
  newContent += '<meta http-equiv="content-type"
  content="text/html;charset=utf-8">';
  newContent += '<title>A New Doc</title>';
  newContent += '<style type="text/css">';
  newContent += 'body { background-color: aqua; }';
  newContent += '</style>';
  newContent += '</head>';
  newContent += '<body>';
  newContent += '<h1>This document is brand new.</h1>';
  newContent += '</body>';
  newContent += '</html>';

  // write HTML to new window document
  newWindow.document.write(newContent);

  // close layout stream
  newWindow.document.close();
}

// apply behaviors when document has loaded
function initialize()
{
  // do this only if the browser can handle DOM methods
  if (document.getElementById)
  {
     // point to the button
     var oButtonRewrite = document.getElementById('writeSubwindow');

          // if it exists...

          if (oButtonRewrite)
          {
              // apply event handler
              addEvent(oButtonRewrite, 'click', subWrite);
          }
  }
}

// initialize when the page has loaded
```


更多资源请访问稀酷客(www.ckook.com)

```
addEvent(window, 'load', initialize);
addEvent(window, 'load', makeNewWindow);
```

注意：
为了运行该脚本，必须临时关闭阻塞弹出窗口。

为使新窗口的引用在两个函数中都可用，本例将 newWindow 变量声明为全局变量。页面载入时，onload 事件处理程序会调用 makeNewWindow()函数，该函数生成一个空的子窗口。另外，本例还在 window.open()方法的第三个参数中加入一个属性，使子窗口的状态栏可见。

页面上的按钮会调用 subWrite()方法，它执行的第一个任务是检查子窗口的 closed 属性。如果关闭了被引用的窗口，该属性返回 true。此时(如果用户手动关闭窗口)，该函数会再次调用 makeNewWindow()函数，重新打开那个窗口。

打开窗口后，新的内容组合为一个字符串变量。与程序清单 10-2 一样，也是一次写入所有内容(但这对单个窗口没有必要)，接下来调用 close()方法。但需要注意一个重要的区别：write()和 close()方法都明确指定了子窗口。

在对象层次结构结构中，document 对象的下一层是表单，这是下一章讨论的主要内容。

10.7 习题

1. 下列哪些引用是有效的，哪些是无效的？对无效引用请解释其原因。

```
a. window.document.form[0]
b. self.entryForm.submit()
c. document.forms[2].name
d. document.getElementByID("firstParagraph")
e. newWindow.document.write("Howdy")
```

2. 编写一个 JavaScript 语句，在 Web 页面上显示一个欢迎用户进行访问的对话框。

3. 编写一个 JavaScript 语句，在页面载入时执行，把第 2 题的信息显示为页面的<hl>层标题。

4. 创建一个页面，在页面载入时(通过一个对话框)提示用户输入用户名，然后在页面中利用该用户名显示欢迎信息。

5. 用自己喜欢的任何内容创建页面，但要求在页面载入后自动弹出一个对话框，显示当前页的 URL。

更多资源请访问稀酷客(www.ckook.com)

第11章

表单和表单元素

本章包含哪些内容？

- 表单对象的含义
- 访问表单对象的关键属性和方法
- 文本、按钮、选择对象的工作方式
- 从脚本中提交表单
- 向函数传递表单元素中的信息

每个浏览器的表单都有许多交互式 HTML 元素：文本域、按钮、复选框和选项列表等。 Web 页面和用户之间的大多数交互操作都在表单中进行。

本章讨论如何在文档树中定位表单及其控件，如何修改它们，如何检查用户的输入，如何提交表单，以及如果输入无效该如何禁止提交它们。

11.1 form 对象

表单以及其中的输入控件是 DOM 对象，其独特的属性是文档中其他对象所没有的。例如，表单对象的 action 属性告诉浏览器，在提交表单时，将输入值发送到什么地方。select 控件(下拉列表)的 selectedIndex 属性指出用户选择了哪个选项。

首先引用页面上的表单。可以采用 3 种方式，在一个包含 3 个表单的页面上，它们都引用如下缩写片段：

```
<div id="header">
   <form id="search" action="...">...</form>
   <form id="join-list" action="...">...</form>
</div>
   ...
   <form id="contact" action="...">...</form>
   ...
```

(1) 如果指定页面上某个特定元素的 id 属性，前面章节介绍的 DOM 方法 getElementById() 就可以提供该元素的句柄：

```
var oForm = document.getElementById('search');
```

更多资源请访问稀酷客(www.ckook.com)

实际上，在大多数情况下，使用getElementById()便足以满足需要，因为网页一般包含一个或几个不同的表单，它们完成完全不同的任务，在标记中给它们赋予独特的 ID 是自然而然的事情。

(2) DOM 方法 getElementsByTagName()用给定的标记名称传送一个表单对象的数组或集合：

```
var aForms = document.getElementsByTagName('form');
var oForm = aForms[0]; // get the first form on the page
```

通常 JavaScript 数组中第一项的下标是 0，所以包含 3 个表单的集合可以表示为数组元素 0、1 和 2。

应考虑一下使用页面上所有表单的集合是否有用。为了定位集合中的某个表单，必须指定它在页面上所有表单中的位置(在富动态站点上该位置可能变化)，否则就必须遍历这些表单，查找标识符(这说明首先应该使用 getElementById())。

getElementsByTagName()的一个优点是，它可以给出某个父元素(而非 document 元素本身)中的对象集合。

```
var oHeader = document.getElementById('header');
var aForms = oHeader.getElementsByTagName('form');
var oForm = aForms[0];
```

这个例子只收集了文档中 header 部分的所有表单。

考虑包含一系列工作坊的页面列表，每个工作坊都有各自的注册表。先引用其父表单 div，而不考虑页面上的其他表单，就可以收集所有这些注册表。

(3) document.forms 语法是收集页面上所有表单的另一种方式，它在目前的 DOM 方法推出之前使用，现代浏览器仍支持它，以免使老网站崩溃。

```
var aForms = document.forms;
// then point to one form:
var oForm = aForms[0];
// or:
var oForm = aForms["search"];
// or:
var oForm = aForms.search;
```

在这个初始的 DOM Level 0 语法中，form 对象可以通过文档中的表单数组下标来引用，也可通过名称来引用(把一个标识符赋给<form>标记中的 id 或 name 特性)。即使文档中只有一个表单，它也是数组成员(单元素数组)，其引用如下：

```
document.forms[0]
```

也可以使用元素名称的字符串作为数组下标：

```
document.forms[formName]
```

注意数组引用使用了单词的复数形式，其后的一组方括号包含了元素下标号(第一个下标号通常为 0) 或元素名称。另外，也可以使用不加引号的表单名，就像它是 document 对象的属性：

```
document.formName
```


更多资源请访问稀酷客(www.ckook.com)

但是，最后这个语法仅在表单 id 或 name 不包含连字符、句点或冒号等 JavaScript 非法字符时有效。例如，上述示例 HTML 的第二种形式是，令 id 是 join-list：

```
var oForm = document.forms.join-list;
```

看起来 JavaScript 似乎要从 id 为 join 的 form 对象中减去 list 变量的值。这个语句会停止 JavaScript 的运行，或者生成结果 NaN (Not a Number)。通常建议使用带引号的元素名称。

访问 form 对象有这么多方式，所以后面的例子常常使用 formObject 表示它们。

11.1.1　将表单作为对象和容器

现代 DOM 要与过去的许多约定兼容，所以表单对象同时是两个不同的树系列的父对象。现代的 DOM Level 2 指定，表单是其所有子节点(包括元素和文本节点)的父节点，而老 DOM Level 0 将 form 对象仅作为其表单控件对象(input、select、button 和 textarea 元素)的容器。图 11-1 显示了这一 DOM 0 层次的结构及其相对于 document 对象的位置。稍后将介绍这个结构对引用表单控件元素的方式的影响。

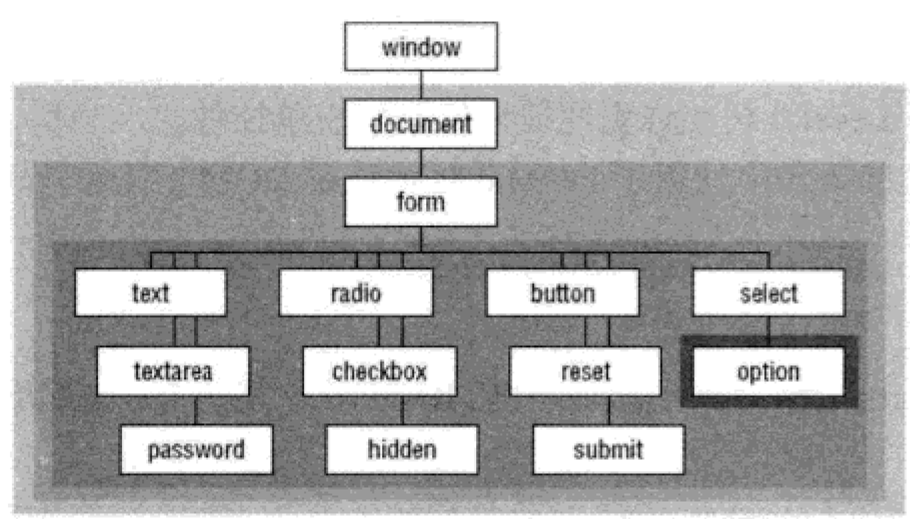


图 11-1　表单和控件的 DOM Level 0 层次结构

用图来表示如程序清单 11-1 所示的 HTML 片段，说明其区别：

程序清单 11-1　示例表单标记

```
<form action="search.php" method="post">
   <p>
      <label for="inputSearch">Search for:</label>
      <input id="inputSearch" name="inputSearch" type="text" value="">
      <input id="submit" type="submit" value="Search">
   </p>
</form>
```

图 11-2 显示了这段标记代码的 DOM Level 2 树。注意它表示该文档段的完整内容。元素之间的回车键、制表符和空格是文本节点(这里用缩写“[whitespace]”来表示)。

图 11-3 显示了这个表单的 DOM Level 0 树。它只包含输入控件，忽略了段落、标签和文本节点。

更多资源请访问稀酷客(www.ckook.com)

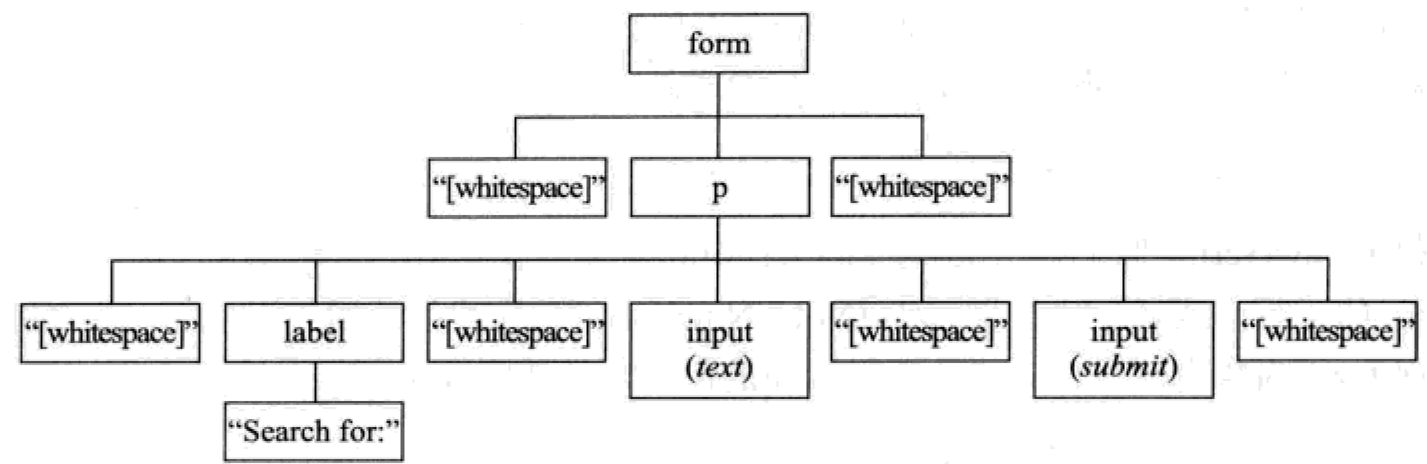


图 11-2　典型表单的 DOM Level 2 树

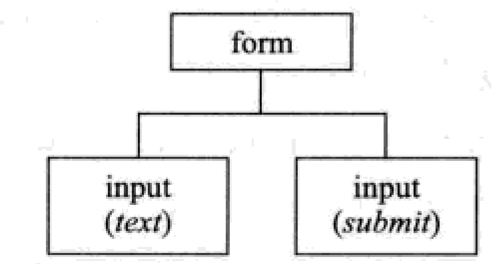


图 11-3　同一表单的 DOM Level 0 树

这两个对象树显然用于完全不同的目的。DOM Level 2 树可用于读写整个文档内容，了解所有的细节，而 DOM Level 0 树允许更便捷地仅读写表单控件。

注意，无论在给定的语句中使用了 DOM 0 技术还是 DOM 2 技术，对象都是相同的。例如：

```
var oForm = document.getElementById('myForm');
var oControl = oForm.fritzy;
```

如果 fritzy 是 myForm 表单中一个输入控件的 ID，那么上述两个语句都是完全有效的 JavaScript，都可以正常工作。

表单对象有大量与所有 HTML 元素对象共享的属性和方法，还有许多特有的项。这些特有的属性几乎都是 form 元素特性(action、target 等)的脚本编程表示。脚本浏览器允许通过脚本控制这些属性的更改，因此脚本可以根据用户在页面上的选择，指引表单的提交行为。

11.1.2　访问表单属性

在创建表单时，可以使用 HTML 页面中的标准标记，也可以使用 JavaScript 中的 DOM 方法。无论采用哪种方法，都可以设置 name、target、action、method 和 enctype 特性。这些都是 form 对象的属性，可以通过这些字的小写形式访问，如下所示：

```
var sURL = formObject.action;
```

要修改这些属性，只需要为它们赋予新值：

```
formObject.action = "http://www.example.com/cgi/login.pl";
```

可以用复合对象引用将这两个 JavaScript 语句重写为：

```
var sURL = document.getElementById('formName').action;
document.forms[0].action = "http://www.example.com/cgi/login.pl";
```


更多资源请访问稀酷客(www.ckook.com)

然而，把多个操作合并为一个表达式，脚本并不能测试其有效性。把操作和测试分开是比较安全的编码方式：

```
var oForm = document.getElementById ('formName');
   if (!oForm)
   {
      // do something if the named form isn't found
   }
var sURL = oForm.action;
```

11.1.3　form.elements[ ]属性

elements[]属性是表单中所有输入控件的集合。这个数组中各项的顺序根据 HTML 标记在源代码中的顺序而定。使用 ID 直接引用单个元素通常更有效，然而，有时脚本需要浏览表单中的所有元素，例如，表单验证例程可能会遍历每个元素，确保用户正确输入了其值。像这样的循环并不需要查看并非表单控件的文本节点和其他元素，实际上它只有对遇到的每个元素执行一系列测试，才能确定它是否是一个控件。在这种情况下，使用 elements[]集合会高效得多。

下面的代码段在 for 循环中使用 form.elements[]属性，查看表单中的所有控件元素，并将文本域的内容设置为空字符串。该脚本不能简单地进入表单内部，把每个元素的内容设置为空字符串，因为有些类型的元素(诸如按钮)的 value 属性具有其他用途。

```
var oForm = document.getElementById('registration-form');
   if (!oForm) return false;

for (var i = 0; i < oForm.elements.length; i++)
{
   if (oForm.elements[i].type == "text")
   {
      oForm.elements[i].value = "";
   }
}
```

第一个语句创建了变量 oForm，它含有所需表单的引用。这样，以后在脚本中多次引用表单元素时，可以把 oForm 变量作为快捷键，给嵌套在表单更深层的项建立引用，使每个引用的长度更短(这样会很快)。

接着开始遍历表单中的 elements 数组元素。每个表单控件元素都有一个 type 属性，它表示表单控件的类型：文本、按钮、单选框、复选框和文本域等。本例对于每个类型为 text 的元素，将其 value 属性设置为空字符串。

这个例子允许使用复合表达式，例如oForm.elements[i].type，因为该表达式已测试过，能确保表单是存在的，DOM 一定能在 elements[]集合的每次迭代中返回一个有效的表单控件对象。即使表单不包含控件，脚本也不会失败；如果 elements[]集合是空数组，上述 for 循环在第一次迭代之前就会正确无误地停止，因为 oForm.elements.length 是 0。

本章后面会介绍如何在没有 Submit 按钮的情况下提交表单，以及如何在客户端上验证表单。

更多资源请访问稀酷客(www.ckook.com)

11.2 将表单控件作为对象

在所有浏览器的 DOM 中，嵌套在<form>标记中的 HTML 元素有三种是脚本对象。多数对象都在页面源代码的<input>标记中，只有赋给<input>标记的 type 特性的值能确定元素是文本框、密码输入域、隐藏域、按钮、复选框还是单选框。其他两种表单控件 textarea 和 select 有各自的标记。

要将某个表单控件作为对象进行引用，可以通过 DOM Level 2 方法使用其 id 或 tagName 直接引用，或者使用 DOM Level 0 语法，将引用构造为一个以 document 开头、后跟 form 和控件的层次结构。仅表单部分的引用就有多种方式——它们都可用于构造表单控件引用。但如果只使用赋予表单和表单控件元素的标识符(而非使用相关的元素数组)，则语法如下：

```
document.getElementById(controlName)
```

或者

```
document.formName.controlName
```

例如，考虑下面的简单表单：

```
<form id="searchForm" action="cgi-bin/search.pl">
   <p>
      <input type="text" id="entry" name="entry">
      <input type="submit" id="sender" name="sender" value="Search">
   </p>
</form>
```

对于文本输入控件，以下引用示例都是有效的：

```
document.getElementById("entry")
document.searchForm.entry
document.searchForm.elements[0]
document.forms["searchForm"].elements["entry"]
document.forms["searchForm"].entry
```

表单控件有一些共同的属性，但某些属性是特定控件类型或相关类型所特有的。例如，只有 select 对象提供了显示其列表中当前所选项的属性，而复选框和单选按钮都有一个表示其当前是否置为打开的属性。但是，所有面向文本的控件在内容的读取和修改方式上都相同。

熟练掌握表单控件对象的脚本编程功能对于成功运用 JavaScript 十分重要。以下几节将学习最重要的表单控件对象以及脚本与它们的交互方式。

11.2.1 与文本相关的输入对象

与文本相关的 HTML 表单元素有 4 个：text、password 和 hidden 类型的 input 元素以及 textarea 元素，它们都是文档对象层次结构中的元素。除了隐藏类型外，它们都显示在页面上，允许用户输入信息，选择选项。

在页面中对这些表单控件对象进行脚本编程时，通常不需要对它们的常规 HTML 标记做任

更多资源请访问稀酷客(www.ckook.com)

何处理，只需要给 id 特性赋值。假如脚本需要读写这些属性或者调用它们的方法，最好给每个与文本相关的表单控件元素指定唯一的 ID 和名称。ID 便于 DOM 操作，并将标签与控件联系起来，而名称是表单正常工作所必需的。假如将表单提交给一个服务器端程序，则必须把控件元素的 name 特性和元素值一起发送给服务器。

对于这个类别中的可见对象，事件处理程序可由许多用户动作触发，如使一个域获得焦点(令文本插入点显示在域中)或者修改文本(输入新文本，然后离开该域)。多数文本域动作都因文本的改变而触发(onchange 事件处理程序)。在当前浏览器中，也可以因按下某个键而触发事件。

毫无疑问，文本相关元素最常用的属性是 value，它表示文本元素的当前内容，脚本可随时读写其内容。value 属性的内容通常是一个字符串。假如文本域的输入值用于数学运算，就需要将其转换为数字(参阅第 8 章)。

文本对象行为

许多脚本编程员希望 JavaScript 能弥补表单中文本相关对象的缺陷或消除其反常行为。我们在刚开始学习脚本编程时就指出这些问题，以防以后发生混淆。

多数浏览器表单都能执行一个操作，这是 Web 先驱以前推荐的一种非正式标准。表单只包含一个文本 input 对象时，如果文本对象拥有焦点，按下回车键就自动提交表单。在除 IE5/Mac 和 Safari 之外的浏览器中，如果表单有两个或多个域，就需要用另一种方法来提交表单(比如 Submit 按钮)。在许多情况下，单域提交机制都是有效的，比如多数 Web 搜索站点的搜索页面。但对于只包含一个域的简单表单，可以通过按下回车键来提交表单。提交表单时如果没有其他的指定动作或目标，页面就会执行无条件重载，清除在表单中输入的所有信息。这个提交操作可以通过表单中的 ousubmit 事件处理程序取消，如本章后面所示。也可以编程，以使在任何文本域中都能用回车键来提交表单(详见第 32 章)。

为了演示如何读写文本域的 value 属性，程序清单 11-2 提供了一个只有单个输入域的完整 HTML 页面。在该域中输入文本，按下 Tab 或回车键，输入的文本就会全部变成大写形式。

程序清单 11-2　获取和设置文本对象的 value 属性

HTML: jsb-11-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Text Object value Property</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-11-02.js"></script>
   </head>
   <body>
      <h1>Text Object value Property</h1>
      <form id="UCform" action="make-uppercase.php">
         <p>
            <input type="text" id="converter" name="converter" value="sample">
         </p>
      </form>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb-11-02.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

var oInput; // (global) input field to make uppercase

// apply behaviors when document has loaded
function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // apply event handler to the button
        oInput = document.getElementById('converter');
            if (oInput)
            {
                addEvent(oInput, 'change', upperMe);
            }

        // apply event handler to the form
        var oForm = document.getElementById('UCform');
            if (oForm)
            {
                addEvent(oForm, 'submit', upperMe);
            }
    }
}

// make the text UPPERCASE
function upperMe(evt)
{
        // consolidate event handling
        if (!evt) evt = window.event;

    // set input field value to the uppercase version of itself
    var sUpperCaseValue = oInput.value.toUpperCase();
    oInput.value = sUpperCaseValue;

    // cancel default behavior (esp. form submission)
        // W3C DOM method (hide from IE)
        if (evt.preventDefault) evt.preventDefault();

    // IE method
    return false;
}
```

下面解释这个示例的工作原理：页面加载到浏览器中时，initialize()函数就把事件处理程序应用于表单及其包含的输入域。表单和输入域的 onchange 事件处理程序都调用了 upperMe()函数，把文本转换为大写形式。

更多资源请访问稀酷客(www.ckook.com)

注意引用输入域 converter 的变量 oInput 在所有函数的外部声明，因此是一个全局变量。这意味着在脚本的任何地方都可以访问它。这么做就可以在两个不同的函数 initialize()和 upperMe()中使用它了(与全局变量相反，局部变量在函数的内部用 var 声明，不能在声明它们的函数之外访问)。

upperMe()函数的核心是两个语句：

```
var sUpperCaseValue = oInput.value.toUpperCase();
oInput.value = sUpperCaseValue;
```

该函数的第一个语句完成了许多操作。赋值语句的右边部分执行了两个重要任务。对象的 value 属性引用(oInput.value)计算为当前文本域中的内容(注意 oInput 是引用 converter 输入域的全局变量)。接着把该字符串传送给 JavaScript 的一个字符串函数 toUpperCase()，该函数将字符串转换为大写形式。然后把该语句右边的计算结果赋予第二个变量 sUpperCaseValue。现在文本框还没有任何变化，但第二个语句就改变了文本框：文本框的 value 属性被赋予 sUpperCaseValue 变量的内容。为便于理解，这里把这个逻辑分解为两个语句。实际上，可将第 1 步和第 2 步的操作合并为一个语句：

```
oInput.value = oInput.value.toUpperCase();
```

在按下 Tab 键，退出输入域时，该域的 onchange 事件就使文本变成大写。这是一个单输入域表单，按下回车键会提交表单，所以把表单的 onsubmit 事件设置为也运行 upperMe()函数。除了把文本变成大写之外，该函数还取消了正常的提交行为(事件捕获详见本章后面的内容)。

如果显示这个页面的用户代理禁用或不支持 JavaScript，输入的文本就提交给服务器端程序 make-uppercase.php，它也能把文本变成大写。

注意：

本章和本书其他许多地方使用的事件处理程序分配函数是 addEvent()，这是一个跨浏览器的事件处理程序，详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，该文件位于配书光盘中，这个文件夹可以在所有章节的脚本中访问。

11.2.2　按钮输入对象

本章的许多例子都使用了按钮类型的 input 元素，在脚本编程中，按钮是最简单的对象之一。在本章的简化对象模型中，按钮对象只有几个属性，它们在日常编程中很少访问或修改。按钮的可见性不受 HTML 或脚本控制，而是受页面访问者使用的操作系统和浏览器控制，这与文本对象一样。到目前为止，按钮对象最有用的事件是 click。当用户单击按钮时，就会触发该事件，非常简单。

11.2.3　复选框输入对象

复选框也是 form 对象的一个简单元素，但它的有些属性可能不很直观。简单按钮对象的 value 属性是按钮的标签文本，而复选框的 value 属性是与该对象相关联的其他任何文本。这

更多资源请访问稀酷客(www.ckook.com)

个文本不显示在页面上，但是脚本可以通过该属性(通过 value 特性初始化)了解表单中复选框的作用。

例如下面的代码：

```
<input type="checkbox" id="memory" name="remember-me" value="yup">
<label for="memory">Remember me on this computer</label>
```

如果选中这个复选框并提交表单，浏览器就给服务器发送名称/值对："remember-me"和"yup"，标签文本"Remember me on this computer"显示在屏幕上，但不发送给服务器。

复选框对象的关键属性 checked 表示该框是否被选中，checked 属性是一个 Boolean 值，所以它可用在 if 或 if...else 条件表达式中：假如选中复选框，则其值为 true，没选中为 false。在程序清单 11-3 中，checked 属性的值用来确定显示哪个警告对话框。

程序清单 11-3　复选框对象的 checked 属性

HTML: jsb-11-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Checkbox Inspector</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-11-03.js"></script>
   </head>
   <body>
      <h1>Checkbox Inspector</h1>
      <form action="">
         <p>
            <input type="checkbox" id="checkThis" name="checkThis">
            <label for="checkThis">Check here</label>
            <input type="button" id="inspectIt" value="Inspect Box">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-11-03.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

var oCheckbox; // checkbox object (global)

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to crucial elements
```


更多资源请访问稀酷客(www.ckook.com)

```
        oCheckbox = document.getElementById('checkThis');
        var oButton = document.getElementById('inspectIt');

            // if they exist, apply event handler
            if (oCheckbox && oButton)
            {
               addEvent(oButton, 'click', inspectBox);
            }
     }
  }

  // report the checked state of the checkbox
  function inspectBox()
  {
     if (oCheckbox.checked)
     {
        alert("The box is checked.");
     }
     else
     {
        alert("The box is not checked at the moment.");
     }
  }
```

复选框通常用于设置参数，而不是触发动作。尽管复选框对象有 onclick 事件处理程序，但单击复选框不会执行很大的动作，比如跳到另一个页面。

11.2.4　单选输入对象

给脚本设置一组单选按钮对象时，只有给组中的每个按钮指定相同的 name 特性，才能让浏览器管理一组相关按钮的突出显示和非突出显示状态。一个表单里可以有多个单选按钮组，但是同一组里的每个成员必须具有相同的名称。

浏览器管理同名元素和不同名元素的方式是不同的。浏览器会维护同名对象的数组列表，数组的名称就是赋予组的名称。有些属性要应用于整个组，有些属性只应用于组中的单个按钮，所以必须通过数组下标来定位。例如，读取组的 length 属性，就可以确定组中的按钮数：

```
formObject.groupName.length
```

如果想通过类似于复选框的 checked 属性来查询某个按钮当前是否被选中，就可以根据该按钮在同名输入域集合中的位置来访问这个元素：

```
formObject.groupName[0].checked
```

程序清单 11-4 演示了单选按钮对象的几个方面，包括如何查询一组按钮，确定选中了哪个按钮，以及如何使用 value 特性和相关的属性。

该页面包括三个单选按钮和一个普通按钮，每个单选按钮的 value 特性都包含 Three Stooges 中一人的全名。用户单击普通按钮时，onclick 事件处理程序就调用 fullName()函数，该函数的第一个语句创建表单的一个快捷引用。接下来 for 循环搜索 stooges 单选按钮组的所有按钮，if

更多资源请访问稀酷客(www.ckook.com)

结构检查每个按钮的 checked 属性。选中一个按钮时，break 语句跳出 for 循环，使 i 循环计数器的值等于循环跳出时的数值，随后，警告对话框引用第 i 个按钮的 value 属性，使完全名称显示在警告框中。

程序清单 11-4 一组单选按钮对象的脚本

HTML: jsb-11-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Extracting Highlighted Radio Button</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-11-04.js"></script>
   </head>
   <body>
      <h1>Extracting Highlighted Radio Button</h1>
      <form action="stooges.php">
         <fieldset>
            <legend>Select your favorite Stooge:</legend>
            <p>
               <input type="radio" name="stooges" id="stooges-1"
               value="Moe Howard" checked>
               <label for="stooges-1">Moe</label>
            </p>
            <p>
               <input type="radio" name="stooges" id="stooges-2"
               value="Larry Fine">
               <label for="stooges-2">Larry</label>
            </p>
            <p>
               <input type="radio" name="stooges" id="stooges-3"
               value="Curly Howard">
               <label for="stooges-3">Curly</label>
            </p>
            <p>
               <input type="submit" id="Viewer" name="Viewer"
               value="View Full Name...">
            </p>
         </fieldset>
      </form>
   </body>
</html>
```

JavaScript: jsb-11-04.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);
```


更多资源请访问稀酷客(www.ckook.com)

```
var aStooges; // radio button array (global)

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to crucial elements
      var oButton = document.getElementById('Viewer');
      var aForms = document.forms;
         if (aForms) aStooges = aForms[0].stooges; // global variable

         // if they exist, apply event handler
         if (oButton && aStooges)
         {
            addEvent(oButton, 'click', showFullName);
         }
   }
}

// display the full name of the selected stooge
function showFullName()
{
   for (var i = 0; i < aStooges.length; i++)
   {
      if (aStooges[i].checked)
      {
         break;
      }
   }
   alert("You chose " + aStooges[i].value + ".");
}
```

11.2.5　select 对象

脚本编程最复杂的表单控件是 select 元素对象。从 DOM Level 0 表单对象层次结构图中可以看出(图 11-1)，select 对象其实是一个包含 option 对象数组的复合对象。此外，可在 HTML 中建立这个对象，把它显示为下拉列表或者滚动列表，滚动列表可以配置为允许进行多选。为了简化本节内容，本章重点讨论只允许单选的下拉列表。

有些属性可用于整个 select 对象，其他属性则仅用于 select 对象中的单个选项。如果想要确定用户选择了哪一项，且编码处理最广泛的浏览器，就必须使用 select 和 option 对象的属性。

select 对象最重要的属性是 selectedIndex，访问方法如下：

```
formObject.selectName.selectedIndex
```

这个值是当前选中项的下标号。和 JavaScript 中的多数下标计数机制一样，第一项(列表顶部的项)的下标号为 0。要访问当前选中项的属性，selectedIndex 值起着至关重要的作用。Option 项的两个重要属性是 text 和 value，访问方法如下：

更多资源请访问稀酷客(www.ckook.com)

```
formObject.selectName.options[n].text
formObject.selectName.options[n].value
```

text 属性是 select 对象的列表显示在屏幕上的字符串，这种信息一般不作为 form 对象的属性，因为在生成 select 对象的 HTML 中，文本定义为<option>标记的嵌套文本。但在<option>标记中，可以设置 value 特性，与前面的单选按钮一样，它能把一些隐藏的字符串信息与列表中的可见项联系起来。

从所有浏览器中，为了更有效地读取选中项的 value 和 text 属性，可以使用 select 对象的 selectedIndex 属性作为这个选中项的下标值。这种操作的引用比较长，需要花些时间来理解它。在下面的函数中，第一个语句创建了 select 对象的快捷引用，接着 select 对象的 selectedIndex 属性替换了同一对象的 options 数组的 index 值。

```
function inspect()
{
   var oList = document.getElementById('choices');
      if (oList)
      {
         var sChosenItemValue = oList.options[oList.selectedIndex].value;
      }
}
```

为了激活 select 对象，需要使用 onchange 事件处理程序。一旦用户在列表中选择了新选项，就运行与 onchange 事件相关的脚本。程序清单 11-5 显示了 select 对象的一个常见应用，它的文本项描述了出入 Web 站点的位置，这些位置的 URL 放在 value 特性中。用户在列表中进行选择时，onchange 事件处理程序触发一个脚本，提取第二个选项的 value 属性，并把该值赋给 location.href 对象属性，来进行导航。

当然，在 HTML 页面所在的文件夹中，这些新网站(网页)大多还不存在。选择一个位置后，页面就会尝试再次定位到该位置上，此时会显示“File not found”错误页面。注意地址栏中的页面，再单击 Back 按钮，返回示例表单。

表单包含一个提交按钮，这样在关闭 JavaScript 或用户代理中没有 JavaScript 时，页面仍可以把表单提交到一个服务器端程序上，来执行重定向。在 JavaScript 的控制下，实现这类导航不需要在页面上放置单独的 Go 按钮。这个例子使用 JavaScript 把提交按钮的显示样式设置为 none，隐藏了该按钮。但注意，一些人认为，只要用户做出了选择，提交表单的选择列表就应可用、可访问。问题是，表单提交并不是选择列表的一般功能，利用 JavaScript 添加这个功能会让一些用户很惊讶，在没有他们许可的情况下切换到一个新页面上。而且，列表的行为假定，用户在列表中做出的第一个选择是他们的真实意图，不允许出现意外选错的情况，其实每个人都可能偶尔选错选项。运行这个表单时关闭 JavaScript，看看会有什么不同。

程序清单 11-5　使用 select 对象导航

HTML: jsb-11-05.html

```
<!DOCTYPE html>
<html>
   <head>
```


更多资源请访问稀酷客(www.ckook.com)

```
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Select Navigation</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-11-05.js"></script>
   </head>
   <body>
      <h1>Select Navigation</h1>
      <form action="redirect.php" method="get">
         <p>
            <label for="urlList">Choose a place to go:</label>
            <select id="urlList" name="urlList">
               <option selected value="index.html">Home Page</option>
               <option value="store.html">Shop Our Store</option>
               <option value="policies.html">Shipping Policies</option>
               <option value="http://www.google.com">Search the Web</option>
            </select>

            <input type="submit" id="submit-button" value="Go">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-11-05.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

var oList; // select list (global)

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to crucial elements
      oList = document.getElementById('urlList');
      oButton = document.getElementById('submit-button');

         // if they exist...
         if (oList && oButton)
         {
            // make the list dynamic
            addEvent(oList, 'change', goThere);

            // make the submit button disappear when JavaScript is running
            oButton.style.display = 'none';
         }
   }
}

// direct the browser to the selected URL
```


更多资源请访问稀酷客(www.ckook.com)

```
function goThere()
{
   location.href = oList.options[oList.selectedIndex].value;
}
```

注意：

当前的浏览器利用 select 对象的 value 属性，提供选中项的 value 属性。这是一种简便合理的快捷方式，如果目标浏览器包括 IE、基于 Mozilla 的浏览器和 Safari，就可以安全地使用这种方式。

select 对象还有许多内容，包括在新浏览器中改变列表的内容等，详见第 37 章。

11.3　用 this 向函数传递元素

在本章前面的所有例子中，事件处理程序调用处理表单元素的函数时，表单或表单控件是通过全局变量明确引用的。还有一种非常有用的捷径，可以把表单或表单控件的信息直接传递给函数，而不必声明全局变量。

JavaScript 有一个关键字 this，它总是指向包含使用这个关键字的脚本的对象。例如，如果通过 click 事件把一个函数和一个按钮关联起来，则单击该按钮后，函数中的关键字 this 就指向按钮本身。使用 this 非常方便：它意味着全局变量更少，两个独立脚本使用同一个全局变量名混淆彼此逻辑的可能性更低，它还意味着可以给多个元素使用相同的函数。下面是一个简单的例子：

```
function identify()
{
   if (this.tagName)
   {
      alert('My tagName is ' + this.tagName);
   }
}
```

如果把 identify()函数应用于页面上的多个元素，再单击它们，每个元素都会报告其 tagName：P、LABEL、INPUT、FORM、BODY 等。

看看程序清单 11-6，其中有两个函数使用了 this 关键字：processData()函数关联到提交按钮上，verifySong()函数关联到歌名输入域上。

程序清单 11-6　用 this 把元素传送给函数

HTML：jsb-11-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Beatle Picker</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <script type="text/javascript" src="jsb-11-06.js"></script>
  </head>
  <body>
    <h1>Beatle Picker</h1>
    <form id="beatles-form" action="beatles.php" method="get">
      <p>
        <label>Choose your favorite Beatle:</label>
        <input type="radio" name="Beatles" id="radio1"
                   value="John Lennon" checked>
        <label for="radio1">John</label>
        <input type="radio" name="Beatles" id="radio2" value="Paul McCartney">
        <label for="radio2">Paul</label>
        <input type="radio" name="Beatles" id="radio3" value="George Harrison">
        <label for="radio3">George</label>
        <input type="radio" name="Beatles" id="radio4" value="Ringo Starr">
        <label for="radio4">Ringo</label>
      </p>
      <p>
        <label for="song">Enter the name of your favorite Beatles song:</label>
        <input type="text" id="song" name="song" value="Eleanor Rigby">
        <input type="submit" id="submit" value="Process Request...">
      </p>
    </form>
  </body>
</html>
```

JavaScript: jsb-11-06.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to crucial elements
      var oForm = document.getElementById('beatles-form');
      var oSong = document.getElementById('song');
      var oButtonSubmit = document.getElementById('submit');

         // if they all exist...
         if (oForm && oSong && oButtonSubmit)
         {
            // apply behavior to input field & button
            addEvent(oSong, 'change', verifySong);
            addEvent(oButtonSubmit, 'click', processData);
         }
   }
}
```


更多资源请访问稀酷客(www.ckook.com)

```
// verify all input & suppress form submission
function processData(evt)
{
      // consolidate event handling
      if (!evt) evt = window.event;

   // point to the activated control's form ancestor
   var oForm = this.form;

   // see which radio button was selected
   for (var i = 0; i < oForm.Beatles.length; i++)
   {
      if (oForm.Beatles[i].checked)
      {
         break;
      }
   }

   // assign values to variables for convenience
   var sBeatle = oForm.Beatles[i].value;
   var sSong = oForm.song.value;

   // this is where a data lookup would go...
   alert("Checking whether " + sSong + " features " + sBeatle + "...");

      // cancel form submission
      if (evt.preventDefault) evt.preventDefault();
   return false;
}

// verify the song name when it's changed
function verifySong()
{
   // get the input song
   // ('this' is the object whose event handler called this function)
   var sSong = this.value;

   // this is where a data lookup would go...
   alert("Checking whether " + sSong + " is a Beatles tune...");
}
```

verifySong()函数想访问输入到歌曲输入域中的值，所以它在下面的语句中使用了 this:

```
var sSong = this.value;
```

换言之，就是获取当前对象(输入域)的值，并赋给变量 sSong。this 用作下述语句的快捷替代:

```
var oInput = document.getElementById('song');
var sSong = oInput.value;
```

processData()函数需要引用几个表单元素，所以它先利用如下事实：每个表单元素都使用 form 属性指向它所属的表单：

```
var oForm = this.form;
```


更多资源请访问稀酷客(www.ckook.com)

无论 this 指向哪个输入控件，this.form 都指向父表单。然后就可以使用 oForm.Beatles 和 oForm.song 等对象引用，按名称指向表单中的各个控件了。

如果在浏览器中运行此脚本时不明白其行为，以下是对其程序逻辑的说明。在文本框中输入新歌名，再按下 Tab 或单击其他地方以退出该输入域时，onchange 事件处理程序就调用 verifySong() 函数，该函数显示一个警告信息，指出它正在检查该歌曲(本例并不对照数据库验证歌曲)。

单击 Process Request 按钮时，它的 onclick 事件处理程序就调用 processData()函数，它会进行检查，看看输入的歌曲是否是在上述单选按钮中选择的 Beatle。

下面尝试一下这两个操作。在输入域中输入一个新歌名，立即单击 Process Request 按钮。此时仅看到一个警告信息——正在检查该歌曲的 verifySong()信息。为什么看不到两个警告信息？因为按钮的 click 动作被文本框的 onchange 事件处理程序打断。换句话说，按钮并未真正被单击，因为 onchange 警告对话框先显示。这就是为什么必须第二次单击按钮才能验证歌曲/Beatle 的原因。如果没有更改域中的文本，按钮的单击操作将不被中断，也会执行 processData()验证。

注意：

IE 和其他浏览器在事件分配和事件处理的处理方式上的差异需要一些说明，这超出了本教程的范围。不过第 25 章和第 32 章将予以介绍。

11.4　提交和预验证表单

在普通的 HTML 文档中，单击 submit 按钮就可以提交表单；接着浏览器收集用户在输入控件中输入或选择的值，把它们发送给指定的 URI。JavaScript 允许我们解释该提交请求，验证输入或者执行其他需要的操作，再继续提交表单或取消提交操作，以便指导用户改进其输入。

在表单的 onsubmit 事件处理调用的函数中，可以执行这类表单提交之前最后的数据验证或其他脚本操作(例如，根据用户的选择来更改表单的 action 属性)。配书光盘中的第 46 章将详细介绍验证例程，这里仅介绍 onsubmit 事件处理程序的工作原理。

考虑到不同浏览器处理事件的方式不同，这里需要先使用两个技术来取消事件，它们对应于 addEvent()函数应用于事件处理程序的两个技术：

```
// add an event to an element
function addEvent(elem, evtType, func)
{
   // first try the W3C DOM method
   if (elem.addEventListener)
   {
      elem.addEventListener(evtType, func, false);
   }
   // otherwise use the 'traditional' technique
   else
   {
      elem["on" + evtType] = func;
   }
}
```

更多资源请访问稀酷客(www.ckook.com)

将这种方式与程序清单 11-7 中的脚本取消提交事件的方式进行比较：

```
function checkForm(evt)
{
      // consolidate event handling
      if (!evt) evt = window.event;
   ...
   // cancel form submission
      // W3C DOM method (hide from IE)
      if (evt.preventDefault) evt.preventDefault();
   // IE method
   return false;
   ...
}
```

对于符合 W3C DOM 标准的浏览器，用 addEventListener()添加事件，用 preventDefault()取消表单提交。尝试调用一个浏览器认为不存在的 DOM 方法会导致一个运行时错误，所以为了保护功能较弱的老式浏览器，下面先进行测试，以确保事件对象存在对应的方法。

对于功能较弱的浏览器，主要是 IE，通过设置事件属性(这里是 onsubmit)把事件添加到对象中，从事件处理程序调用的函数中返回 false 值，就可以取消表单提交操作。

本书后面会说明原因，但这里仅使用这两个技术来处理事件。

程序清单 11-7 显示的页面具有简单的验证例程，这个例程在允许提交之前，确保所有的域都有值。HTML 标记了一个带有 4 个输入域和一个提交按钮的表单。注意 form 元素的 action 属性是假想的服务器端程序 validate.php。如果这个页面发送给不运行 JavaScript 的用户代理，表单就把输入值提交给服务器端程序进行验证。基本表单验证总是在服务器上进行，我们在客户端的 JavaScript 上重复这个验证过程，让用户能更快地响应它们的动作(有时，这表示脚本中是否存在拼写错误：如果提交这个表单时，表单上的某个输入域是空的，且浏览器窗口尝试执行 validate.php，就说明输入 HTML 或 JavaScript 代码时可能有拼写错误，因为脚本不会阻止表单的提交)。

页面加载完毕后，脚本就把 checkForm()函数与表单的 onsubmit 事件关联起来。注意，这里没有把 onclick 行为应用于 submit 按钮，而是让按钮正常工作，并在表单对象上拦截提交过程。

单击 submit 时，下一个正常步骤是浏览器提交表单，但它首先要执行 onsubmit 事件处理程序，调用 checkForm()函数。这个函数的主要工作是遍历所有的表单控件，查看是否有空白的输入域。如果找到一个空输入域，就显示错误信息，并取消表单提交操作。

在循环中，if 语句执行两个测试。第一个测试是确保检查 type 属性为 text 的表单控件(这样就不必检查其他控件，如按钮)；接下来，它检查文本域的值是否为空。& & 运算符(称为布尔 AND 运算符)表示，只有该运算符两边的条件都为 true，括号内的整个条件表达式才为 true。如果有一个子测试失败，整个条件就失败了。

程序清单 11-7　表单提交之前的最后一次检查

HTML: jsb-11-07.html

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Form Field Validator</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-11-07.js"></script>
   </head>
   <body>
      <h1>Form Field Validator</h1>
      <form id="theForm" action="validate.php" method="get">
         <p>Please enter all requested information:</p>
         <p>
            <label for="firstName">First Name:</label>
            <input type="text" id="firstName" name="firstName">
         </p>
         <p>
            <label for="lastName">Last Name:</label>
            <input type="text" id="lastName" name="lastName">
         </p>
         <p>
            <label for="age">Age:</label>
            <input type="text" id="age" name="age">
         </p>
         <p>
            <label for="favoriteColor">Favorite Color:</label>
            <input type="text" id="favoriteColor" name="favoriteColor">
         </p>
         <p>
            <input type="submit" id="submit">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-11-07.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to the form
      var oForm = document.getElementById('theForm');

         // if it exists, apply the behavior
         if (oForm)
         {
            addEvent(oForm, 'submit', checkForm);
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
        }
    }

    // when the form is submitted,
    // check to make sure all input fields have been filled in
    function checkForm(evt)
    {
        // consolidate event handling
        if (!evt) evt = window.event;

      // 'this' is the current object, in this case the form
      for (var i = 0; i < this.elements.length; i++)
      {
        if (this.elements[i].type == "text" && this.elements[i].value == "")
        {
          alert("Fill out ALL fields.");

          // cancel form submission
            // W3C DOM method (hide from IE)
            if (evt.preventDefault) evt.preventDefault();
          // IE method
          return false;
        }
      }
      // allow form submission
      return true;
    }
```

submit()的说明

提交表单的等效脚本编写方法是表单对象的 submit()方法。在该语句中，只需要引用表单和这个方法：

```
formObject.submit();
```

submit()方法和 onsubmit 事件处理程序的一个古怪行为需要解释一下。用户可能以为，submit()方法应当与单击 submit 按钮完全等效，但不是这样。submit()方法根本不会触发表单的 submit 事件。如果要通过 submit()方法执行所提交表单的有效性检验，可在调用 submit()方法的脚本函数中执行有效性检验。

有关表单和表单控件的基础知识就讲这些。第 12 章将暂且离开 HTML 这个话题，介绍更高级的 JavaScript 核心语言内容：String、Math 和 Date 对象。

11.5 习题

1. 用 this 关键字替代全局变量，重写程序清单 11-2、程序清单 11-3、程序清单 11-4 和程序清单 11-5。

更多资源请访问稀酷客(www.ckook.com)

2. 对于下列表单(假设页面上只有一个表单)，在所有现代的可编程浏览器中，写出把文本输入域引用为对象的至少 10 种方式。

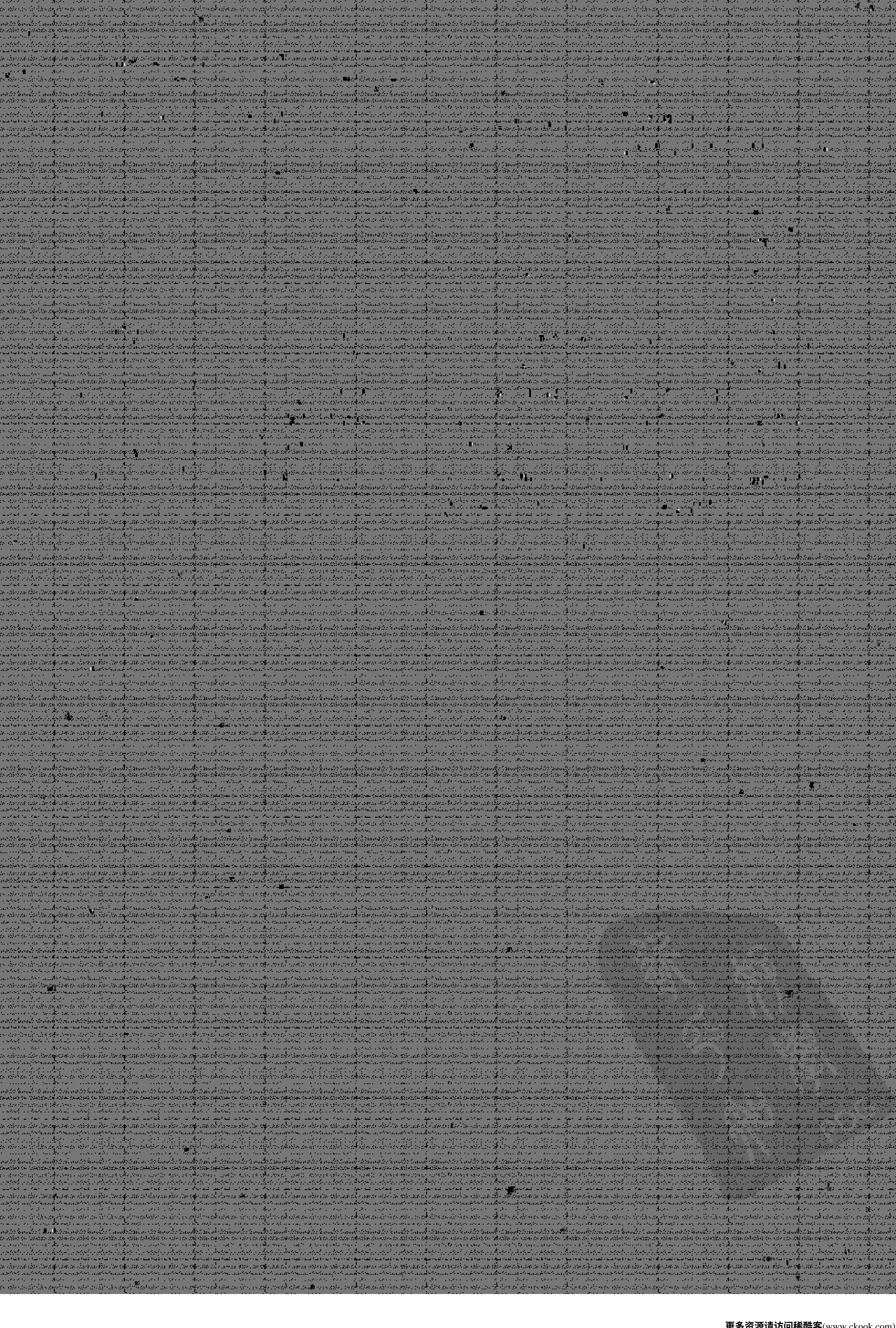
```
<form name="subscription" action="cgi-bin/maillist.pl" method="post">
<p>
   <input type="text" id="email" name="email">
   <input type="submit">
</p>
</form>
```

3. 编写一个函数，在警告对话框中显示一个输入域的值，再添加一些脚本，在修改该输入域时调用该函数。

4. 一个文档包含两个表单 specifications 和 accessories，在 accessories 表单中有一个域 accl。至少写出两个不同的语句，将该域的内容设置为 Leather Carrying Case。

5. 创建一个页面，该页面包含一个 select 对象，用于改变当前页面的背景色。给这个对象设置 document.body.style.backgroundColor 属性，作为选项的三个值为 red、yellow 和 green。在 select 列表中，颜色标签应该显示为 Stop、Caution 和 Go。

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第12章

String、Math 和 Date 对象

到目前为止，本书的大部分章节重点关注的对象都属于文档对象模型(DOM)。但是如第 2 章所述，在 DOM 和 JavaScript 语言之间存在着一条明显的界线，JavaScript 语言有一些独立于 DOM 的对象。这些对象的设计目标是：假如供应商想用 JavaScript 作为编程语言来建立不同类型的产品，该语言仍会使用这些核心对象去处理文本、高级数学(高于简单的算术)和日期。ECMA-262 标准正式描述了这些对象的规范。

本章包含哪些内容？
使用常用的字符串方法来修改字符串
使用Math对象的场合和方式
使用 Date 对象

12.1 核心语言对象

万事开头难，新手常常觉得编程非常困难，甚至是有经验的程序员，如果以前没有接触过面向对象语言，也很难理解对象，更不用说那些表示实际上不存在的“东西”的对象。例如，很容易理解页面上的按钮是一个对象，按钮有几个具有实际意义的物理属性。但是字符串呢？如本章所述，在类似 JavaScript 这样基于对象的环境中，从 Boolean 值到日期的所有东西都是对象。每个对象或许有一个或多个属性，来帮助定义其内容；每个对象可能还有一些相关联的方法，来定义对象能做什么或能在对象上做什么。

JavaScript 中的所有非 DOM 对象都称为核心语言对象，附录 A 包含相关的补充内容，本章只讨论 String、Math 和 Date 对象。

12.2 String 对象

前面的章节多次使用过 String 对象。字符串是放在一对引号中的文本，引号可以是双引号或单引号。这样就允许字符串互相嵌套。在下面的例子中，alert()方法需要一个带引号的字符串作为参数：

更多资源请访问稀酷客(www.ckook.com)

```
alert('You cannot lose. ');
```

如果带引号的表达式包含一个撇号，就应该将外面的引号改为双引号：

```
alert("You can't lose.");
```

但问题没这么简单，因为如果在带引号的字符串中同时出现了撇号和引号，就需要使用转义序列，详见第 15 章。

JavaScript 不限制字符串内的字符数。然而，大多数旧式浏览器将脚本语句的长度限制在 255 字符以内。如果脚本包含一个用于显示在页面内容的长字符串，有时会超过这个限制，这就需要使用后面提到的方法，把这些代码分割成小块。

可以采用两种方法将字符串值赋给变量。最简单的就是基本赋值语句：

```
var myString = "Howdy";
```

除了少数例外，这种方法很有效。也可以使用更正规的语法创建字符串对象，这种方法要使用 new 关键字和构造函数(即，它“构造”一个新对象)：

```
var myString = new String("Howdy");
```

字符串变量的初始化不管使用什么方法，这个赋值的变量都能响应所有 String 对象方法。

12.2.1 连接字符串

把两个字符串合并成一个字符串称为连接字符串，这是第 8 章学习的术语。字符串连接需要使用两个 JavaScript 操作符之一。第 3 章给出的第一个脚本曾使用加号(+)把多个字符串连接为一个字符串：

```
var today = new Date();
var msg = "This is JavaScript saying it's now " + today.
    toLocaleString();
```

另一个操作符+=与+操作符同样有用，但更适于在脚本环境中使用。在简单的变量中组合大字符串时，这个操作符很有用。这种字符串可能很长，必须将构造过程分解成多个语句；或者，把连接过程分解到几个语句中，以便人们理解。这些字符串可能组合了字符串字面量(引号内的字符串)和变量值。还有一种笨方法(在 JavaScript 中是完全可行的)是使用+运算符给现有的块追加更多的文本：

```
var msg = "Four score";
msg = msg + " and seven";
msg = msg + " years ago,";
```

所谓的“加值”操作符+=提供了一种捷径，这个操作符的符号由加号和等号组合而成，其含义是“将右边的内容加到左边内容的后面”。因此，上面的语句可简写为：

```
var msg = "Four score";
msg += " and seven";
msg += " years ago,";
```


更多资源请访问稀酷客(www.ckook.com)

如有必要，可以组合操作符：

```
var msg = "Four score";
msg += " and seven" + " years ago";
```

12.2.2 字符串方法

在所有 JavaScript 核心对象中，String 对象的方法最多，许多方法都用来帮助脚本提取字符串的一部分，但一个很少用、目前在 CSS 中已废弃的方法是使用几个面向样式的标记封装字符串(与字体大小、样式等等价的脚本标记)。

在字符串方法中，被操作的字符串应该是引用的一部分，后面跟着方法名：

```
myString.methodName();
```

所有字符串方法都返回某种类型的值，该返回值大多是方法调用中被引用的字符串对象的转换版本，但原字符串保持不变。为了得到这个修改后的版本，需要把方法的结果赋给一个变量：

```
var result = myString.methodName();
```

下面介绍几个重要的字符串方法，它们可用于所有浏览器产品和版本。

1. 改变字符串的大小写

可以采用两种方法将字符串转换为大写或小写：

```
var result = string.toUpperCase();
var result = string.toLowerCase();
```

如果希望这些方法起作用，就一定要注意方法名中每个字母的大小写。脚本需要比较大小写不一致的字符串时，使用这两个方法非常方便(例如，比较查询表中的字符串和用户输入的字符串)。因为这些方法并不改变表达式中的原始字符串，所以只需比较这些方法返回的结果：

```
var foundMatch = false;
   if (stringA.toUpperCase() == stringB.toUpperCase())
   {
      foundMatch = true;
   }
```

2. 字符串搜索

可以使用 String.indexof()方法来确定一个字符串是否包含在另一个字符串中。例如，假定在网站上建立一个联系表单，用于把访问者的信息发送给公关部门，但希望仅给监察员复制包含“complaint”的信息，这个单词可能隐藏在长信息字符串的任何位置，但此时，并不需要知道这个单词在什么地方，只需知道它是否在长字符串中。

string.indexof()方法返回一个数值，表示短字符串在长字符串中的起始下标值(从 0 开始)。注意，假如短字符串不在长字符串中，其返回值为-1。所以，为了判定短字符串是否在长字符串中，只需测试该方法的返回值是否为-1。

这个方法涉及两个字符串：一个短字符串和一个长字符串。长字符串位于方法名的左边；

更多资源请访问稀酷客(www.ckook.com)

而短字符串是 indexof()方法的参数。为了演示方法的运行情况，下面的代码段判断输入信息是否包含 complaint：

```
var sMsg = '';
var oInput = document.getElementById('message');
   if (oInput)
   {
      sMsg = oInput.value;
   }
var isComplaint = false;
   if (sMsg.indexOf("complaint") != -1)
   {
      isComplaint = true;
   }
```

if 结构条件中的操作符(! =)是一个不等操作符，表示“不等于”。

3. 提取字符和子字符串的副本

可以使用 charAt()方法从字符串的指定位置提取单个字符。这个方法的参数是要提取的字符的下标(从 0 开始)，这里的提取不是删除，而是获取字符的一个快照，原始字符串保持不变。

例如，假设主窗口中的一个脚本可以检测变量 stringA，另一个窗口则显示不同公司的建筑图。该窗口显示 C 建筑的地图时，stringA 变量包含“Building C”。因为建筑物名称中的字母 C 总是位于这个字符串的第 10 个位置(如果从 0 开始计数，它就是 9)，所以脚本可以检查这个字符，来判定另一个窗口当前载入了哪个建筑物的地图。

```
var stringA = "Building C";
var bldgLetter = stringA.charAt(9);
   // result: bldgLetter = "C"
```

如果知道要提取的子字符串的起始位置，就可以使用另外两个类似的方法 string.substr()和 string.substring()提取连续几个字符。这两个方法的区别是 string.substr()需要指定子字符串的长度，而 string.substring()希望知道子字符串的终止位置。

```
string.substr(startingPosition [, length])
string.substring(startingPosition [, endingPosition])
```

如果没有提供第二个参数(长度或终止位置)，这两个方法都会提取直到原始字符串末尾的所有字符。被提取的字符串在引用中位于方法名的左边，起始和终止位置参数是下标值(从 0 开始)：

```
var stringA = "banana daiquiri";

var excerpt = stringA.substr(2,4);      // result: "nana"
var excerpt = stringA.substr(2);        // result: "nana daiquiri"

var excerpt = stringA.substring(2,6);   // result: "nana"
var excerpt = stringA.substring(2);     // result: "nana daiquiri"
```

相对于其他脚本语言来说，JavaScript 中的字符串操作是相当麻烦的，它没有更高一层的字、

更多资源请访问稀酷客(www.ckook.com)

句子或段落的概念。因此，有时需要使用字符串方法，来完成看起来简单的任务。但假如要编写使用大量嵌套结构的表达式，则可以利用表达式求值方法来完成。

例如，假定要从一个多字表达式中提取除第一个字之外的所有字符：

```
var stringA = "The Painted Bird";
var firstSpace = stringA.indexOf(" ");
var excerpt = stringA.substring(firstSpace + 1);
   // result: excerpt = "Painted Bird"
```

假定第一个字的长度可以任意，则第二个语句利用 string.indexof()方法来查找第一个空格字符的位置，并将其值加 1 作为 string.substring()方法的起始下标值。忽略第二个 length 参数，以提取直到 stringA 末尾的所有字符。

也许读者不希望重复创建这样的语句，因此第 23 章会介绍如何创建字符串函数库，以便在需要这些字符串处理功能的脚本中重用它们，很多强大的字符串匹配工具都用正则表达式的形式内置在目前的浏览器中(参阅第 15 章和第 45 章)。

12.3　Math 对象

JavaScript 提供了丰富的数学处理工具——远远多于大多数没有计算机和数学背景知识的脚本开发人员在日常生活中使用的知识。但每种编程语言都需要这些功能，以使编程人员让窗口在屏幕上更加生动。

Math 对象包含了所有这些能力，这个对象不同于 JavaScript 的大多数其他对象，因为不能创建这个对象的副本，脚本总是使用 Math 对象的属性和方法(在技术上，Math 对象实际上存在于每个窗口或框架中，但对脚本没有影响)。这种固定的对象叫做静态对象。Math 对象(开头是一个大写 M)是这些属性和方法引用的一部分，Math 对象的属性是常数，比如 π 和 2 的平方根：

```
var piValue = Math.PI;
var rootOfTwo = Math.SQRT2;
```

Math 对象方法包括各种三角函数，它的其他一些函数处理已在脚本中定义的数值。例如，可以利用 Math 对象的方法确定两个数值中的较大者：

```
var larger = Math.max(value1, value2);
```

也可以计算出一个数的 10 次方：

```
var result = Math.pow(value1, 10);
```

更常用的方法是把一个值圆整为最近的整数值：

```
var result = Math.round(value1);
```

随机数也常常使用 Math 对象的方法来生成。Math.random()方法返回 0 和 1 之间的浮点数。假如设计一个扑克牌游戏的脚本，就需要 1～52 之间的随机数；对于骰子游戏，每个骰子的范围是 1～6。为了生成 0 和任意上限之间的随机整数，可使用下列公式：

更多资源请访问稀酷客(www.ckook.com)

```
Math.floor(Math.random() * (n + 1))
```

其中 n 是上限(Math.floor 返回浮点数的整数部分)。为了生成 1 和任意数之间的随机数，可以使用下列公式：

```
Math.floor(Math.random() * n) + 1
```

其中 n 是范围的上限。对骰子游戏，每个骰子的公式是：

```
newDieValue = Math.floor(Math.random() * 6) + 1;
```

为了看到这个结果，请在 Evaluator Jr.的顶部文本框中输入上述语句的右侧部分，并重复单击 Evaluate 按钮。

12.4 Date 对象

在 JavaScript 中，使用日期处理稍微复杂的任务是很困难的。其原因是，假设访问者的 PC 内部时钟和控制面板的设置是正确的，则日期和时间是按照格林威治标准时间(Greenwich Mean Time，GMT)计量的。由于其复杂性，第 17 章将详细讲述它，本章这一部分只介绍 JavaScript 中有关 Date 对象的基础知识。

脚本浏览器包含一个全局 Date 对象(事实上，每个窗口都有一个 Date 对象)，这个对象总是存在，可以随时调用。Date 对象也是静态的。处理日期时，比如显示当天的日期，需要调用 Date 对象的构造函数，获取一个与当天时间和日期关联的 Date 对象实例。例如，调用这个构造函数的无参数版本：

```
var today = new Date();
```

Date 对象取得了 PC 内部时钟的一个快照，并返回此时的 Date 对象。注意静态 Date 对象和 Date 对象实例的差别，后者包含一个实际的日期值。变量 today 包含的不是一个滴答时钟，而是一个值，可以根据脚本的需要检查、分解和重组这个值。

Date 对象实例的内部值是时间，以毫秒为单位，从 GMT 时区(它是所有时间转换的世界标准参考点)的 1970 年 1 月 1 日 0 时开始计算，所以 Date 对象包含日期和时间信息。

在 Date 对象的构造函数中，将过去或将来的某个日期和时间作为 Date 对象构造函数的参数，就可以得到该时刻的 Date 对象快照：

```
var someDate = new Date("Month dd, yyyy hh:mm:ss");
var someDate = new Date("Month dd, yyyy");
var someDate = new Date(yyyy,mm,dd,hh,mm,ss);
var someDate = new Date(yyyy,mm,dd);
var someDate = new Date(GMT milliseconds from 1/1/1970);
```

如果要查看原始 Date 对象的内容，JavaScript 就将这个值转换为本地时区字符串，这是 PC 中控制面板设置指定的。为了看到运行情况，在 Evaluator Jr.的顶部文本框中输入下列内容：

```
new Date();
```


更多资源请访问稀酷客(www.ckook.com)

PC 时钟一边计算一边提供当前的时间和日期(JavaScript 仍然保留了 GMT 时区的 Date 对象毫秒数)，然而，通过 Date 对象实例的一系列方法，可提取 Date 对象的组成部分。表 12-1 是这些值的属性和信息简表。

表 12-1　一些 Date 对象方法

方　法	取值范围	说　明
dateobj.getTime()	0～…	格林尼治标准时间 1970 年 1 月 1 日 0 时后的毫秒数
dateobj.getYear()	70～…	指定的年份减去 1900；2000 年后则是 4 位数的年份
dateobj.getFullYear()	1970～…	4 位数的年份(Y2K 兼容)：版本 4 以上的浏览器
dateobj.getMonth()	0～11	年中的月份(1 月为 0)
dateobj.getDate()	1～31	月中的日期
dateobj.getDay()	0～6	星期几(星期日为 0)
dateobj.getHours()	0～23	24 小时制的小时
dateobj.getMinutes()	0～59	特定小时内的分钟
dateobj.getSeconds()	0～59	特定分钟内的秒
dateobj.setTime(*val*)	0～…	格林尼治标准时间 1970 年 1 月 1 日 0 时后的毫秒数
dateobj.setYear(*val*)	70～…	指定的年份减去 1900；2000 年后则是 4 位数的年份
dateobj.setMonth(*val*)	0～11	年中的月份(1 月为 0)
dateobj.setDate(*val*)	1～31	月中的日期
dateObj.setDay(*val*)	0～ 6	星期几(星期日为 0)
dateobj.setHours(*val*)	0～23	24 小时制的小时
dateobj.setMinutes(*val*)	0～59	特定小时内的分钟
dateObj.setSeconds(*val*)	0～59	特定分钟内的秒

警告：

注意以 0 开始计数的值，特别是月份。getMonth()和 setMonth()方法值是基于 0 的，因此这些数值比我们习惯使用的月份值少 1(例如，1 月为 0，12 月为 11)。

设置 Date 对象值的方法有一个不同点，这些方法不返回新值，而是直接修改调用方法的 Date 对象实例的值。

12.5　日期计算

对日期执行计算常常要求处理 Date 对象的毫秒值，这是加、减和比较日期值的可靠方法。为了演示 Date 对象的机制，程序清单 12-1 显示了当前日期和时间，并根据当前时间和日期值计算出 7 天后的日期和时间。

更多资源请访问稀酷客(www.ckook.com)

程序清单 12-1　Date 对象计算

HTML: jsb-12-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date Calculation</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-12-01.js"></script>
   </head>
   <body>
      <h1>Date Calculation</h1>
      <form id="date-form" action="date-calc.php">
         <p>
            <label for="today">Today is:</label>
            <input type="text" id="today" name="today" size="80">
         </p>
         <p>
            <label for="nextWeek">Next week will be:</label>
            <input type="text" id="nextWeek" name="nextWeek" size="80">
         </p>
         <p>
            <input type="submit" id="submit">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-12-01.js

```
// run script when the page has loaded
addEvent(window, 'load', calcDates);

// calculate dates and plug into document
function calcDates()
{
   var dTodaysDate = new Date();

   // plug in today
   var oToday = document.getElementById('today');
      if (oToday)
      {
         oToday.value = dTodaysDate;
      }

   // plug in next week
   var oNextWeek = document.getElementById('nextWeek');
      if (oNextWeek)
      {
         // today in milliseconds
```


更多资源请访问稀酷客(www.ckook.com)

```
            var msToday = dTodaysDate.getTime();

            // one week = 1000 milliseconds * 60 seconds * 60 minutes
            //             * 24 hours * 7 days
            var msOneWeek = 1000 * 60 * 60 * 24 * 7;

            // next week in milliseconds
            var msNextWeek = msToday + msOneWeek;

            // next week as date object
            var dAWeekFromNow = new Date(msNextWeek);

            // plug in string value
            oNextWeek.value = dAWeekFromNow;
        }
    }
```

脚本首先创建一个新的 Date 对象，并赋予 dTodaysDate 变量。接着把输入域 today 的值设置为等于 dTodaysDate，以显示当前的日期和时间。尽管我们希望这会把 Date 对象的一个副本插入输入域，但实际上脚本使用 Date 对象的 toString()方法输出一个字符串，例如：

```
Sun Dec 05 2010 16:47:20 GMT-0800 (Pacific Standard Time)
```

然后显示一星期之后的日期和时间。为了处理毫秒值，首先把当前日期和时间的毫秒值赋予 msToday 变量。为了给这个毫秒值加上一个星期，需要知道一星期是多少毫秒：1 秒=1000 毫秒，60 秒=1 分钟，60 分钟=1 小时，24 小时=1 天，7 天=1 星期。把这些数值相乘，存储在 msOneWeek 变量中。把 1 星期的毫秒值加到当前日期和时间的毫秒值上，再使用该结果创建一个新的 Date 对象，存储在 dAWeekFromNow 变量中。显示得到的日期和时间与以前一样——把新 Date 对象赋予输入域的 value 特性。

在 Date 对象上加减时间间隔，可以使用不需要进行毫秒转换的快捷方式。组合 Date 对象的 set 和 get 方法，可让 Date 对象计算出结果。例如，在程序清单 12-1 中，可以用如下语句替换“下一个星期”的逻辑代码：

```
dTodaysDate.setDate(dTodaysDate.getDate() + 7);
oNextWeek.value = dTodaysDate;
```

由于 JavaScript 在内部将日期和时间记录为毫秒，因此即使新的日期是在下一个月中，也会显示精确的日期。JavaScript 自动计算一个月有多少天以及闰年的情况。

假如在页面中对日期进行脚本编程，会遇到许多奇怪和复杂的现象，然而，如后面的章节所述，它们是值得研究的。

12.6　习题

1. 创建一个 Web 页面，它包含一个表单域(用于输入用户的电子邮件地址)和一个 Submit 按钮。在随附的 JavaScript 脚本中，编写一个预提交验证例程，它在允许提交表单之前，验证文本域是否包含所有电子邮件地址都使用的@符号。

更多资源请访问稀酷客(www.ckook.com)

2. 给定“Internet Explorer”字符串，在空格处填入 string.substring()方法的参数，以产生每个方法调用右侧显示的结果：

```
var myString = "Internet Explorer";
myString.substring(___,___)       // result = "Int"
myString.substring(___,___)       // result = "plorer"
myString.substring(___,___)       // result = "net Exp"
```

3. 下面的代码提取字符串中前两个字之后的部分：

```
var stringA = "The Painted Bird";
var firstSpace = stringA.indexOf(" ");
var excerpt = stringA.substring(firstSpace + 1);
```

如果 stringA 不包含空格，这个代码段会出现什么情况？解决这个问题，再运行脚本，看看自己是否正确。

4. 在下列程序清单中填入函数的其余部分，使函数遍历输入域的每个字符，并计算域中有多少个字母 e(大小写均可，提示：函数缺少的是 for 循环)。

```
var inputString = 'Elephantine';
var count = 0;

MISSING CODE

var msg = "The number of e's in '" + this.mainstring.value + "'";
msg += " is " + count;
alert(msg);
```

5. 创建一个有两个域和一个按钮的页面。单击该按钮，会触发一个函数，生成 1～6 之间的两个随机数，并把每个数放在一个域中(可以使用这个页面替换棋盘游戏中的滚动骰子部分)。

6. 创建一个脚本，显示当天到下一个圣诞节之间的天数。

更多资源请访问稀酷客(www.ckook.com)

第13章

编写框架和多窗口脚本

本章包含哪些内容？

- 浏览器窗口中框架之间的关系
- 访问其他框架中的对象和值
- 控制多框架间的导航
- 不同窗口间的通信技巧

在客户端的一些应用程序上，JavaScript 可使一个框架或窗口中的动作影响其他窗口和框架，这正是 JavaScript 的一个富有吸引力的方面。本章将目前所学的对象引用知识扩展到多框架和窗口领域。

13.1 框架：父框架和子框架

在前面的顶级层次结构图中(参见图 6-2)，window 对象在顶部，它还有几个同义词，它们在特殊情况下代表 window 对象。例如，第 10 章提到，引用包含脚本文档的同一个窗口时，self 与 window 同义。本章则学习其他三个表示窗口对象的引用：frame、top 和 parent。

把一个普通的 HTML 文档载入浏览器，会在浏览器中创建一个模型，刚开始时，这个模型只有 window 对象和它包含的文档。这个层次模型相当简单，如图 13-1 所示，window 或 self 引用(或 document 引用，因为假定处于当前窗口)就从这里开始。

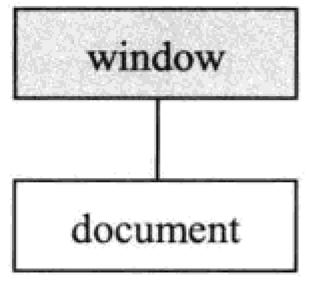


图 13-1　单框架窗口和文档层次结构

是否使用框架?

Frames 和 iFrames 都是有效的 HTML，其用途都是完成某些任务。然而，基于框架的网站存在严重的可用性和可访问性问题，因此用于一般的网站开发不大理想。

框架集的根本问题是，框架页面的任何给定组合都不能直接寻址，即如果导航一个框架集，希望显示框架中特定内容的组合，浏览器地址栏中的 URL 将仍是父框架集文档的 URL，而不

更多资源请访问稀酷客(www.ckook.com)

是实际所查看内容的 URL。由于缺乏框架页面内容的唯一地址，所以搜索引擎无法索引内容，其他人也不能给框架内容发布链接，在计算机上给框架内容加上书签，或者在电子邮件中共享它们。移动设备和辅助技术(如放大镜和屏幕阅读器)的用户也会在框架集中失去方向，或者完全无法访问框架集。

Web 可用性专家 Jakob Nielsen 在 1996 年写了“Why Frames Suck (Most of the Time)”文章。多年来，情况仍没有什么改进。

避免这些问题的替代方法是使用服务器端脚本来组合多个源页面，使用 CSS 属性 overflow: auto 创建滚动内容区，使用 position: fixed 建立其他页面滚动时不移动的部分。

本书假定用户考虑了使用框架的优缺点，而且有一个需要使用框架来解决的 Web 开发问题，或者仅仅对此感到好奇。用户应知道如何在需要时用 JavaScript 处理它们，它们有助于练习 DOM 的导航，但在大多数情况下不建议使用框架。

把框架集文档载入浏览器，浏览器就开始建立一个稍微不同的层次模型，该模型的具体结构完全依赖于在框架集文档中定义的框架集结构。考虑下面的框架集定义：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
        "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Title of all pages in this frameset</title>
   </head>
   <frameset cols="50%,50%">
      <frame name="leftFrame" src="somedoc1.html" title="Frame 1">
      <frame name="rightFrame" src="somedoc2.html" title="Frame 2">
      <noframes>
         ...
      </noframes>
   </frameset>
</html>
```

这些标记把浏览器窗口分为两个并排的框架，每个框架载入不同的文档。该模型只考虑结构(父元素和子元素)和布局(框架的相对大小，以及它们是放在行中还是列中)。框架标记不是向后兼容的，在不支持框架的浏览器中什么都不显示，所以我们提供了一个无框架版本，它可能是实际内容或无框架页面的超链接。

框架集在集合中建立了框架之间的关系。在面向对象编程的术语中，该框架集文档加载到父窗口中，在父窗口文档中定义的框架是子框架。图 13-2 显示了有两个框架的层次模型，它揭示了框架集和框架之间的微妙关系。

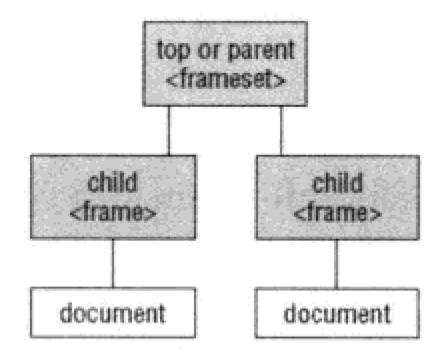


图 13-2　双框架窗口和文档的层次结构

一开始常常很难把框架集看作层次结构中的 window 对象。毕竟，除了在浏览器的 Location/Address 域中显示的 URL 外，在浏览器中看不到任何框架集信息，但这个 window 对象在对象模型中确实存在。另外，结构图中的框

更多资源请访问稀酷客(www.ckook.com)

架集父窗口没有 document 对象，这有点奇怪，因为窗口显然需要一个 HTML 文件来包含框架集说明。实际上，父窗口有一个关联的 document 对象，但结构图略去了它，以更好地描述父子窗口的关系。框架集父窗口的文档本身不能包含大多数典型的 HTML 对象，比如表单和控件，因此父窗口的文档引用很少使用。

如果在框架集文档中加入脚本，来访问 window 对象的属性或方法，其引用就类似于单框架文档。窗口中脚本的直接容器就是这个窗口。

子框架比较有趣。每个子框架都包含一个 document 对象，它们的内容显示在浏览器窗口中，每个子框架的文档都完全独立于其他文挡，就好像每个文档都位于自己的浏览器窗口中。因此，每个子框架也是一个窗口类型的对象，框架的属性和方法和占据整个浏览器的 window 对象相同。

在图 13-2 中，每个子窗口的直接容器是 parent 窗口。当 parent 窗口位于载入浏览器的层次模型顶部时，该窗口就称为 top 对象。

13.2　家庭成员之间的引用

有了图 13-2 的框架结构，下面看看一个窗口中的脚本如何访问其他窗口的对象、函数或变量。该功能的一个要点是，如果脚本可访问自己窗口的对象、函数和全局变量，则层次结构的另一个框架中的脚本就可以访问它们(假设两个文档来自同一个 Web 服务器)。

在前面的二代层次结构中，脚本引用可能需要三个可能的路径：父到子、子到父或子到子。这些窗口之间的每个路径都需要不同的引用格式。

13.2.1　父到子的引用

父文档中的脚本访问其框架中的一些元素是最少见的引用路径。父文档可以包含一个或多个框架，也就是说父文档包含子框架对象的数组。要定位一个框架，可以使用数组语法或框架名，而框架名可以用<frame>标记中的 id 或 name 特性来设置。在下面的引用语法例子中，用 objFuncVarName 占位符替代要在其他窗口或框架中访问的对象、函数或全局变量。每个可见框架都包含一个 document 对象，它一般是脚本使用的元素的容器，所以元素的引用应当包括 document。从父框架到子框架的引用如下所示：

```
[window.]frames[n].ObjFuncVarName
[window.]frames["frameName"].ObjFuncVarName
[window.]frameName.ObjFuncVarName
```

框架的下标值根据其<frame>标记在框架集文档中的顺序而定。但如果给每个框架指定容易识别的名称，并在引用中使用该框架名，引用就会更简单。

13.2.2　子到父的引用

脚本通常放在父框架文档中(在 head 部分)，多个子框架或一个框架的多个文档将这个父框架作为公共脚本库。这些脚本在载入框架集时，仅在框架集可见时载入一次。如果以后把同一个

更多资源请访问稀酷客(www.ckook.com)

服务器上的其他文档载入框架，它们就可以利用父框架的脚本，而不必将自己的副本载入浏览器。

子框架的上一层称为 parent。因此，从子框架到父框架中元素的引用就很简单：

```
parent.ObjFuncVarName
```

如果父框架中被访问的元素是有返回值的函数，这个返回值就会直接传给子框架：

```
var sValue = parent.FuncName();
```

如果父窗口也位于当前载入浏览器的对象层次结构的顶部，就可以把它作为顶窗口，如下所示：

```
top.ObjFuncVarName
```

如果因为某种原因，Web 页面显示在其他 Web 站点的框架集中，使用 top 引用可能会导致错误。如果顶部窗口不是主框架集的顶部窗口，该怎么办？因此，建议尽量使用 parent 引用(除非脚本嵌套在 Web 站点中一个不想要的框架中，且希望该脚本失败)。

13.2.3　子到子的引用

浏览器在使一个子窗口与另一个子窗口通信时，需要一些帮助。窗口或框架都有 parent 属性(对于单个窗口，它的值是 null)。引用必须使用 parent 属性跳出当前框架，并转到两个子框架共有的父框架上。之后就可以执行该引用的其余部分，就像在父框架上开始执行一样。因此，从一个子框架到另一个子框架的引用可使用下列格式：

```
parent.frames[n].ObjFuncVarName
parent.frames["frameName"].ObjFuncVarName
parent.frameName.ObjFuncVarName
```

从另一个子框架回到第一个子框架的引用也使用这个格式，但是 frames[]数组下标或引用的 frameName 部分不同。当然，在 HTML 中存在更复杂的框架层次。即使这样，对象模型和引用机制也为嵌套最复杂的框架结构提供了解决方案——遵循上述规则。

13.3　有关框架脚本编程的提示

框架脚本编程新手易犯的第一个错误是直接编写在载入页面时调用其他框架的脚本语句。但是文档的载入顺序和框架集源代码的顺序可能不一致。父文档肯定先载入，但无论＜frame＞标记的顺序如何，子框架都可以用任何顺序载入。另外，框架载入时间还取决于文档中的其他元素，比如图像或 Java applet。

幸好，可使用某种技术，在完全载入框架集的所有文档后启动脚本。窗口的 load 事件在该窗口的文档完全载入后触发，同样，父框架的 load 事件也在子框架的 load 事件触发后触发。在链接到框架集文档的脚本中指定 window.onload 事件处理程序，就相当于把该事件处理程序应用于 frameset 元素。该处理程序可能会调用框架集文档中的函数，来处理对象层次中所有框架的对象、函数和变量。

更多资源请访问稀酷客(www.ckook.com)

注意，将框架引用为窗口对象类型与 frame 元素对象的引用是完全不同的。元素对象是文档节点树中的一个 DOM 元素节点(参见第 6 章)，此节点的属性和方法与窗口类型对象的属性和方法不同。这个区别可能难以把握，但十分重要。引用框架的方式(作为窗口对象或元素节点)决定了哪组属性和方法对脚本可用。

交叉引用：

元素节点的脚本编程详见第 26 章。

从 frame 元素对象的引用开始，仍然可以获得载入该框架的 document 对象引用，但语法不同，这取决于浏览器。IE4＋和 Safari 允许使用与窗口相同的 document 引用，而基于 Mozilla 的浏览器更严格地遵循 W3C DOM 标准：使用框架元素的 contentDocument 属性。按以下方式构造引用，就可以兼顾这两种语法：

```
var docObj;
var frameObj = document.getElementById("myFrame");
   if (frameObj.contentDocument)
   {
      docObj = frameObj.contentDocument;
   }
   else
   {
      docObj = frameObj.document;
   }
```

13.4　iframe 元素简介

在 IE4+、基于 Mozilla 的浏览器以及 Safari 中，iframe 元素可用作脚本对象，它通常可以从服务器中获取和载入 HTML，而不会打乱当前的 HTML 页面。因此，在脚本处理 iframe 对象与主文档之间的所有工作时，iframe 对象往往不可见(目前，数据从服务器异步加载到已下载页面上的技术常常使用 XMLHttpRequest()来实现，参见配书光盘上的第 39 章)。

iframe 元素已经成为当前窗口的 frames 集合中的另一个成员，要将 iframe 引用为元素对象，也可以使用 W3C DOM 的 document.getElementById()方法。作为窗口的传统框架对象和 DOM 元素对象是有区别的，同样，对 iframe 元素对象中的 document 对象的脚本引用也需要经过特殊的处理。详见第 27 章。

13.5　突出显示脚注：框架集脚本示例

为了演示窗口交互脚本编程，考虑一个框架集，在文档文本中单击对应的链接时，就会突出显示一个框架中的脚注。图 13-3 显示了这个框架集，程序清单 13-1 显示了标记和脚本。

更多资源请访问稀酷客(www.ckook.com)

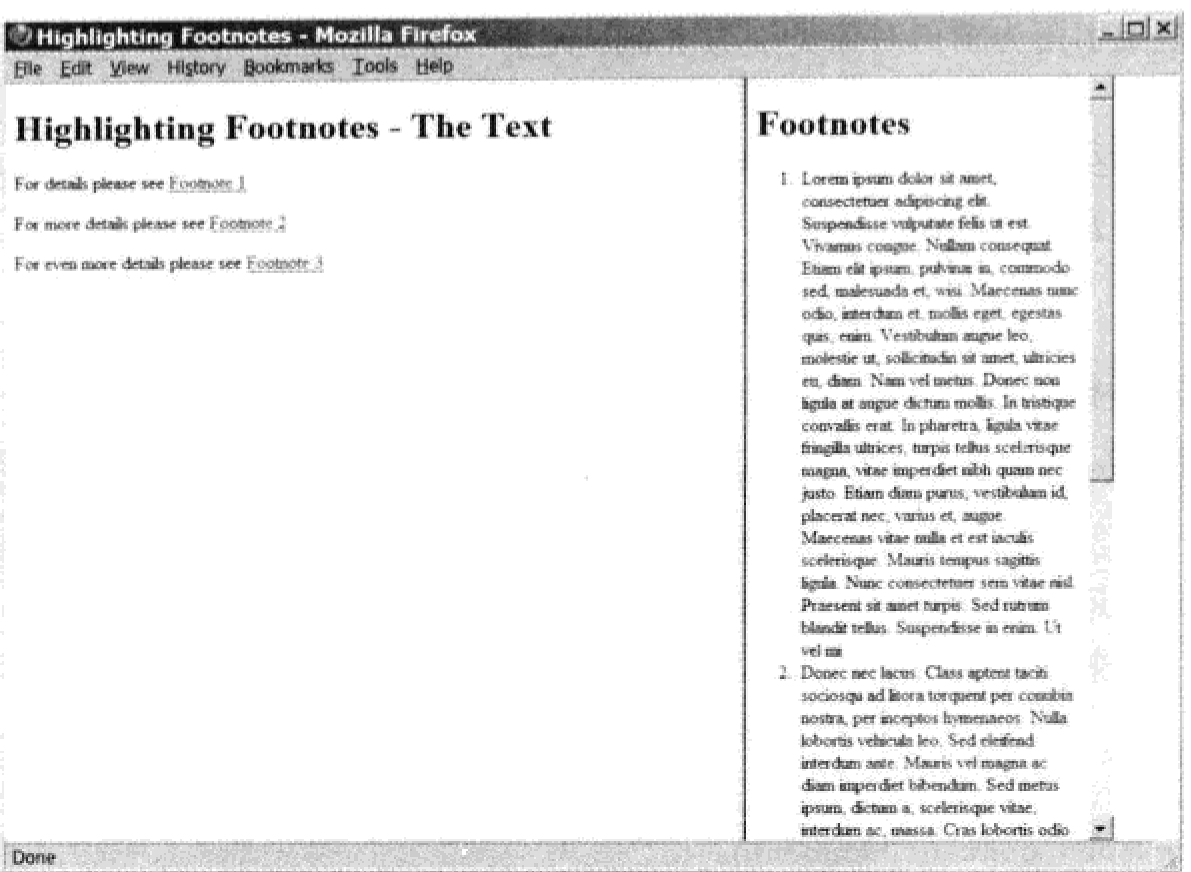


图 13-3　突出显示脚注的框架集

这个例子用 3 个 HTML 文件建立了一个带两个子框架的框架集，一个子框架用于显示主文本，另一个子框架则显示脚注。单击文本中的一个脚注链接，会将所选的脚注显示在脚注框架的顶部。为此，需要把所选脚注的元素 ID 添加到 URL 的“散列”上：

```
<a target="frameFootnotes" ⮐
href="jsb-13-01-frame-footnotes.html#footnote-01">Footnote 1</a>
```

target 特性指定子框架，而 href 指定在该框架上显示哪个文档。href URL #footnote-01 的最后部分则匹配脚注标记：

```
<li id="footnote-01">Lorem ipsum dolor ...
```

这是浏览器的基本功能：元素的 ID 设置为 URL 的散列时，页面会向上滚动，定位到窗口顶部的该元素上。

结合使用 JavaScript 和 CSS 来突出显示所选的脚注：用户在文本窗口中单击一个脚注链接时，JavaScript 就把一个 selected 类赋予该脚注元素。CSS 用一个背景色给该类添加样式。

程序清单 13-1　突出显示脚注

HTML: jsb-13-01-frameset.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Highlighting Footnotes</title>
      <link href="jsb-13-01-frameset.css" rel="stylesheet"
         type="text/css" media="all">
   </head>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <frameset cols="66%,33%">
       <frame name="frameText" src="jsb-13-01-frame-text.html" title="Frame 1">
       <frame name="frameFootnotes" src="jsb-13-01-frame-footnotes.html"
          title="Frame 2">
       <noframes>
          <body>
             <h1>Text with Footnotes</h1>
             <p>Proceed to <a href="jsb-13-01-frame-text-footnotes.html">
                   combined text and footnotes</a>.</p>
          </body>
       </noframes>
    </frameset>
</html>
```

Stylesheet: jsb-13-01-frameset.css

```
/* limit the width of the page */
html
{
   width: 60em;
   max-width: 100%;
}
```

HTML: jsb-13-01-frame-text.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Highlighting Footnotes - The Text</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-13-01-frame-text.js"></script>
   </head>
   <body>
      <h1>Highlighting Footnotes - The Text</h1>

      <p>For details please see <a target="frameFootnotes"
            href="jsb-13-01-frame-footnotes.html#footnote-01">Footnote 1</a></p>

      <p>For more details please see <a target="frameFootnotes"
            href="jsb-13-01-frame-footnotes.html#footnote-02">Footnote 2</a></p>

      <p>For even more details please see <a target="frameFootnotes"
            href="jsb-13-01-frame-footnotes.html#footnote-03">Footnote 3</a></p>

   </body>
</html>
```

JavaScript: jsb-13-01-frame-text.js

```
// run script when the page has loaded
addEvent(window, 'load', initialize);
```


更多资源请访问稀酷客(www.ckook.com)

```
// global memory of the last footnote highlighted
var oFootnote = null;
var sFootnoteClass = '';

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // collect anchors
      var aAnchors = document.getElementsByTagName('a');

      // for each anchor...
      for (var i = 0; i < aAnchors.length; i++)
      {
         // get the target of the link
         var sTarget = aAnchors[i].getAttribute('target');

            // if there is a target and it's the footnotes frame
            if (sTarget && sTarget == 'frameFootnotes')
            {
               addEvent(aAnchors[i], 'click', highlightFootnote);
            }
      }
   }
}

// highlight the selected footnote
function highlightFootnote()
{
      // restore previously highlighted footnote (if any)
      if (oFootnote)
      {
         oFootnote.className = sFootnoteClass;
      }

   // get the HREF of the clicked anchor
   var sHref = this.getAttribute('href');
   // get the 'hash' of the URL
   var iHash = sHref.indexOf('#');
   // everything after the hash mark is the element Id
   var sFootnoteId = sHref.substr(iHash + 1);

   // the target of the link is the frame Id
   var sFrameId = this.getAttribute('target');

      // if the frame exists...
      if (parent[sFrameId])
      {
         // point to the indicated footnote
         oFootnote = parent[sFrameId].document.getElementById(sFootnoteId);
            // if it exists...
```


更多资源请访问稀酷客(www.ckook.com)

```
            if (oFootnote)
            {
               // remember its original class to restore it later
               sFootnoteClass = oFootnote.className;

               // set its new class
               oFootnote.className = 'selected';
            }
      }
}
```

HTML: jsb-13-01-frame-footnotes.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Footnotes</title>
      <link href="jsb-13-01-frame-footnotes.css" rel="stylesheet"
         type="text/css" media="all">
   </head>
   <body>
      <h1>Footnotes</h1>

      <ol class="footnotes">
         <li id="footnote-01">Lorem ipsum dolor sit amet, consectetuer
adipiscing elit. Suspendisse vulputate felis ut est. Vivamus congue.
Nullam consequat. Etiam elit ipsum, pulvinar in, commodo sed, malesuada
et, wisi. Maecenas nunc odio, interdum et, mollis eget, egestas quis, enim.
Vestibulum augue leo, molestie ut, sollicitudin sit amet, ultricies eu, diam.
Nam vel metus. Donec non ligula at augue dictum mollis. In tristique convallis
erat. In pharetra, ligula vitae fringilla ultrices, turpis tellus scelerisque
magna, vitae imperdiet nibh quam nec justo. Etiam diam purus, vestibulum id,
placerat nec, varius et, augue. Maecenas vitae nulla et est iaculis scelerisque.
Mauris tempus sagittis ligula. Nunc consectetuer sem vitae nisl. Praesent sit
amet turpis. Sed rutrum blandit tellus. Suspendisse in enim. Ut vel mi.</li>

         <li id="footnote-02">Donec nec lacus. Class aptent taciti sociosqu ad
litora torquent per conubia nostra, per inceptos hymenaeos. Nulla lobortis
vehicula leo. Sed eleifend interdum ante. Mauris vel magna ac diam imperdiet
bibendum. Sed metus ipsum, dictum a, scelerisque vitae, interdum ac, massa.
Cras lobortis odio at augue. Aenean fermentum bibendum purus. Ut congue
interdum turpis. Ut quis mi a ligula viverra hendrerit. Nulla facilisi.
Nullam lacinia, ligula ac congue feugiat, risus diam fringilla tellus,
condimentum volutpat eros risus sit amet dolor.</li>

         <li id="footnote-03">Proin et urna. Morbi in odio. In mauris tellus,
tincidunt vitae, ultrices a, sollicitudin sit amet, nulla. Praesent consequat,
lectus auctor imperdiet dapibus, risus turpis feugiat orci, sit amet tempus quam
erat a purus. Nullam neque nulla, fringilla ut, dictum id, luctus placerat,
urna. Sed vel velit. Fusce eget erat a quam auctor pulvinar. Phasellus vel
```


更多资源请访问稀酷客(www.ckook.com)

```
neque eu risus feugiat sagittis. Nulla est. Duis sed massa. Duis ac leo quis
sapien fringilla posuere. Nam facilisis lorem. Nullam bibendum, wisi quis
ultrices bibendum, sem pede volutpat sem, ut ultrices odio nisl ut ligula.
Cras ligula est, tempus et, tempus vitae, pretium eget, ipsum. Integer
Blandit mattis wisi. Curabitur eget tellus at neque vulputate dignissim.
Phasellus leo nibh, euismod ac, congue vel, interdum et, turpis. Fusce
viverra aliquet wisi.</li>
      </ol>
   </body>
</html>
```

Stylesheet: jsb-13-01-frame-footnotes.css

```
/* style the selected footnote */
ol.footnotes li.selected
{
   background-color: aqua;
}
```

下面分析这些 JavaScript 代码，看看突出显示机制的工作方式。脚本链接到 frame-text.html 文档上，因为单击该文档上的链接，就会触发突出显示事件。addEvent()函数调用告诉 JavaScript，在加载了该文档后运行 initialize()函数，该函数会遍历文档中的所有 anchor 元素。如果它发现 target 特性设置为 frameFootnotes，就触发 onclick 事件，来运行 highlightFootnote()函数。

用户在文档文本中单击其中一个链接时，highlightFootnote()函数就给 anchor 元素赋予 selected 类。为了便于以后把链接恢复为它以前的类名，脚本把当前的 anchor 元素指针及其类特性存储在全局变量 oFootnote 和 sFootnoteClass 中。所以，单击链接时，它首先恢复上一次单击的链接(如果存在)。

要在另一个子框架中设置脚注的 class 特性，脚本需要确定框架和脚注元素的名称。框架名称与被单击链接的 target 特性相同。脚注元素是脚注框架中其 ID 等于链接 URL 尾部的散列的元素。

如果指定的框架存在，指定的元素也存在，脚本就可以安全地存储元素的已有类，并把该类设置为 selected。最后，链接到脚注文档上的样式表把一个背景色应用于带 selected 类的脚注。

程序清单 13-1 演示了两个重要的脚本编程原则。首先，这个页面的基本操作(把选中的脚注显示在脚注框架的顶部)完全由 HTML 处理，无需 JavaScript 的帮助。这将确保使用不支持 JavaScript 的用户代理(移动设备)查看该页面时，或者使用禁用了 JavaScript 的浏览器查看该页面时，该页面是有效的。这是一个逐步改进的例子，逐步改进是指，脚本编程应改进已具备一定功能的页面，而不是提供非常重要的功能。我们希望改进运行 JavaScript 的用户的体验，而不是让页面在不支持 JavaScript 的浏览器上崩溃。

其次，使用 JavaScript 是为了指出，脚注被选中，而不是脚注是如何设置样式的。可以这样编写脚本：

```
oFootnote.style.backgroundColor = 'aqua';
```

样式设置使用 CSS 完成。这是分解开发层的一个例子，即 HTML 结构和内容与 JavaScript 代码和 CSS 显示分开，可以使用这种开发模式便捷地创建新代码，修改已有的代码，把旧代码应用于新项目。例如，网站设计人员修改网站的外观时——几乎每个网站都会在开发过程中或开

更多资源请访问稀酷客(www.ckook.com)

发完成后进行修改——网站的美化工作应由掌握 CSS 的人完成，无需 JavaScript 程序员的帮助。

13.6　多窗口引用

第 10 章提到，window.open()方法返回了 window 对象引用，该引用创建了一个新窗口，并与它通信。本节将介绍子窗口如何与创建它的窗口或框架中的对象、函数和变量通信。

每个 window 对象都有一个 opener 属性，该属性包含窗口或框架的引用，这些窗口或框架包含一些脚本，脚本中的 window.open()语句生成了子窗口。对于主浏览器窗口和框架，这个 opener 属性值为 null。opener 属性是一个有效的窗口引用(其值为非 null)，可以用于引用原始窗口的元素，就像子框架的脚本使用 parent 来访问父文档的元素一样，不过 parent 关键字不能应用于子窗口的 opener 属性。

程序清单 13-2 包含的两个文档在两个窗口中协同工作。jsb-13-02-main 显示了两个按钮，一个按钮会打开一个较小的窗口，把 jsb-13-02-sub.html 加载到该窗口中，另一个按钮关闭该子窗口。主窗口文档还包含一个文本域，在子窗口的对应域中输入文本时，就会填充主窗口中的文本域。

注意：

练习这些例子时，可能需要暂时关闭弹出阻塞功能。

因为许多人都把浏览器设置为禁止弹出窗口，而弹出窗口可能令使用辅助技术如屏幕阅读器的人迷失方向，给人带来烦恼，所以在实际的应用程序中最好不要使用 JavaScript 创建子窗口，尤其是重要的函数，而应使用其他技术，例如 CSS 弹出窗口，从容地降级为使用简单的 HTML 导航，而不添加样式和脚本。

在主窗口的脚本中，openSubWindow()函数生成了新窗口。closeSubWindow()函数需要知道关闭哪个窗口，所以把指向新窗口的指针存储在全局变量中。

程序清单 13-2　主窗口文档

HTML：jsb-13-02-main.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Opener</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-13-02-main.js"></script>
   </head>
   <body>
      <h1>Opener</h1>
      <form action="">
         <p>
            <input type="button" value="Open Sub Window" id="open-sub-window">
            <input type="button" value="Close Sub Window" id="close-sub-window">
```


更多资源请访问稀酷客(www.ckook.com)

```
            </p>
            <p>
               <label>Text incoming from subwindow:</label>
               <input type="text" id="output">
            </p>
         </form>
      </body>
   </html>
```

JavaScript: jsb-13-02-main.js

```
// run script when the page has loaded
addEvent(window, 'load', initialize);

// global pointer to sub-window
var oSubWindow;

// apply behaviors when document has loaded
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to button
      var oButton = document.getElementById('open-sub-window');

         // if it exists, apply behavior
         if (oButton)
         {
            addEvent(oButton, 'click', openSubWindow);
         }

      // point to button
      oButton = document.getElementById('close-sub-window');

         // if it exists, apply behavior
         if (oButton)
         {
            addEvent(oButton, 'click', closeSubWindow);
         }
   }
}

// open (create) sub window
function openSubWindow()
{
   // store sub-window pointer in global variable
   oSubWindow = window.open("jsb-13-02-sub.html", "sub", "height=200,width=300");
}

// close (destroy) sub window
function closeSubWindow()
{
```


更多资源请访问稀酷客(www.ckook.com)

```
        // if the sub-window has been opened, close it
        if (oSubWindow)
        {
            oSubWindow.close();
        }
}
```

HTML: jsb-13-02-sub.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Sub-Document</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb-13-02-sub.js"></script>
    </head>
    <body>
        <h1>Sub-Document</h1>
        <form id="sendForm" action="">
            <p>
                <label for="input">Enter text to be copied to the main window:</label>
                <input type="text" id="input">
            </p>
            <p>
                <input type="button" value="Send Text to Opener" id="send-text">
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-13-02- sub.js

```
// run script when the page has loaded
addEvent(window, 'load', initialize);

// apply behaviors when document has loaded
function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to button
        var oButton = document.getElementById('send-text');

            // if it exists, apply behavior
            if (oButton)
            {
                addEvent(oButton, 'click', copyToOpener);
            }
    }
}
```


更多资源请访问稀酷客(www.ckook.com)

```
// copy input text to output in parent
function copyToOpener()
{
   // find the input field in the current window
   var oInput = document.getElementById('input');

   // find the output field in the parent window
   var oOutput = opener.document.getElementById('output');

      // if they both exist, copy input text
      if (oInput && oOutput)
      {
         oOutput.value = oInput.value;
      }
}
```

子窗口文档中的所有动作都来自 Send Text 按钮的 onclick 事件处理程序，它触发了 copyToOpener() 函数，把子窗口中输入域的值赋给主窗口文档中的输出域。注意，每个窗口和框架的内容都属于文档。因此即使引用的目标是特定的窗口或框架，也必须引导浏览器找到最终的目标，通常这些目标是文档的某个元素。

在学习本书的其他内容之前，还有一章需要学习，本部分的最后一章介绍在 Web 页面中可以完成的有趣操作，如用户的鼠标滑过一幅图片时改变图像等。

13.7　习题

回答下面的前三个问题前，请研究下述网站的框架集结构，这个框架集用于列举大学课程：

```
<html>
   <head>
      ...
   </head>
   <frameset rows="85%,15%">
      <frameset cols="20%,80%">
         <frame name="mechanics" src="history101M.html">
         <frame name="description" src="history101D.html">
      </frameset>
      <frameset cols="100%">
         <frame name="navigation" src="navigator.html">
      </frameset>
   </frameset>
   <noframes>
      ...
   </noframes>
</html>
```

1. 每个载入 description 框架的文档都链接了一个 JavaScript 脚本，该脚本使用 onload 事件处理程序，将课程标识符值存放在框架集文档的 currCourse 全局变量中。写出这个 onload 事件处理程序，将 currCourse 的值设置为 history101。

更多资源请访问稀酷客(www.ckook.com)

2. 画框图来描述框架集定义中窗口和框架的层次结构。

3. 写出 navigation 框架中的 HTML 标记和 JavaScript 语句，执行两个动作：将文件 french201M.html 载入 mechanics 框架，把文件 french201D.html 载入 description 框架。

4. 在框架集载入过程中，突然出现了一个 JavaScript 错误消息，显示 window.document.navigation.form.selector is undefined 信息。在应用程序的脚本中发生了什么？该如何解决这个问题？

5. 在主窗口的子框架中，脚本使用 window.open()生成了第二个窗口，第二个窗口的脚本如何访问主浏览器窗口中父窗口的 location 对象(URL)？

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第14章

图像和动态 HTML

前 8 章透彻讲述了许多编程概念和 JavaScript 的基础知识，现在使用这些基础知识学习更高级的技术。这涉及两个方面，首先讨论如何实现常见的鼠标滚动，即用户在屏幕上移动光标时变换图像，然后介绍加载页面后如何修改页面样式和内容。

根据鼠标的滚动变换图像通过 CSS(Cascading Style Sheets)更容易实现，但这里介绍如何使用 JavaScript 来实现，因为旧脚本中有许多鼠标滚动，这也是说明如何操作图像对象的简便方式。

本章包含哪些内容？

- 预缓存图像
- 根据鼠标的滚动变换图像
- 更改样式表设置
- 动态修改主体内容

14.1 image 对象

image 对象包含在文档中，Netscape Navigator 3 和 Internet Explorer 4 以及之后推出的所有脚本浏览器都支持这个对象。文档的 image 对象引用在对象模型中存储为 document 对象的数组，因此可以用数组下标或图像名来引用图像，而且，数组下标可以是图像名的字符串版本。下列都是 image 对象的合法引用：

```
document.getElementById("imageName")
document.getElementsByTagName("img")[n]
document.images[n]
document.images["imageName"]
document.imageName
```

最后一个引用不能包含任何不允许在 JavaScript 名称中使用的、但对 HTML ID 有效的字符，例如连字符或句点。

老脚本浏览器要求图像必须装入 a 元素，才能接收鼠标事件，但现在不再施加这个限制。如果要通过访问者的鼠标单击操作导航到新的 URL，则仍可能需要这样做，但如果要通过单击操作来启动本地 JavaScript 代码的执行，则最好让 img 元素的事件处理程序完成所有工作。

更多资源请访问稀酷客(www.ckook.com)

14.1.1　可互换的图像

拥有一个支持脚本的 image 对象，脚本就可改变矩形空间中的已有图像。在当今的浏览器中，甚至可改变图像的尺寸，其周围的内容会自动改变，以容纳尺寸改变后的图像。

这种图像变换的脚本非常简单，只需把一个新的 URL 赋给 img 元素对象的 src 属性即可。页面上图像的大小由<img>标记中设置的 height 和 width 特性，以及样式表中设置的 height 和 width 属性控制。最常见的图像滚动对每个滚动状态使用相同大小的图像。

14.1.2　图像的预缓存

图像要花费额外的时间从 Web 服务器上下载，并存储在浏览器的缓存中。假如页面设计为根据用户的动作改变图像，一般希望图像改变的反应速度与其他程序一样快。让用户花很长时间等待图像改变，会影响页面的效果。

JavaScript 允许脚本把图像载入浏览器内存的缓存，但不显示图像，这项技术称为预缓存图像。最好是在第一次载入页面时，把图像预载到浏览器的图像缓存中。相对于等待某个鼠标动作来启动图像的下载，用户等待图像载入主页面时要耐心得多，但注意要把握好尺度。目前的用户一般希望页面的下载时间不超过 10 秒，否则它们就会转向其他页面。尽管预缓存图像意味着对鼠标动作的快速响应，但如果预缓存了太多的图像，网站访问者就可能在看到为响应鼠标动作而变换的漂亮图像之前退出网站。

预缓存图像需要在内存中构造 image 对象。在内存中创建的 image 对象在几个方面不同于用<img>标记创建的文档 image 元素对象。内存对象是由脚本创建的，它们不在页面上显示。但它们存在于文档代码中，迫使浏览器在载入页面时载入图像。对象模型提供了一个 image 对象构造函数，来创建内存类型的 image 对象，如下：

```
var myImage = new Image(width, height);
```

构造函数的参数是图像的宽度和高度(以像素为单位)，这些尺寸应该匹配<img>标记的 width 和 height 特性。一旦 image 对象存在于内存中，就可以给 image 对象的 src 属性赋予文件名或 URL：

```
myImage.src = "someArt.gif";
```

浏览器遇到上述语句时，就会把图像取出并载入图像缓存。用户只能在状态栏上看到一些附加的载入信息，就像页面上有另一个图像。载入整个页面后，以这种方式生成的所有图像就存放在图像缓存中。然后把缓存图像的 src 属性或实际图像的 URL 赋给用<img>标记创建的文档图像的 src 属性：

```
document.images[0].src = myImage.src;
```

或

```
document.getElementById("myImg").src = myImage.src ;
```

文档中图像的改变是瞬间发生的。

程序清单 14-1 演示的页面由一个<img>标记和选择列表组成，该列表允许把文档中的图

更多资源请访问稀酷客(www.ckook.com)

像替换为 4 个预缓存图像(包括为标记指定的原始图像)中的任何一个。假如输入这个程序清单，就可从配书光盘的 Chap 14 目录里获得这 4 个图像文件的副本(但必须输入 HTML 和代码)。

程序清单 14-1　预缓存图像

HTML: jsb-14-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Image Object</title>
      <style type="text/css">
         img
         {
            height:90px; width:120px;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-14-01.js"></script>
   </head>
   <body >
      <h2>Image Object</h2>
      <img src="desk1.gif" id="thumbnail">
      <form>
         <select id="imageChooser">
            <option value="image1">Bands</option>
            <option value="image2">Clips</option>
            <option value="image3">Lamp</option>
            <option value="image4">Erasers</option>
         </select>
      </form>
   </body>
</html>
```

JavaScript: jsb-14-01.js

```
// initialize when the page has loaded
var oImageChooser;
addEvent(window, "load", initialize);
function initialize()
{
   // get W3C DOM or IE4+ reference for an ID
   // imageChooser is the select element
   if (document.getElementById)
   {
      oImageChooser = document.getElementById("imageChooser");
   }
   else
   {
      oImageChooser = document.imageChooser;
```


更多资源请访问稀酷客(www.ckook.com)

```
      }

      addEvent(oImageChooser, 'change', loadCached);
   }

   // initialize empty array
   var imageLibrary = new Array();
   // pre-cache four images
   imageLibrary["image1"] = new Image(120,90);
   imageLibrary["image1"].src = "desk1.gif";
   imageLibrary["image2"] = new Image(120,90);
   imageLibrary["image2"].src = "desk2.gif";
   imageLibrary["image3"] = new Image(120,90);
   imageLibrary["image3"].src = "desk3.gif";
   imageLibrary["image4"] = new Image(120,90);
   imageLibrary["image4"].src = "desk4.gif";

   // load an image chosen from select list
   function loadCached()
   {
      var imgChoice = oImageChooser.options[oImageChooser.selectedIndex].value;
      document.getElementById("thumbnail").src = imageLibrary[imgChoice].src;
   }
```

页面载入时，会立刻执行几个语句，来创建一个数组。该数组包含 4 个新的内存 image 对象，每个 image 对象的文件名都赋给其 src 属性。页面载入时，这些图像就载入到图像缓存。在文档的 body 部分，<img>标记标出图像在页面中的范围，并载入一个图像作为初始图像。

当 select 元素包含预存在内存中的 image 对象名时，就列出便于用户使用的图片名称。用户在列表中选择某个选项时，loadCached()函数就提取所选项的值，该值是 imageLibrary 数组中图像的字符串下标。所选 image 对象的 src 属性赋给页面上可见 img 元素对象的 src 属性，预缓存图像会立即显示出来。

14.1.3　图像变换的创建

要在页面中加入令人兴奋的内容，最好是在画面上移动鼠标时，变换按钮图像。图像的改变很大程度上是一个品位问题，其效果有时不明显，比如将原始图像的边缘微微加亮或使之发光；有时很明显，比如剧烈的颜色变化。不管效果如何，脚本编程都是一样的。

一组中有几个这样的图形按钮时，最好把内存中的 image 对象组织为数组，并创建便于处理该数组的命名和编号机制。程序清单 14-2 就提供了这样一个机制，其中的 4 个按钮控制幻灯片的放映。程序清单中的代码只用于应用程序中图像变换的部分，这是本书最复杂的程序清单，而且内容较多，因此需要做一些解释。首先给出一个样式表规则，用于 controller 容器中的每个 img 元素。

程序清单 14-2　像变换

```
<!DOCTYPE html>
<html>
  <head>
```

更多资源请访问稀酷客(www.ckook.com)

```
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>Slide Show/Image Rollovers</title>
<style type="text/css">
   div#controller img {
      height: 70px; width: 136px; padding: 5px;
      }
</style>
<script type="text/javascript" src="../jsb-global.js"></script>
<script type="text/javascript">
```

只有能够处理 image 对象的浏览器才能执行预缓存图像的语句，因此，整个语句嵌套在一个 if 结构中，来测试 document.images 数组是否存在。在旧版浏览器中，条件求值的结果是 undefined，if 条件将其视为 false：

```
if (document.images)
{
```

在预缓存图像时，首先要构建 image 对象的两个数组，一个数组用于表示图形按钮的 off 状态的图像的信息；另一个用于表示 on 状态的图像的信息。这些数组把字符串(不是整数)作为下标值，这些字符串名对应于可见的 img 元素对象名，该元素对象的标记在后面的源代码中给出，这样代码将更清晰可读(例如，offImgArray["first"]项显然与 First 按钮图像相关)。另外，如程序清单的后面所示，滚动图像和页面上的其他可见图像没有冲突(假如表示用于变换的可见图像时只使用数值下标，就可能有冲突)。

创建数组，并把新的空 image 对象赋给数组的前 4 个元素后，再次遍历数组，这次把文件路径名赋给数组中每个对象的 src 属性。这些代码行在载入页面时执行，迫使图像在页面载入过程中也载入到图像缓存中：

```
// pre-cache all 'off' button images
var offImgArray = new Array();
offImgArray["first"] = new Image(136,70);
offImgArray["prev"] = new Image(136,70);
offImgArray["next"] = new Image(136,70);
offImgArray["last"] = new Image(136,70);

// off image array -- set 'off' image path for each button
offImgArray["first"].src = "firstoff.png";
offImgArray["prev"].src = "prevoff.png";
offImgArray["next"].src = "nextoff.png";
offImgArray["last"].src = "lastoff.png";

// pre-cache all 'on' button images
var onImgArray = new Array();
onImgArray["first"] = new Image(136,70);
onImgArray["prev"] = new Image(136,70);
onImgArray["next"] = new Image(136,70);
onImgArray["last"] = new Image(136,70);

// on image array -- set 'on' image path for each button
onImgArray["first"].src = "firston.png";
```


更多资源请访问稀酷客(www.ckook.com)

```
onImgArray["prev"].src = "prevon.png";
onImgArray["next"].src = "nexton.png";
onImgArray["last"].src = "laston.png";
```

看出了上述语句中的重复模式吗？如果不使用这些语句，可利用该重复模式编写如下语句：

```
var offImgArray = new Array();
var onImgArray = new Array();
var ButtonArray = ["first", "prev", "next", "last"];

for (var i=0; i < ButtonArray.length; i++)
{
   var Button = ButtonArray[i];

   // pre-cache 'off' button image
   offImgArray[Button] = new Image(136,70);
   offImgArray[Button].src = Button + "off.png";

   // pre-cache 'on' button image
   onImgArray[Button] = new Image(136,70);
   onImgArray[Button].src = Button + "on.png";
}
```

在下面的代码中，用户在文档中任何可见的 image 对象上移动鼠标时，onmouseover 事件处理程序就调用 imageOn()函数，并给它传递指定的图像名。imageOn()函数通过该图像名使 document.images 数组项(可见图像)和 onImgArray 数组的内存图像数组项同步。数组项的 src 属性赋给相应文档图像的 src 属性。同时，光标也变化了，表示位于活动链接上。

```
}
// functions that swap images & status bar
function imageOn(imgName)
{
   if (document.images)
   {
      document.images[imgName].style.cursor = "pointer";
      document.images[imgName].src = onImgArray[imgName].src;
   }
}
```

同样，onmouseout 事件处理程序通过相同的下标值调用 imageOff()函数，来关闭图像。

```
function imageOff(imgName)
{
   if (document.images)
   {
      document.images[imgName].style.cursor = "default";
      document.images[imgName].src = offImgArray[imgName].src;
   }
}
```

onmouseover 和 onmouseout 事件处理程序都设置状态栏文本，以显示便于他人理解的描述性内容——至少在仍支持在状态栏上显示定制文本的浏览器上是这样。onmouseout 事件处理程

更多资源请访问稀酷客(www.ckook.com)

序将状态栏消息设置为空字符串。

```
function setMsg(msg)
{
   window.status = msg;
   return true;
}
```

为了进行演示，这个例子禁用了控制幻灯片放映的函数。但这里定义了一个空函数，它们能捕获与图像相关的链接被单击时执行的调用操作。

```
// controller functions (disabled)
function goFirst()
{
}
function goPrev()
{
}
function goNext()
{
}
function goLast()
{
}
// event handler assignments
function init()
{
   if (document.getElementById)
   {
      oImageFirst = document.getElementById("first");
      oImagePrev = document.getElementById("prev");
      oImageNext = document.getElementById("next");
      oImageLast = document.getElementById("last");
   }
   else
   {
      oImageFirst = document.first;
      oImagePrev = document.prev;
      oImageNext = document.next;
      oImageLast = document.last;
   }

   addEvent(oImageFirst, 'click', goFirst);
   addEvent(oImageFirst, 'mouseover', function()
      {
         imageOn("first");
         return setMsg("Go to first picture");
      });
   addEvent(oImageFirst, 'mouseout', function()
      {
         imageOff("first");
```


更多资源请访问稀酷客(www.ckook.com)

```
                return setMsg("");
            });
        addEvent(oImagePrev, 'click', goPrev);
        addEvent(oImagePrev, 'mouseover', function()
            {
                imageOn("prev");
                return setMsg("Go to previous picture");
            });
        addEvent(oImagePrev, 'mouseout', function()
            {
                imageOff("prev");
                return setMsg("");
            });
        addEvent(oImageNext, 'click', goNext);
        addEvent(oImageNext, 'mouseover', function()
            {
                imageOn("next");
                return setMsg("Go to next picture");
            });
        addEvent(oImageNext, 'mouseout', function()
            {
                imageOff("next");
                return setMsg("");
            });
        addEvent(oImageLast, 'click', goLast);
        addEvent(oImageLast, 'mouseover', function()
            {
                imageOn("last");
                return setMsg("Go to last picture");
            });
        addEvent(oImageLast, 'mouseout', function()
            {
                imageOff("last");
                return setMsg("");
            });
    }

    // initialize when the page has loaded
    addEvent(window, "load", init);
  </script>
</head>

<body>
<h1>Slide Show Controls</h1>
```

这里选择将控制器图像放在一个 div 元素中，以使图像能作为一个组来定位或设置样式。每个 img 元素的 onmouseover 事件处理程序(在前面定义)都会调用 imageOn()函数，并给它传递要变换的图像名称。由于 onmouseover 和 onmouseout 事件处理程序都需要一条 return true 语句，才能在较旧的浏览器中工作，所以这里将第二个函数调用(调用 setMsg())与 return true 语句结合在一起。setMsg()函数总是返回 true，与调用它之前的 return 关键字结合起来，就是 return true。

更多资源请访问稀酷客(www.ckook.com)

这只是减少这些事件处理程序的代码量的一个小技巧。

```
            <div id="controller">
               <img src="images/firstoff.png" name="first" id="first" >
               <img src="images/prevoff.png" name="prev" id="prev" >
               <img src="images/nextoff.png" name="next" id="next" >
               <img src="images/lastoff.png" name="last" id="last" >
            </div>
      </body>
   </html>
```

图 14-1 显示了这个长脚本的结果。用户在图像上移动鼠标时，整个图像会从浅色变到深色。配书光盘提供了这些图像文件，建议输入这段冗长的代码，看看这个奇妙的结果。

图 14-1　典型的鼠标滚动图像变换

14.2　无需脚本的图像变换

尽管变换效果很酷，但有了 CSS 技术，就不总是需要 JavaScript 来实现变换效果。结合使用 CSS 和 JavaScript 也可以实现相同的效果。程序清单 14-3 是程序清单 14-2 的另一个版本，它使用 CSS 来实现变换效果，而 JavaScript 仍然用于幻灯片放映的控制。

按钮的 HTML 由 li 元素组成，li 元素是 off 版本按钮的背景图像，但图像的大小做了调整。每个 li 元素的文本都放在一个 a 元素中，以便指定 CSS: hover 伪元素(直到 Internet Explorer 版本 7 都需要这样做，而 W3C DOM 浏览器能识别所有元素的:hover)。把鼠标指针停放在 a 元素上，背景图像就变为 on 版本。onclick 事件处理程序的分配语句仍位于页面的脚本部分，它们在页面载入后执行，以确保元素存在。

程序清单 14-3　CSS 图像变换

HTML: jsb-14-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>Slide Show/Image Rollovers</title>
        <link rel="stylesheet" href="jsb-14-03.css" type="text/css">
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb-14-03.js"></script>
        <script type="text/javascript">
        </script>
    </head>
    <body>
        <h1>Slide Show Controls</h1>
        <ul id="controller">
            <li id="first"><a href="#">First</a></li>
            <li id="prev"><a href="#">Previous</a></li>
            <li id="next"><a href="#">Next</a></li>
            <li id="last"><a href="#">Last</a></li>
        </ul>
    </body>
</html>
```

CSS: jsb-14-03.css

```
#controller {
    position: relative;
    }
#controller li {
    position: absolute; list-style: none; display: block;
    height: 70px; width: 136px;
    }
#controller a {
    display: block; text-indent: -999px; height: 70px;
    }

#first {
    left: 0px;
    }
#prev {
    left: 146px;
    }
#next {
    left: 292px;
    }
#last {
    left: 438px;
    }

#first a {
    background-image: url("firstoff.png");
    }
#first a:hover {
    background-image: url("firston.png");
    }
#prev a {
    background-image: url("prevoff.png");
```


更多资源请访问稀酷客(www.ckook.com)

```
    }
#prev a:hover {
    background-image: url("prevon.png");
    }
#next a {
    background-image: url("nextoff.png");
    }
#next a:hover {
    background-image: url("nexton.png");
    }
#last a {
    background-image: url("lastoff.png");
    }
#last a:hover {
    background-image: url("laston.png");
    }
```

JavaScript: jsb-14-03.js

```
// controller functions (disabled)
function goFirst()
{
   alert("in gofirst");
}
function goPrev()
{
   alert("in goprev");
}
function goNext()
{
   alert("in gonext");
}
function goLast()
{
   alert("in golast");
}

// event handler assignments
function init()
{
   if (document.getElementById)
   {
      oImageFirst = document.getElementById("first");
      oImagePrev = document.getElementById("prev");
      oImageNext = document.getElementById("next");
      oImageLast = document.getElementById("last");
   }
   else
   {
      oImageFirst = document.first;
      oImagePrev = document.prev;
```


更多资源请访问稀酷客(www.ckook.com)

```
        oImageNext = document.next;
        oImageLast = document.last;
    }
        addEvent(oImageFirst, 'click', goFirst);
        addEvent(oImagePrev, 'click', goPrev);
        addEvent(oImageNext, 'click', goNext);
        addEvent(oImageLast, 'click', goLast);
}

// initialize when the page has loaded
addEvent(window, "load", init);
```

Internet Explorer 需要将 li 元素的文本(CSS 将其完全隐藏，因为我们不需要文本)封装起来，这迫使脚本开发人员编写更多的代码。在此应用程序中，点击 li 元素会运行本地脚本，而不是载入外部 URL，但 a 元素的默认行为是载入另一个 URL。程序清单 14-3 中的＃占位符会重新载入当前页面，这将清除 onclick 事件处理函数的所有效果。因此，必须为每个幻灯片放映导航函数编写一些代码，以防 a 元素执行其默认行为，详见第 32 章(对于不兼容的 W3C DOM 和 IE 事件模型，它需要不同的语法)。

还要注意，对于程序清单 14-3 中的 CSS 方法，如果没有预先缓存图像，可以把程序清单 14-2 中用于 on 图像的预缓存代码添加进来，以便浏览器变换不同的背景图像。这是结合使用 CSS 和 JavaScript 的一种情况。

14.3　JavaScript：伪 URL

前面章节中的示例演示了如何将所谓的 javascript:伪 URL 应用于＜a＞和＜area＞标记中的 href 特性。这一技巧不应用于可能由关闭脚本编程功能的浏览器所访问的公共 Web 站点(此时，链接不是活动的)。

实现此技巧是为了把它作为超链接对象的 onclick 事件处理程序。尤其在脚本编程的早期，图像等元素没有自己的事件处理程序，包含这些非活动元素的超链接元素利用该技术允许用户直接与图像等元素交互。为了调用脚本函数(而不是导航到另一个 URL，超链接元素通常会导航到另一个 URL)，语言设计人员发明了 javascript:协议，用于给超链接元素的 href 特性赋值(而不是让该特性为空)。

支持脚本编程的浏览器遇到了指向 javascript:伪 URL 的 href 特性时，如果用户单击元素，浏览器就执行冒号后的脚本内容。例如，程序清单 14-3 中的 a 元素可以写为指向调用页面上脚本函数的 javascript:伪 URL，如：

```
<a href="javascript:goFirst()">
```

浏览器厂商广泛采用了 javascript:协议，但它不是一个公开发布的标准。在一个公共 Web 站点中，如果访问者关注该网站的可访问性，JavaScript 功能极少或没有该功能的浏览器也要访问它，则链接应该指向执行动作的服务器 URL(例如，通过服务器程序)，对于使用脚本浏览器的访问者，该服务器会完成与客户端 JavaScript 同样的功能(更快)。

更多资源请访问稀酷客(www.ckook.com)

14.4 主流的动态 HTML 技术

由于今天的脚本浏览器都允许脚本访问文档的每个元素，并在页面改变时自动刷新页面的内容，因此可以在应用程序中实现高度的动态化。动态 HTML(DHTML)是个非常宽泛的主题，有大量浏览器特定的细节。本节将介绍在 Internet Explorer 和 W3C DOM 兼容浏览器中可用的技术，重点讨论使用 DHTML 完成的两项最常见的任务：更改元素样式和修改文档内容。

14.4.1 样式表设置的修改

页面上显示的每个元素(甚至一些不在页面上显示的元素)都有 style 属性。此属性允许脚本访问当前浏览器支持的该元素的所有 CSS 属性。其属性值与 CSS 说明的相同，但其语法往往不同于 HTML 标记特性实现的类似设置。例如，要设置 ID 为 FranklinQuote 的 blockquote 元素的文本颜色，语法为：

```
document.getElementById("FranklinQuote").style.color = "rgb(255, 255, 0)";
```

由于 CSS color 属性接受其他的颜色指定方式(如传统的十六进制三元组—#ffff00)，所以也可以使用这些方式。

但一些 CSS 属性名不符合 JavaScript 命名约定，有个别的 CSS 属性名包含连字符，此时，属性的等价脚本表示会去掉连字符，并将每个词的首字母大写。例如，CSS 属性 font-weight 在脚本中设置为：

```
document.getElementById("highlight").style.fontWeight = "bold";
```

一个相关的技术可以用 CSS 代码实现文档的一些设计。例如，如果为两个不同的类定义了 CSS 规则，就可以通过元素对象的 className 属性来切换应用于元素的类定义。假设定义了两个有不同背景色的 CSS 类：

```
.normal {background-color: #ffffff};
.highlighted {background-color: #ff0000};
```

在 HTML 页面中，元素首先接收其默认的类，如下所示：

```
<p id="news" class="normal">...</p>
```

此后脚本语句可以更改该元素对象的类，将 highlighted 样式应用于该元素：

```
document.getElementById("news").className = "highlighted";
```

恢复原有类名也将恢复其外观。此方法也是用单个语句更改元素的多个样式属性的快捷方式。

14.4.2 通过 W3C DOM 节点实现动态内容

第 10 章介绍了 document.createElement()和 document.createTextNode()方法。这些方法可以创建新的 DOM 对象，随后可以设置其属性来修改对象，再将新对象插入文档树供所有人查看。

为了介绍这种技术，下面列出将一个元素及其文本添加到页面上 span 占位元素的步骤。在

更多资源请访问稀酷客(www.ckook.com)

此示例中，把一个属于 centered 类的段落元素添加到 id 为 placeholder 的 span 元素上。段落内容的部分文本来自于表单中的一个文本域(访问者的名)。以下是完整的代码序列：

```
var newElem = document.createElement("p");
newElem.className = "centered";
var newText = document.createTextNode("Thanks for visiting, " +
   document.forms[0].firstName.value);
// insert text node into new paragraph
newElem.appendChild(newText);
// insert completed paragraph into placeholder
document.getElementById("placeholder").appendChild(newElem);
```

创建元素和文本节点后，必须将文本节点插入元素节点。由于新元素节点在创建时为空，所以使用 DOM 方法 appendChild()将文本节点插入到元素中(在其首尾标记之间)。段落元素组合好后，就插入到最初为空的 span 元素的末尾。其他 W3C DOM 方法(参见第 26 章和第 27 章)提供了插入、删除和替换节点的更多方式。

14.4.3　通过 innerHTML 属性实现动态内容

在 W3C DOM 规范发布之前，Microsoft 发明了一个所有元素对象都有的属性 innerHTML，此属性首次在 Internet Explorer 4 中推出，并由于其实用性而流行起来。该属性的值是一个包含 HTML 标记和其他内容的字符串，如 HTML 文档中当前元素的标记所示。尽管 W3C DOM 工作组没有在其发布的标准中实现这一属性，但此属性仍非常实用、流行，现代浏览器厂商都不能忽视它。它在基于 Mozilla 的浏览器和 Safari 中实现为事实标准。

为说明 W3C DOM 方式和 innerHTML 属性的区别，以下代码示例给出了与前面 W3C DOM 一节相同的内容创建和插入功能，但这次是用 innerHTML 属性实现的：

```
// accumulate HTML as a string
var newHTML = "<p class='centered'>Thanks for visiting, ";
newHTML += document.forms[0].firstName.value;
newHTML += "</p>";
// blast into placeholder element's content
document.getElementById("placeholder").innerHTML = newHTML;
```

innerHTML 版本似乎更直接，HTML 代码编写者更容易可视化所添加的内容，但需要修改文档中的大量内容时，DOM 节点方法更高效。大量的 JavaScript 字符串连接运算会减缓浏览器的脚本处理速度。有时，最短的脚本并不一定最快。

本部分的最后一章到此结束了。读者如果学完每一课并完成了练习，就可以阅读本书的余下部分，学习 DOM 和 JavaScript 语言的细节和更多功能。可以按顺序学习第III部分和第IV部分的章节，还应该看看光盘上的第 48 章，学习一些重要的调试技术。

14.5　习题

1. 说明文档 image 对象和内存 image 对象之间的差别。

更多资源请访问稀酷客(www.ckook.com)

2. 编写 JavaScript 语句，来预缓存图像文件 jane.jpg，这个图像以后用于替换由下列 HTML 定义的 document 图像：

```
<img name="people" src="john.jpg" height="120" width="100" alt="people">
```

3. 使用题 2 中编写的代码，写出用内存图像替换文档图像的 JavaScript 语句。

4. 向下兼容的 img 元素对象没有鼠标事件的处理程序，如何为鼠标滚动触发变换图像所需的脚本？

5. 假设 table 元素包含一个 ID 为 forwardLink 的空表单元格(td)元素。使用 W3C DOM 节点创建技术，编写一系列脚本语句，创建下面的超链接，并插入表单元格：

```
<a href="page4.html">Next Page</a>
```

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第 III 部分

JavaScript 核心语言参考

本部分内容

第 15 章　String 对象

第 16 章　Math、Number 和 Boolean 对象

第 17 章　Date 对象

第 18 章　Array 对象

第 19 章　JSON — Native JavaScript Object Notation

第 20 章　E4X — Native XML Processing

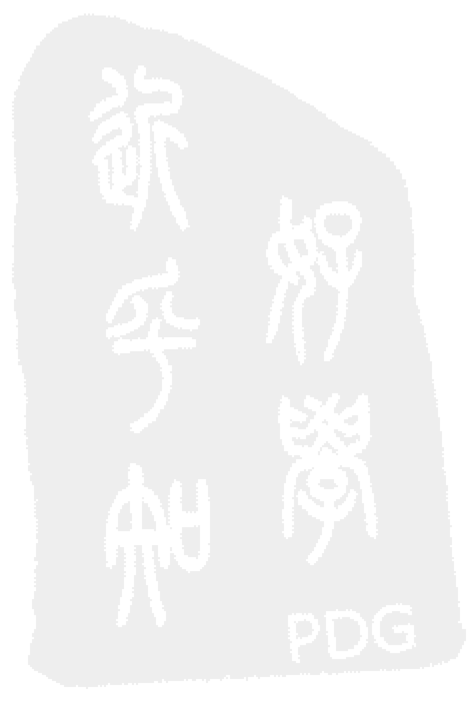
第 21 章　控制结构和异常处理

第 22 章　JavaScript 操作符

第 23 章　函数和自定义对象

第 24 章　全局函数和语句

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第15章

String 对象

第 8 章介绍了 JavaScript 中值的概念和类型，如字符串、数值和 Boolean 值等。第 12 章描述了字符串的几个特征和方法。本章将详细讨论非常重要的字符串数据类型，以及这种类型与数值数据类型的关系，并提供 JavaScript 中使用脚本处理字符串的许多方式。

本章包含哪些内容？
分析和处理文本
执行查找和替换操作
设置文本格式的脚本方式

注意：

本章大部分语法都和 Java 编程语言相同，因为 JavaScript 的应用范围比 Java 窄得多，所以要学习的 JavaScript 知识比 Java 少得多。

15.1 字符串以及数值数据类型

JavaScript 是所谓的“弱类型”语言，但要了解几个数据类型，因为这些数据类型会影响处理该类信息的方法。本节介绍字符串和两个数值类型。

15.1.1 简单字符串

字符串由包含在一对引号内的一个或多个标准文本字符组成。JavaScript 允许使用单引号或双引号，只要字符串外的引号配对即可。在字符串中包含引用文本时，可看出这种方式的另一个优点。例如，假设要在变量中组合一行 HTML 代码，并在 JavaScript 的控制下将其写入新窗口。于是，要赋予变量的文本行如下：

```
<input type="checkbox" name="candy" />Chocolate
```

为了给变量赋予整行文本，需要把这些文本放在引号中。但该行文本中包含引号，JavaScript(或其他语言)就无法确定字符串的开头和结尾。如果小心地使用另一种引号对，就可以成功执行赋值语句。下面是两种有效的方式：

```
result = '<input type="checkbox" name="candy" />Chocolate';
```

PDG

更多资源请访问稀酷客(www.ckook.com)

```
result = "<input type='checkbox' name='candy' />Chocolate";
```

注意在这两个语句中，整个字符串外部的引号对都是唯一的，字符串内两个带引号的字符串由 JavaScript 处理。若使用其中的一种格式，则在整个脚本中都应使用这种格式。

如果要加引号的表达式包含单引号和双引号，就需要用反斜杠转义其中一个引号：

```
Pat's favorite word is "syzygy."
result = "Pat's favorite word is \"syzygy.\""
result = 'Pat\'s favorite word is "syzygy." '
```

转义字符不容易阅读和校对，所以应尽可能地选择在表达式中不使用的引号字符。

15.1.2 建立长字符串变量

将字符串合并在一起(连接)，可以将几个小片段组成一个长字符串。这个功能对某些脚本非常重要，例如将页面写入其他框架之前，需要用一个语句在变量中构建 HTML 页面的详细说明。在一行代码的一个字符串内包括很长的信息通常是不明智、不实用的，因此要通过子字符串来构造大字符串。

在这个构建过程中应使每个语句比较短，以便文本编辑器读取。这种方法使用加值赋值操作符(+=)将操作符右边的部分加到左边。下面的简单示例首先将 newDocument 变量初始化为空字符串：

```
var newDocument = "";
newDocument += "<!DOCTYPE html>";
newDocument += "<html>" ;
newDocument += "<head>";
newDocument += '<meta http-equiv="content-type"';
newDocument += ' content="text/html;charset=utf-8">';
newDocument += "<title>Prodigal Summer</title>";
newDocument += "</head>";
newDocument += "<body>";
newDocument += "<h1>Prodigal Summer</h1>";
newDocument += '<p class="byline">by Barbara Kingsolver</p>';
```

从第二行开始，每个语句都给存储在 newDocument 中的字符串加入更多数据，直到 newDocument 变量包含整个页面的详细说明。

注意：

过度使用加值操作符，会大大降低大量文本的处理效率。在组合大字符串时，如果性能很低，可将字符串段放到数组项中(参见第 18 章)，然后使用数组的 join()方法生成最终的大字符串值。

15.1.3 连接字符串字面量和变量

有时，需要从字符串字面量(包含在引号中的字符)和字符变量值中创建字符串。连接这类字符串与连接多个字符串字面量的方法一样，都是使用加号操作符。因此，在下面的示例中，

更多资源请访问稀酷客(www.ckook.com)

变量 teamName 包含一个名称，而长字符串包含该变量的值和字符串字面量：

```
teamName = prompt("Please enter your favorite team:","");
var msg = "The " + teamName + " are victorious!";
alert(msg);
```

在进行连接操作时，经常遇到的问题包括：

- 忘记了字面量字符串一边的引号。
- 没有在字符串字面量中插入空格，将各个单词分隔开。
- 忘记连接变量值后面的标点符号。

另外，本例的变量值可以是值为字符串的任何表达式，包括属性引用和一些方法的结果。例如：

```
var msg = "The name of this document is " + document.title + ".";
alert(msg);
```

15.1.4　特殊的内嵌字符

JavaScript 创建字符串字面量的方式，使得很难在字符串中添加某些字符——主要是引号、回车、撇号及 Tab 字符。但 JavaScript 提供了一种将这些字符加入字符串字面量的机制：在反斜杠符号的后面加上要显示为内嵌的字符。对于“不可见”的字符，反斜杠后面的特殊字符集会将做法告诉 JavaScript。

最常用的反斜杠对如下：

- \"双引号
- \'单引号(撇号)
- \\反斜杠
- \b 退格
- \t Tab
- \n 换行
- \r 回车
- \f 换页

在带引号的字符串字面量中使用这些“内嵌字符”(有时称为“转义字符”，但对于 Internet 字符串，这个称呼有不同的含义)，JavaScript 就可以识别它们。如果组合需要新段落的文本块，可以插入\n 字符对。下面的示例使用了这些特殊字符：

```
msg = "I say \"trunk\" and you say \"boot.\"";
msg = 'You\'re doing fine. ';
msg = "This is the first line.\nThis is the second line.";
msg = document.title + "\n" + document.links.length + " links present.";
```

在技术上，打字时，完整的回车包含一个换行符(另起一行)和一个回车符(将回车移动到最左边)。虽然 JavaScript 字符串将换行(\n)作为完整的回车符，但要组合一个字符串以返回给服务器上的 cgi 脚本，应该使用\r\n，其格式取决于 cgi 程序的字符串解析能力(参见配书光盘上第 36 章中对 textarea 对象的特殊要求)。

更多资源请访问稀酷客(www.ckook.com)

为 textarea 对象和警告框组合的字符串很容易与编写为 HTML 的字符串混淆。在大多数情况下，浏览器会忽略回车符，或者把它们显示为空格。对于 HTML 字符串，要使用标准的 HTML 标记，例如段落间隔标志(<p>…</p>)和行间断标记(<br>)，而不是内嵌回车或者换行符号。

15.2　String 对象

属　性	方　法	属　性	方　法
Constructor	anchor()		replace()
Length	big()		search()
Prototype*	blink()		slice()
	bold()		small()
	charAt()		split()
	charCodeAt()		strike()
	concat()		sub()
	fixed()		substr()
	fontcolor()		substring()
	fontsize()		sup()
	fromCharCode()*		toLocaleLowerCase()
	index0f()		toLocaleUpperCase()
	italics()		toLowerCase()
	lastIndex0f()		toString()
	link()		toUpperCase()
	localeCompare()		valueOf()
	match()		

*静态 String 对象的成员。

15.2.1　语法

创建 String 对象：

```
var myString = new String("characters");
```

创建字符串值：

```
var myString = "characters";
```

访问静态 String 对象的属性和方法：

```
String.property | method([parameters])
```

访问 String 对象的属性和方法：

PDG

更多资源请访问稀酷客(www.ckook.com)

```
string.property | method([parameters])
```

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

15.2.2　关于 String 对象

JavaScript 在字符串值和 String 对象之间划出了很清晰的界限。它们允许使用相同的方法处理其内容，所以在每次为变量赋字符串值时，基本上不用创建 String 对象(用 new String ()构造函数)。简单的赋值操作(var myString= "fred")就可以创建字符串值，这个字符串值看起来就像完整的 String 对象一样。

使用 String 对象的对象特征时，字符串值和 String 对象就有了区别，这将在本章后面讨论 string.prototype 属性时详细解释。向 Java applet 传递字符串数据时，也可能需要使用完整的 String 对象。若 applet 不把字符串值接收为 Java 的 String 数据类型，则应该在给 applet 传递值之前，用 JavaScript 构造函数创建新的 String 对象。

字符串数据通常用来在脚本中传送字符串文本。除连接字符串外，有时还需要提取字符串片段，删除字符串的一部分，或者用其他文本替换部分字符串。与许多纯粹的脚本语言不同，JavaScript 只内置了一些低级的字符串操作功能。也就是说除非利用 IE4+/Mozl+的正则表达式功能或高级数组技术，否则必须从 JavaScript 内嵌的基本功能中创建自己的字符串处理例程。本章后面提供了几个函数，可以用在脚本中，以与旧浏览器完全兼容的方式对字符串进行通用处理。

在处理字符串的值时，应将字符串值看作拥有属性和方法的对象，就像其他 JavaScript 对象一样。JavaScript 定义了字符串值的一些方法和属性(还有一个属性属于静态 String 对象，它总是出现在浏览器窗口的内容中)，字符串方法的语法和其他对象方法相同：

```
stringObject.method()
```

这个引用的 *stringObject* 部分可以是任何值为字符串的表达式，包括字符串字面量、包含字符串的变量、返回字符串值的函数或方法或者其他对象属性。因此，下面几个调用 toUpperCase()方法的示例都是正确的：

```
"blah blah blah".toUpperCase()
yourName.toUpperCase() // yourName is a variable containing a string
window.prompt("Enter your name","").toUpperCase()
document.forms[0].entry.value.toUpperCase() // entry is text field object
```

注意，调用字符串方法并不改变引用中的 String 对象；相反，该方法返回一个字符串值，该值可以作为其他方法或函数的参数，也可以赋给一个变量。

所以，要将字符串变量的内容改为方法的结果，必须使用赋值操作符，如：

```
yourName = yourName.toUpperCase(); // variable is now all uppercase
```

更多资源请访问稀酷客(www.ckook.com)

15.2.3　属性

constructor

值：函数引用，　　读/写

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

constructor 属性是用来创建当前字符串的函数的引用。对基本的 JavaScript 字符串对象，构造函数是内置的 String()构造函数。

当使用 new String()构造函数创建 String 对象时，构造函数的返回值是 object 类型(这意味着 typeof 操作符返回 object)。因此，可使用 constructor 属性判断对象是否为 String 对象：

```
if (typeof someValue == "object" )
{
   if (someValue.constructor == String)
   {
      // statements to deal with string object
   }
}
```

虽然 constructor 属性可以读/写，并可以为 String.prototype 分配不同的构造函数，但 String 对象的内在行为仍由 new 构造函数决定。

示例

使用第 4 章介绍的 The Evaluator，测试 constructor 属性的值。在顶端的文本框中输入下面的语句，一次一行，并求值：

```
a = new String("abcd")
a.constructor == String
a.constructor == Number
```

相关主题：prototype 属性。

Length

值：整型，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

最常用的字符串属性是 length。要得到字符串长度，可读取字符串的 length 属性，就像读取其他对象的 length 属性一样：

```
string.length
```

length 值表示字符串包含的字符数，这是一个整数。空格和标点符号也算作字符，嵌入字符串的反斜杠特殊符号也算作一个字符，包括换行符和 Tab。下面是一些示例：

```
"Lincoln".length // result = 7
"Four score".length // result = 10
"One\ntwo".length // result = 7
"".length // result = 0
```


更多资源请访问稀酷客(www.ckook.com)

在重复循环中进行具体的字符串操作时，通常要用到 length 属性。例如，如果要遍历字符串中的每个字符，并检查或修改每个字符，就可以把字符串的长度用作循环计数器的基础。

prototype

值：String 对象，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

与赋予字符串值的简单变量相比，new String("*stringValue*")构造函数定义的 string 对象更为强大。不一定要在脚本中为每个字符串创建这种 String 对象，但如果变量中的字符串出错，使用这些对象会很方便。试图将字符串信息保存为其他窗口或框架中的脚本变量时，偶尔需要使用 String 对象。使用 String 对象构造函数，可以保证字符串值在需要时可用于其他框架。

使用 String 对象的另一个优点在于，可以给文档中所有的 String 对象分配原型属性和方法。prototype(原型)是指一个属性和方法集合，增加该原型项后，它会成为任何新建对象的属性和方法。例如，可以定义新的方法来传输 JavaScript String 对象未定义的字符串。程序清单 15-1 演示了如何创建和应用原型。

注意：

本书和本章代码使用的属性赋值事件处理技术是 addEvent()，这个跨浏览器的事件处理程序详见第 32 章。

addEvent()函数在脚本文件 jsb-global.js 中，该文件位于配书光盘中，所有章节的脚本都可以访问这个文件。

程序清单 15-1　使用 String 对象原型

HTML: jsb-15-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Using the String Object Prototype</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-01.js"></script>
   </head>
   <body>
      <h1>Using the String Object Prototype</h1>
      <form action="initialCap.php">
         <p>
            <label for="entry">Enter one or more words to be Initial Capped:</label>
         </p>
         <p>
            <input type="text" id="entry" name="entry" value="">
            <input type="button" id="do-it" value="Initial Caps">
         </p>
      </form>
   </body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

JavaScript: jsb-15-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to critical elements
      var oButton = document.getElementById('do-it');
      var oEntry = document.getElementById('entry');

         // if they all exist...
         if (oButton && oEntry)
         {
            // apply behavior to button
            addEvent(oButton, 'click', doIt);
         }
   }
}

// try out the new String method
function doIt()
{
   // point to input field
   var oEntry = document.getElementById('entry');

   // use the new method to update the input text
   oEntry.value = oEntry.value.initialCaps();
}

// add a new method to the String prototype
// to capitalize the first letter of each word

String.prototype.initialCaps = function()
{
   // get the input text
   var sText = this.toString();

   // separate individual words
   var aWords = sText.split(' ');

   // capitalize each word
   for (var i = 0; i < aWords.length; i++)
   {
      var sInitial = aWords[i].charAt(0);
      var sRemainder = aWords[i].substr(1);
      aWords[i] = sInitial.toUpperCase() + sRemainder.toLowerCase();
   }
```


更多资源请访问稀酷客(www.ckook.com)

```
    // reassemble the array of words back into a string
    return aWords.join(' ');
}
```

注意：

本书的许多 HTML 示例页面都变换一个表单动作：把输入粘贴到一个服务器端 PHP 程序中，以说明即使不能运行 JavaScript，也一定要确保页面能起作用。服务器端脚本并不存在，它超出了本书的讨论范围。如果编写了这个程序，它们应该执行与 JavaScript 逻辑相同的操作。

程序清单 15-1 中 JavaScript 代码的核心是最后一个语句块，它给 String 对象原型添加了 initialCaps()方法，并说明了该方法的作用。与大多数对象方法一样，这个新方法也没有修改原字符串，而是返回一个可用于任何目的的值。这里提取输入域 entry 的内容，使用 initialCaps()对其进行转换，再使用这个新值重置输入域的内容。

修改对象原型是随时给 JavaScript 语言添加新功能的一种强大方式。

下面将 String 对象方法分为两类，第一类是解析方法，主要是字符串的分析和字符串中字符的处理；第二类是格式化方法，主要介绍用 HTML 语法在脚本中组合字符串，然后将其写入新的文档或其他框架。

15.2.4 解析方法

string.charAt(*index*)

返回值：单字符的字符串

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

如果知道某个字符的位置，就可以使用 *string*.charAt()方法从字符串中读取这个字符。这个方法应把字符串中的一个索引值指定为参数。字符串中首字符的索引值是 0。要得到字符串的最后一个字符，需要使用多个字符串方法：

```
myString.charAt(myString.length - 1)
```

若脚本需要得到大量字符，应使用 *string*.substring()方法。一般不使用 *string*.substring()从字符串中提取单个字符，此时 *string*.charAt()方法更有效。

示例

在 The Evaluator 的顶端文本框中，输入下面的语句：

```
a = "banana daiquiri"
a.charAt(0)      // result = 'b'
a.charAt(5)      // result = 'a' (the third 'a')
a.charAt(6)      // result = '' (space)
a.charAt(20)     // result = '' (empty string)
```

相关主题：*string*.lastIndexOf()、*string*.indexOf()和 *string*.substring()方法。

string.charCodeAt([*index*])，*string*.fromCharCode(*num1* [, *num2* [, ... *numn*]])

更多资源请访问稀酷客(www.ckook.com)

返回值：若参数是单个字符，就返回该字符的整数代码号；若参数是一系列代码号，就返回连接起来的字符串值。

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

将普通的字符转化为等价的数值是计算机编程的习惯。多年来，最常用的编号机制是 ASCII 标准，它将基本的英语字母、数字字符和标点符号用 128 个值表示(0~127)。它的扩展版本共有 256 个字符(它有一些变体，这取决于操作系统)，包括了其他语言中的拉丁字符，特别是重音元音。为了支持所有的语言，包括象形语言和其他非拉丁字母表，人们制定了 Unicode 标准，它包含上千个字符。所有现代浏览器都使用 Unicode 系统。

在 JavaScript 中，字符转换由 string 方法完成。执行字符转换的两个方法以非常不同的方式进行。第一个方法 string.charCodeAt()将一个字符串转换为等效的数字。被转换的字符串位于方法名的左边，这个字符串可以是字面量字符串，也可以是计算结果为字符串值的其他任何表达式。如果没有传递参数，被转换的字符默认为字符串的第一个字符。另外，还可以为字符串中的另一个字符指定索引值(第 1 个字符的索引值为 0)，如下所示：

```
"abc".charCodeAt()  // result = 97
"abc".charCodeAt(0) // result = 97
"abc".charCodeAt(1) // result = 98
```

若字符串值是空，或索引值超出了最后一个字符，结果就是 NaN。

要将数值转换成字符，应使用 String.fromCharCode()方法。注意调用此方法的是静态 String 对象，而不是字符串值。该方法的参数可以是一个整数值或用逗号分开的多个整数值。在转换过程中，该方法将所有参数表示的字符组合为一个字符串。下面是一个示例：

```
String.fromCharCode(97, 98, 99) // result "abc"
```

注意：

尽管多数现代浏览器都支持 Unicode 中的所有字符值，但浏览器不会正确显示超过 255 的字符，除非计算机装有支持语言和指定语言的字体。为了显示 Unicode 字符，所有 HTML 文档都应先声明 UTF-8 字符编码。

示例

程序清单 15-2 在一个页面上提供了两个方法的一些示例。此外，因为其中一个示例依赖于页面上所选文本的自动捕获，所以脚本包括一些代码，以在各种浏览器中对选择事件进行不同的处理，并捕获所选的文本。

在加载页面后，选择页面上主体文本的部分。如果首先选择小写字母“a”，字符码就显示为 97。

在页面底部的三个域中尝试输入数字值。低于 32 的值是 ASCII 控制符，多数字体将它们显示为空白或空心框。尝试其他值，并查看结果。注意，这个脚本将这三个值作为一组传递给 String.fromCharCode()的方法，结果是一个组合字符串。因此，图 15-1 显示了为一个 3 字母动物名输入大写 ASCII 值时的结果。

更多资源请访问稀酷客(www.ckook.com)

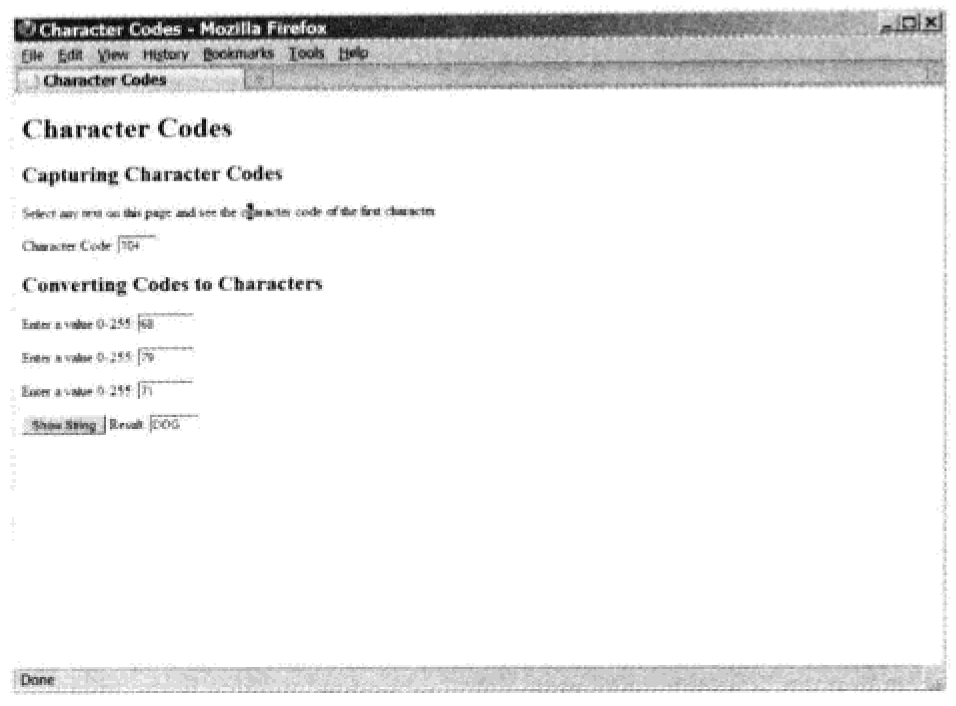


图 15-1　从文本字符转换为 ASCII 值，从 ASCII 值转换为文本字符

程序清单 15-2　字符转换

HTML: jsb-15-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Character Codes</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-02.js"></script>
   </head>
   <body>
      <h1>Character Codes</h1>

      <form action="show-string.php">
         <h2>Capturing Character Codes</h2>
         <p>Select any text on this page and see the character code of the first
         character.</p>
         <p>
            <label for="display">Character Code:</label>
            <input type="text" id="display" name="display" size="3" />
         </p>

         <h2>Converting Codes to Characters</h2>
         <p>
            <label for="entry1">Enter a value 0-255:</label>
            <input type="text" id="entry1" name="entry1" size="6" />
         </p>
         <p>
            <label for="entry2">Enter a value 0-255:</label>
            <input type="text" id="entry2" name="entry2" size="6" />
         </p>
         <p>
```


更多资源请访问稀酷客(www.ckook.com)

```
                <label for="entry3">Enter a value 0-255:</label>
                <input type="text" id="entry3" name="entry3" size="6" />
            </p>
            <p>
                <input type="button" id="showstr" value="Show String" />
                <label for="result">Result:</label>
                <input type="text" id="result" name="result" size="5" />
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-15-02.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

// show character code when text is selected
addEvent(document, 'mouseup', showCharCode);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {

        // point to critical element
        var oButton = document.getElementById('showstr');

            // if it exists...
            if (oButton)
            {
                // apply behavior to button
                addEvent(oButton, 'click', showString);
            }
    }
}

// show the text characters represented by the numbers entered
function showString()
{
    var oEntry1 = document.getElementById('entry1');
    var oEntry2 = document.getElementById('entry2');
    var oEntry3 = document.getElementById('entry3');
    var oResult = document.getElementById('result');

    oResult.value = String.fromCharCode(oEntry1.value, oEntry2.value, oEntry3.value);
}
// display the character code of the selected text
// (first character only)
function showCharCode()
{
    var theText = "";
```


更多资源请访问稀酷客(www.ckook.com)

```
    var oDisplay = document.getElementById('display');

      // get the selected text using various browsers' methods
      if (window.getSelection)
      {
         theText = window.getSelection().toString();
      }
      else if (document.getSelection)
      {
         theText = document.getSelection();
      }
      else if (document.selection && document.selection.createRange)
      {
         theText = document.selection.createRange().text;
      }

      // display the result if any
      if (theText)
      {
         oDisplay.value = theText.charCodeAt();
      }
      else
      {
         oDisplay.value = " ";
      }
}
```

string.concat(*string*2)

返回值：组合的字符串

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

JavaScript 的加值操作符(+=)提供了一种连接字符串的简便方式。但大多数浏览器都使用一个 String 对象方法来完成这个任务。基本字符串(向其添加文本)是放在句点左边的对象或值。方法的参数是要添加的字符串，如下：

```
"abc".concat("def") // result: "abcdef"
```

与加值操作符一样，concat()方法不处理字之间的空格。如果两个字符串中间需要空格，就必须在两个字之间添加空格。

相关主题：加值(+=)操作符。

string.indexOf(*searchString* [, *startIndex*])

返回值：*searchString* 的首字符在字符串中的索引值

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

类似于一些编程语言中的偏移字符串函数，JavaScript 的 indexOf()方法可以得到搜索字符串的首字符在主字符串中的位置。也可以酌情指定在主字符串中开始搜索的位置，但返回值总是相对于主字符串的首字符。像所有的 string 对象方法一样，索引值从 0 开始计数。若在主字

更多资源请访问稀酷客(www.ckook.com)

符串中找不到搜索字符串，返回值就是-1。因此可以使用这个方法方便地判断一个字符串是否包含另一个字符串，无论其位置如何。

示例

在 The Evaluator 的顶端文本框中输入下面的每个语句(输入到“//”注释符号之前)，对于第一个语句后的每个语句，可以只替换 indexOf()方法的参数，将得到的结果与下面的结果进行比较：

```
a = "bananas"
a.indexOf("b")        // result = 0 (index of 1st letter is zero)
a.indexOf("a")        // result = 1
a.indexOf("a",1)      // result = 1 (start from 2nd letter)
a.indexOf("a",2)      // result = 3 (start from 3rd letter)
a.indexOf("a",4)      // result = 5 (start from 5th letter)
a.indexOf("nan")      // result = 2
a.indexOf("nas")      // result = 4
a.indexOf("s")        // result = 6
a.indexOf("z")        // result = -1 (no "z" in string)
```

相关主题：*string*.lastIndexOf()、*string*.charAt()和 *string*.substring()方法。

string.lastIndexOf(*searchString*[, *startIndex*])

返回值：*searchString* 的首字符在字符串中最后一次匹配的索引值

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

string.lastIndexOf()方法与 *string*.indexOf()的关系十分密切，唯一的区别是它从字符串尾部(*string*.length-1)开始向前搜索匹配的字符，索引值仍然从字符串开头计数，从 0 开始。下面的示例与 *string*.indexOf()示例使用相同的值，以便比较两个方法的结果。如果只找到了搜索字符串的一个实例，两者的结果就是相同的；但存在搜索字符串的多个实例时，结果就完全不同——因此这个方法是必需的。

示例

在 The Evaluator 的顶端文本框中输入下面的语句(输入到“//”注释符号前即可)，对第一个语句后的每个语句，可以只替换 lastIndexOf()方法的参数，将得到的结果与下面的结果比较：

```
a = "bananas"
a.lastIndexOf("b")        // result = 0 (index of 1st letter is zero)
a.lastIndexOf("a")        // result = 5
a.lastIndexOf("a",1)      // result = 1 (from 2nd letter toward the front)
a.lastIndexOf("a",2)      // result = 1 (start from 3rd letter working toward front
a.lastIndexOf("a",4)      // result = 3 (start from 5th letter)
a.lastIndexOf("nan")      // result = 2 [except for -1 Nav 2.0 bug]
a.lastIndexOf("nas")      // result = 4
a.lastIndexOf("s")        // result = 6
a.lastIndexOf("z")        // result = -1 (no "z" in string)
```

相关主题：*string*.lastIndexOf()、*string*.charAt()和 *string*.substring()方法。

更多资源请访问稀酷客(www.ckook.com)

string.localeCompare(*string2*)

返回值：整型

兼容性：WinIE5.5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

localeCompare()方法允许脚本比较两个字符串的累加 Unicode 值，它考虑了浏览器的语言系统，只有几个民族的语言需要这个方法(土耳其语是其中一种)。若两个字符串(已针对语言系统进行了调整)相同，则返回值是 0；若调用方法的字符串值(即句点左边的字符串)排在参数字符串的前面，那么返回值是负整数，否则返回值是正整数。

该方法的 ECMA 标准将返回正数或负数的决定权交给浏览器的设计者；NN6+为两个字符串计算累加的 Unicode 值，并从字符串值的总和中减去字符串参数的总和；而 IE5.5+和 FF1+在字符串不相等时，返回-1 或 1。

相关主题：*string*.toLocaleLowerCase()、*string*.toLocaleUpperCase()方法。

string.match(*regExpression*)

返回值：匹配的子字符串数组

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

string.match()方法依赖 RegExp 对象(正则表达式)在字符串中进行匹配。待检查的字符串值在圆点的左边，而方法使用的正则表达式是参数。参数必须是一个正则表达式对象，它用两种方式创建。

至少有一个匹配时，这个方法返回数组值；否则，返回值是 null。数组中的每一项是符合正则表达式规则的部分字符串的副本。使用该方法可以显示子字符串或字符序列在大字符串中出现的次数。要查找匹配的偏移位置，应使用其他字符串解析方法。

示例

为了帮助理解 *string*.match()方法，程序清单 15-3 提供了一个正则表达式练习。有 3 个输入控件用于测试：第一个控件可输入方法要检查的长字符串，第二个输入正则表达式，第三个是一个复选框，用于指定字符串匹配是否考虑大小写。现在，因为读者还不熟悉正则表达式的语法(参见配书光盘中的第 45 章)，所以提供了一些默认值。单击 Execute match ()按钮后，脚本根据输入创建一个正则表达式对象，对大字符串执行 *string*.match()方法，并在页面上报告两类结果。主要结果是方法返回的数组的字符串版本，另一个结果是返回项的数目。

程序清单 15-3　正则表达式匹配练习

HTML：jsb-15-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Regular Expression Match Workshop</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-03.js"></script>
   </head>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <body>
        <h1>Regular Expression Match Workshop</h1>

        <form action="regexp-match.php">
            <p>
                <label for="string">Enter a main string:</label>
                <input type="text" id="string" name="string" size="60"
                value="Many a maN and womAN have meant to visit GerMAny." />
            </p>
            <p>
                <label for="pattern">Enter a regular expression to match:</label>
                <input type="text" id="pattern" name="pattern" size="25" value="\wa\w" />

                <input type="checkbox" id="caseSens" name="caseSens" />
                <label for="caseSens">Case-sensitive</label>
            </p>
            <p>
                <input type="button" id="button" value="Execute match()" />
                <input type="reset" />
            </p>
            <p>
                <label for="result">Result:</label>
                <input type="text" id="result" name="result" size="40" />
            </p>
            <p>
                <label for="count">Count:</label>
                <input type="text" id="count" name="count" size="3" />
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-15-03.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to critical element
        var oButton = document.getElementById('button');

            // if it exists...
            if (oButton)
            {
                // apply behavior to button
                addEvent(oButton, 'click', doMatch);
            }
    }
}
```


更多资源请访问稀酷客(www.ckook.com)

```
// perform regular expression match using input values
function doMatch()
{
   // gather input values
   var oString = document.getElementById('string');
   var sString = oString.value;

   var oPattern = document.getElementById('pattern');
   var oCaseSensitivity = document.getElementById('caseSens');
   var sFlags = 'g'; // g = find all
      if (!oCaseSensitivity.checked) sFlags += 'i'; // i = case-insensitive

   // create regular expression object
   var oRegExp = new RegExp(oPattern.value, sFlags);

   // perform match
   var aResult = sString.match(oRegExp);

   // display results
   var oResult = document.getElementById('result');
   var oCount = document.getElementById('count');

      // got matches?
      if (aResult)
      {
         oResult.value = aResult.toString();
         oCount.value = aResult.length;
      }
      else
      {
         oResult.value = "<no matches>";
         oCount.value = "";
      }
}
```

主要字符串的默认值特意设置了特殊的大写方式。这样，可以清楚地看出一些匹配来自哪里。例如，默认的正则表达式查找中间字母是 a 的 3 字符字符串。有 6 个子字符串匹配这个表达式。由于采用了大写形式，所以可以看出，4 个包含 man 的字符串是从主字符串的什么地方提取出来的。表 15-1 列出了默认主字符串可以尝试使用的其他一些正则表达式。

表 15-1 正则表达式

正则表达式	说　明
man	区分大小写，以及不区分大小写
man\b	man 在词尾
\bman	man 在词头
me*an	在 m 和 a 之间有 0 个或多个 e 字母
.a.	a 的两边是任何单字符，包括空格
\sa\s	a 的两边都是一个空格
z	z 出现的位置(默认字符串中没有)

更多资源请访问稀酷客(www.ckook.com)

在程序清单 15-3 的脚本中，如果 *string*.match()方法返回 null，则应通知用户，且计数域为空。

相关主题：RegExp 对象(参阅配书光盘中的第 45 章)。

string.replace(*regExpression, replaceString*)

返回值：修改后的字符串

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

正则表达式常用来处理查找替换操作。JavaScript 的 *string*.replace()方法与 *string*.search()方法一起提供了一个对任意字符串进行此类操作的简单框架。

使用 RegExp 执行替换操作需要 3 个部分，第一部分是操作目标，即主字符串；第二部分是要查找的正则表达式模式；第三部分是用来替换找到的文本实例的字符串。

```
stringToSearch.replace(pattern, replacementString);
```

对于 *string*.replace()方法，主字符串是句点左边的字符串值或引用的对象，也可以是字面量字符串(即用引号括起来的文本)。要搜索的正则表达式是第一个参数，而替换字符串是第二个参数。

正则表达式的定义决定了是替换主字符串中的第一个匹配还是替换全部匹配。若在正则表达式后面加上 g 标志，replace ()方法便进行整个主字符串的查找替换。

只要知道如何生成正则表达式，就很容易使用 *string*.replace()方法处理简单的替换操作。但使用正则表达式可以执行更强大的操作。例如下面的句子：

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

若用 exist 替换 be，可指定：

```
var regexp = /be/g;
soliloquy.replace(regexp, "exist");
```

但不能保证把 be 看作一个独立的单词。如果主字符串包含字 being 或 saber，上面的示例也会将其中的 be 字母替换掉。

使用特殊字符有助于更好地定义要查找的内容。下面的示例查找 be，所以正则表达式使用字边界包括要查找的文本(特殊字符 \b)，如：

```
var regexp = /\bbe\b/g;
soliloquy.replace(regexp, "exist");
```

注意，前两个 be 后面跟的是标点符号，而非空格。正则表达式的更多内容请参见配书光盘中的第 45 章。

示例

程序清单 15-4 创建的页面可在友好的环境中练习 *string*.replace()、*string*.search()方法以及正则表达式。源文本是从“哈姆雷特”中摘录的 5 行内容。在页面中可以输入要搜索的正则表达式和替换文本。注意，脚本创建了正则表达式对象，这样，可以只考虑用来定义匹配字符串的其他特殊字符。所有的替换活动都是全局范围的，因为 g 标志会自动附加到输入的任何表达式上。

更多资源请访问稀酷客(www.ckook.com)

在单击 Execute replace()按钮后，域中的默认值就用 it is 替换了缩写形式的'tis(参见图 15-2)。注意，'tis(在 mainString 中组合的字符串)前面的反斜杠使撇号变成一个非字边界，这样，\B't 正则表达式就可以在那里找到一个匹配。前面在 *string*.search()方法一节中介绍过，与该方法连接的按钮返回匹配字符串的偏移字符数(如果没有匹配，则为-1)。

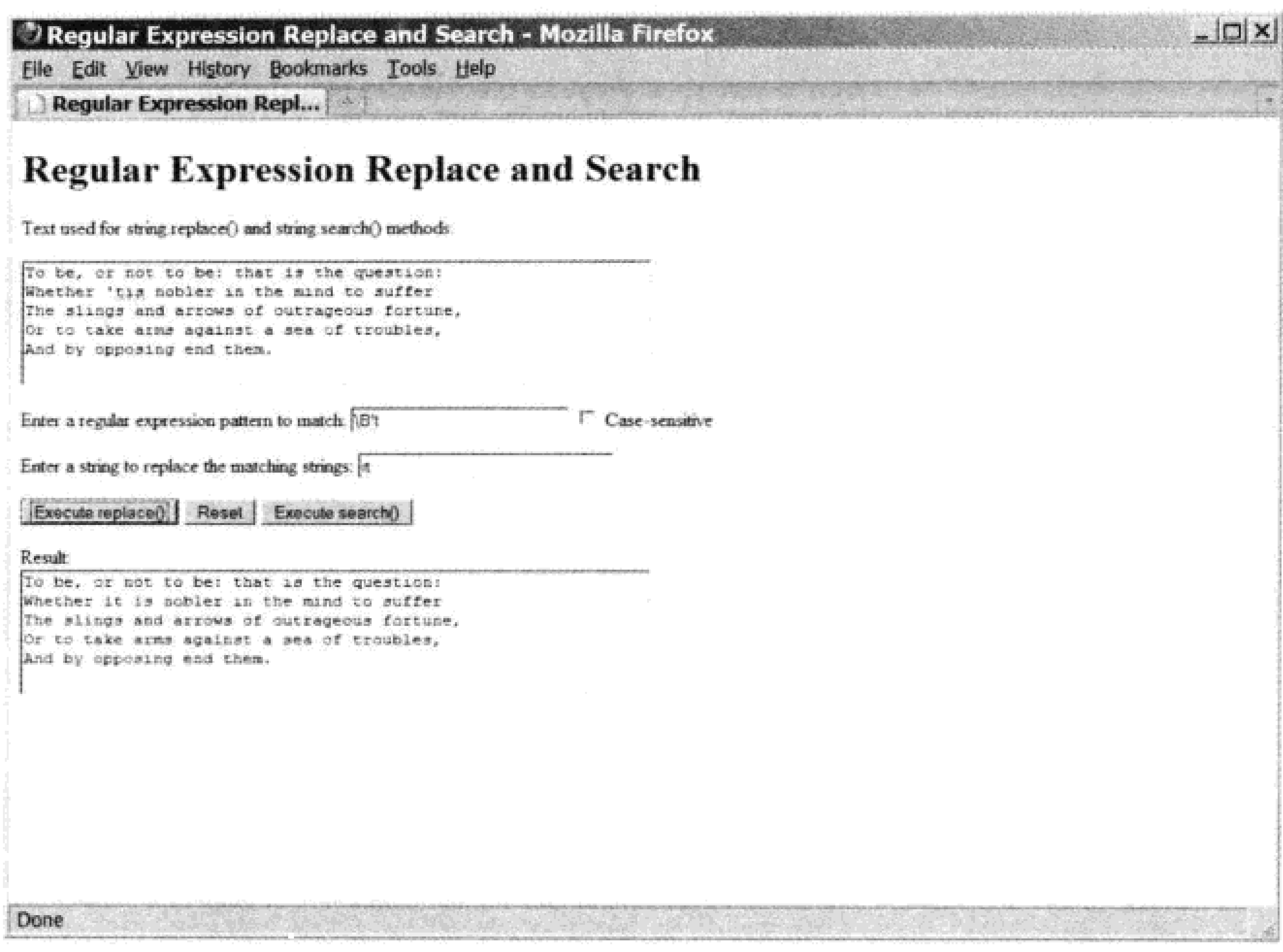


图 15-2 使用默认的替换正则表达式

程序清单 15-4 练习 *string*.replace()和 *string*.search()

HTML:jsb-15-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Regular Expression Replace and Search</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-04.js"></script>
   </head>
   <body>
      <h1>Regular Expression Replace and Search</h1>

      <form action="regexp-replace.php">
         <p>
            <label for="source">Text used for string.replace() and
                        string.search() methods:</label>
         </p>
         <p>
            <textarea id="source" name="source" cols="60" rows="5">
To be, or not to be: that is the question:
```


更多资源请访问稀酷客(www.ckook.com)

```
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them.
</textarea>
         </p>
         <p>
           <label for="pattern">Enter a regular expression pattern to match:</label>
           <input type="text" id="pattern" name="pattern" size="25" value="\B't" />

           <input type="checkbox" id="caseSens" name="caseSens" />
           <label for="caseSens">Case-sensitive</label>
         </p>
         <p>
           <label for="replacement">Enter a string to replace
                        the matching strings:</label>
           <input type="text" id="replacement" name="replacement"
                        size="30" value="it " />
         </p>
         <p>
           <input type="button" id="button-replace" value="Execute replace()"
                        onclick="doReplace(this.form)" />
           <input type="reset" />
           <input type="button" id="button-search" value="Execute search()"
                        onclick="doSearch(this.form)" />
         </p>
         <p>
           <label for="result">Result:</label><br />
           <textarea id="result" name="result" cols="60" rows="5"></textarea>
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-15-04.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // check for critical elements
      var oSource = document.getElementById('source');
      var oPattern = document.getElementById('pattern');
      var oCaseSensitivity = document.getElementById('caseSens');
      var oReplacement = document.getElementById('replacement');
      var oButtonReplace = document.getElementById('button-replace');
      var oButtonSearch = document.getElementById('button-search');
      var oResult = document.getElementById('result');
```


更多资源请访问稀酷客(www.ckook.com)

```
        // if they all exist...
        if (oSource && oPattern && oCaseSensitivity && oReplacement
           && oButtonReplace && oButtonSearch && oResult)
        {
           // apply behaviors
           addEvent(oButtonReplace, 'click', doReplace);
           addEvent(oButtonSearch, 'click', doSearch);
        }
        else
        {
           alert('Critical element not found');
        }
   }
}

function doReplace()
{
   var sSource = document.getElementById('source').value;
   var sReplacement = document.getElementById('replacement').value;
   var sPattern = document.getElementById(' pattern' ).value;
   var sFlags = 'g'; // g = find all; i = case-insensitive
      if (!document.getElementById('caseSens').checked) sFlags += 'i';

   // create regular expression object
   var oRegExp = new RegExp(sPattern, sFlags);

   // perform replace
   var sResult = sSource.replace(oRegExp, sReplacement);

   // display results
   document.getElementById('result').value = sResult;
}

function doSearch()
{
   var sSource = document.getElementById('source').value;
   var sPattern = document.getElementById('pattern').value;
   var sFlags = 'g';    // g = find all; i = case-insensitive
      if (!document.getElementById('caseSens').checked) sFlags += 'i';

   // create regular expression object
   var oRegExp = new RegExp(sPattern, sFlags);

   // perform replace
   var sResult = sSource.search(oRegExp);

   // display results
   document.getElementById('result').value = sResult;
}
```

相关主题：*string*.match()方法；RegExp 对象。

string.search(*regExpression*)

更多资源请访问稀酷客(www.ckook.com)

返回值：整数偏移量

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

string.search()方法的结果与 *string*.indexOf()方法相同，它们的返回值都是匹配字符串在主字符串中第一次出现时的字符编号。如果没有匹配，则返回-1。当然最大的区别是 *string*.search()的匹配字符串是正则表达式。

示例

程序清单 15-4 演示了 *string*.replace()方法，也提供了使用 *string*.search()方法的练习。

相关主题：*string*.match()方法；RegExp 对象。

string.slice(*startIndex* [, *endIndex*])

返回值：字符串

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

string.slice()方法像 *string*. substring ()一样，可以提取字符串的一个部分，并创建一个新的字符串作为结果，但不改变原字符串。但 *string*.slice()方法有一个很好的改进：它更容易指定相对于主字符串结尾处的索引值。

用 *string*.substring()来提取一个在主字符串结束前结束的子字符串需要一些技巧，如下所示：

```
string.substring(4, (string.length-2))
```

还可以给 *string*.slice ()的第二个参数指定一个负数，来表示从字符串结尾偏移：

```
string.slice(4, -2)
```

第二个参数是可选的，如果省略第二个参数，返回值就是主字符串中从开始偏移值到结尾的字符串。

示例

使用程序清单 15-5，可以尝试 *string*.slice()方法的几个参数组合，如图 15-3 所示。这里提供了基本字符串和字符数。请选择不同的参数，研究提取的结果。

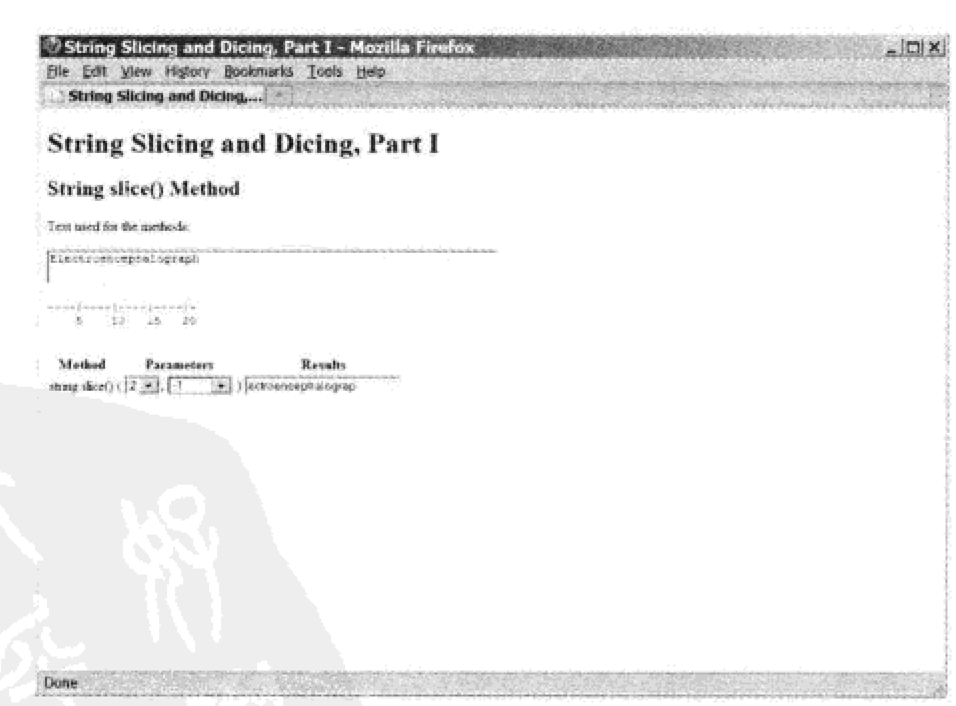


图 15-3 练习 string.slice()方法

更多资源请访问稀酷客(www.ckook.com)

程序清单 15-5 提取字符串

HTML: jsb-15-05.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>String Slicing and Dicing, Part I</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-05.js"></script>
   </head>
   <body>
   <h1>String Slicing and Dicing, Part I</h1>
      <b>String slice() Method</b>

      <form action="string-slice.php">
         <p>
            <label for="source">Text used for the methods:</label>
         </p>
         <p>
            <textarea id="source" name="source" cols="60" rows="1">
Electroencephalograph</textarea>
            <pre>
----|----|----|----|-
    5    10   15   20
            </pre>
         </p>

         <table>
            <thead>
               <tr>
                  <th>Method</th>
                  <th>Parameters</th>
                  <th>Results</th>
               </tr>
            </thead>
            <tbody>
               <tr>
                  <td>string.slice()</td>
                  <td>(
                     <select id="parameter1" name="parameter1">
                        <option value="0">0</option>
                        <option value="1">1</option>
                        <option value="2">2</option>
                        <option value="3">3</option>
                        <option value="5">5</option>
                     </select>,
                     <select id="parameter2" name="parameter2">
                        <option>(None)</option>
                        <option value="5">5</option>
```


更多资源请访问稀酷客(www.ckook.com)

```
                  <option value="10">10</option>
                  <option value="-1">-1</option>
                  <option value="-5">-5</option>
                  <option value="-10">-10</option>
               </select>
            )</td>
            <td>
               <input type="text" id="result" name="result" size="25" />
            </td>
         </tr>
      </tbody>
   </table>
  </form>
 </body>
</html>
```

JavaScript: jsb-15-05.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // check for critical elements
      var oSource = document.getElementById('source');
      var oParameter1 = document.getElementById('parameter1');
      var oParameter2 = document.getElementById('parameter2');
      var oResult = document.getElementById('result');

         // if they all exist...
         if (oSource && oParameter1 && oParameter2 && oResult)
         {
            // apply behaviors
            addEvent(oParameter1, 'change', showResults);
            addEvent(oParameter2, 'change', showResults);
         }
         else
         {
            alert('Critical element not found');
         }
   }
}

// execute the method and display the result
function showResults()
{
   // get source text
   var oSource = document.getElementById('source');
   var sSource = oSource.firstChild.nodeValue;
```


更多资源请访问稀酷客(www.ckook.com)

```
    // get paramaters
    var oParameter1 = document.getElementById('parameter1');
    var iParameter1 = parseInt(oParameter1.options[oParameter1.selectedIndex].value);

    var oParameter2 = document.getElementById('parameter2');
    var iParameter2 = parseInt(oParameter2.options[oParameter2.selectedIndex].value);

    // generate result & display it
    var oResult = document.getElementById('result');

        if (!iParameter2)
        {
            oResult.value = sSource.slice(iParameter1);
        }
        else
        {
            oResult.value = sSource.slice(iParameter1, iParameter2);
        }
}
```

相关主题：*string*.substr()、*string*.substring()方法。

string.split("*delimiterCharacter*" [, *limitInteger*])

返回值：分隔项数组

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

split()方法的功能与 array.join()方法相反(见第 18 章)。对于 String 对象而言，JavaScript 以指定的字符作为分隔符，将一个长字符串分为多个片段，并用这些片段创建一个密集数组。这里不需要使用 new Array()构造函数来初始化数组，因为 array 对象的方法很强大，如 array.sort()，所以可以将一系列字符串项转换为数组，来利用这些数组方法。另外，如果要将字符串分为单个字符的数组，仍可以使用 split()方法，但把一个空字符串指定为参数。对于 NN3 和 IE4 等老浏览器，只能指定第一个参数。

在现代浏览器中，可以把正则表达式对象用作第一个参数，提高在字符串中查找分隔符的能力。例如下面的字符串：

```
var nameList = "Fred - 123-4567,Jane - 234-5678,Steve - 345-6789";
```

将该字符串转换为只包含名字的数组时，如果不在 string.split()中使用正则表达式，就需要完成许多解析工作。但是，如果把正则表达式作为参数：

```
var regexp = / - [\d-]+,?\b/;
var newArray = nameList.split(regexp);
   // result = an array "Fred", "Jane", "Steve", ""
```

新数组项只包含名称，没有电话号码或标点符号。

可选的第二个参数是一个整数值，可限定该方法生成的数组元素个数。

示例

使用第 4 章介绍的 The Evaluator，可以查看 string.split()方法的工作情况。首先，给变量赋

更多资源请访问稀酷客(www.ckook.com)

予一个用逗号分隔的字符串：

```
a = "Anderson,Smith,Johnson,Washington"
```

现在，在逗号位置拆分字符串，把字符串的各个部分作为数组 b 中的项：

```
b = a.split(",")
```

为了证明这个数组包含 4 项，可以检查数组的 length 属性：

```
b.length     // result: 4
```

相关主题：array.join()方法。

string.substr(*start* [, *length*])

返回值：字符串

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

string.substr()方法提供了从一开始就存在于 JavaScript 语言中的 *string*.substring()方法的一个变体，其区别是 *string*.substr()方法的参数指定了开始索引值，以及要提取的字符个数(包括开始位置上的字符)；而 *string*.substring()方法的参数指定主字符串中开始字符和结束字符的索引值。

与需要索引值的所有字符串方法一样，*string*.substr()的第一个参数也是基于 0 的，若不指定第二个参数，返回的子字符串就从索引点开始，直到字符串的结尾。如果第二个参数值超过字符串的结尾点，方法就返回直到字符串结尾的子字符串。

虽然这个方法比 *string*.substring()方法更新，它也没有包含在 ECMA 语言标准的最新版(第 5 版)中。但是由于该方法得到了广泛运用，标准也承认了该方法，这样其他脚本就能实现这个方法来兼容浏览器了。

示例

程序清单 15-6 可以实际尝试 *string*.substr()方法的各种参数值，查看其工作情况。

程序清单 15-6　读取字符串的一部分

HTML：jsb-15-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>String Slicing and Dicing, Part II</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-06.js"></script>
   </head>
   <body>
      <h1>String Slicing and Dicing, Part II</h1>
      <h2>String substr() Method</h2>

      <form action="string-substr.php">
```


更多资源请访问稀酷客(www.ckook.com)

```
            <p>
               <label for="source">Text used for the methods:</label>
            </p>
            <p>
               <textarea id="source" name="source" cols="60" rows="1">
 Electroencephalograph</textarea>
               <pre>
 ----|----|----|----|-
     5    10    15    20
               </pre>
            </p>

            <table>
               <thead>
                  <tr>
                     <th>Method</th>
                     <th>Parameters</th>
                     <th>Results</th>
                  </tr>
               </thead>
               <tbody>
                  <tr>
                     <td>string.substr()</td>
                     <td>(
                        <select id="parameter1" name="parameter1">
                           <option value="0">0</option>
                           <option value="1">1</option>
                           <option value="2">2</option>
                           <option value="3">3</option>
                           <option value="5">5</option>
                        </select>,
                        <select id="parameter2" name="parameter2">
                     <option>(None)</option>
                     <option value="5">5</option>
                     <option value="10">10</option>
                     <option value="20">20</option>
                        </select>
                     )</td>
                     <td>
                        <input type="text" id="result" name="result" size="25" />
                     </td>
                  </tr>
               </tbody>
            </table>
         </form>
      </body>
   </html>
```

JavaScript: jsb-15-06.js

```
// initialize when the page has loaded
```


更多资源请访问稀酷客(www.ckook.com)

```
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // check for critical elements
      var oSource = document.getElementById('source');
      var oParameter1 = document.getElementById('parameter1');
      var oParameter2 = document.getElementById('parameter2');
      var oResult = document.getElementById('result');

         // if they all exist...
         if (oSource && oParameter1 && oParameter2 && oResult)
         {
            // apply behaviors
            addEvent(oParameter1, 'change', showResults);
            addEvent(oParameter2, 'change', showResults);
         }
         else
         {
            alert('Critical element not found');
         }
   }
}

// execute the method and display the result
function showResults()
{
   // get source text
   var oSource = document.getElementById('source');
   var sSource = oSource.firstChild.nodeValue;

   // get paramaters
   var oParameter1 = document.getElementById('parameter1');
   var iParameter1 = parseInt(oParameter1.options[oParameter1.selectedIndex].value);

   var oParameter2 = document.getElementById('parameter2');
   var iParameter2 = parseInt(oParameter2.options[oParameter2.selectedIndex].value);

   // generate result & display it
   var oResult = document.getElementById('result');

      if (!iParameter2)
      {
         oResult.value = sSource.substr(iParameter1);
      }
      else
      {
         oResult.value = sSource.substr(iParameter1, iParameter2);
      }
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：*string*.substring()方法。

string.substring(*indexA*[, *indexB*])

返回值：索引 A 和索引 B 之间的字符串

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

string.substring()方法可以从任意字符串中提取一系列连续字符的副本，该方法的参数是从主字符串中提取字符的开始和结束索引值(String 对象中第一个字符的索引值是 0)。注意这种提取操作会在较大的索引值处结束，但不包括这个索引位置的字符。

哪个索引值参数更大并没有区别，因为方法从较小的索引值开始，一直提取到较大的索引值(但不包括该索引值)。若两个索引值相同，则方法返回空字符串；若忽略第二个参数，则一直提取到字符串的结尾处。

示例

可以通过程序清单 15-7 来练习 *string*.substring()方法的各种参数值，查看其工作情况。

程序清单 15-7　读取字符串的一部分

HTML: jsb-15-07.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>String Slicing and Dicing, Part III</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-07.js"></script>
   </head>
   <body>
      <h1>String Slicing and Dicing, Part III</h1>
      <h2>String substring() Method</h2>

      <form action="string-substring.php">
         <p>
            <label for="source">Text used for the methods:</label>
         </p>
         <p>
            <textarea id="Textarea1" name="source" cols="60" rows="1">
Electroencephalograph</textarea>
            <pre>
----|----|----|----|-
    5   10   15   20
            </pre>
         </p>

         <table>
            <thead>
               <tr>
                  <th>Method</th>
```


更多资源请访问稀酷客(www.ckook.com)

```
                    <th>Parameters</th>
                    <th>Results</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>string.substring()</td>
                    <td>(
                        <select id="parameter1" name="parameter1">
                            <option value="0">0</option>
                            <option value="1">1</option>
                            <option value="2">2</option>
                            <option value="3">3</option>
                            <option value="5">5</option>
                        </select>,
                        <select id="parameter2" name="parameter2">
                            <option>(None)</option>
                            <option value="3">3</option>
                            <option value="5">5</option>
                            <option value="10">10</option>
                        </select>
                    )</td>
                    <td>
                        <input type="text" id="result" name="result" size="25" />
                    </td>
                </tr>
            </tbody>
        </table>
    </form>
  </body>
</html>
```

JavaScript: jsb-15-07.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // check for critical elements
        var oSource = document.getElementById('source');
        var oParameter1 = document.getElementById('parameter1');
        var oParameter2 = document.getElementById('parameter2');
        var oResult = document.getElementById('result');

            // if they all exist...
            if (oSource && oParameter1 && oParameter2 && oResult)
            {
                // apply behaviors
```


更多资源请访问稀酷客(www.ckook.com)

```
            addEvent(oParameter1, 'change', showResults);
            addEvent(oParameter2, 'change', showResults);
         }
         else
         {
            alert('Critical element not found');
         }
      }
   }

   // execute the method and display the result
   function showResults()
   {
      // get source text
      var oSource = document.getElementById('source');
      var sSource = oSource.firstChild.nodeValue;

      // get paramaters
      var oParameter1 = document.getElementById('parameter1');
      var iParameter1 = parseInt(oParameter1.options[oParameter1.selectedIndex].value);

      var oParameter2 = document.getElementById('parameter2');
      var iParameter2 = parseInt(oParameter2.options[oParameter2.selectedIndex].value);

      // generate result & display it
      var oResult = document.getElementById('result');

         if (!iParameter2)
         {
            oResult.value = sSource.substring(iParameter1);
         }
         else
         {
            oResult.value = sSource.substring(iParameter1, iParameter2);
         }
   }
```

相关主题： *string*.substr()、*string*.slice()方法。

string.toLocaleLowerCase()，*string*.toLocaleUpperCase()

返回值： 字符串

兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Opera+，Chrome+

这两个方法都可以改变字符串的大小写，但与标准方法有所不同。它们也可以用于某些不需要把特殊字符映射为拉丁字母的语言系统。

相关主题： *string*.toLowerCase()、*string*.toUpperCase()方法。

string.toLowerCase()，*string*.toUpperCase()

返回值： 根据所调用的方法，返回全部小写或全部大写的字符串

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

Internet 和 JavaScript 中的许多对象都要区分大小写。例如，在一些服务器的 URL 中，目录名和文件名都是区分大小写的。

这两个最简单的字符串方法返回转换为全部大写或全部小写的字符串副本。任何混合大小写的字符串都会转变为统一的大小写。若希望对域中的用户输入和一些编写好的代码字符串进行比较，但不考虑字符串的大小写，可将两个字符串都转换为同样的大小写，再进行比较。

示例

可以对字面量字符串使用 toLowerCase ( )和 toUpperCase ()方法，如下：

```
var newString = "HTTP://www.Mozilla.ORG".toLowerCase();
    // result = "http://www.mozilla.org"
```

这些方法也可用于大小写不重要时的字符串比较，如下所示：

```
if (guess.toUpperCase() == answer.toUpperCase()) {...}
    // comparing strings without case sensitivity
```

注意这些方法不会改变字符串本身的大小写，只是输出改变了大小写的副本，原来的字符串保持不变。

相关主题： *string*.toLocaleLowerCase()、*string*.toLocaleUpperCase()方法。

string.toString()，*string*.valueOf()

返回值： 字符串值

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

这两个方法都返回字符串值(而不是完整的 String 对象)。若使用 new String()构造函数创建 String 对象，该对象的类型就是 object。因此，若想更加精确地确定对象值的类型，可使用 valueOf()方法得到这个值，然后用 typeof 操作符检查。这个对象目前有 toString()方法，主要是因为 String 对象继承了 JavaScript 根对象的方法。

示例

使用第 4 章介绍的 The Evaluator 来测试 valueOf()方法。在顶端文本框中输入下面的语句，在 Results 域中查看结果：

```
a = new String("hello")
typeof a
b = a.valueOf()
typeof b
```

因为其他所有的 JavaScript 核心对象都有 valueOf()方法，所以可建立通用函数，它的参数接收各种对象类型，而且脚本代码可以根据对象中存储的值类型，执行不同的处理。

相关主题： typeof 操作符(第 22 章)。

更多资源请访问稀酷客(www.ckook.com)

15.3　字符串使用函数

确定如何将各种字符串对象方法应用于字符串操作并不总是很简单。如果要使用有限的 JavaScript 功能来满足旧式或移动浏览器的要求，难度就更大了。很难预料需要在脚本中处理字符串的每种可能方式。但是，为了帮助读者起步，程序清单 15-8 包含了一个完全向后兼容的字符串函数库，来插入、删除和替换字符串中的文本块。如果用户使用的浏览器能够包括外部.js 库文件，则最好使这些函数可用于脚本。

程序清单 15-8　使用字符串处理函数

JavaScript: jsb-15-08.js

```
// extract front part of string prior to searchString
function getFront(sMain, sSearch)
{
   iOffset = sMain.indexOf(sSearch);

      if (iOffset == -1)
      {
         return null;
      }
      else
      {
         return sMain.substring(0, iOffset);
      }
}

// extract back end of string after searchString
function getEnd(sMain, sSearch)
{
   iOffset = sMain.indexOf(sSearch);

      if (iOffset == -1)
      {
         return null;
      }
      else
      {
         return sMain.substring(iOffset + sSearch.length, sMain.length);
      }
}

// insert insertString immediately before searchString
function insertString(sMain, sSearch, sInsert)
{
   var sFront = getFront(sMain, sSearch);
   var sEnd = getEnd(sMain, sSearch);

      if (sFront == null || sEnd == null)
```


更多资源请访问稀酷客(www.ckook.com)

```
        {
            return null;
        }
        else
        {
            return sFront + sInsert + sSearch + sEnd;
        }
}

// remove sDelete from sMain
function deleteString(sMain, sDelete)
{
    return replaceString(sMain, sDelete, "");
}

// replace sSearch with sReplace
function replaceString(sMain, sSearch, sReplace)
{
    var sFront = getFront(sMain, sSearch);
    var sEnd = getEnd(sMain, sSearch);

        if (sFront == null || sEnd == null)
        {
            return null;
        }
        else
        {
            return sFront + sReplace + sEnd;
        }
}
```

前两个函数根据需要取出字符串的前一部分或后一部分以用于其他函数。最后三个函数是这些字符串处理函数的核心。若打算在脚本中使用这些函数，应注意函数彼此之间的依赖性。将 5 个函数组合起来，确保它们像所设计的那样工作。

程序清单 15-8 的一种现代替换方式是，组合字符串和 array 方法，在一个单语句函数中执行全局替换操作：

```
function replaceString(sMain, sSearch, sReplace)
{
    return sMain.split(sSearch).join(sReplace);
}
```

下一步，创建一个自定义方法，在脚本中使用所有的字符串值或者对象。在页面加载时，执行下面的语句：

```
String.prototype.replaceString = function replaceString(sMain,
        sSearch, sReplace)
{
    return sMain.split(sSearch).join(sReplace);
}
```


更多资源请访问稀酷客(www.ckook.com)

然后，在页面的其他脚本中，给字符串值调用此方法：

```
myString = myString.replaceString(" CD ", " MP3 ");
```

注意也可以使用正则表达式完成全局查找和替换，参见第 45 章。

15.3.1　HTML 标记方法

现在学习另一组 String 对象方法，如表 15-2 所示。它们可以有效地在 HTML 元素中封装文本。这些方法都不在 ECMAScript 第 5 版标准中，它们存在于 JavaScript 实现中，仅是为了支持老版本。现代网页使用样式表给元素指定修饰格式，使用 DOM 方法插入新节点。

表 15-2　string 对象方法

string.anchor("*anchorName*")	*string*.link(*locationOrURL*)
string.blink()	*string*.big()
string.bold()	*string*.small()
string.fixed()	*string*.strike()
string.fontcolor(*colorValue*)	*string*.sub()
string.fontsize(*integer1to7*)	*string*.sup()
string.italics()	

首先看看不需要任何参数的方法，它们的共同点是：所有方法都是具有开关设置的字体样式特性。在 HTML 文档中，要启用这些特性，可将文本放在适当的标记对中，如...用于粗体文本。这些方法提取 String 对象，给它们加上标记，并返回得到的文本，然后就可以把这些文本放入脚本创建的 HTML 中。因此表达式：

```
"Good morning!".bold()
```

等同于：

```
<b>Good morning!</b>
```

程序清单 15-9 把这些字符串方法应用于页面上的文本(Internet Explorer、Chrome 和 Safari 不支持<blink>元素，所以它们执行 *string*.blink()方法时，将显示不会闪烁的文本）。

程序清单 15-9　应用简单的字符串方法

HTML：jsb-15-09.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>HTML by JavaScript</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-15-09.js"></script>
   </head>
   <body>
      <h1>HTML by JavaScript</h1>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <ul id="tag-list">
           <li id="anchor">anchor</li>
           <li id="big">big</li>
           <li id="blink">blink</li>
           <li id="bold">bold</li>
           <li id="fixed">fixed</li>
           <li id="fontcolor">fontcolor</li>
           <li id="fontsize">fontsize</li>
           <li id="italics">italics</li>
           <li id="link">link</li>
           <li id="small">small</li>
           <li id="strike">strike</li>
           <li>sub H<span id="sub">2</span>O</li>
           <li>sup E=mc<span id="sup">2</span></li>
        </ul>
        <form action="html-by-javascript.php">
           <p>
              <button id="apply-markup">Apply Markup</button>
           </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-15-09.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // check for button
      var oButton = document.getElementById('apply-markup');

         // if it exists...
         if (oButton)
         {
            // apply behavior
            addEvent(oButton, 'click', applyMarkup);
         }
   }
}

// apply markup to text elements
function applyMarkup(evt)
{
      // consolidate event handling
      if (!evt) evt = window.event;

   var oList = document.getElementById('tag-list');
```


更多资源请访问稀酷客(www.ckook.com)

```
    var aElements = oList.getElementsByTagName('*');

    for (var i = 0; i < aElements.length; i++)
    {
        // get this element's id & text value
        sId = aElements[i].id;

        //var sText = aElements[i].firstChild.nodeValue;
        var sText = aElements[i].innerHTML;

        switch (sId)
        {
            case 'anchor': sText = sText.anchor('anchor-name'); break;
            case 'big': sText = sText.big(); break;
            case 'blink': sText = sText.blink(); break;
            case 'bold': sText = sText.bold(); break;
            case 'fixed': sText = sText.fixed(); break;
            case 'fontcolor': sText = sText.fontcolor('red'); break;
            case 'fontsize': sText = sText.fontsize(7); break;
            case 'italics': sText = sText.italics(); break;
            case 'link': sText = sText.link('http://example.com/'); break;
            case 'small': sText = sText.small(); break;
            case 'strike': sText = sText.strike(); break;
            case 'sub': sText = sText.sub(); break;
            case 'sup': sText = sText.sup(); break;
            default: continue;
        }

        // replace the element's contents with the marked-up text
        aElements[i].innerHTML = sText;
    }
    alert('Here is the modified markup:\n' + oList.innerHTML);

    // cancel form submission
        // W3C DOM method (hide from IE)
        if (evt.preventDefault) evt.preventDefault();
    // IE method
    return false;
}
```

string.fontsize()和 *string*.fontcolor()方法会影响显示在 HTMT 页面上的字体特征。其参数非常简单，是 1～7 之间的整数(对应于 7 种浏览字体大小)和指定文本的颜色值(3 个十六进制值或颜色常量名)。

最后两个字符串方法可以用字符串创建锚和链接，*string*.anchor()方法使用参数为锚创建名称。因此下面的表达式：

```
"Table of Contents".anchor("toc")
```

等同于：

```
<a name="toc">Table of Contents</a>
```


更多资源请访问稀酷客(www.ckook.com)

同样，*string*.link()方法的参数是一个有效的位置或 URL，可以用字符串创建一个真正的 HTML 链接：

```
"Back to Home".link("index.html")
```

这等同于：

```
<a href="index.html">Back to Home</a>
```

在目前的网站脚本中很少使用这些方法的主要原因是，人们一直在努力分离开发层——这里是把 HTML 结构、CSS 显示和 JavaScript 代码分开。这些层之间的分隔越清晰，开发、调试和修改的模块化程度就越高，效率也就越高。例如，如果所有的字体大小和颜色细节都在样式表中指定，而不是硬编码到 HTML 或 JavaScript 中，页面上文本的外观就可以由拥有不同技能的不同人来修改，独立于页面的标记和行为。同样，把标记保存在 HTML 文档中，并允许 JavaScript 操作该标记；而且除非绝对必须，不会做硬性规定。

通过修改标记来改变页面的外观是非常个性化的行为。它依赖浏览器为给定标记指定的默认样式，而不是在每个人的样式表中指定它们。现代脚本可以通过指定 id 或 className 来改变元素的外观，并让样式表决定元素如何显示。

15.4 URL 字符串编码及解码

浏览器与服务器通信时，许多常见的非字母数字字符(如空格)不能以其原来的格式传输，只能传输字母、数字和标点符号。为了传输其他字符，必须使用特殊的符号(%)和十六进制的 ASCII 值来编码。例如，空格字符值是十六进制的 20 (ASCII 十进制 32)，编码为%20。这个符号可在浏览器历史记录或 URL 中见到。

JavaScript 包含两个函数 encodeURIComponent()和 decodeURIComponent()，它们可以即时转换整个字符串。将一般字符串转换为转义代码字符串时，应使用转义函数，如：

```
encodeURIComponent("Howdy Pardner"); // result = "Howdy%20Pardner"
```

decodeURIComponent()函数将转义代码转换为可读的格式。

交叉引用：
第 24 章讲解了这两个函数。

更多资源请访问稀酷客(www.ckook.com)

第16章

Math、Number和Boolean对象

本章包含哪些内容？
高级数学操作
基本数值转换
处理整型和浮点型数值

第8章简述了数据类型、值以及JavaScript中数值和Boolean的功能。本章将详细讨论JavaScript使用数值和Boolean数据的方式。

数学常常令初学者望而生畏，但如本书所述，并不是必须成为数学天才才能编写JavaScript程序。在需要数学时，本章的有关知识将很有帮助。即使不擅长数学，也不要被这些术语吓退。

就像字符串值和string对象一样，本章介绍的数值和Boolean也是值和对象。对脚本编写者来说，除非要进行非常复杂的编程，否则这种差异很小。但对于编写浏览器中JavaScript解释器的人来说，这种区别则显得非常重要。

数值数据类型和转换以及Math对象的信息对于大多数脚本编写者都非常重要。

16.1 JavaScript中的数值

功能强大的编程语言有许多类型的数值，每类数值都和它占用的计算机内存有关。处理这些不同的数据类型可能比较有趣，但是不利于快速编写脚本。JavaScript数值只有两种类型：整型和浮点型。整型是没有小数部分的任意整数，其取值范围很大。整型从不包含小数点；JavaScript的浮点数也在这个取值范围内，有小数点和小数值。有经验的程序员可以参考本章后面number对象的讨论，看看JavaScript数值类型对应于其他编程环境中的哪些数值数据类型。

16.1.1 整数和浮点数

计算机内部的微处理器处理整型值的数学操作比带小数的数值更容易。微处理器需要完成额外的工作，才能对两个浮点数执行相加操作。很不幸，脚本的编写者要背负这个历史的包袱，必须知道某些计算所用的数值类型。

更多资源请访问稀酷客(www.ckook.com)

JavaScript 生成的大多数内部值(如索引值和 length 属性)都是整数。浮点值一般是数值除法的结果、特殊的值(如 pi)和用户输入的值(如美元和美分)。幸好 JavaScript 允许执行复合数值数据类型的数学操作。注意下面的示例如何将数据转换为合适的数据类型：

```
3 + 4 = 7 // integer result
3 + 4.1 = 7.1 // floating-point result
3.9 + 4.1 = 8 // integer result
```

在上面的三个示例中，只有最后一个结果是出人意料的。两个浮点数相加得到一个整数时，结果显示为整数。

处理浮点数时，注意不是所有的浏览器返回的值都精确到小数点的最后一位小数。例如，表 16-1 是各种脚本浏览器计算 8/9 的结果，转换为字符串显示。

表 16-1　字符串显示

NN3＆NN4	.8888888888888888
WinIE3	0.888888888888889
NN6+/Moz+/Safari+/opera+/Chrome+	0.8888888888888888
WinIE4+	0.8888888888888888

很明显，不应在 JavaScript 浏览器中使用浮点数来计算空间飞行轨迹，或进行其他精度很高的重要计算。甚至是日常的数学运算，PC 算法都有可能产生浮点错误。

在 Navigator 中，JavaScript 依靠操作系统的数学运算能力来完成自己的数学计算。操作系统可以使计算结果精确到小数点后面很多位，但是 JavaScript 很少全部显示出来。从上表可以看出，现代浏览器在显示的位数和数值的舍入方面是一致的。这对数学计算有帮助，但需要以特殊格式显示时，它并不实用。

在 IE5.5、基于 Mozilla 的浏览器和其他 W3C 兼容的浏览器推出之前，JavaScript 没有提供内置工具，来格式化浮点运算的结果(现代浏览器的用户可参见本章后面的 Number 对象来了解格式化方法)。程序清单 16-1 演示了正数的通用格式化例程，它还调用这个例程，将一个值转换为美元。删除注释后，例程就会更加简短。

交叉引用：

在本章和本书许多章节的代码中，指定事件处理程序的函数是 addEvent()，这个跨浏览器的事件处理程序详见第 32 章。

程序清单 16-1　通用的数值格式化例程

HTML: jsb-16-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Number Formatting</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-16-01.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
    </head>
    <body>
        <h1>How to Make Money</h1>
        <form>
            <p>
                <label for="entry">Enter a positive floating point value or arithmetic
                expression to be converted to a currency format:</label>
            </p>
            <p>
                <input type="text" id="entry" name="entry" value="1/3">
                <input type="button" id="dollars" value="&gt; Dollars and Cents &gt;">
                <input type="text" id="result" name="result">
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-16-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to critical elements
        var oButton = document.getElementById('dollars');
        var oEntry = document.getElementById('entry');
        var oResult = document.getElementById('result');

            // if they all exist...
            if (oButton && oEntry && oResult)
            {
                // apply behavior to button
                oButton.onclick =
                    function() { oResult.value = dollarize(oEntry.value); }
            }
    }
}
// turn incoming expression into a dollar value
function dollarize(expr)
{
    return "$" + format(expr,2);
}

// generic positive number decimal formatting function
function format(expr, decplaces)
{
    // evaluate the incoming expression
    var val = eval(expr);
```


更多资源请访问稀酷客(www.ckook.com)

```
    // raise the value by power of 10 times the number of decimal places;
    // round to an integer; convert to string
    var str = "" + Math.round(val * Math.pow(10, decplaces));

    // pad small value strings with zeros to the left of rounded number
    while (str.length <= decplaces)
    {
       str = "0" + str;
    }

    // establish location of decimal point
    var decpoint = str.length - decplaces;

    // assemble final result from:
    // (a) the string up to the position of the decimal point;
    // (b) the decimal point; and
    // (c) the balance of the string. Return finished product.
    return str.substring(0,decpoint) + "." + str.substring(decpoint, str.length);
}
```

这个例程似乎完成了大量工作；若应用程序需要为所有的浏览器处理浮点值，并进行指定的格式化，这个例程就是必需的。

注意：

本例使用的全局方法 eval()并不适合实际工作。因为 eval()会解释所有 JavaScript 表达式，处理脚本、对象、变量值以及 DOM 的所有细节，以便具备理解力的访问者考察和操作。JavaScript 语言故意在客户计算机上设置了安全限制，但 XMLHttpRequest 允许访问服务器。最好不要给陌生人提供开放、未过滤的 eval()输入通道。

还可以输入指数形式的浮点数。指数用字母 e(大写或小写) 和加上符号(+或-)的指数值来表示。下面的示例用指数形式表示浮点数：

```
1e6 // 1,000,000 (the "+" symbol is optional on positive exponents)
1e-4 // 0.0001 (plus some error further to the right of the decimal)
-4e-3 // -0.004
```

JavaScript 把 le-5 和 1e15 之间的值显示为非指数形式(在现代浏览器上可以强制显示指数形式)。该范围外的其他值在所有浏览器中都显示为指数形式。

16.1.2 十六进制和八进制整数

JavaScript 可以处理十进制、十六进制和八进制值，处理这些值只需遵循几个规则。

十进制值不以 0 开头，因此若页面要求用户输入以 0 开头的十进制值，脚本就必须从输入字符串中去掉这些零，或在对这些值执行数学处理前，使用数值全局解析函数(参见下一节)。

十六进制整数以 0x 或 0x(是 0 而不是字母 O)开头，A-F 可以大写或小写。下面是几个十六进制的值：

```
OX2B
```


更多资源请访问稀酷客(www.ckook.com)

```
OX1a
OXcc
```

不要将用于运算的十六进制值与 Web 文档的颜色属性规范中的十六进制值混淆。那些值用特殊的三个十六进制格式表示：#号后跟三个串在一起的十六进制数，如#c0c0c0。

八进制值以 0 开头，后跟 0～7 之间的数字，只包含整数。

在数学表达式中，可以自由混合使用不同进制的数值，但 JavaScript 将所有结果显示为十进制数。进制之间的转换必须使用用户在脚本中定义的函数。例如，程序清单 16-2 中的函数可以将 0～255 之间的任意十进制数转换为 JavaScript 十六进制值。

程序清单 16-2　十进制与十六进制的转换函数

```
function toHex(dec)
{
   hexChars = "0123456789ABCDEF";
   if (dec > 255)
   {
      return null;
   }
   var i = dec % 16;
   var j = (dec - i) / 16;
   result = "0X";
   result += hexChars.charAt(j);
   result += hexChars.charAt(i);
   return result;
}
```

toHex()转换函数假定，传递给它的参数是十进制整数，若需将字符串格式的数值表示为十六进制，请参看第 15 章的 toString()方法。

16.1.3　将字符串转换成数值

迄今为止，还没有讨论过将字符串表示的数值转换成 JavaScript 算术操作符可以处理的数值的方法。其实，大多数 JavaScript 操作符和数学方法都接受数值的字符串表示，并可以处理这些值。在使用+操作符(它有时是字符串连接符)进行加法运算时，常会遇到数值类型不匹配的情况；如果对从表单文本框中提取的值执行数学操作，这些对象的 value 属性就是字符串。所以在许多情况下，需将这些字符串值转换为数值类型，以进行数学操作。

要转换为数值，需要使用以下两个 JavaScript 函数中的一个：

```
parseInt(string [,radix])
parseFloat(string [,radix])
```

这些函数最初用在 Java 语言中，“解析”(parsing)在编程领域有许多含义。其中一个含义与“提取”(extracting)相同。parseInt()函数可以从传给它的字符串中提取并返回整型值；parseFloat()函数可以从字符串中提取并返回浮点型值。下面是示例及其结果：

```
parseInt("42")        // result = 42
```


更多资源请访问稀酷客(www.ckook.com)

```
parseInt("42.33")        // result = 42
parseFloat("42.33")      // result = 42.33
parseFloat("42")         // result = 42
parseFloat("fred")       // result = NaN (Not a Number)
```

parseFloat()函数还可以处理整数，返回整型值。根据用户在文本域中输入的字符串，可以在脚本中使用这个函数处理各种类型的数值。

在这两个函数中，第二个可选的参数可以指定字符串表示的数值的进制。如果需要从以一个或多个 0 开始的字符串中提取出十进制数，这个参数尤其有用。一般情况下，前导 0 表示八进制值。但如果强制将字符串值转换为十进制，也能按照希望的那样转换：

```
parseInt("010")        // result = 8
parseInt("010",10)     // result = 10
parseInt("F2")         // result = NaN
parseInt("F2", 16)     // result = 242
```

在需要整型值或浮点值时，可以使用这些函数。例如：

```
var result = 3 + parseInt("3");      // result = 6
var ageVal = parseInt(document.forms[0].age.value);
```

后一个函数确保这个属性的字符串值转换为一个数值，但在对用户输入的数值进行数学运算前，要验证这些数值，参见配书光盘中的第 46 章。

parseInt()和 parseFloat()方法都从字符串的第一个字符开始搜索，直到字符串中再也没有数字和小数点字符为止，因此可以用它们处理字符串(例如 navigator.appVersion 属性返回的字符串，如 6.0(Windows；en-US))，来得到字符串的前导数值部分。若字符串不以可接受的字符开头，这两个方法就返回 NaN(不是数值)。

16.1.4　将数值转换成字符串

JavaScript 不允许将数值数据类型传递给第 15 章讨论的许多字符串方法。因此，需要先将数值转换成字符串，再确定数值的位数。

可采用几种方法将数值强制转换为字符串，老方法是在数值前加上一个空字符串和连接操作符。例如，假设 dollars 变量包含整数值 2500，为了使用字符串对象的 length 属性(在本章后面讨论)确定该数值的位数，可使用下面的结构：

```
("" + dollars).length        // result = 4
```

在提取 length 属性值之前，圆括号强制 JavaScript 执行连接操作。

较好的办法是使用 toString()方法。调用该方法的语句结构与调用任何对象方法是一样的，例如要将 dollars 变量值转换为字符串，可使用下面的语句：

```
dollars.toString()           // result = "2500"
```

在现代浏览器中，该方法有一个额外的功能：可为字符串表示的数值指定基数，称为 radix，这个基数在方法名中指定为一个参数。下面的示例创建一个数值，并转换为十六进制数的字符串：

更多资源请访问稀酷客(www.ckook.com)

```
var x = 30;
var y = x.toString(16);    // result = "1e"
```

使用参数 2 得到二进制结果，使用参数 8 得到八进制，默认为基数 10。注意不要把这些转换和真正的数值转换相混淆，toString()方法返回的结果不能作为其他语句中的数值操作数。

最后，在 IE5.5+、基于 Mozilla 的浏览器和其他 W3C 浏览器中，Number 对象还有三个方法 toExponential()、toFixed()和 toPrecision()；它们根据规则和传递给方法的参数，返回格式化后的数值的字符串表示，参见本章后面的讨论。

16.1.5　数值不是数值型时

在上一节的几个示例中，一些操作的结果是 NaN，这不是一个字符串，而是一个特殊值，表示 Not a Number(不是数值)。例如，如果使用 parseFloat()将字符串 joe 转换为整型，就不会成功，而是报告原字符串不是一个数值。

在设计应用程序时，如果请求用户输入，或者从服务器端的数据库中检索数据，就不能保证该值是所需要的数值型或者可以转换为数值。此时，在对这个值执行数学操作之前，需要查看该值是否是一个数值。JavaScript 提供了一个特殊的全局函数 isNaN ()，可检查某个值是否是数值，如果该值不是数值，函数就返回 true；否则返回 false。例如，可检查某个表单域是否是一个数值：

```
var ageEntry = parseInt(document.forms[0].age.value);
   if (isNaN(ageEntry))
   {
   alert("Try entering your age again.");
   }
```

16.2　Math 对象

进行比简单运算更高级的数学处理时，请参见 Math 对象方法列表。

16.2.1　语法

访问 Math 对象的属性和方法：

```
Math.property
Math.method(value [, value])
```

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

16.2.2　关于 Math 对象

除了一般的算术操作(详见第 22 章)外，JavaScript 还有高级数学处理能力，如果以前没有

更多资源请访问稀酷客(www.ckook.com)

在真正的面向对象环境下编写过程序，这种处理能力可能看起来比较奇怪。大多数算术操作(如 var result=2+2)是立即执行的，其他操作就需要使用 JavaScript 内部的 Math(M 大写)对象了。Math 对象有一些属性(就像其他语言的常量)和方法(就像其他语言的数学函数)。

在语句中使用 Math 对象的方式和使用其他 JavaScript 对象相同：创建一个引用，它以 Math 对象名开头，后跟一个句点和所需的属性或方法名：

```
Math.property | method([parameter]. . . [,parameter])
```

属性引用返回内置的值(例如 pi)，方法引用需要把一个或几个值作为参数，每个方法都返回一个结果。

16.2.3　属性

JavaScript 的 Math 对象属性表示数学中许多有用的常量值。表 16-2 列出了这些方法和值，保留 16 位小数。

表 16-2　JavaScript 的 Math 对象属性

属　　性	数　　值	说　　明
Math.E	2.718281828459045091	欧拉常数
Math.LN2	0.6931471805599452862	2 的自然对数
Math.LN10	2.302585092994045901	10 的自然对数
Math.LOG2E	1.442695040888963387	E 的对数，以 2 为基
Math.LOG10E	0.4342944819032518167	E 的对数，以 10 为基
Math.PI	3.141592653589793116	π
Math.SQRT1_2	0.7071067811865475727	0.5 的平方根
Math.SQRT2	1.414213562373095145	2 的平方根

由于这些属性表达式返回常量值，所以可将它们用于常规的算术表达式。例如已知直径 d，求圆的周长，就可以使用下面的语句：

```
circumference = d * Math.PI;
```

脚本编写者使用这些属性时，最常见的错误是没有大写 Math 对象名，而属性名称是区分大小写的。

16.2.4　方法

方法增强了 JavaScript 中 Math 对象的功能，除 Math.random()方法外，其他所有的 Math 对象方法都需要一个或几个参数。三角函数通常操作一个参数值，其他函数决定在所传递的参数中哪一个是组中的最大值或最小值。Math.random()方法没有参数，但返回 0～1 之间的随机浮点数。表 16-3 列出了所有 Math 对象方法的语法和返回值。

更多资源请访问稀酷客(www.ckook.com)

表 16-3 Math 对象的方法

方法语法	返 回 值
Math.abs(*val*)	*val* 的绝对值
Math.acos(*val*)	*val* 的反余弦(弧度为单位)
Math.asin(*val*)	*val* 的反正弦(弧度为单位)
Math.atan(*val*)	*val* 的反正切(弧度为单位)
Math.atan2(*vall*，*va12*)	极坐标 *x* 和 *y* 的角度
Math.ceil(*val*)	大于或等于 *val* 的下一个整数
Math.cos(*val*)	*val* 的余弦
Math.exp(*val*)	欧拉常数的 *val* 次方
Math.floor(*val*)	小于或等于 *val* 的下一个整数
Math.log(*val*)	*val* 的自然对数(e 为基)
Math.max(*val1*,*val2*)	*val1* 和 *va12* 中的较大一个
Math.min(*val1*,*val2*)	*val1* 和 *va12* 中的较小一个
Math.pow(*val1*,*va12*)	*val1* 的 *va12* 次方
Math.random()	0～1 之间的随机数
Math.round(*val*)	*val*≥n.5 时为 N+l；否则为 N
Math.sin(*val*)	*vat* 的正弦(弧度为单位)
Math.sqrt(*val*)	*val* 的平方根
Math.tan(*val*)	*val* 的正切(弧度为单位)

HTML 不是绘图艺术家梦想中的环境，但使用 JavaScript 三角函数可以给 HTML 生成的图表得到一系列值。有了可定位的元素，脚本设计者就可以利用这些函数定义移动元素的轨迹。对不具有编程基础的脚本编写者，数学常常是一个主要的障碍。但如前所述，使用简单的算术，加上一点逻辑，就可以用 JavaScript 完成大量工作——把复杂的数学操作留给喜欢它们的人。

16.2.5 创建随机数

Math 对象中最简便的一个方法是 Math.random()，它返回 0～1 之间的浮点数。若设计一个扑克牌游戏脚本，则需要 1～52 之间的随机数；掷骰子游戏则需要 1～6 之间的随机数。要生成 0 和任意上限(n)之间的随机数，可使用以下公式：

```
Math.floor(Math.random() * n)
```

这会生成 0～n-1 之间的整数。要生成 0～n 之间的整数，可以加上 1 或把 floor()改为 ceil()：

```
Math.floor(Math.random() * n) + 1
Math.ceil(Math.random() * n)
```

为使随机数所在的范围不从 0 开始，可使用下面的公式：

```
Math.floor(Math.random() * (n - m + 1)) + m
```

这里 m 是范围的整数下限，n 是上限。对于骰子游戏，每个骰子的公式如下：

更多资源请访问稀酷客(www.ckook.com)

```
newDieValue = Math.floor(Math.random() * 6) + 1;
```

16.2.6　Math 对象的快捷引用

第 21 章详细说明了一种 JavaScript 结构，它可以简化在语句中定位多个 Math 对象属性和方法的方式，其技巧是使用 with 语句。

其核心是，with 语句告诉 JavaScript 下一组语句(在花括号中)指向特殊的对象。对于 Math 对象，基本结构如下：

```
with (Math)
{
   //statements
}
```

花括号中的所有语句都可以忽略具体的 Math 对象引用。计算圆面积中的长引用如下(半径为 6 个单位)：

```
result = Math.pow(6,2) * Math.PI;
```

使用快捷引用则为：

```
with (Math)
{
   result = pow(6,2) * PI;
}
```

虽然后者包含更多的代码行，但对象引用更短，阅读起来更自然。在涉及 Math 对象属性和方法的一长串计算中，with 结构可以减少输入，降低引用中对象名大小写出错的可能性。还可在 with 结构中包括其他对象的引用，JavaScript 会将缺少对象名的引用与对象名相关联。其缺点是，with 结构在 JavaScript 中不是非常有效，因为它需要处理许多内部跟踪。所以最好不要在重复许多次的循环中使用它。

16.3　Number 对象

属　　性	方　　法
constructor	toExponential()
MAX_VALUE	toFixed()
MIN_VALUE	toLocaleString()
NaN	toString()
NEGATIVE_INFINITY	toPrecision()
POSITIVE_INFINITY	valueOf()
prototype	

更多资源请访问稀酷客(www.ckook.com)

16.3.1　语法

创建 Number 对象：

```
var val = new Number(number);
```

访问数值和 Number 对象的属性和方法：

```
number.property | method([parameters])
Number.property | method([parameters])
```

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

16.3.2　关于 Number 对象

Number 对象很少使用，因为在多数情况下，JavaScript 使用简单数值即可完成日常数值计算，但 Number 对象包含程序员为完成一些严谨工作需要的一些信息和功能。

首先考虑在语言中定义数值范围的属性，最大值是 1.79E+308；最小值是 2.22E-308。比最大值更大的值是 POSITIVE_INFINITY；比最小值更小的值是 NEGATIVE_INFINITY，但很少会遇到这些值。

对于 JavaScript 对象，还要说明的是 prototype 属性。第 15 章说明了如何向 String 对象的原型添加一个方法，使每个新建的对象都包含这个方法，这同样适用于 Number.prototype 属性。如果要为每个 Number 对象加入通用功能，就可以使用这个属性。这个原型工具只能用于完整的 Number 对象，不能用于一般的数值。有经验的程序员知道，JavaScript 的 number 对象和值在内部定义为 IEEE 的双精度 64 位值。

16.3.3　属性

constructor

(参见第 15 章中的 string.constructor)

MAX_VALUE，MIN_VALUE，NEGATIVE_INFINITY，POSITIVE_INFINITY

值：数值，　　只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

Number.MAX_VALUE 和 Number.MIN_VALUE 属性属于静态 Number 对象，代表 JavaScript 和 ECMAScript 可用的最大和最小正数值，其值分别是 $1.7976931348623157\times10^{308}$ 和 5×10^{-324}。超出该范围的值用常量 Number.POSITIVE_INFINITY 或 Number.NEGATIVE_INFINITY 表示。

示例

在 The Evaluator(第 4 章)的顶端文本框中，输入 4 个 Number 对象表达式，查看浏览器如何报告每个值。

```
Number.MAX_VALUE
Number.MIN_VALUE
```


更多资源请访问稀酷客(www.ckook.com)

```
Number.NEGATIVE_INFINITY
Number.POSITIVE_INFINITY
```

相关主题： NaN 属性；isNaN()全局函数。

NaN

值： NaN， 只读

兼容性： WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

NaN 属性是一个常数，当与数值相关的函数或方法试图处理不是数值的值，或者结果不是数值时，JavaScript 将它显示给用户。在使用 parseInt()和 parseFloat()函数时，如果要转换的字符串的第一个字符不是数值，则返回 NaN 值。使用 isNaN()全局函数可以确定某个值是否为 NaN 值。

示例

参看第 24 章中的 isNaN ()函数。

相关主题： isNaN()全局函数。

prototype

详见第 15 章中的 String.prototype。

16.3.4 方法

number.toExponential(*fractionDigits*)，number.toFixed(*fractionDigits*)，number.toPrecision(*precisionDigits*)

返回值： 字符串

兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

这三个方法允许脚本控制数值的格式化，把它们显示为字符串文本。每个方法的作用都不同，但它们都返回字符串。应给非格式化的 Number 对象执行所有的数学操作，因为这些值的精度最高。只有准备显示结果时，才能使用这些方法将数值转换为字符串，将它们显示为主题文本或赋予文本域。

toExponential()方法强制数值用指数形式显示，即使 JavaScript 可以使用标准形式显示这些数值也同样如此。其整数参数指明返回值在小数点后有多少位，即使小数点后的所有数字都是 0，也必须返回这些 0。例如，如果变量包含数值为 345，使用 toExponential (3)使该值变为 3.450e+2，它是 3.45×10^2 的 JavaScript 指数形式。

要将数值格式化为小数点后保留指定的位数，可使用 toFixed()方法。例如，该方法可以把财务计算的结果显示为百分制单位(如美元和美分)。该方法的参数是一个整数，表示小数点后面的位数。如果要格式化的数值的小数位数比参数指定的小数位数多，方法就圆整后面的值(但只圆整下一个数字)。例如，要保留两位小数，值 123.455 就显示为 123.46。但对于值 123.4549，该方法就忽略 9，圆整后去掉了 5 右边的 4，结果是 123.45。toFixed()方法不能返回准确的值，但在大多数情况下，会得到满意的结果。

最后一个方法是 toPrecision()，它可以定义数值总共显示多少位(包括小数点左右两边的数

更多资源请访问稀酷客(www.ckook.com)

字)，即定义数值的精度。下面演示了参数指定不同精度的结果：

```
var num = 123.45
num.toPrecision(1) // result = 1e+2
num.toPrecision(2) // result = 1.2e+2
num.toPrecision(3) // result = 123
num.toPrecision(4) // result = 123.5
num.toPrecision(5) // result = 123.45
num.toPrecision(6) // result = 123.450
```

注意 toPrecision ()和 toFixed ()的圆整是相同的。

示例

可使用第 4 章介绍的 The Evaluator，用各种参数值练习这三个方法。在调用方法之前，要保证在 The Evaluator 中给一个内置的全局变量(a～z)指定一个数值。

```
a = 10/3
a.toFixed(4)
"$" + a.toFixed(2)
```

这些方法都不处理数值字面量，比如 123.toExponential(2)是无效的。

相关主题： Math 对象。

number.toLocaleString()

返回值： 字符串

兼容性： WinIE5.5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

number.toLocaleString()方法返回当前数字的字符串版本，其格式取决于浏览器的本地设置。根据 ECMA 第 5 版的标准，浏览器可以自行确定 toLocaleString()方法返回的字符串值是否符合客户系统或浏览器的语言标准。

相关主题： number.toFixed()、number.toString()方法。

number.toString([*radix*])

返回值： 字符串

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

number.toString()方法返回当前数值的字符串版本，如果初始值不是 10 进制数，默认的 radix 参数(10)将该值转换成十进制。还可以指定其他基数(如二进制的基数 2，十六进制的基数 16)，将原数值转换为其他基数的字符串值(而不是数值)，用于以后的计算。

示例

使用第 4 章介绍的 The Evaluator，练习使用 toString ()方法。将数字 12 指定给变量 a，查看在各种进制中，这个数字如何转换为字符串：

```
a = 12
a.toString()      // base 10
a.toString(2)
```


更多资源请访问稀酷客(www.ckook.com)

```
a.toString(16)
```

相关主题：toLocaleString ()方法。

number.valueOf()

参见第 15 章中的 string.valueOf()。

16.4 Boolean 对象

属　　性	方　　法
constructor	toString()
prototype	valueOf()

16.4.1 语法

创建 Boolean 对象：

```
var val = new Boolean(BooleanValue);
```

访问 Boolean 对象的属性：

```
BooleanObject.property | method
```

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

16.4.2 关于 Boolean 对象

在 JavaScript 中，Boolean 值很常用，一般是作为条件检测的结果。字符串值可以使用与 String 对象相关的属性和方法，同样，Boolean 值也可以从 Boolean 对象中获得帮助。例如，在文本框中显示 Boolean 值时，true 或 false 字符串由 Boolean 对象的 toString ()方法提供，这样就不必直接调用这个方法。

只有需要将一些属性或方法加入用 new Boolean()构造函数创建的 Boolean 对象时，才使用 Boolean 对象。构造函数的参数可以是值的字符串版本，数字(0 表示 false；其他整数表示 true)和计算结果为 Boolean 值的表达式。如果将一些新属性或方法加入核心 Boolean 对象的 prototype 属性，其他新 Boolean 对象也会包含这些新的属性或方法。

有关 Boolean 对象属性和方法的细节，请参见第 15 章中 String 对象的相应列表。

更多资源请访问稀酷客(www.ckook.com)

第17章

Date 对象

> **本章包含哪些内容？**
> - 在 JavaScript 中处理日期和时间值
> - 进行日期计算
> - 验证日期输入表单域

JavaScript 的日期和时间处理能力并未得到完全的应用，在早期的 JavaScript 中，脚本编写者忽略 Date 对象有很正当的理由：在早期的脚本浏览器中，如果不做彻底的检测，许多重大错误和操作平台异常会使日期和时间编程非常危险。即使更正了这些错误，也需要了解世界时区及其与标准参考点(称为 Greenwich Mean Time(GMT)或 Coordinated Universal Time(UTC))之间的关系，才能正确处理日期。

现代浏览器增强了时间和日期的处理能力。我们希望脚本编写者认真研究如何在页面中合并这些计算。例如，光盘上的第 57 章介绍了一个应用程序，可使网站高亮显示每个访问者上次浏览页面后更新的区域，这个应用程序极大地依赖于日期算法和时区转换功能。

但在开始讨论 JavaScript 的日期之前，本章先简要介绍时区的要点及其对浏览器中时间和日期脚本编程的影响。若不知道 GMT 和 UTC 的含义，应阅读下一节。

17.1 时区和 GMT

根据国际协议，世界分为不同的时区，这样每个时区的居民都可以认为，太阳出现在头顶时，正好是中午，即一天的中间时刻。时区中的当前时间就是时钟的时间——当地时间。

如果一个人的活动范围没有超出本地的时区，就没有什么问题。但要在世界各地进行即时通讯，其范围就超出了本地时区。要定期注意其他时区的本地时间。毕竟，生活在纽约的人不会在天亮之前从办公室打电话到洛杉矶以免惊醒对方。

注意：

在本节的其余部分提到，太阳在“移动”，就像地球是太阳系的中心一样。这样说是为了方便，在我们日常的感觉中，天空是固定的，太阳从天空中穿越而过。当然，我们相信哥白尼的理论，所以，如果你在这个方面有异议，不必给我们发送电子邮件。

太阳在任何给定的时刻都位于某个时区，从这个时区观察，东边的所有时区已过了中午，

更多资源请访问稀酷客(www.ckook.com)

所以此时区晚于这些时区——每个时区滞后一小时(有几个时区的滞后时间少于一小时)。这就是为什么美国电视网络同时向东部时区和中部时区广播的原因，节目的播出时间表一般是“东部时间 10 点、中部时间 9 点”。

许多跨国公司调整各个事件的时间表时，都要考虑时区的差异(有时还要考虑季节变化，例如夏令时)，这简直是一场噩梦。为了完成这项工作，人们设计了一个标准参考点：时区以格林威治天文台为中心分布。这个时区称为 Greenwich Mean Time，简称 GMT。Mean 的含义是该时区在地球另一面的对应时区是国际日期变更线，这是另一个世界标准位置，是地球上下一天开始的地方，这样 GMT 位于一天的中间或平均点。多年以前，GMT 有另一个缩写 UTC，它不基于地球上的任何语言，其英语版本是 Coordinated Universal Time，它的实际含义与 GMT 相同。

若个人计算机的系统时钟设置正确，计算机就以 GMT 时间进行计时。但因为当地时区在控制面板上设置，所有的文件时间和时钟都显示当地时间。计算机知道当地时间和 GMT 的差值，对于夏令时，必须检查首选项设置，相应地调整偏移量；在基于 Windows 的操作系统中，系统知道何时发生改变，并提醒用户偏移量的调整是否合适。拿着笔记本电脑穿越时区时，应改变计算机的时区设置，而不是时钟。

JavaScript 内部处理日期和时间的方式类似于 PC 时钟(程序以 PC 时钟为准)。脚本中生成的日期值在内部存储为 GMT 时间；但几乎所有显示和提取出的值都是访问者的当地时间(不是网站服务器)。记住，日期值是脚本在访问者的计算机上创建的，不必从服务器向客户端发送当前日期对象。在使用 JavaScript 时间和日期时，这个概念也许是最难理解的。

只要使用 JavaScript 为公共网页编写时间和日期程序，都必须有全局观点，它要求通过访问者的计算机设置来完成 GMT 和当地时间的精确转换。还要假设自己暂时在世界上的其他地区，将计算机的时钟改为这些地区的时间，做一些测试，这一般不容易做到。作者在澳大利亚的悉尼访问时，晚上睡觉前打开饭店的电视，电视正在通过卫星现场转播美国电视节目 Today，这个节目在纽约是早上播出，而此时悉尼恰好是晚上。是的，这个时区的信息令人感到混乱。

17.2 Date 对象

与 JavaScript 和 DOM 中的其他对象相同，每个窗口(或框架)中的单个静态 Date 对象和包含具体时间和日期的 date 对象之间是有区别的。静态 Date 对象(大写的 D)仅在几种情况下使用：主要是创建日期的新实例，调用 Date 对象提供的方法，用于一些通用的转换。

我们处理的大多数日期和时间都是 Date 对象的实例，这些实例一般称为 date 对象(小写的 d)。每个 date 对象都是毫秒级时间的快照，不管该快照是生成对象的快照，还是需要计算的过去或将来某个时间点的快照。若需要让时钟计时，脚本要反复生成新的 date 对象，来得到计算机时钟的毫秒级快照。要在网页上显示时间，可从 date 对象的快照中提取小时、分钟、秒，按照需要的样式显示(例如，数字读出器、条形图等)；一般使用脚本调用 date 对象实例的方法，来读取和更改 date 对象的各个组成部分(如月或小时)。

不管其名称如何，每个 date 对象都包含日期和时间信息，即使只关心对象数据的日期部分，时间数据也存在。随着学习的进一步深入，时间元素会在一些操作上让人措手不及。

更多资源请访问稀酷客(www.ckook.com)

17.2.1　创建 date 对象

JavaScript 脚本使用特殊的对象构造关键字 new 来生成对象。生成新 date 对象的基本语法如下：

```
var dateObjectName = new Date([parameters]);
```

date 对象是 object 数据类型，而不是字符串或数值。

将 date 对象引用放在变量名中，就可以使用自己熟悉的圆点语法访问所有面向日期的方法：

```
var result = dateObjectName.method();
```

脚本通过变量(如 result)来计算或显示 date 对象的数据(一些方法提取该对象中的日期和时间部分)。若想将新的值放入 date 对象(如将年份加入 date 对象)，可以通过设置值的方法，为对象分配新的值：

```
dateObjectName.method(newValue);
```

本语句不像一般的 JavaScript 赋值语句，因为一般的赋值语句都包含一个等号操作符，但该语句是设置 date 对象数据的方法的工作方式。

没有彻底理解时区计算之前，不要急着给日期编写脚本。JavaScript 可以显示 date 对象在本地时区的字符串形式，但实际内部存储的却是严格的 GMT 形式数据。

尽管尚未详细介绍 date 对象的方法，但下面说明了如何使用两个方法将当前的日期增加一年：

```
var oneDate = new Date();           // creates object with current GMT date
var theYear = oneDate.getYear();    // stores the current four-digit year
theYear = theYear + 1;              // theYear now is next year
oneDate.setYear(theYear);           // the new year value now in the object
```

在最后一个语句中，oneDate 对象自动为下一年的日期调整其他日期成分。例如，这一天的星期将会不同，如果需要提取星期数据，JavaScript 会自动处理。oneDate 对象包含下一年的数据后，可将新的日期提取为字符串值，显示在网页的一个域中，或将它提交给服务器上的 CGI 程序。

创建新 date 对象的参数比较复杂，这主要是因为 JavaScript 为脚本编写者提供了很多灵活性。newDate()语句可在内存中创建一个空间，用于存储日期所需的所有数据。这项任务中缺少的是数据——把什么日期和时间放进内存中，而参数就提供了日期和时间。

若参数为空，JavaScript 自动将当天的日期和当前的时间赋给新的 date 对象。JavaScript 并没有那么智能，它只是使用了页面访问者的计算机时钟设置。如果时钟不正确，JavaScript 也不能正确地确定日期和时间。

注意：

在创建新的 date 对象时，还包含当前时间。因为当前日期包含时间 16:03:19(24 小时制)，所以在计算日期之间的天数时可能会产生误差。

要为具体的日期和时间创建 date 对象，可使用 5 种方式将值作为参数传递给 new Date()构造函数：

更多资源请访问稀酷客(www.ckook.com)

```
new Date("Month dd, yyyy hh:mm:ss")
new Date("Month dd, yyyy")
new Date(yy,mm,dd,hh,mm,ss)
new Date(yy,mm,dd)
new Date(milliseconds)
```

前 4 种方式分为两类：长字符串和逗号分隔的数据列表，每种方式都有可选的时间设置。若忽略时间设置，则无论输入什么日期，data 对象中的时间都是 0；在参数中不能忽略日期值，不管需要与否，每个 date 对象都必须有一个真实日期。

在长字符串版本中，月份应该用英语全部拼出，不允许使用缩写。剩下的数据是表示天、年、小时、分钟和秒的数值，有时其顺序可能不同于本地表示日期的方法。如果数据只有一位数字，则可使用一位或两位数字，如 4:05:00，冒号将小时、分钟和秒分开。

短版本包含不带引号的整数值列表，按指定的顺序排列，如果偶尔将 30 放在月份中，JavaScript 不会将它识别为日期。

只有包含日期和时间的毫秒值，才使用最后一种方法。一般在执行数学计算后，会返回毫秒格式的时间和日期(参见本章后面的讨论)。要将该数值转换为 date 对象，可利用 new Date()构造函数。从新建的日期对象中可以得到许多关于时间和日期的有用值。

17.2.2 内部对象的属性和方法

与 String 和 Array 对象一样，Date 对象也有少量 JavaScript 内部对象共有的属性和方法。另外，Date 对象还有 prototype 属性，可将新的属性和方法用于当前页面中创建的每个 date 对象。在讨论 String 对象和 Array 对象的 prototype 属性时，介绍了使用该属性的示例(参见第 15 章和第 18 章)。同时，在现代浏览器中，date 对象的每个实例都有 constructor 属性，它引用生成该对象的构造函数。

date 对象有许多将 date 对象类型转换为字符串的方法，这些方法的针对性大多比常规的 toString()更强。valueOf()方法返回毫秒级的整数，来存储特定日期。

17.2.3 日期方法

date 对象的方法一般用来读取日期和时间信息，改变存储在对象中的日期和时间。这两类方法很容易区分，因为它们都以 get 或 set 开头。

表 17-1 列出了静态 Date 对象的所有方法，以及 date 对象实例继承的方法。它们非常有趣，有些也令人感到恐惧，但其模式很容易理解。大多数方法只处理日期和时间的单个部分：年、月、日等。每组 get 和 set 方法也有两类：一类将当地时间和日期转换为存储在对象中的日期，一类用于存储在对象中的实际 UTC 日期。了解到这些内容后，下表就很容易管理。除非另外提示，否则这些方法从第一代脚本浏览器开始就是 Date 对象的一部分，新的浏览器也支持它们。

表 17-1 date 对象的方法

方 法	值的范围	说 明
dateObj.getFullYear()	1970～…	获取指定的年份(NN4+，Moz1+，IE3+)
dateObj.getYear()	70～…	参见正文
dateObj.getMonth()	0～11	获取月份((1 月＝0)

更多资源请访问稀酷客(www.ckook.com)

(续表)

方　法	值的范围	说　明
dateObj.getDate()	1～31	获取一个月的第几日
dateObj.getDay()	0～6	获取星期几(星期日=0)
dateObj.getHours()	0～23	获取小时数(24 小时制)
dateObj.getMinutes()	0～59	获取分钟数
dateObj.getSeconds()	0～59	获取秒数
dateObj.getTime()	0～…	获取 1/1/70 00:00:00 GMT 至今的毫秒
dateObj.getMilliseconds()	0～999	获取前一个完整秒至今的毫秒数(NN4+、Moz1+、IE3+)
dateObj.getUTCFullYear()	1970～…	获取指定的 UTC 年(NN4+、Moz1+、IE3+)
dateObj.getUTCMonth()	0～11	获取 UTC 月份(1 月＝0) (NN4+、Moz1+、IE3+)
dateObj.getUTCDate()	1～31	获取 UTC 月份的第几日(NN4+、Moz1+、IE3+)
dateObj.getUTCDay()	0～6	获取 UTC 星期几(星期日＝0)(NN4+、Moz1+、IE3+)
dateObj.getUTCHours()	0～23	获取 UTC 小时数，24 小时制(NN4+、Moz1+、IE3+)
dateObj.getUTCMinutes()	0～59	获取 UTC 分钟数(NN4+、Moz1+、IE3+)
dateObj.getUTCSeconds()	0～59	获取 UTC 秒数(NN4+、Moz1+、IE3+)
dateObj.getUTCMilliseconds()	0～999	获取前一个完整秒至今的 UTC 毫秒数(NN4+、Moz1+、IE3+)
dateObj.setYear(*val*)	1970～…	注意，总是指定 4 位数字的年份
dateObj.setFullYear(*val*)	1970～…	设置指定的年(NN4+、Moz1+、IE3+)
dateObj.setMonth(*val*)	0～11	设置月份(1 月＝0)
dateObj.setDate(*val*)	1～31	设置月中的第几日
dateObj.setDay(*val*)	0～6	设置星期几(星期日＝0)
dateObj.setHours(*val*)	0～23	设置小时数，24 小时制
dateObj.setMinutes(*val*)	0～59	设置分钟数
dateObj.setSeconds(*val*)	0～59	设置秒数
dateObj.setMilliseconds(*val*)	0～999	设置前一个完整秒至今的毫秒数(NN4+，Moz1+，IE3+)
dateObj.setTime(*val*)	0～…	设置 1/1/70 00:00:00 GMT 至今的毫秒数
dateObj.setUTCFullYear(*val*)	1970～…	设置指定的 UTC 年(NN4+、Moz1+、IE3+)
dateObj.setUTCMonth(*val*)	0～11	设置 UTC 年的月份(1 月=0) (NN4+、Moz1+、IE3+)
dateObj.setUTCDate(*val*)	1～31	设置 UTC 月中的第几天(NN4+、Moz1+、IE3+)
dateObj.setUTCDay(*val*)	0～6	设置 UTC 星期几(星期日＝0) (NN4+、Moz1+、IE3+)
dateObj.setUTCHours(*val*)	0～23	设置 UTC 小时数，24 小时制(NN4+、Moz1+、IE3+)
dateObj.setUTCMinutes(*val*)	0～59	设置 UTC 分钟数(NN4+、Moz1+、IE3+)
dateObj.setUTCSeconds(*val*)	0～59	设置 UTC 秒数(NN4+、Moz1+、IE3+)
dateObj.setUTCMilliseconds(*val*)	0～999	设置前一个完整秒至今的 UTC 毫秒数(NN4+、Moz1+、IE3+)

更多资源请访问稀酷客(www.ckook.com)

(续表)

方　法	值的范围	说　明
dateObj.getTimezoneoffset()	0～…	获取 GMT/UTC 的分钟偏移量
dateObj.toDateString()		用浏览器确定的格式表示只包含日期的字符串(WinIE5.5+)
dateObj.toGMTString()		用通用格式表示日期/时间字符串
dateObj.toLocaleDateString()		用系统的本地格式表示只包含日期的字符串(NN6+、Moz1+、WinIE5.5+)
dateObj.toLocaleString()		用系统的本地格式表示日期/时间字符串
dateObj.toLocaleTimeString()		用系统的本地格式表示只包含时间的字符串(NN6+、Moz1+、WinIE5.5+)
dateObj.toString()		用浏览器确定的格式表示日期/时间字符串
dateObj.toTimeString()		用浏览器确定的格式表示只包含时间的字符串(WinIE5.5+)
dateObj.toUTCString()		用通用格式表示日期/时间字符串(NN4+、Moz1+、IE3+)
Date.parse("*dateString*")		将日期字符串转换为毫秒级整数
Date.UTC(*date values*)		将 GMT 日期字符串转换为毫秒级整数

使用方法的 UTC 版本还是当地时间版本取决于几个因素。若支持的浏览器要向后兼容，就必须使用当地时间版本。但对于较新的浏览器而言，计算两个日期之间的天数或为测验创建一个倒数计时器等操作就不必关心使用什么版本，只要计算中使用的标准相同即可。若混合使用当地时间版本和 UTC 版本的 date 方法，一定会出错。若日期计算必须相对于另一个时区的绝对时间来考虑客户机的时区，则使用 UTC 版本的方法最方便，比如计算现在到英国伦敦议会大厦的大钟新年钟声敲响的时间。

JavaScript 以 GMT(UTC)时区中从 1970 年 1 月 1 日开始计数的毫秒值来保存日期信息，这个起始点以前的日期保存为负值(参见本章后面的 17.2.10 一节)。不管在哪个国家、不管计算机指定了什么日期和时间格式，毫秒都是 JavaScript 的时间统一度量标准。为确保精度，任何涉及时间和日期加减的操作都用毫秒值来计算。虽然在文本域或对话框中不显示毫秒值，脚本也经常在变量中处理毫秒值。要得到 date 对象中时间和日期的毫秒值，可使用 dateObj.getTime()方法：

```
var startDate = new Date();
var started = startDate.getTime();
```

尽管方法名中有 time，但其值是从 1970 年 1 月 1 日开始计数的整毫秒数，这就是说该值还包括了日期。

date 对象的其他 get 方法读取日期或时间的特定部分，但使用时要非常小心，因为一些值从 0 开始计算，这可能是用户想不到的。例如，在 JavaScript 标准中，一月是 0；十二月是 11；小时、分钟和秒都从 0 开始，这是符合逻辑的。但日期使用的是墙上日历中的实际值，一个月的第一天是日期值 1。对于 20 世纪中的年，年份值是真实的年数减去 1900，例如 1996 年的年份值为 96；但 1900 年之前和 1999 年之后的年份，JavaScript 使用另一种格式：显示完整的年份值。这就是说在显示年份之前，必须检查年份值是否小于 100，如果是，就加上 1900。

更多资源请访问稀酷客(www.ckook.com)

```
var today = new Date();
var thisYear = today.getYear();
   if (thisYear < 100)
   {
      thisYear += 1900;
   }
```

当然，这要假设不处理公元 100 年以前的年份，若用户使用现代的浏览器(其可能性很大)，就只需使用 getFullYear()方法，它返回所有范围内的完整年份。

要调整日期值的任何部分，可在赋值语句中使用相应的 set 方法。如果新值强制调整其他元素，JavaScript 会自动予以处理。例如，考虑下面示例中的一些值是如何改变的：

```
myBirthday = new Date("July 4, 1776");
result = myBirthday.getDay();        // result = 4, a Thursday
myBirthday.setYear(1777);            // bump up to next year
result = myBirthday.getDay();        // result = 5, a Friday
```

同一个日期在下一年中是不同的日子，JavaScript 会自动处理。

17.2.4 处理时区

dateObj.getTimezoneOffset()方法涉及操作系统的时区控制面板设置，以及表示日期和时间的国际认可格式(在计算机界)。若忽略控制面板上设置的当地时区，该属性的返回值对于大部分时间和日期来说是不对的。例如，北美东部标准时间比格林尼治标准时间早 5 小时，getTimezone Offset()方法会生成 GMT 和 PC 时区之差(用分钟表示)，东部标准时间 5 小时的差异会显示为 300 分钟。在 Windows 平台上，值会自动改变，来反映用户区域的夏令时间。向 GMT 东部偏移(相对于日期变更线)用负值表示。

17.2.5 字符串日期

生成 date 对象时，要在页面或警告框中显示日期，JavaScript 会自动对 date 对象应用 toString()方法。该字符串的格式因浏览器和操作系统平台而异。例如在 Windows XP 的 IE8 中，字符串的格式如下：

```
Sun Dec 5 16:47:20 PST 2010
```

但在 Windows XP 的 Firefox 中，字符串为：

```
Sun Dec 05 2010 16:47:20 GMT-0800 (Pacific Standard Time)
```

对于这个字符串，其他浏览器会返回自己的版本。要点是，对于日期的组成部分，不要依赖这个字符串的具体格式和字符位置。应使用 date 对象的方法读取 date 对象的组成部分。

但 JavaScript 使用两个方法返回常量字符串格式的 date 对象。一个方法是 dateObj. toGMTString()，它可以将日期和时间转换为 GMT 形式，并赋给存储所提取数据的变量，数据如下所示：

```
Mon, 06 Dec 2010 00:47:20 GMT
```

若不熟悉 GMT 数据，也不明白这些转换可能会产生意料不到的日期，在测试应用程序时

更多资源请访问稀酷客(www.ckook.com)

要特别小心。北美西海岸一个星期日下午的 5 点差一刻，正好是 GMT 的星期一午夜。

时区转换可能令人头疼，但可以使用第二个字符串方法 dateObj.toLocaleString()。在北美 Windows 用户的 Firefox 上，返回值如下：

```
Sunday, December 05, 2010 4:47:20 PM
```

从 IE5.5 和 NN6/Moz1 开始，可以用 JavaScript 转换 date 对象，使其只包含时间或日期，且以希望的格式显示。最好的方法是 toLocaleDateString()和 toLocaleTimeString()，因为这些方法可以根据用户操作系统和浏览器的本地设置，返回对用户最有意义的值。

17.2.6 用于以前浏览器的日期格式

若编写的脚本不只用于现代的浏览器，就可以创建自己的格式化函数，以用于多数浏览器。程序清单 17-1 是从 date 对象中创建这种字符串的方法，其形式可以向后兼容版本 4 浏览器。

注意：

本章和本书许多章节的代码中指定事件处理程序的函数是 addEvent()，这个跨浏览器的事件处理程序详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，该文件可在配书光盘中找到，所有章节的脚本都可以访问它。

程序清单 17-1 创建友好的日期字符串

HTML: jsb-17-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date String Maker</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-17-01.js"></script>
   </head>
   <body>
      <h1>Date String Maker</h1>
      <p id="output">Today’s date</p>
   </body>
</html>
```

JavaScript: jsb-17-01.js

```
// initialize when the page has loaded
addEvent(window, ‘load’, initialize);

monthNames = ["January", "February", "March", "April", "May", "June", "July",
   "August", "September", "October", "November", "December"];
dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
   "Saturday"];
```


更多资源请访问稀酷客(www.ckook.com)

```
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to the paragraph to contain the date
      oOutput = document.getElementById(' output' );

         // if it exists, replace its contents with the date
         if (oOutput)
         {
            // remove all child nodes from the paragraph
            while (oOutput.firstChild)
            {
               oOutput.removeChild(oOutput.firstChild);
            }

            // create a text node with the date
            var sDate = customDateString(new Date());
            var oNewText = document.createTextNode(sDate);

            // insert that content into the paragraph
            oOutput.appendChild(oNewText);
         }
   }
}

// generate a formatted date
function customDateString(oDate)
{
   var theDay = dayNames[oDate.getDay()];
   var theMonth = monthNames[oDate.getMonth()];
   var theYear = oDate.getFullYear();
   return theDay + ", " + theMonth + " " + oDate.getDate() + ", " + theYear;
}
```

假设用户的 PC 时钟设置正确，则标题下面的日期是当前日期——就好像文档是当天更新的。这种方法的缺陷(相对于较新的 toLocaleDateString()方法)是国外用户被迫使用设计好的日期格式，这可能与他们当地的习惯不同。

17.2.7　更多转换

程序清单 17-1 中的最后两个方法是静态 Date 对象的方法，这些方法将日期从字符串或数值格式转换为日期的毫秒值。其主要受益者是 dateObj.setTime()方法，该方法要求将日期的毫秒值作为参数，可将一个完全不同的日期放入已有的 date 对象。

Date.parse()接收日期字符串参数，如本节前面所述，它包括国际承认的标准。另一方面，Date.UTC()需要 GMT 时区中以逗号分隔的值，顺序是 yy,mm,dd,hh,mm,ss。Date.UTC()方法以向后兼容的方式对 GMT 时间进行硬编码(可通过 UTC 方法在版本 4 的浏览器中实现相同的功能)。下面是在 WinIE8 中为 2011 年 10 月 1 日下午 6 点的 GMT 时间创建新的 date 对象：

更多资源请访问稀酷客(www.ckook.com)

```
var newObj = new Date(Date.UTC(2011, 9, 1, 18, 0, 0));
result = newObj.toString(); // result = "Sat Oct 1 11:00:00 PDT2011"
```

第二个语句返回本地时区的值，因为所有非 UTC 方法都自动将保存在对象中的 GMT 时间转换为客户机的本地时间。

17.2.8　日期和时间运算

对日期执行数学计算有很多原因。也许是计算固定天数或星期后的日期，或计算两个日期之间的天数。在涉及这些类型的计算中，应使用 JavaScript 日期值的通用单位：毫秒。

在涉及大量日期的脚本中，应创建一些表示分钟、小时、日、星期的毫秒值的变量，然后在计算中使用这些变量。下面是根据前一个变量创建下一个变量的示例：

```
var oneMinute = 60 * 1000;
var oneHour = oneMinute * 60;
var oneDay = oneHour * 24;
var oneWeek = oneDay * 7;
```

在脚本中使用这些值，即可计算从今天开始的一个星期后的日期：

```
var targetDate = new Date();
var dateInMs = targetDate.getTime();
dateInMs += oneWeek;
targetDate.setTime(dateInMs);
```

另一个示例利用 date 对象的组成部分，根据用户 PC 时钟的本地时间，来决定在文档中设置什么类型的问候消息。程序清单 17-2 在程序清单 17-1 的基础上加入几行脚本，使脚本的处理更趋智能化。

程序清单 17-2　动态欢迎消息

HTML: jsb-17-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date String Maker</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-17-02.js"></script>
   </head>
   <body>
      <h1>Welcome!</h1>
      <p id="date">Today’s date</p>
      <p>We hope you are enjoying the <span id="day-part">day</span>.</p>
   </body>
</html>
```

JavaScript: jsb-17-02.js

更多资源请访问稀酷客(www.ckook.com)

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

monthNames = ["January", "February", "March", "April", "May", "June", "July",
   "August", "September", "October", "November", "December"];
dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
   "Saturday"];

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to the target paragraphs
      var oDateDisplay = document.getElementById('date');
      var oDayPart = document.getElementById('day-part');

         // if they exist, plug in new content
         if (oDateDisplay && oDayPart)
         {
            // plug in date
            var oNow = new Date();
            var sDate = customDateString(oNow);
            replaceTextContent(oDateDisplay, sDate);

            // plug day-part into greeting
            var sDayPart = dayPart(oNow);
            replaceTextContent(oDayPart, sDayPart);
         }
   }
}

// generate a formatted date
function customDateString(oDate)
{
   var theDay = dayNames[oDate.getDay()];
   var theMonth = monthNames[oDate.getMonth()];
   var theYear = oDate.getFullYear();
   return theDay + ", " + theMonth + " " + oDate.getDate() + ", " + theYear;
}

// get the part of the day
function dayPart(oDate)
{
   var theHour = oDate.getHours();
   if (theHour < 6 )
      return "wee hours";
   if (theHour < 12)
      return "morning";
   if (theHour < 18)
      return "afternoon";
   return "evening";
```


更多资源请访问稀酷客(www.ckook.com)

```
}

// replaces the text contents of a page element
function replaceTextContent(oElement, sContent)
{
   // if the object exists
   if (oElement)
   {
      // remove all child nodes
      while (oElement.firstChild)
      {
         oElement.removeChild(oElement.firstChild);
      }

      // create a text node with the new content
      var oNewText = document.createTextNode(sContent);

      // insert that content
      oElement.appendChild(oNewText);
   }
}
```

脚本将一天分为 4 个阶段，为每个阶段显示不同的问候消息，问候消息非常简单，date 对象基于 date 对象的小时部分，表示浏览器载入页面的时间。因为问候消息嵌入在页面中，不管用户在页面上停留多久，消息也不会改变。但每次页面重新加载时，问候消息都会更新。

17.2.9　计算天数

现在，介绍一两个比较实用的日期计算应用程序。一个是计算圣诞节前所剩的天数(在用户的时区中)；另一个是到 2100 年的倒计时计时器。

程序清单 17-3 说明如何计算现在与未来某个日期之间的天数，该应用程序假设所有的计算都在用户的时区内进行。示例显示了到下一个圣诞节(12 月 25 日)的天数，基本操作是，将当前日期和下一个 12 月 25 日转换成毫秒，计算毫秒的差值所表示的天数。如果使用毫秒值表示日期，JavaScript 将自动处理闰年的情况。

唯一困难的部分是正确设置下一个圣诞节的年份。不能只处理当年的日期，因为如果程序在 12 月 26 日运行，下一个圣诞节的年份就必须加一。所以，圣诞节的 date 对象构造函数不将固定日期作为参数，而必须分别设置 date 对象中的每个组成部分。

另外注意，setDate()的参数是整数 1～31，而 setMonth()的参数是整数 0～11，这个方法调用把 11 作为 12 月。

程序清单 17-3　到下一个圣诞节的天数

HTML: jsb-17-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>Christmas Countdown</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb-17-03.js"></script>
    </head>
    <body>
        <h1>Christmas Countdown</h1>
        <p>You have <em id="days-left">too few</em> shopping
            <span id="day-word">days</span> until Christmas.</p>
    </body>
</html>
```

JavaScript: jsb-17-03.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to the target paragraphs
        var oOutput = document.getElementById('days-left');
        var oDayWord = document.getElementById('day-word');

            // if they exist, plug in new content
            if (oOutput && oDayWord)
            {
                // plug in days left
                var sDaysLeft = getDaysUntilXmas();
                replaceTextContent(oOutput, sDaysLeft);
                // pluralize day if not one
                var sDayWord = (sDaysLeft == 1) ? 'day' : 'days';
                replaceTextContent(oDayWord, sDayWord);
            }
    }
}

// calculate the number of days till next Christmas
function getDaysUntilXmas()
{
    var oneMinute = 60 * 1000;
    var oneHour = oneMinute * 60;
    var oneDay = oneHour * 24;

    var today = new Date();
    var nextXmas = new Date();
    nextXmas.setMonth(11);
    nextXmas.setDate(25);
        if (today.getMonth() == 11 && today.getDate() > 25)
        {
            nextXmas.setFullYear(nextXmas.getFullYear() + 1);
```


更多资源请访问稀酷客(www.ckook.com)

```
      }
   var diff = nextXmas.getTime() - today.getTime();
   diff = Math.floor(diff/oneDay);
   return diff;
}

// replaces the text contents of a page element
function replaceTextContent(oElement, sContent)
{
   [see listing 17-2]
}
```

第二个示例计算距离特定事件发生还有多长时间，这需要考虑时区。为此，设计一个页面，显示到 2008 年北京夏季奥运会圣火点燃那一刻的倒计时。这个事件所在的时区不同于页面浏览者所在的时区，倒计时必须计算相应的时差。

程序清单 17-4 是一个简化的示例，它仅在文本域中显示计时的时钟。当然，输出可以根据页面使用的动态 HTML 的数量，采用不同的方式进行定制。这个演示将点燃圣火的时间设为 GMT 时间，2008 年 8 月 8 日的 11:00，日期是准确的，而当地官方可能设置一个更接近实际事件的时间。

这个应用程序实现为一个实时的时钟，所以代码先设置一些只计算一次的全局变量，这样重复调用的函数的计算量最小(更高效)。Date.UTC()方法采用标准时间格式提供目标时间和日期；getTimeUntil()函数接收毫秒值(targetDate 变量提供)，并计算目标日期和客户 PC 时钟内部实际的毫秒值之差。

getCountDown()函数的核心是计算现在和目标日期之间的整数毫秒差，从中提取出天数、小时、分钟和秒。注意每一块都从总数中减去，这样下一个块可以从剩下的毫秒数中计算。

在这个页面上，还要显示实际事件的本地日期和时间。

程序清单 17-4 夏季奥运会的倒计时

HTML: jsb-17-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Time Since the First Moon Landing</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-17-04.js"></script>
   </head>
   <body>
      <h1>Time Since the First Moon Landing</h1>
      <p>It is now <span id="now">the present</span> in your timezone.</p>
      <p><span id="time-since">Many days</span> have passed since the first
         moon landing on <span id="past-day">a day long ago</span>.</p>
   </body>
</html>
```

JavaScript: jsb-17-04.js

更多资源请访问稀酷客(www.ckook.com)

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to the target elements
      oOutputNow = document.getElementById('now');
      oOutputTimeSince = document.getElementById('time-since');
      var oOutputPastDay = document.getElementById('past-day');

         // if they exist, plug in new content
         if (oOutputNow && oOutputTimeSince && oOutputPastDay)
         {
            // globals -- calculate only once
            // set target date to 20:17 UTC on July 20, 1969
            targetDate = Date.UTC(1969, 6, 20, 20, 17, 0, 0);
            oneSecond = 1000;
            oneMinute = oneSecond * 60;
            oneHour = oneMinute * 60;
            oneDay = oneHour * 24;

            // plug in the past date
            var sPastDay = (new Date(targetDate)).toLocaleString();
            replaceTextContent(oOutputPastDay, sPastDay);

            // plug in current time & time since
            updateCounter();
         }
   }
}

// timer loop to display changing values
function updateCounter()
{
   // plug in current time
   var sNow = (new Date()).toLocaleString();
   replaceTextContent(oOutputNow, sNow);

   // plug in days since
   var sTimeSince = formatTimeSince();
   replaceTextContent(oOutputTimeSince, sTimeSince);

   setTimeout("updateCounter()", 1000);
}

// format elapsed time
function formatTimeSince()
{
   var ms = getTimeDiff(targetDate);
   var days, hrs, mins, secs;
```


更多资源请访问稀酷客(www.ckook.com)

```
    days = Math.floor(ms/oneDay);
    output = days + " Day";
       if (days != 1) output += 's';

    ms -= oneDay * days;
    hrs = Math.floor(ms/oneHour);
    output += ", " + hrs + " Hour";
       if (hrs != 1) output += 's';
    ms -= oneHour * hrs;
    mins = Math.floor(ms/oneMinute);
    output += ", " + mins + " Minute";
       if (mins != 1) output += 's';

    ms -= oneMinute * mins;
    secs = Math.floor(ms/oneSecond);
    output += ", and " + secs + " Second";
       if (secs != 1) output += 's';

    return output;
}

// get the difference between two datetimes in milliseconds
function getTimeDiff(targetMS)
{
    var today = new Date();
    var diff = Math.abs(today.valueOf() - targetMS);
    return Math.floor(diff);
}

// replaces the text contents of a page element
function replaceTextContent(oElement, sContent)
{
    [see listing 17-2]
}
```

17.2.10　早期浏览器中日期的错误和漏洞

每个新浏览器版本都会改进 date 脚本对象的稳定性和可靠性。例如，Netscape Navigator 2 有许多错误和致命问题，因此几乎不可能为这个浏览器编写世界时间的复杂应用程序。NN3 有所改进，但仍存在一些明显的问题。选择使用 Netscape，是因为 Internet Explorer 的早期版本有大量日期和时间问题。IE3 不能处理 1970 年 1 月 1 日(GMT)以前的日期，时区偏移量的计算完全错误，它还有 NN2 的问题。问题是，如果必须处理大量日期和时间，同时要支持以前的浏览器，就会遇到麻烦。

注意，Navigator 版本 4 之前的 Mac 和 Windows 版本之间还有一个差异。在 Windows 中，如果为某年的另一日期生成 date 对象，浏览器就会根据该年份的时区设置，设置那个对象的时区偏移量。在 Mac 中，控制面板的当前设置确定对日期是应用普通时间，还是夏令时间偏移，而不管这个日期在那一年的什么季节。这种差异对 Navigator 3 和 4 有影响，而且在计算时，会产生一个小时的差异。

看来，在 Date 对象的脚本编程中，处处都埋着地雷。尽管日期和时间的脚本编程还有问题，

更多资源请访问稀酷客(www.ckook.com)

但只要仔细计划，大量测试，就可以避免这些问题。比较好的情况是，如果可以假定大量用户都使用现代浏览器(WinIE6+，NN6+，Mozl+，FF1+，Caml+，Safaril+等)，脚本编程就会比较顺利。

17.3　在表单中验证日期项

上一节介绍了这么多的错误，那么，如何在表单中验证数据项？其实数据计算的问题并不多，而可接受的日期格式有不少问题。无论如何指导用户按照指定的格式输入日期，用户也常常遵循自己的习惯和风俗。而且，怎样确定输入的 03/04/2010 是北美的 2010 年 3 月 4 日，还是欧洲的 2010 年 4 月 3 日呢？无法确定！

建议将日期域分为三部分：月、日、年。让用户在每个域中输入值，分别验证每个域中的输入是否在有效范围内。程序清单 17-5 演示了一个验证示例。页面包括一个提交前进行验证的表单，每个域在用户输入时执行各自的有效范围验证。但因为这样的验证可能失败，所以页面中的表单 onsubmit 事件处理程序会执行进一步的检查。只要任何一个域失败，就取消表单的提交。

程序清单 17-5　在表单中验证日期

HTML：jsb-17-05.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Date Entry Validation</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-17-05.js"></script>
   </head>
   <body>
      <h1>Date Entry Validation</h1>
      <form id="birthdate" action="example.php" method="post">
         <p>Please enter your birthdate...</p>
         <p>
            <label for="month">Month:</label>
            <input type="text" id="month" name="month" value="1" size="2"
               maxlength="2">

            <label for="day">Day:</label>
            <input type="text" id="day" name="day" value="1" size="2"
               maxlength="2">

            <label for="year">Year:</label>
            <input type="text" id="year" name="year" value="1900" size="4"
               maxlength="4">
         </p>
         <p>Thank you for entering <span id="fullDate">your birthdate</span>.</p>
         <p>
            <input type="submit">
```


更多资源请访问稀酷客(www.ckook.com)

```
            <input type="Reset">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-17-05.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to the target elements
      oForm = document.getElementById('birthdate');
      oMonth = document.getElementById('month');
      oDay = document.getElementById('day');
      oYear = document.getElementById('year');
      oFullDate = document.getElementById('fullDate');

         // if they exist, plug in new content
         if (oForm && oMonth && oDay && oYear && oFullDate)
         {
            // apply behaviors to form elements
            oForm.onsubmit = checkForm;
            oMonth.onchange = function() { return validateMonth(oMonth); }
            oDay.onchange = function() { return validateDay(oDay); }
            oYear.onchange = function() { return validateYear(oYear); }
         }
   }
}

function checkForm(evt)
{
      if (validateMonth(oMonth))
      {
         if (validateDay(oDay))
         {
            if (validateYear(oYear))
            {
               // do nothing
            }
         }
      }
   return false;
}

// validate input month
function validateMonth(oField, bBypassUpdate)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   var sInput = oField.value;

      if (isEmpty(sInput))
      {
         alert("Be sure to enter a month value.");
         selectField(oField);
         return false;
      }
      else
      {
         sInput = parseInt(oField.value, 10);
            if (isNaN(sInput))
            {
               alert("Entries must be numbers only.");
               selectField(oField);
               return false;
            }
            else
            {
               if (!inRange(sInput,1,12))
               {
                  alert("Enter a number between 1 (January) and 12 (December).");
                  selectField(oField);
                  return false;
               }
            }
      }

      if (!bBypassUpdate)
      {
         calcDate();
      }
   return true;
}

// validate input day
function validateDay(oField)
{
   var sInput = oField.value;

   if (isEmpty(sInput))
   {
      alert("Be sure to enter a day value.");
      selectField(oField);
      return false;
   }
   else
   {
      sInput = parseInt(oField.value, 10);
      if (isNaN(sInput))
      {
```


更多资源请访问稀酷客(www.ckook.com)

```
                alert("Entries must be numbers only.");
                selectField(oField);
                return false;
            }
            else
            {
                var monthField = oMonth;

                    if (!validateMonth(monthField, true))
                        return false;

                var iMonthVal = parseInt(monthField.value, 10);
                var iMonthMax = new Array(31,31,29,31,30,31,30,31,31,30,31,30,31);
                var iTop = iMonthMax[iMonthVal];

                    if (!inRange(sInput,1,iTop))
                    {
                        alert("Enter a number between 1 and " + iTop + ".");
                        selectField(oField);
                        return false;
                    }
            }
        }
        calcDate();
        return true;
    }

    // validate input year
    function validateYear(oField)
    {
        var iCurrentYear = (new Date).getFullYear();

        var sInput = oField.value;
            if (isEmpty(sInput))
            {
                alert("Be sure to enter a year value.");
                selectField(oField);
                return false;
            }
            else
            {
                sInput = parseInt(oField.value, 10);
                    if (isNaN(sInput))
                    {
                        alert("Entries must be numbers only.");
                        selectField(oField);
                        return false;
                    }
                    else
                    {
                        if (!inRange(sInput, 1900, iCurrentYear))
                        {
```


更多资源请访问稀酷客(www.ckook.com)

```
                        alert("Enter a number between 1900 and " + iCurrentYear + ".");
                        selectField(oField);
                        return false;
                    }
                }
            }
    calcDate();
    return true;
}

// place the edit cursor in the requested field
function selectField(oField)
{
    oField.focus();
    oField.select();
}

// format a complete date
function calcDate()
{
    var mm = parseInt(oMonth.value, 10);
    var dd = parseInt(oDay.value, 10);
    var yy = parseInt(oYear.value, 10);
    sDate = mm + "/" + dd + "/" + yy;
    replaceTextContent(oFullDate, sDate);
}

// **BEGIN GENERIC VALIDATION FUNCTIONS**
// general purpose function to see if an input value has been entered at all
function isEmpty(sInput)
{
        if (sInput == "" || sInput == null)
        {
            return true;
        }
    return false;
}

// determine if value is in acceptable range
function inRange(sInput, iLow, iHigh)
{
    var num = parseInt(sInput, 10);

        if (num < iLow || num > iHigh)
        {
            return false;
        }
    return true;
}
// **END GENERIC VALIDATION FUNCTIONS**
```

根据一般规则，JavaScript 验证仅是第一道防线，且必须得到使用 PHP 或另一种服务器脚

更多资源请访问稀酷客(www.ckook.com)

本语言进行服务器端验证的支持。因为一些用户和用户代理要在没有 JavaScript 的情况下运行此程序，所以服务器端程序需要再次执行所有验证，但对于使用 JavaScript 的用户和用户代理而言，验证的反馈是即时的。

并不是每个日期项都必须这样分开验证。例如，Intranet 应用程序可能对用户输入数据的方式有更多的要求，因此可以用一个域输入日期，但其验证需要的解析过程与程序清单 17-5 完全不同。单域日期的验证例程请参见配书光盘上的第 46 章。

数据输入验证也是一个从异步 JavaScript(也称为 Ajax)中获益的脚本编程领域，在服务器上查找值来验证输入时，Ajax 是必需的。但其优势应仔细考虑，因为每个 Ajax 操作都需要执行服务器的一次往返，可能比提交整个页面快不了多少。

交叉引用：

要了解 Ajax 如何用于动态数据项的验证，可查阅第 39 章。

更多资源请访问稀酷客(www.ckook.com)

第18章

Array 对象

<table><tr><td>本章包含哪些内容？
使用数据的有序集合
模拟多维数组
操作数组中的信息
新增的 JavaScript 数组处理功能</td></tr></table>

数组是 JavaScript 中的一个主要数据结构，用来存储和操作有序的数据集。但与其他编程语言不同，JavaScript 的数组对存储在数组中每个单元或者数组项中的数据类型要求不是很严格。例如，它允许在数组中定义数组，这相当于为应用程序所需的数据类型定制多维数组。

如果以前没有编过什么程序，数组就可能是一个高级主题。但是，如果忽视它们的能力，就很难执行很多任务。编写脚本时，首先应考虑由应用程序控制的数据，以及将其处理为数组能否为创建文档和处理用户交互提供一些捷径。

希望通过学习本章，读者不但可以熟悉 JavaScript 数组的属性和方法，而且可以开始寻找处理数组的方式。

18.1 结构化的数据

在编程时，数组定义为有序的数据集。最好将数组想象成一个表，与电子数据表类似。在 JavaScript 中，数组是一个只有一列数据的表，但有足够的行，来容纳所有数据。如第Ⅳ部分的许多章节所述，支持 JavaScript 的浏览器为 HTML 文档中的对象和浏览器属性创建了许多内部数组。例如，如果文档含有 5 个链接，浏览器就保留一张链接的表。使用数组语法：数组名后跟方括号中的下标，就可以访问它们，例如 document.links[0]代表文档中的第一个链接(0 是第一个链接)。

许多 JavaScript 应用程序根据用户与表单元素的交互方式，使用数组作为存储用户访问页面数据的“仓库”。配书光盘上第 53 章中的应用程序演示了在页面中使用这种方式的扩展版本，用户可以搜索小型数据表，在美国社会保险号的前三位数字与机构注册的州之间查找匹配。在 JavaScript 增强的页面中，可以使用数组来重建更复杂的服务器端应用程序，例如 CGI 脚本和 Java servlet。嵌入脚本的数据集不大于普通的.gif 图像文件时，用户就感觉不到页面加载的明显

更多资源请访问稀酷客(www.ckook.com)

延迟，但仍可以使用小数据集的功能进行即时搜索，而不必返回服务器。这种面向数据库的数组是 JavaScript 用于无服务器的 CGI 的重要应用。

在设计应用程序时，可以查找一些使用数组的潜在线索。如果有大量的对象或数据点采用相同的方式与脚本交互，就可以考虑使用数组结构。例如，可在一列订单表中，为每个文本域指定类似的名称。在这个序列中，具有类似名称的对象都处理为数组的元素。为了对订单表执行重复的行计算，脚本可以使用数组语法，通过少量的 JavaScript 语句执行所有的扩展，而不是使用大量语句硬编码每个域名。配书光盘中的第 54 章列举了这种应用的例子。

还可以创建类似 Java 哈希表的数组。哈希表是一个查找表，如果知道表项的名称，立刻就能找到想要的数据。如果数据可以使用表格形式，就可以使用数组。

18.2　创建空数组

在 JavaScript 中，成熟的数组对象可以追溯到 NN3 和 IE4。在更早的浏览器中，可以模拟一些数组特性。但是，因为大多数用户已不再使用第一代浏览器，所以本章将重点介绍现代数组及其强大的功能。

要创建新的数组对象，可使用静态 Array 对象的构造函数方法。例如：

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

数组对象自动拥有 length 属性(0 表示空数组)。

如果需要先设置数组的大小，例如，使用 null 值预先加载数组项，可以指定数组的初始大小作为构造函数的参数。例如，下面创建一个新数组，它可容纳 500 条唱片信息。

```
var myCDCollection = new Array(500);
```

与其他许多编程语言不同，预先设置 JavaScript 数组的大小没有什么特别的好处，因为可以随时对任何数组项指定值，length 属性会随之调整。例如，如果为 myCDCollection[700]指定一个值，则数组对象会增加其长度，以包含这一项(起始计数为 0):

```
myCDCollection [700] = "The Smiths/Louder Than Bombs";
collectionSize = myCDCollection.length;     // result = 701
```

因为数组元素从 0 开始计数，所以，对位置 700 指定一个值，会使数组包含 701 项。

实际数组对象具有大量方法和功能，可以添加 prototype 属性，如本章后面所述。

18.3　填充数组

在数组中输入数据就像创建一系列赋值语句一样简单，每个数组元素使用一个赋值语句。程序清单 18-1 生成了一个数组，它包含太阳系的八大行星。

程序清单 18-1　生成新数组并添加数据

```
solarSys = new Array();
```


更多资源请访问稀酷客(www.ckook.com)

```
solarSys[0] = "Mercury";
solarSys[1] = "Venus";
solarSys[2] = "Earth";
solarSys[3] = "Mars";
solarSys[4] = "Jupiter";
solarSys[5] = "Saturn";
solarSys[6] = "Uranus";
solarSys[7] = "Neptune";
```

编写代码时，这种填充单个数组的方法有点麻烦，但设置完数组后，只要使用数组引用就可以访问其中的信息：

```
onePlanet = solarSys[4];      // result = "Jupiter"
```

这个方法的一个变体利用了一个事实：因为 JavaScript 数组是基于 0 的，数组的 length 属性总是把数组的尾部指针指向要填充的下一项。下面的代码生成的数组与上述程序清单相同：

```
solarSys = new Array();
solarSys[solarSys.length] = "Mercury";
solarSys[solarSys.length] = "Venus";
solarSys[solarSys.length] = "Earth";
solarSys[solarSys.length] = "Mars";
solarSys[solarSys.length] = "Jupiter";
solarSys[solarSys.length] = "Saturn";
solarSys[solarSys.length] = "Uranus";
solarSys[solarSys.length] = "Neptune";
```

如果新建的空数组的长度为 0，且添加到其中的第一项是 array[0]，这种做法就是合理的。于是数组的长度是 1，下一个要添加的项是 array[1]，依此类推。

像这样给数组元素赋值将便于以后编辑脚本，以添加或删除元素，而不必修改后续的所有下标：

```
solarSys = new Array();
solarSys[solarSys.length] = "Mercury";
solarSys[solarSys.length] = "Venus";
solarSys[solarSys.length] = "Earth";
solarSys[solarSys.length] = "Mars";
solarSys[solarSys.length] = "asteroid belt";
solarSys[solarSys.length] = "Jupiter";
solarSys[solarSys.length] = "Saturn";
solarSys[solarSys.length] = "Uranus";
solarSys[solarSys.length] = "Neptune";
```

如果知道数据的顺序(如前面的 solarSys 数组)，就可以使用一种更简洁的方式来创建数组。这种方式不编写一系列赋值语句(如程序清单 18-1 所示)，但可以把数据作为用逗号隔开的参数传给 Array()构造函数，来创建所谓的紧凑数组：

```
solarSys = new Array("Mercury","Venus","Earth","Mars","Jupiter","Saturn",
   "Uranus","Neptune");
```


更多资源请访问稀酷客(www.ckook.com)

“紧凑数组”意味着数据被压入数组，中间没有间隙，下标从 0 开始。

程序清单 18-1 的示例显示了所谓的纵向数据集，每个数据点都包含和其他数据点相同类型的数据(行星的名字)，数据点的显示顺序是与太阳距离近的行星排在前面，距离远的行星排在后面。

18.4　JavaScript 数组创建功能的增强

JavaScript 提供了另一种创建紧凑数组的方式，而且消除了以前浏览器数组中的错误。这种改进方式不需要 Array 对象构造函数。JavaScript(1.2 版本)接受所谓的字面量符号，以生成数组。为了说明其区别，请看下面的语句，这是常规的紧凑数组构造函数，它可以追溯到 NN3 版本：

```
solarSys = new Array("Mercury","Venus","Earth","Mars","Jupiter","Saturn",
   "Uranus","Neptune");
```

JavaScript 1.2+完全接受上述语法，也接受新的字面量符号：

```
solarSys = ["Mercury","Venus","Earth","Mars","Jupiter","Saturn",
   "Uranus","Neptune"];
```

方括号表示对 Array 构造函数的调用。如果使用这个精简方式创建数组，则应注意，它可能使老浏览器停止运行 JavaScript。

要修复前面提及的错误，就必须考虑到，如果脚本编写者只输入 1 作为参数——new Array(l)，该如何处理早期的紧凑数组构造函数。在 NN3 和 IE4 中，JavaScript 会错误地创建一个长度为 1 的数组，但没有定义那个元素。在 NN4 和所有更新的浏览器中，该语句会创建单元素数组，且给该元素赋值 1。

18.5　删除数组项

把数组项的值设为 null 或者空字符串，可以清除该数组元素中的数据。但是直到版本 4 的浏览器引入了 delete 操作符，才能完全删除元素。

删除一个数组元素，会从可访问的索引值列表中清除索引，但是数组的长度不会减少，如下所示：

```
myArray.length          // result: 5
delete myArray[2]
myArray.length          // result: 5
myArray[2]              // result: undefined
```

删除一个数组项不见得会释放数据占用的内存。JavaScript 解释器的内部垃圾回收机制(超出了本书的讨论范围)会自动释放内存。delete 操作符详见第 22 章。

如果要更加严格地控制数组元素的删除，可以考虑使用 splice()方法，现代浏览器支持这个方法。splice()方法可用于删除任何数组中的一个或多个数组项，数组长度也相应地调整为新数组项的数目。本章后面将讨论 splice()方法。

更多资源请访问稀酷客(www.ckook.com)

18.6 并行数组

使用数组存储数据，常常允许用一个脚本查询数组中是否有某一个值(可能验证用户输入到文本框中的数据是否是可接受的)。另外，在查找匹配的项时，脚本可以在另一个数组中查找一些相关的信息。完成这个任务的一种方式是使用两个或者多个并行数组：每个数组的相同索引项包含彼此相关的信息。

考虑下面的三个数组：

```
var regionalOffices = ["New York", "Chicago", "Houston", "Portland"];
var regionalManagers = ["Shirley Smith", "Todd Gaston",
   "Leslie Jones", "Harold Zoot"];
var regOfficeQuotas = [300000, 250000, 350000, 225000];
```

这些语句假定，Shirley Smith 是纽约办公室的区域经理，她的办公经费是 300 000。这些数据放在一个文档中，可以通过一个服务器端程序得到，此程序从 SQL 数据库中获得最新的数据，并通过数组构造函数嵌入这些数据。程序清单 18-2 把这些数据显示在一个简单的页面上，并通过 select 元素查找经理姓名和办公经费值。select 元素列表中数据项的顺序不是随机的：为了便于查找脚本，该顺序与数组的顺序相同。

程序清单 18-2 通过 getData()函数进行查找。select 元素中选项的索引值等于并行数组的索引值，所以通过 select 元素的 selectedIndex 属性可以方便地获得其他数组中相应的数据。

交叉引用：

本章和本书许多章节的代码中指定事件处理程序的函数是 addEvent()，这个跨浏览器的事件处理程序详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，可在配书光盘中找到该文件，所有章节的脚本都可以访问它。

程序清单 18-2 一个简单的并行数组查找

HTML: jsb-18-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Parallel Array Lookup</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-18-02.js"></script>
   </head>
   <body>
      <h1>Parallel Array Lookup</h1>
      <form id="officeData" action="" method="post">
         <p>
            <label for="offices">Select a regional office:</label>
            <select id="offices" name="offices">
```


更多资源请访问稀酷客(www.ckook.com)

```
            </select>
         </p>
         <p>
            <label for="manager">The manager is:</label>
            <input type="text" id="manager" name="manager" size="35" />
         </p>
         <p>
            <label for="quota">The office quota is:</label>
            <input type="text" id="quota" name="quota" size="8" />
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-18-02.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);
function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // set up the data in global variables (by omitting 'var')
      aRegionalOffices = ["New York", "Chicago", "Houston", "Portland"];
      aRegionalManagers = ["Shirley Smith", "Todd Gaston", "Leslie Jones",
                           "Harold Zoot"];
      aRegOfficeQuotas = [300000, 250000, 350000, 225000];

      // point to the critical input fields & save in global variables
      oSelect = document.getElementById('offices');
      oManager = document.getElementById('manager');
      oQuota = document.getElementById('quota');

         // if they all exist...
         if (oSelect && oManager && oQuota)
         {
            // build the drop-down list of regional offices
            for (var i = 0; i < aRegionalOffices.length; i++)
            {
               oSelect.options[i] = new Option(aRegionalOffices[i]);
            }

            // set the onchange behavior
            addEvent(oSelect, 'change', getData);
         }
      // plug in data for the default select option
      getData();
   }
}

// when a new option is selected, do the lookup into parallel arrays
```


更多资源请访问稀酷客(www.ckook.com)

```
function getData(evt)
{
   // get the offset of the selected option
   var index = oSelect.selectedIndex;

   // get data from the same offset in the parallel arrays
   oManager.value = aRegionalManagers[index];
   oQuota.value = aRegOfficeQuotas[index];
}
```

另一方面，如果要查找的内容由用户输入到文本框中，就必须遍历一个数组，来得到匹配的索引。程序清单 18-3 是程序清单 18-2 的一个变体，但它不用 select 元素，而是通过一个文本域请求用户输入地区的名称。假如用户的输入总是正确的，程序清单 18-3 中的 getData()更类似于一个查找函数：在数组中查找一个匹配，并显示从并列数组中获得的相应结果。for 循环会遍历 aRegionalOffices 数组中的数据项。if 条件先比较输入和数组中每一项的所有大写版本。如果匹配，就中断 for 循环，此时 i 的值会指向匹配的索引值。在 for 循环的外面，另一个 if 条件确保索引值没有超过数组的长度，否则就表示没有匹配的数组项。只有 i 值指向一个数组项，脚本才能从其他两个数组中得到相应的数据项。

程序清单 18-3　一个循环数组查询

HTML: jsb-18-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Parallel Array Lookup II</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-18-03.js"></script>
   </head>
   <body>
      <h1>Parallel Array Lookup II</h1>
      <form id="officeData" action="" method="post">
         <p>
            <label for="officeInput">Enter a regional office:</label>
            <input id="officeInput" name="officeInput" size="35">
            <input id="officeSearch" type="button" value="Search">
         </p>
         <p>
            <label for="manager">The manager is:</label>
            <input type="text" id="manager" name="manager" size="35" />
         </p>
         <p>
            <label for="quota">The office quota is:</label>
            <input type="text" id="quota" name="quota" size="8" />
         </p>
      </form>
   </body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

JavaScript: jsb-18-03.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // set up the data in global variables (by omitting 'var')
      aRegionalOffices = ["New York", "Chicago", "Houston", "Portland"];
      aRegionalManagers = ["Shirley Smith", "Todd Gaston", "Leslie Jones",
                           "Harold Zoot"];
      aRegOfficeQuotas = [300000, 250000, 350000, 225000];

      // point to the critical input fields & save in global variables
      oOffice = document.getElementById('officeInput');
      oSearch = document.getElementById('officeSearch');
      oManager = document.getElementById('manager');
      oQuota = document.getElementById('quota');

         // if they all exist...
         if (oOffice && oSearch && oManager && oQuota)
         {
            // apply behavior to the search button
            addEvent(oSearch, 'click', getData);
         }
   }
}

// when the Search button is clicked, do the lookup into parallel arrays
function getData(evt)
{
   // make a copy of the text box contents
   var sInputText = oOffice.value;

      // nothing entered?
      if (!sInputText)
      {
         alert('Please enter an office location.');
      }
      else
      {
         // loop through all entries of regionalOffices array
         for (var i = 0; i < aRegionalOffices.length; i++)
         {
            // compare entered text against each item in regionalOffices
            // (make both uppercase for easy comparison)
            if (sInputText.toUpperCase() == aRegionalOffices[i].toUpperCase())
            {
```


更多资源请访问稀酷客(www.ckook.com)

```
                // if they're the same, then break out of the for loop
                break;
            }
        }

        // make sure the i counter hasn't exceeded the max index value
        if (i < aRegionalOffices.length)
        {
            // display corresponding entries from parallel arrays
            oManager.value = aRegionalManagers[i];
            oQuota.value = aRegOfficeQuotas[i];
        }
        // otherwise the loop went all the way with no matches
        else
        {
            // empty any previous values
            oManager.value = "";
            oQuota.value = "";

            // advise user
            alert("No match found for '" + sInputText + "'.");
        }
    }
    // return the focus to the office input field
    oOffice.focus();
}
```

18.7 多维数组

并列数组的一个替代方法是模拟多维数组，尽管 JavaScript 数组是一维的，但可以创建数组的一维数组或其他对象的一维数组。一个合理的方法是创建自定义对象的数组，因为对象属性的命名很容易理解，所以多维数组中的数据引用更易于读取(自定义对象详见第 23 章和第 19 章)。

下面的语句使用与并列数组示例中相同的数据，为每个“数据记录”定义了一个对象构造函数，接着在主数组中为 4 项中的每一项指定了新的对象：

```
// custom object constructor
function officeRecord(city, manager, quota)
{
    this.city = city;
    this.manager = manager;
    this.quota = quota;
}

// create new main array
var regionalOffices = new Array();

// stuff main array entries with objects
regionalOffices[0] = new officeRecord("New York", "Shirley Smith", 300000);
regionalOffices[1] = new officeRecord("Chicago", "Todd Gaston", 250000);
```


更多资源请访问稀酷客(www.ckook.com)

```
regionalOffices[2] = new officeRecord("Houston", "Leslie Jones", 350000);
regionalOffices[3] = new officeRecord("Portland", "Harold Zoot", 225000);
```

对象构造函数 officeRecord()通过输入参数为对象的属性指定了值。因此，要访问数组中的一个数据，可使用两种数组语法访问数组中的数据项和该项的属性名：

```
var eastOfficeManager = regionalOffices[0].manager;
```

对于此类数组，还可以指定字符串索引值，如：

```
regionalOffices["east"] = new officeRecord("New York", "Shirley Smith",
   300000);
```

通过同一个索引访问数据：

```
var eastOfficeManager = regionalOffices["east"].manager;
```

如果更喜欢使用传统的多维数组(根据以往在其他编程语言中的经验)，还可以用较少的代码将上面的示例实现为一个数组的数组：

```
// create new main array
var regionalOffices = new Array();
// stuff main array entries with arrays
regionalOffices[0] = new Array("New York", "Shirley Smith", 300000);
regionalOffices[1] = new Array("Chicago", "Todd Gaston", 250000);
regionalOffices[2] = new Array("Houston", "Leslie Jones", 350000);
regionalOffices[3] = new Array("Portland", "Harold Zoot", 225000);
```

或使用非常简洁的字面量记号：

```
// create new main array
var regionalOffices = [
                        ["New York", "Shirley Smith", 300000],
                        ["Chicago", "Todd Gaston", 250000],
                        ["Houston", "Leslie Jones", 350000],
                        ["Portland", "Harold Zoot", 225000]
                      ];
```

访问数组中数组的单个值需要两个数组引用。例如，获得 Houston 办公室经理的姓名要使用以下语法：

```
var HoustonMgr = regionalOffices[2][1];
```

括号中的第一个索引是最外面的数组(regionalOffices)；第二个索引指向 regionalOffices[2] 返回的数组项。

18.8 模拟 Hash 表

至此为止，本章讨论的几乎所有数组都使用整数索引值。JavaScript 数组是一种特殊类型的对象(第 23 章介绍 Object 类型)。所以，也可以对数组的自定义属性赋值，而不会影响数组中存

更多资源请访问稀酷客(www.ckook.com)

储的数据或者数组长度。换句话说，可在数组对象中“背负”数据。使用“句点”语法(array.propertyName)，或者使用包含方括号和属性名的数组语法(属性名是放在方括号内的字符串，比如 array["propertyName"])，都可以引用这些属性值。以这种方式使用的数组也叫做联合数组(associative array)。如果使用句点语法，就不能在 propertyName 值中使用连字符、空格、句点和引号等字符，而方括号语法的唯一限制是，值包含的引号不能与把表达式括起来的引号相同。

下面的语句是无效的：

```
// Dot notation:
oArray.Chris Smith                    // space
oArray.Chris-Smith                    // hyphen
oArray.Chris.Smith                    // period
oArray.Smith,Chris                    // comma
oArray.Chris"Punky"Smith              // quotes

// Quoted bracket notation:
oArray["Chris "Punky" Smith"]         // matching quotes
oArray['Chris 'Punky' Smith']         // matching quotes
```

下面的语句是有效的：

```
// Dot notation:
oArray.ChrisSmith                    // one word
oArray.Chris_Smith                   // underscore

// Quoted bracket notation (nearly everything works):
oArray["Chris_Smith"]
oArray["Chris.Smith"]
oArray["Chris Smith"]
oArray["Smith, Chris"]
oArray["Chris 'Punky' Smith"]         // non-matching quotes
oArray['Chris "Punky" Smith']         // non-matching quotes
```

利用字符串索引查找对象属性有时非常有用。例如，前一节中介绍的多维数组由 4 个对象构成。如果页面包含的一个表单可在数组中查找与 select 列表选择的城市相匹配的数组项，则通常的数组查找操作就是遍历这个数组，比较所选的值和每个对象的 city 属性，如果有匹配，就提取其他属性。这个列表只有 4 项，所以工作量不大。但如果列表有 100 项，这个处理就相当耗时。更快捷的方法是直接跳到其 city 属性是所选值的数组项。此时可使用模拟的 hash 表(一些编程语言具有正式的 hash 表结构，专门用作查找表)。

填充数组后，遍历数组，并给数组对象的属性赋予字符串值，就创建了一个模拟的 hash 表。再使用期望用来查找的字符串值。例如，在 regionalOffices 数组指定了它的组件对象后，运行下面的例程就会生成 hash 表：

```
for (var i = 0; i < regionalOffices.length; i++)
{
   regionalOffices[regionalOffices[i].city] = regionalOffices[i];
}
```


更多资源请访问稀酷客(www.ckook.com)

可以检索休斯敦办公对象的 manager 属性，如下：

```
var HoustonMgr = regionalOffices["Houston"].manager;
```

有了数组的 hash 表组件，脚本就可以方便地查找数值(如果脚本需要遍历所有项)，还可以立即跳转到某项。

交叉引用：
有关在 JavaScript 中表示数据结构的更多方式，详见第 19 章和第 20 章。

18.9 Array 对象的属性和方法

属　性	方　法
constructor	concat()
length	every()*
prototype	filter()*
	forEach()*
	indexOf()*
	join()
	lastIndexOf()*
	map()*
	pop()
	push()
	reduce()
	reduceRight()
	reverse()
	shift()
	slice()
	some()*
	sort()
	splice()
	toLocaleString()
	toString()
	unshift()

*Array 对象属性和方法列表中标有*的项是最近添加到 JavaScript 1.6 及其以后版本中的。Mozilla 浏览器(如 Firefox 2.0+)支持它们，但 Internet Explorer 8 之前的版本不支持它们，Internet Explorer 浏览器的市场份额很大，不能等闲视之。本章提供了把这些新方法添加到 Array 对象中的代码(假定它们不存在)。

更多资源请访问稀酷客(www.ckook.com)

18.9.1 Array 对象属性

constructor

详见第 15 章的 string.constructor。

length

值：整型， 读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

数组对象的 length 属性表示数组中数据项的数目。数据项可以是任何类型的 JavaScript 值，包括 null。如果第 10 个单元中有一个数据项，其他单元都是 null，数组长度就是 10。注意，数组的索引值从 0 开始，所以最后一个单元的索引值(这里是 9)比数组的长度小 1。把这一属性作为自动计数器，可以很方便地在数组中追加一个新数据项：

```
myArray[myArray.length] = valueOfAppendedItem;
```

因此，普通函数没必要知道数组附加项的索引值。

prototype

值：变量或函数， 读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

在 JavaScript 中，数组对象有自己的方法和 length 属性，所有的数组对象都有这些项。prototype 属性允许脚本加入一些附加的属性或方法，应用于在当前载入文档中创建的所有数组。然而，任何单个对象都可以覆盖这个原型。

示例

为了演示 prototype 属性的工作方式，程序清单 18-4 创建了一个 prototype 属性，用于在静态 Array 对象中生成的所有数组对象。当脚本生成新的数组时(Array 对象的实例，正如 date 对象是 Date 对象的实例)，这个属性自动成为新数组的一部分。清单中的数组 c 覆盖了原型属性 sponsor 的值。但改变这个对象的值，并不能改变 Array 对象的原型值，因此，在其后创建的另一个数组 d 仍然具有初始的 sponsor 属性值。

程序清单 18-4　增加一个 prototype 属性

HTML：jsb-18-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Array prototypes</title>
      <script type="text/javascript" src="jsb-18-04.js"></script>
   </head>
   <body>
      <h1>Array prototypes</h1>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb-18-04.js

```
// add prototype to all Array objects
Array.prototype.sponsor = "DG";

// create new arrays
var a = new Array();
var b = new Array();
var c = new Array();

// override prototype property for one 'instance'
c.sponsor = "JS";

// this one picks up the original prototype
var d = new Array();

// display results
var sMsg = "Array a is brought to you by: " + a.sponsor + "\n";
sMsg += "Array b is brought to you by: " + b.sponsor + "\n";
sMsg += "Array c is brought to you by: " + c.sponsor + "\n";
sMsg += "Array d is brought to you by: " + d.sponsor;
alert(sMsg);
```

可为原型指定属性和函数。为了指定函数，可以按照 JavaScript 中正常定义函数的方式定义一个函数。接着通过名称把这个函数指定给 prototype 属性：

```
function newFunc(param1)
{
   // statements
}
Array.prototype.newMethod = newFunc; // omit parentheses in this reference
```

其中 newMethod 是方法名(函数名不会保留)。

可以像调用任何对象方法那样调用这个函数(它实质上变成了 Array 对象的一个新的临时方法)。因此，如果已经创建了 CDCollection 数组，并给它增加了原型方法 showCoverImage()，给数组中的第 10 项调用这个方法：

```
CDCollection.showCoverImage(9);
```

这个函数的参数使用索引值来得到一幅图像，其 URL 是赋给数组第 10 项的对象属性。

18.9.2 Array 对象的方法

将信息存储在数组中后，JavaScript 就可以用多种途径管理这些数据。这些方法都属于已创建的数组对象，且已随着时间的推移发生了变化，因此如果需要支持旧版本的浏览器(版本 4 之前)，应特别注意浏览器的兼容性。

更多资源请访问稀酷客(www.ckook.com)

array.concat(*array2*)

返回值： Array 对象

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

array.concat()方法可以把两个数组对象连接为一个新的数组对象。数组的连接操作并不改变两个原始数组的内容或行为。要连接数组，应把第一个数组对象放在方法前句点的左边，第二个数组的引用作为方法的参数。例如：

```
var array1 = new Array(1,2,3);
var array2 = new Array("a","b","c");
var array3 = array1.concat(array2);
    // result: array with values 1,2,3,"a","b","c"
```

如果数组元素是一个字符串或数值(不是 string 对象或者 number 对象)，这些值就从原始数组复制到新数组中，它们和原始数组的关联完全断开。但如果原始数组中的元素是某类对象的引用，JavaScript 就从原始数组的元素中将引用复制到新数组中。因此，如果改变了数组中的数据项，这个对象将发生改变，两个数组的数据项也会反映这个改变。

示例

程序清单 18-5 有些复杂，但它演示了数组如何与 array.concat()方法连接，源数组中的数值和对象如何根据其数据类型传播或不传播。这个页面如图 18-1 所示。

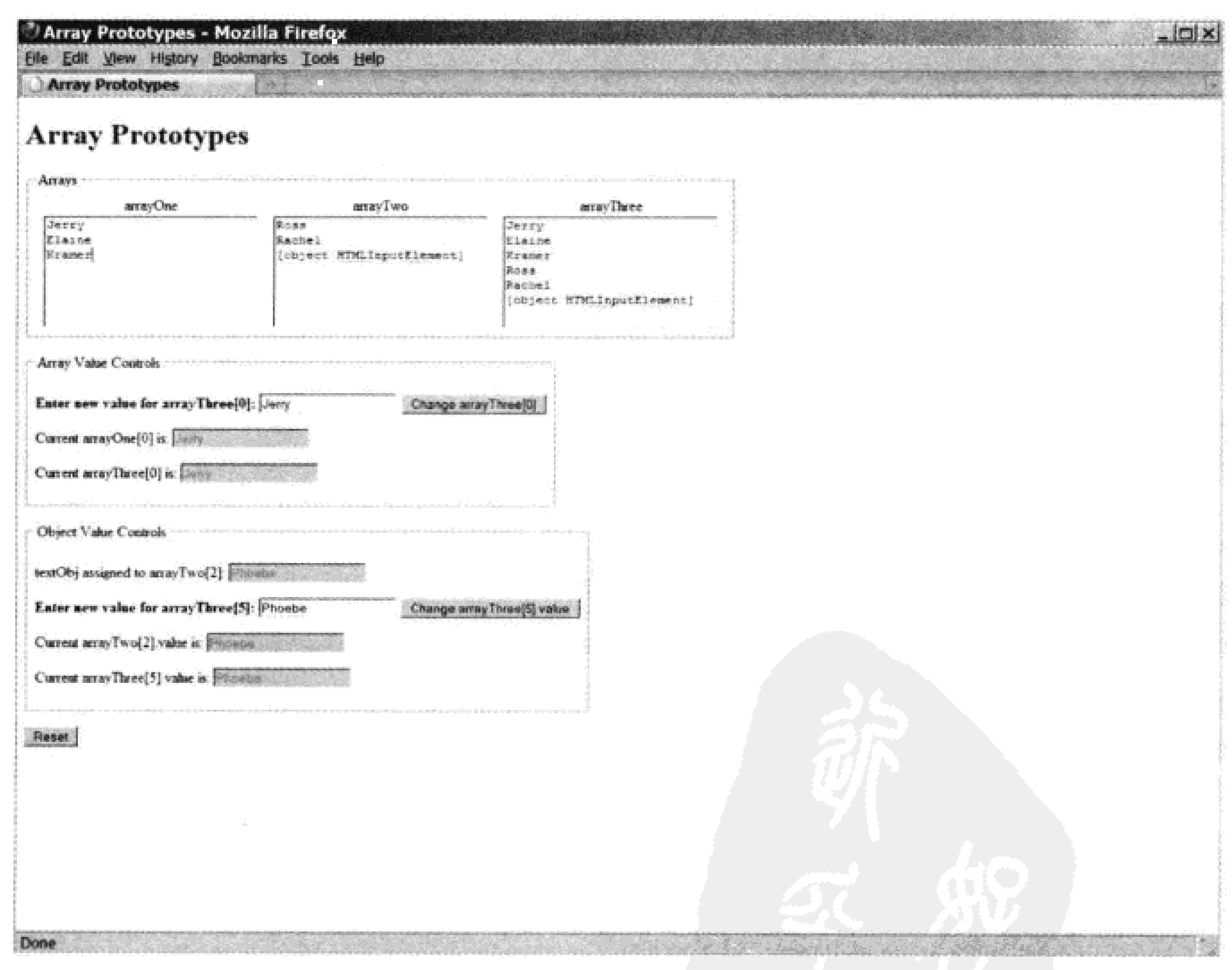


图 18-1　在连接的数组中，对象引用保持“活动”

加载页面后，会输出三个数组。第一个数组全部由字符串值构成，第二个数组有两个字符串值，还有页面上表单对象的引用(HTML 中的一个文本框 original)。在这个页面的初始化例程

PDG

更多资源请访问稀酷客(www.ckook.com)

中，不仅创建了两个源数组，它们还与 array.concat()方法连接，在第三个框中显示结果。为了分栏显示这三个数组的内容，这里使用 array.join()方法，它将数组元素连接为一个字符串，每个元素之间用一个回车来分隔，这样数据就显示为一列。

本例有两个系列的域和按钮，可在连接的数组中练习链接数值和对象引用的方式。在第一组中，如果输入一个新值，赋给 arrayThree[0]，这个新值就会替换连接数组中的字符串值。因为常规数值不会保留到原始数组的链接，所以，只有连接数组中的项才会改变。对 showArrays()的调用证明了，只有第三个数组才受到这个替换操作的影响。

更复杂的情况是这个演示中的对象关系。把第二组第一个文本框的引用赋予 arrayTwo 的第三项。连接后，这个引用就在连接数组的最后一项中。如果在 arrayThree 的最后一项中输入对象属性的新值，这个改变就会影响到原始对象——第一组中的第一个文本框。这样，初始域文本的改变，响应了 arrayThree[5]的变化。而且，因为该对象的所有引用都产生相同的结果，所以，arrayTwo[2]中的引用指向同一文本对象，产生相同的响应。所显示的数组内容不会改变，因为两个数组仍然包含同一对象的引用(“列”列表的<input>标签中显示的 value 特性引用这个标签的默认值，而不是页面最后两个域中显示的当前可通过执行算法检索到的值)。

程序清单 18-5　数组连接

HTML: jsb-18-05.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Array Prototypes</title>
      <link type="text/css" rel="stylesheet" href="jsb-18-05.css">
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-18-05.js"></script>
   </head>
   <body>
      <h1>Array Prototypes</h1>
      <form>
         <fieldset id="arrays">
         <legend>Arrays</legend>
         <p>
            <label>arrayOne</label>
            <textarea name="array1" cols="25" rows="6"></textarea>
         </p>
         <p>
            <label>arrayTwo</label>
            <textarea name="array2" cols="25" rows="6"></textarea>
         </p>
         <p>
            <label>arrayThree</label>
            <textarea name="array3" cols="25" rows="6"></textarea>
         </p>
      </fieldset>
         <fieldset>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <legend>Array Value Controls</legend>
            <p>
               <label class="input" for="source1">Enter new value for
                                               arrayThree[0]:</label>
               <input type="text" name="source1" value="Jerry">
               <input id="button1" type="button" value="Change arrayThree[0]">
            </p>
            <p>
               <label>Current arrayOne[0] is:</label>
               <input id="result1" type="text" disabled="disabled">
            </p>
            <p>
               <label>Current arrayThree[0] is:</label>
               <input id="result2" type="text" disabled="disabled">
            </p>
            </fieldset>

            <fieldset>
            <legend>Object Value Controls</legend>
            <p>
               <label>textObj assigned to arrayTwo[2]:</label>
               <input type="text" name="original" disabled="disabled">
            </p>
            <p>
               <label class="input">Enter new value for arrayThree[5]:</label>
               <input type="text" name="source2" value="Phoebe">
               <input id="button2" type="button" value="Change arrayThree[5].value">
            </p>
            <p>
               <label>Current arrayTwo[2].value is:</label>
               <input type="text" name="result3" disabled="disabled">
            </p>
            <p>
               <label>Current arrayThree[5].value is:</label>
               <input type="text" name="result4" disabled="disabled">
            </p>
            </fieldset>
         <p class="reset">
            <input id="buttonReset" type="button" value="Reset">
         </p>
         </form>
      </body>
</html>
```

JavaScript: jsb-18-05.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

// global variables
var arrayOne, arrayTwo, arrayThree, textObj;
```


更多资源请访问稀酷客(www.ckook.com)

```
function initialize()
{
  // do this only if the browser can handle DOM methods
  if (document.getElementById)
  {
     form = document.forms[0];

     var oButton = document.getElementById('button1');
        if (oButton) oButton.onclick = update1;

     oButton = document.getElementById('button2');
        if (oButton) oButton.onclick = update2;

     oButton = document.getElementById('buttonReset');
        if (oButton) oButton.onclick = function() { location.reload(); };

     // populate arrays
        textObj = form.original;
        arrayOne = new Array("Jerry", "Elaine", "Kramer");
        arrayTwo = new Array("Ross", "Rachel", textObj);
        arrayThree = arrayOne.concat(arrayTwo);

        // perform initial update
        update1();
        update2();
        showArrays();
     }
}

// display current values of all three arrays
function showArrays()
{
    form.array1.value = arrayOne.join("\n");
    form.array2.value = arrayTwo.join("\n");
    form.array3.value = arrayThree.join("\n");
}

// change the value of first item in Array Three
function update1(evt)
{
    arrayThree[0] = form.source1.value;
    form.result1.value = arrayOne[0];
    form.result2.value = arrayThree[0];
    showArrays();
}

// change value of object property pointed to in Array Three
function update2(evt)
{
    arrayThree[5].value = form.source2.value;
    form.result3.value = arrayTwo[2].value;
    form.result4.value = arrayThree[5].value;
    showArrays();
```


更多资源请访问稀酷客(www.ckook.com)

```
}
```

Stylesheet: jsb-18-05.css

```
fieldset
{
   clear: left;
   float: left;
   margin-bottom: 1em;
   padding: .5em;
}
fieldset#arrays p
{
   float: left;
   margin: 0 .5em;
}
fieldset#arrays label
{
   display: block;
   text-align: center;
}
label.input
{
   font-weight: bold;
}
p.reset
{
   clear: left;
}
```

相关主题：array.join()方法。

array.every(*callback[*，*thisObject]*)，*array*.some(*callback[*，*thisObject]*)

返回值：Boolean

兼容性：FF2+，Safari3+，Chrome1+，Opera 9.5+，WinIE−，MacIE−

every()和 some()允许用回调函数测试数组的元素(回调函数是作为一个参数传送给另一个函数的函数)。这两个方法之间的区别在于，只有每个数组元素的测试结果都是 true，every()才返回 true，而只要一个数组元素的测试结果都是 true，some()就返回 true。原数组不会改变，除非回调函数明确修改了它。

回调函数可包含任何需要的逻辑，但这里需要返回一个布尔值(true 或 false)。它带 3 个参数：要检查的数组元素值，它在数组中的索引，以及要遍历的数组对象引用。every()使用回调函数依次测试数组的每个成员，直到返回 false 值或者到达数组的尾部。some()使用回调函数测试数组的每个成员，直到返回 true 值或者到达数组的尾部。注意如果数组为空，every()就返回 true，因为它没有遇到测试结果为 false 的值。

下面的简单示例检查数组包含的数字是否均为正数：

```
function isPositive(iValue, iIndex, aArray)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   return (iValue > 0);
}

var a = new Array(5, 4, 3, 2, 1);

var bResult = a.every(isPositive);
// returns true because there are no non-positive elements

bResult = a.some(isPositive);
// returns true because there is at least one positive element

a = [5, 4, 3, 2, 1, 0];

bResult = a.every(isPositive);
// returns false because zero is non-positive
bResult = a.some(isPositive);
// returns true because there is at least one positive element

a = new Array();

bResult = a.every(isPositive);
// returns true since there are no non-positive elements in an empty array!

bResult = a.some(isPositive);
// returns false because there are no positive elements in an empty array
```

下面的例子利用回调函数的其他参数，通过其他数组元素计算当前数组元素。在 Fibonacci 序列中，每个数字都是前两个数字之和。当然，该序列中的前两个数字是这个算法的例外情况，它们分别是 0 和 1，但其他数字都是可以计算出来的：

```
function isFibonacci(iValue, iIndex, aArray)
{
   switch (iIndex)
   {
      case 0:
         return (iValue == 0);
      break;

      case 1:
         return (iValue == 1);
      break;

      default:
         return (iValue == aArray[iIndex-2] + aArray[iIndex-1]);
   }
}

var a = new Array(0, 1, 1, 2, 3, 5, 8, 13, 21);
var bResult = a.every(isFibonacci);
// returns true

a = [0, 1, 1, 2, 3, 5, 8, 13, 22];
var bResult = a.every(isFibonacci);
```


更多资源请访问稀酷客(www.ckook.com)

```
// returns false because 22 isn't 8 + 13
```

也可以通过 for()循环实现这个功能。使用 every()的优点在于，可以把程序逻辑封装在各个函数中，以便在脚本的不同地方重复使用。

对于不支持 JavaScript 1.6 中 every()方法的浏览器，可使用 Mozilla.org 中的代码给数组对象提供这个功能：

```
if (!Array.prototype.every)
{
   Array.prototype.every = function(fun /*, thisp*/)
   {
      var len = this.length >>> 0;
      if (typeof fun != "function")
      {
         throw new TypeError();
      }
      var thisp = arguments[1];
      for (var i = 0; i < len; i++)
      {
         if (i in this &&
               !fun.call(thisp, this[i], i, this))
            return false;
      }

      return true;
   };
}
```

相关主题： array.filter()，array.forEach()，array.map()方法。

array.filter(*callback[，thisObject]*)

返回值： 数组

兼容性： FF2+，Safari3+，Chrome1+，Opera 9.5+，WinIE−，MacIE−

与 every()类似，filter()方法也允许用所选回调函数测试数组的每个元素，并返回一个新数组，其中包含测试结果为 true 的所有元素。原数组保持不变，除非回调函数明确修改了它。

回调函数检查单个数组元素，并返回布尔值 true 或 false。它带有 3 个参数：要检查的数组元素值、它在数组中的索引和要遍历的数组对象的引用。每个检测为 true 的元素都添加到结果数组中。

下面的例子使用前面的回调函数检查正数：

```
function isPositive(iValue, iIndex, aArray)
{
   return (iValue > 0);
}

var a = new Array(1, -2, -3, 4, 5, -6);
var aResult = a.filter(isPositive);
// result: [1, 4, 5]
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.6 中 filter()方法的浏览器实现了该功能：

更多资源请访问稀酷客(www.ckook.com)

```
if (!Array.prototype.filter)
{
   Array.prototype.filter = function(fun /*, thisp*/)
   {
      var len = this.length >>> 0;
      if (typeof fun != "function")
         throw new TypeError();

      var res = new Array();
      var thisp = arguments[1];
      for (var i = 0; i < len; i++)
      {
         if (i in this)
         {
            var val = this[i]; // in case fun mutates this
            if (fun.call(thisp, val, i, this))
               res.push(val);
         }
      }

      return res;
   };
}
```

相关主题： array.every()，array.forEach()，array.map()，array.some()方法。

array.forEach(*callback[，thisObject]*)

返回值： 数组

兼容性： FF2+，Safari3+，Chrome1+，Opera 9.5+，WinIE−，MacIE−

forEach()在数组的每个元素上执行一个函数。它没有返回值，原数组不会改变，除非回调函数明确修改了它。

回调函数带 3 个参数：要检查的数组元素值、它在数组中的索引和要遍历的数组对象引用。

下面的例子建立了一个信息，以显示数组的内容。注意在回调函数 buildMsg()中，关键字 this 表示传送给 forEach()的定制对象 oMsg：

```
function sendMessages(sValue, iIndex, aArray)
{
   sendMsg(sValue, this.text); // where sendMsg() is defined elsewhere . . .
}

var a = new Array('chris@example.com', 'pat@example.net', 'jessie@example.org');

var oMessage = {text: "Dear friend,\nYou are cordially invited..."};
a.forEach(sendMessages, oMessage);
```

另外，上例可以使用更通用的 for()方法编写：

```
var a = new Array('apple', 'banana', 'cardamom', 'dandelion');
var oMsg = {buffer: 'This array contains: '};
for (var i = 0; i < a.length; i++)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   oMsg.buffer += '\n' + i + ':' + a[i];
}
alert(oMsg.buffer);
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.6 中 forEach()方法的浏览器实现了该功能：

```
if (!Array.prototype.forEach)
{
   Array.prototype.forEach = function(fun /*, thisp*/)
   {
      var len = this.length >>> 0;
      if (typeof fun != "function")
         throw new TypeError();

      var thisp = arguments[1];
      for (var i = 0; i < len; i++)
      {
         if (i in this)
            fun.call(thisp, this[i], i, this);
      }
   };
}
```

相关主题：array.every()，array.filter()，array.map()，array.some()方法。

array.indexOf(*searchString[，startFrom]*)，*array*.lastIndexOf(*searchString[，startFrom]*)

返回值：其值匹配 searchString 的数组元素的索引值，如果没有找到匹配，就返回-1.

兼容性：FF2+，Safari3+，Chrome1+，Opera 9.5+，WinIE−，MacIE−

这两个方法搜索精确匹配 searchString 的数组元素，并返回第一个(或最后一个)匹配元素的的索引，没有找到匹配，就返回-1。

这两个方法的主要区别在于，indexOf()从数组的开头开始，以递增顺序从左到右地搜索，而 lastIndexOf()从数组的尾部开始，以递减顺序从右到左地搜索。在这两个方法中，返回值都是所找到元素的索引——与数组开头的偏移量。因此，如果数组中只有一个匹配的元素，且没有指定 startFrom，indexOf()和 lastIndexOf()就会返回相同的值，因为它们找到的是同一个匹配元素。

可选的 startFrom 参数告诉方法，开始搜索的位置距离数组的开头或结尾有多远，如果忽略这个参数，就假定它为 0，即从第一个元素(对于 indexOf())或最后一个元素(对于 lastIndexOf())开始搜索。如果 startFrom 是整数，就用作与数组开头的偏移量，如果它是负数，就用作与数组尾部的偏移量。

例如：

```
var a = new Array('cat', 'dog', 'cat');

// indexOf searches left to right:
a.indexOf('cat');          // result = 0 (search elements 0, 1, and 2)
a.indexOf('cat', 1);       // result = 2 (search elements 1 and 2)
```


更多资源请访问稀酷客(www.ckook.com)

```
a.indexOf('cat', 2);       // result = 2 (search element 2)
a.indexOf('cat', 3);       // result = -1 (no elements to search)
a.indexOf('cat', -1);      // result = 2 (search element 2)
a.indexOf('cat', -2);      // result = 2 (search elements 1 and 2)
a.indexOf('cat', -3);      // result = 0 (search elements 0, 1, and 2)

// lastIndexOf searches right to left:
a.lastIndexOf('cat');      // result = 2 (search elements 2, 1, and 0)
a.lastIndexOf('cat', 1);  // result = 0 (search elements 1 and 0)
a.lastIndexOf('cat', 2);  // result = 2 (search elements 2, 1, and 0)
a.lastIndexOf('cat', 3);  // result = 2 (search whole array)
a.lastIndexOf('cat', -1); // result = 2 (search elements 2, 1, and 0)
a.lastIndexOf('cat', -2); // result = 0 (search elements 1 and 0)
a.lastIndexOf('cat', -3); // result = 0 (search element 0)
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.6 中 indexOf()方法的浏览器实现了该功能：

```
if (!Array.prototype.indexOf)
{
   Array.prototype.indexOf = function(elt /*, from*/)
   {
      var len = this.length >>> 0;

      var from = Number(arguments[1]) || 0;
      from = (from < 0)
             ? Math.ceil(from)
             : Math.floor(from);
      if (from < 0)
         from += len;

      for (; from < len; from++)
      {
         if (from in this &&
               this[from] === elt)
            return from;
      }
      return -1;
   };
}
```

Mozilla 还提供了如下代码，为不支持 lastIndexOf()方法的浏览器实现了该功能：

```
if (!Array.prototype.lastIndexOf)
{
   Array.prototype.lastIndexOf = function(elt /*, from*/)
   {
      var len = this.length;

      var from = Number(arguments[1]);
      if (isNaN(from))
      {
         from = len - 1;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
        else
        {
            from = (from < 0)
                    ? Math.ceil(from)
                    : Math.floor(from);
            if (from < 0)
                from += len;
            else if (from >= len)
                from = len - 1;
        }

        for (; from > -1; from--)
        {
            if (from in this &&
                    this[from] === elt)
                return from;
        }
        return -1;
    };
}
```

array.join(*separatorString*)

返回值：通过 separatorString 值分界的数组项的字符串

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

无法直接查看存储在数组中的数据，也无法把数组放在一个表单元素中，传递到一个需要文本字符串的服务器端 CGI 程序中。为将离散的数组元素转换成字符串，必须使用 array.join()方法来处理繁琐的字符串操作。

这个方法的唯一参数是一个或多个字符组成的字符串，用作数组元素之间的分隔符。例如，如果想用逗号分隔数组项的文本版本，语句如下：

```
var arrayText = myArray.join(",");
```

调用这个函数不会对原始数组做任何改动。因而，需要把这个方法的结果赋给另一个变量或者表单元素的 value 属性。

示例

程序清单 18-6 中的脚本将行星名称数组转换为一个文本字符串。这个页面提供了一个域，用于输入所选的分界字符串，并在文本域中显示了结果。

程序清单 18-6　使用 Array.join()方法

HTML：jsb-18-06.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>Array.join()</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb-18-06.js"></script>
    </head>
    <body>
        <h1>Array.join(): Converting arrays to strings</h1>
        <form>
            <p>This document contains an array of planets in our solar system.</p>
            <p>
                <label for="delimiter">Enter a string to act
                                        as a delimiter between entries:</label>
                <input type="text" id="delimiter" name="delim" value="," size="5">
            </p>
            <p>
                <input id="displayButton" type="button" value="Display as String">
                <input type="reset">
            </p>
            <p>
                <textarea id="output" name="output" rows="4" cols="40"
                                wrap="virtual"></textarea>
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb-18-06.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // do this only if the browser can handle DOM methods
    if (document.getElementById)
    {
        // point to critical elements
        oDelimiter = document.getElementById('delimiter');
        oOutput = document.getElementById('output');
        var oButton = document.getElementById('displayButton');

            // if they all exist...
            if (oDelimiter && oOutput && oButton)
            {
                // apply behavior to button
                oButton.onclick = convert;

                // set global array
                solarSys = new Array("Mercury", "Venus", "Earth", "Mars", "Jupiter",
                                     "Saturn", "Uranus", "Neptune");
            }
    }
}
```


更多资源请访问稀酷客(www.ckook.com)

```
// join array elements into a string
function convert(evt)
{
   var delimiter = oDelimiter.value;
   oOutput.value = decodeURIComponent(solarSys.join(delimiter));
}
```

注意，这个方法只接受字面量参数。如果要包括非字母数字的字符，比如换行符或制表符，可使用 URL 编码的字符(%0D 表示回车，%09 表示制表符)，而不是使用内联的字符串字面量。在程序清单 18-7 中，array.join()方法的结果传送给 decodeURIComponent()函数，以便在文本域中显示。

相关主题：string.split()方法。

array.map(*callback[，thisObject]*)

返回值：数组

兼容性：FF2+，Safari3+，Chrome1+，Opera 9.5+，WinIE−，MacIE−

map()在原数组的每个元素上运行用户提供的函数，以创建一个新数组。

回调函数的返回值用于建立新数组。它有 3 个参数：要检查的数组元素值、它在数组中的索引和要遍历的数组对象的引用。

如果提供了 thisObject，它就用作回调函数中的 this 值。

例如：

```
function square(iValue, iIndex, aArray)
{
   return iValue * iValue;
}

var a = new Array(0, 1, 2, 3, 4, 5, 6, 7);
var b = a.map(square);
// result: b == [0, 1, 4, 9, 16, 25, 36, 49]
```

也可以即时创建回调函数，将上述代码改为：

```
var a = new Array(0, 1, 2, 3, 4, 5, 6, 7);
var b = a.map(function(i) { return i * i; });
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.6 中 map()方法的浏览器实现了该功能：

```
if (!Array.prototype.map)
{
   Array.prototype.map = function(fun /*, thisp*/)
   {
      var len = this.length >>> 0;
      if (typeof fun != "function")
         throw new TypeError();

      var res = new Array(len);
      var thisp = arguments[1];
```


更多资源请访问稀酷客(www.ckook.com)

```
        for (var i = 0; i < len; i++)
        {
            if (i in this)
                res[i] = fun.call(thisp, this[i], i, this);
        }

        return res;
    };
}
```

相关主题： array.every()，array.filter()，array.forEach()，array.some()方法。

array.pop()，*array*.push(*valueOrObject*)，*array*.shift()，*array*.unshift(*valueOrObject*)

返回值： 一个数组项的值

兼容性： WinIE5.5+，MacIE−，NN4+，Moz+，Safari+，Opera+，Chrome+

有经验的程序员非常熟悉堆栈的概念，特别是那些了解汇编语言在 CPU 级别的内部工作原理的程序员。即使以前编程时从来没有使用过堆栈，也可能在现实生活中多次遇到这个概念。它类似于一堆装有发条的自助餐盘子。如果一次把一个盘子放入这堆盘子中，每个盘子都会放在盘子堆栈的顶部。客户进来时，就取走最上面的盘子(这个盘子是最后一个放入堆栈中的)，最后放入堆栈中的盘子是第一个被取走的盘子。

现代浏览器中的 JavaScript 能把数组转换成一个装有发条的堆栈。但它不是把盘子放在堆上，而可以把任何类型的数据放在堆栈的任何一端，这取决于使用堆栈的方法。同样，也可以从堆栈的任何一端获得数据。

堆栈最常见的术语是 push 和 pop。使用 push()将一个值放入数组中时，这个值会作为最后一个数据项追加到数组中。使用 array.pop ()方法，可从堆栈中移出并返回数组中的最后一项，并给数组的长度减 1。在下面的代码中，将数组用作堆栈，可看到其值的变化：

```
var source = new Array("Homer","Marge","Bart","Lisa","Maggie");
var stack = new Array();
    // stack = <empty>
stack.push(source[0]);
    // stack = "Homer"
stack.push(source[2]);
    // stack = "Homer","Bart"
var Simpson1 = stack.pop();
    // stack = "Homer" ; Simpson1 = "Bart"
var Simpson2 = stack.pop();
    // stack = <empty> ; Simpson2 = "Homer"
```

push()和 pop()在数组的末端工作，另一对方法在数组的前端工作，但它们的命名并不像 push()和 pop()那样明确。可以使用 array.unshift()方法在数组的前面插入一个值；使用 array.shift()方法可以获得第一个元素并把它从数组中删去。当然，不需要成对使用这些方法。如果把一系列值放到数组的末端，可以使用 shift()把它们从前面取出，这完全依赖处理数据的方式。

相关主题： array.concat()，array.slice()方法。

更多资源请访问稀酷客(www.ckook.com)

array.reduce(*callback[, initialValue]*)，*array*.reduceRight(*callback[, initialValue]*)

返回值：一个值

兼容性：FF3+，Safari4+

这两个方法把数组缩减为一个值。reduce()从第一个元素开始操作数组元素，reduceRight()而从最后一个元素开始。这两个方法的参数都是回调函数和一个可选的初始值(因为第一次迭代只有一个数组元素要处理)。回调函数有两个参数(firstValue, secondValue)，它以某种方式处理这些参数，以派生出一个返回值。

例如，以下代码将一个数值数组缩减为其总和：

```
function addEmUp(a, b)
{
   return a + b;
}

var a = new Array(1, 2, 3, 4, 5);
var b = a.reduce(addEmUp);
// result: b == 15
```

对于简单的相加，缩减操作从前往后还是从后往前并不重要，但把这个回调函数应用于文本，会得到什么结果？让我们来看一下；注意在 JavaScript 中，加号会对数值执行相加操作，而对文本执行连接操作。

```
var a = new Array('d', 'e', 's', 's', 'e', 'r', 't', 's');

var b = a.reduce(addEmUp);
// result: b == 'desserts'

var b = a.reduceRight(addEmUp);
// result: b == 'stressed'
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.8 中 reduce()方法的浏览器实现了该功能：

```
if (!Array.prototype.reduce)
{
  Array.prototype.reduce = function(fun /*, initial*/)
  {
    var len = this.length >>> 0;
    if (typeof fun != "function")
      throw new TypeError();

    // no value to return if no initial value and an empty array
    if (len == 0 && arguments.length == 1)
      throw new TypeError();

    var i = 0;
    if (arguments.length >= 2)
    {
      var rv = arguments[1];
    }
    else
```


更多资源请访问稀酷客(www.ckook.com)

```
      {
        do
        {
          if (i in this)
          {
            rv = this[i++];
            break;
          }

          // if array contains no values, no initial value to return
          if (++i >= len)
            throw new TypeError();
        }
        while (true);
      }

      for (; i < len; i++)
      {
        if (i in this)
          rv = fun.call(null, rv, this[i], i, this);
      }

      return rv;
    };
}
```

Mozilla 提供了如下代码，为不支持 JavaScript 1.8 中 reduceRight()方法的浏览器实现了该功能：

```
if (!Array.prototype.reduceRight)
{
  Array.prototype.reduceRight = function(fun /*, initial*/)
  {
    var len = this.length >>> 0;
    if (typeof fun != "function")
      throw new TypeError();

    // no value to return if no initial value, empty array
    if (len == 0 && arguments.length == 1)
      throw new TypeError();

    var i = len - 1;
    if (arguments.length >= 2)
    {
      var rv = arguments[1];
    }
    else
    {
      do
      {
        if (i in this)
        {
          rv = this[i--];
```


更多资源请访问稀酷客(www.ckook.com)

```
          break;
        }

        // if array contains no values, no initial value to return
        if (--i < 0)
          throw new TypeError();
      }
      while (true);
    }

    for (; i >= 0; i--)
    {
      if (i in this)
        rv = fun.call(null, rv, this[i], i, this);
    }

    return rv;
  };
}
```

array.reverse()

返回值：和原数组顺序相反的数组项

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

有时，以倒序的方式处理数组中的数据更加方便。虽然可通过索引值使用循环反向计数，但是服务器端的程序更喜欢以与脚本相反的方式处理数据。

使用 JavaScript 可以变换数组的内容：使数组中的最后一个元素变成这个数组的第一项。记住，虽然这个方法返回数组的一个反向副本，但如果这样做，要重建原来的数组，而不是复制它。HTML 文档的重载会恢复它在文档中编写的顺序。

示例

程序清单 18-7 是程序清单 18-6 的增强版本，它包括另一个按钮和函数，用于将数组反转，在文本区域显示为字符串。

程序清单 18-7　Array.reverse()方法

HTML：jsb-18-07.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Array.reverse()</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-18-07.js"></script>
   </head>
   <body>
      <h1>Array.reverse(): Reversing array element order</h1>
      <form>
         <p>This document contains an array of planets in our solar system.</p>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <p>
               <label for="delimiter">Enter a string to act as
                                       a delimiter between entries:</label>
               <input type="text" id="delimiter" name="delim" value="," size="5">
            </p>
            <p>
               <input type="button" id="showAsIs" value="Array as-is">
               <input type="button" id="reverseIt" value="Reverse the array">
               <input type="reset">
               <input type="button" id="reload" value="Reload">
            </p>
            <p>
               <textarea id="output" name="output" rows="4" cols="40"
                              wrap="virtual"></textarea>
            </p>
         </form>
      </body>
   </html>
```

JavaScript: jsb-18-07.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to critical elements
      oDelimiter = document.getElementById('delimiter');
      oOutput = document.getElementById('output');
      var oButtonShowAsIs = document.getElementById('showAsIs');
      var oButtonReverseIt = document.getElementById('reverseIt');
      var oReload = document.getElementById('reload');

         // if they all exist...
         if (oDelimiter && oOutput && oButtonShowAsIs && oButtonReverseIt)
         {
            // apply behaviors to buttons
            oButtonShowAsIs.onclick = showAsIs;
            oButtonReverseIt.onclick = reverseIt;
            oReload.onclick = function() { location.reload(); };

            // set global array
            solarSys = new Array("Mercury", "Venus", "Earth", "Mars", "Jupiter",
                                 "Saturn", "Uranus", "Neptune");
         }
   }
}

// show array as currently in memory
function showAsIs(evt)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
   var delimiter = oDelimiter.value;
   oOutput.value = decodeURIComponent(solarSys.join(delimiter));
}

// reverse array order, then display as string
function reverseIt(evt)
{
   var delimiter = oDelimiter.value;
   solarSys.reverse();   // reverses original array
   oOutput.value = decodeURIComponent(solarSys.join(delimiter));
}
```

注意 solarSys.reverse()方法是独立的，也就是说，不捕获返回值，因为这个方法修改了 solarSys 数组。然后，可通过 array.join()方法反转 solarSys 数组，以便显示文本。

相关主题：array.sort()方法。

array.slice(*startIndex [，endIndex]*)

返回值：数组

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

其行为类似于名称相似的字符串方法，array.slice()可以从数组中提取连续的数据项，并把提取的数据项变成一个全新的数组对象。从原来数组中提取的值和对象与使用 array.concat()方法创建的数组的行为相同。

这个方法需要一个参数：开始提取的索引位置。如果不指定第二个参数，就一直提取到数组的结尾，否则，只提取到第二个参数指定的索引值的位置，但不包括那个位置的值。例如，下面的代码从行星名称数组中获得地球的相邻行星。

```
var solarSys = new Array("Mercury","Venus","Earth","Mars",
   "Jupiter","Saturn","Uranus","Neptune","Pluto");
var nearby = solarSys.slice(1,4);
    // result: new array of "Venus", "Earth", "Mars"
```

相关主题：array.splice()，string.slice()方法。

array.sort(*[compareFunction]*)

返回值：以 compareFunction 算法控制顺序的数组项

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

JavaScript 的数组排序功能是非常强大的，但如果没有使用过数组排序的方法，学起来就有点困难。很明显，数组排序的目的是按照与数据项有关的条件来排列数组中的数据项。对于字符串型的数据项，这个条件可以是字母顺序或字符串长度；对于数值型数据项，这个条件可以是数字顺序。

首先看看使用 array.sort()自动(例如，不需要调用比较函数)进行排序的方法。不指定参数时，JavaScript 会获得数组内容的一个快照，并把它的各项转换成字符串，然后执行字符串排序。它以字符的 ASCII 值为排序条件，这意味着数字通过它们的字符串值，而不是数值来排序。如果数组由数值组成，上面的排序方法会把数值 201 排在 88 的前面，因为这种排序机制比较字符串的第一个字符(2 对 8)，来决定排列的顺序。对于数组中简单的字符串字母排序，普通的

更多资源请访问稀酷客(www.ckook.com)

array.sort()方法就可以完成。

幸好，可以在数组排序时利用其他信息。关键的策略是定义一个函数，它帮助 sort()方法比较数组项。这个比较函数接受数组中的两个值(array.sort()方法迅速地向比较函数传递数组中的若干值，来帮助排列所有的数据项，这一点在前台看不到)。比较函数根据其返回值，告诉 sort()方法两个数据项哪一项在前。假如函数比较两个值 a 和 b，则返回值将向 sort()方法传递有关信息，如表 18-1 所示。

表 18-1 比较函数的返回值

返回值的范围	含 义
<0	b 应当排在 a 的后面
0	a 和 b 的顺序不应当改变
>0	a 应当排在 b 的后面

考虑下面的示例：

```
myArray = new Array(12, 5, 200, 80);
function compare(a,b)
{
   return a - b;
}
myArray.sort(compare);
```

这个数组有 4 个数值。为了按数值顺序排列它们，定义一个比较函数(随意命名为 compare())，它在 sort()方法中调用。注意，不像调用别的函数，sort()方法的参数使用了这个函数的引用，且比较函数没有圆括号。

在调用 compare()函数时，JavaScript 自动快速、连续地给这个函数发送两个参数，直到数组中的每个元素都和别的元素比较完毕为止。每次调用 compare()函数，JavaScript 都将数组中的两个值赋给参数变量(a 和 b)。在前面的示例中，返回值是 a 和 b 之差。如果 a 比 b 大，就给 sort()方法返回一个正值，说明 a 在 b 后面(即，a 位置的索引值比 b 位置的索引值大)。因此，b 可能在 myArray[0]，而 a 可能在一个较高的索引位置上。另一方面，如果 a 比 b 小，返回的负值告诉 sort()方法，a 的索引值位置比 b 小。

在比较函数中，可以采用多种比较方式，只要数据与数组值相关联，就能进行比较。例如，如上所示，可以不比较数值的大小，而进行字符串的比较。下面的函数比较字符串数组中的每一项的最后一个字符，按字母顺序进行排序：

```
function compare(a,b)
{
  // last character of array strings
  var aComp = a.charAt(a.length - 1);
  var bComp = b.charAt(b.length - 1);
  if (aComp < bComp)
     return -1;
  if (aComp > bComp)
     return 1;
  return 0;
}
```


更多资源请访问稀酷客(www.ckook.com)

首先，这个函数从传递给它的两个值中获得每个值的最后一个字符。接着，因为字符串不能像数字一样相加减，所以比较两个字符的 ASCII 值，给 sort()方法返回相应的值，让 sort()方法知道怎样处理这两个值。

数组的数据项是对象时，甚至可以通过这些对象的属性来排序。排序函数的 a 和 b 参数是两个数组项的引用，所以可以通过扩展，引用那些对象的属性。例如，如果数组所含对象的属性定义了雇员的信息，其中一个属性是雇员的年龄，定义为字符串类型，则下面的比较函数比较对象的 age 属性，就可以对这个数组排序：

```
function compare(a,b) {
   return parseInt(a.age) - parseInt(b.age);
}
```

不像在其他脚本语言中的排序例程，JavaScript 数组排序不是一个稳定排序。不稳定意味着在同一个数组上重复排序，其效果并不是累积的。还有，排序改变了原始数组的排列顺序。如果不想改变原始数组，就要在排序前复制它，或重载文档，将数组恢复到原来的顺序。如果一个数组元素是 null，该方法会把这个元素放在数组的最后。

JavaScript 数组排序能力相当强大。数组排序是加载包含 IE XML 数据孤岛的页面常常要花费许多时间的一个原因，例如，要使数据的 JavaScript 副本作为对象数组(参见配书光盘中的第 60 章)。与在 XML 元素上执行自己的排序例程相比，将 XML 转换为 JavaScript 数组，会使数据排序工作更简捷。

示例

在程序清单 18-8 中，有几个字符串数组排序的例子，如图 18-2 所示。其中有 4 个按钮用于不同的排序，其中的 3 个调用了比较函数。这个程序清单通过行星名称的最后一个字符以及行星名称的长度，按字母顺序对行星数组排序(从前向后，从后向前)。每个比较函数都演示了排序期间比较数据的不同方式。

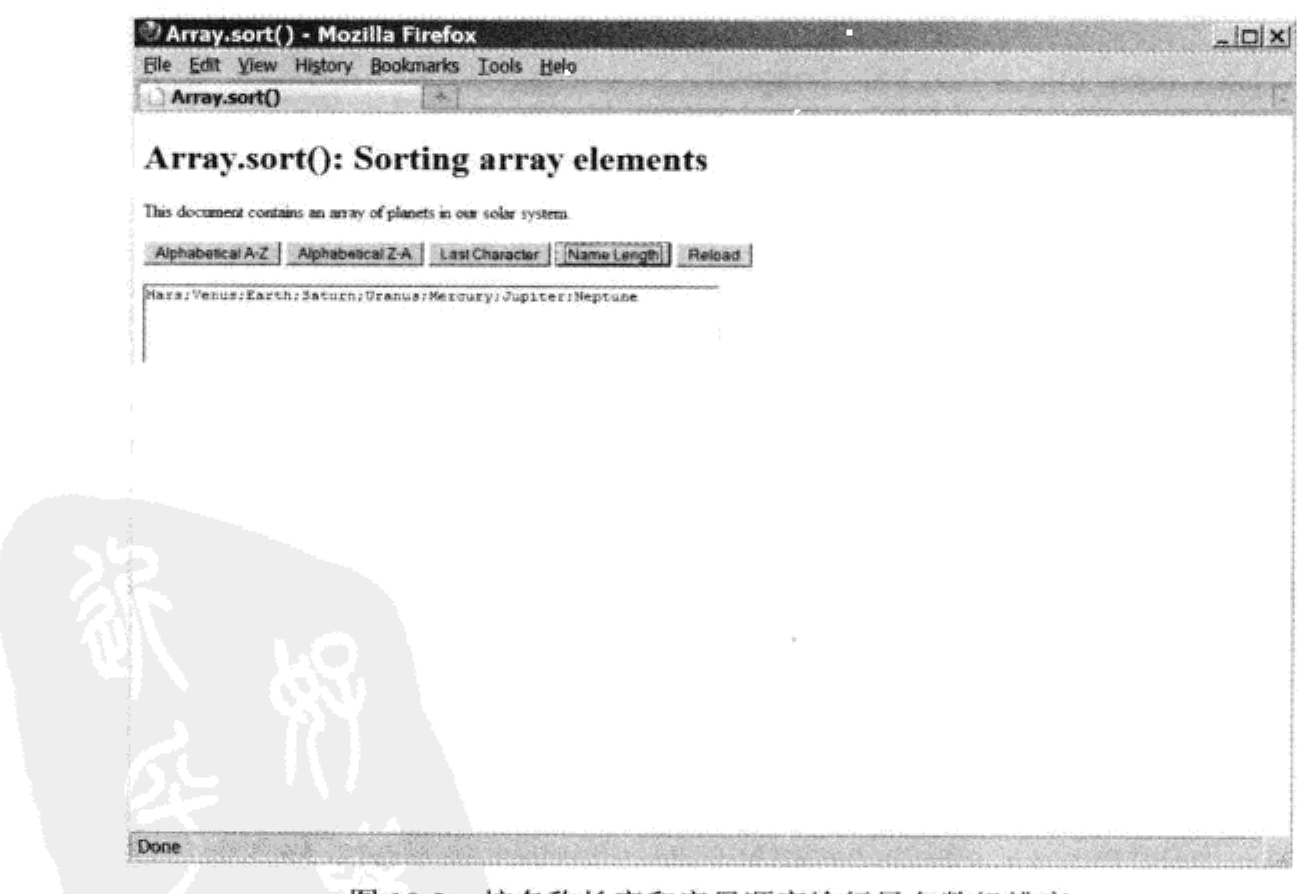


图 18-2　按名称长度和字母顺序给行星名数组排序

更多资源请访问稀酷客(www.ckook.com)

程序清单 18-8　Array.sort()的不同用法

HTML: jsb-18-08.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Array.sort()</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-18-08.js"></script>
   </head>
   <body>
      <h1>Array.sort(): Sorting array elements</h1>
      <form>
         <p>This document contains an array of planets in our solar system.</p>
         <p>
            <input type="button" id="sortAsc" value="Alphabetical A-Z">
            <input type="button" id="sortDesc" value="Alphabetical Z-A">
            <input type="button" id="sortLastChar" value="Last Character">
            <input type="button" id="sortLength" value="Name Length">
            <input type="button" id="reload" value="Reload">
         </p>
         <p>
            <textarea id="output" name="output" rows="3" cols="60"
                          wrap="virtual"></textarea>
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-18-08.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // do this only if the browser can handle DOM methods
   if (document.getElementById)
   {
      // point to critical elements
      oOutput = document.getElementById('output');
      var oButtonSortAsc = document.getElementById('sortAsc');
      var oButtonSortDesc = document.getElementById('sortDesc');
      var oButtonSortLastChar = document.getElementById('sortLastChar');
      var oButtonSortLength = document.getElementById('sortLength');
      var oReload = document.getElementById('reload');

         // if they all exist...
         if (oOutput && oButtonSortAsc && oButtonSortDesc && oButtonSortLastChar
                  && oButtonSortLength && oReload)
```


更多资源请访问稀酷客(www.ckook.com)

```
            {
               // apply behaviors to buttons
               oButtonSortAsc.onclick = function() { sortIt(null) };
               oButtonSortDesc.onclick = function() { sortIt(compare1) };
               oButtonSortLastChar.onclick = function() { sortIt(compare2) };
               oButtonSortLength.onclick = function() { sortIt(compare3) };
               oReload.onclick = function() { location.reload(); };

               // set global array
               solarSys = new Array("Mercury", "Venus", "Earth", "Mars", "Jupiter",
                                    "Saturn", "Uranus", "Neptune");
            }
        }
    }
    // comparison functions
    function compare1(a,b)
    {
       // reverse alphabetical order
       if (a > b)
          return -1;
       if (b > a)
          return 1;
       return 0;
    }

    function compare2(a,b)
    {
       // last character of planet names
       var aComp = a.charAt(a.length - 1);
       var bComp = b.charAt(b.length - 1);
       if (aComp < bComp)
          return -1;
       if (aComp > bComp)
          return 1;
       return 0;
    }

    function compare3(a,b)
    {
       // length of planet names
       return a.length - b.length;
    }

    // sort and display array
    function sortIt(compFunc)
    {
       if (compFunc == null)
       {
             solarSys.sort();
       }
       else
       {
```


更多资源请访问稀酷客(www.ckook.com)

```
        solarSys.sort(compFunc);
    }

    // display results in field
    var delimiter = ";";
    oOutput.value = decodeURIComponent(solarSys.join(delimiter));
}
```

相关主题：array.reverse()方法。

注意：如配书光盘中的第 45 章所述，许多正则表达式对象方法都以数组作为结果(例如，用字符串中的匹配值生成数组)。这些特殊数组有一个自定义的命名属性集，帮助脚本分析这种方法的结果。除此之外，正则表达式结果数组与其他数组一样。

array.splice(*startIndex*，*deleteCount*[，*item1*[，*item2*[，...*itemN*]]])

返回值：数组

兼容性：WinIE5.5+，MacIE−，NN4+，Moz+，Safari+，Opera+，Chrome+

如果需要从数组中删去一些项，使用 array.splice()方法能简化这一任务，否则就需要用原始数组的选项建立一个新数组。两个必选参数中的第一个是基于 0 的索引整数，指向当前数组中被删除的第一项；第二个参数也是一个整数，它指定要从数组中删除多少个连续项。删除数组项影响了数组的长度，被删除的项由 splice()方法返回为一个数组。

还可以使用 splice()方法替换数组项。从第三个参数开始是可选参数，它们提供了插入数组、替换被删除项的数据元素。加入的每个数据项可以是任意 JavaScript 数据类型，新数据项的数量不一定等于被删除项的数量。实际上，将第二个参数指定为 0，可以使用 splice()在数组的任何位置插入一个或多个数据项。

示例

使用第 4 章介绍的 The Evaluator，练习 splice()方法。首先使用一系列数字创建一个数组：

```
a = new Array(1,2,3,4,5)
```

随后，删除中间 3 项，使用一个字符串项替换它们：

```
a.splice(1, 3, "two/three/four")
```

Results 框显示方法返回的 3 项数组元素的字符串版本。要查看数组的当前内容，可在顶端文本框中输入 a。

要将初始数字返回到数组中，可使用 3 个数字项交换字符串项：

```
a.splice(1, 1, 2, 3, 4)
```

该方法返回单个字符串，现在，a 数组又有 5 项了。

相关主题：array.slice()方法。

array.toLocaleString()

返回值：字符串

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE5.5+，MacIE–，NN6+，Moz+，Safari+，Opera+，Chrome+

array.toString()

返回值： 字符串
兼容性： WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

使用 array.toLocaleString()方法与兼容性更强的旧方法 array.toString()都能以字符串形式显示数组内容。在文本框中显示数组时，浏览器会自动使用 toString()方法，此时，数组中的每一项使用逗号分开。

toLocaleString()的准确字符串转换方式取决于具体的浏览器版本。不同的浏览器在一些细节上有所不同，这并不奇怪，甚至在操作系统和浏览器的美国英语版本中，也有这种情况。例如，如果数组包含整数值，IE5.5+中的 toLocaleString()方法就返回逗号和空格分隔的数字，它们有两位小数，就像美元和美分一样。另一方面，基于 Mozilla 的浏览器只返回整数，但它们也是以逗号分隔的。

如果需要把一个数组转换成字符串，以便把数组中的数据传递到其他地点(例如，把不可见文本框中的数据提交给服务器，或把检索的字符串数据传递到另一页)，可以改用 array.join()方法。Array.join()提供了更可靠、更灵活的方法来控制数据项分隔符，无论在什么地方都能确保产生相同的结果。

相关主题： array.join()方法。

18.10　数组包含

兼容性： FF2+。

JavaScript 1.7 的一个功能是对 JavaScript 语法的改进，称为数组包含，这是在一个数组的基础上建立另一个数组的快捷方式。数组包含常可以替换 map()和 filter()方法。

下面的例子使用数组包含功能生成一个数组中所有元素的修改版本：

```
var aIntegers = new Array(0, 1, 2, 3, 4, 5, 6, 7);
var aResults = [i * i for each (i in aIntegers)];

// result: aResults == [0, 1, 4, 9, 16, 25, 36, 49]
```

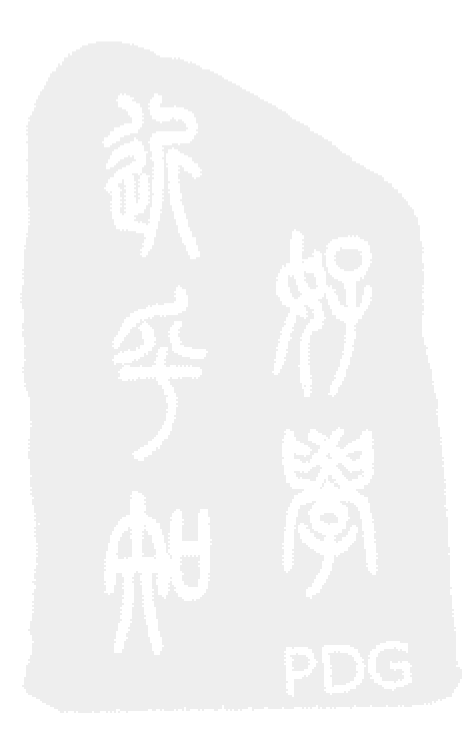
下面是对应的 map()语法：

```
function square(iValue)
{
   return iValue * iValue;
}

var aIntegers = new Array(0, 1, 2, 3, 4, 5, 6, 7);
var aResults = aIntegers.map(square);
```

数组包含还可以用于从数组中选择某些元素：

```
var aNumbers = new Array(1, -2, -3, 4, 5, -6);
```



更多资源请访问稀酷客(www.ckook.com)

```
var aResult = [i for each (i in aNumbers) if (i > 0)];

// result: aResult == [1, 4, 5]
```

下面是对应的 filter()语法：

```
function isPositive(iValue, iIndex, aArray)
{
   return (iValue > 0);
}

var aNumbers = new Array(1, -2, -3, 4, 5, -6);
var aResult = aNumbers.filter(isPositive);
```

数组包含中的方括号引入了一个隐含的作用域。新变量(例如上例中的 i)有本地作用域(就像它们是用 let 声明的)，不能用于数组包含的外部。

数组包含的输入未必是数组，也可以使用迭代器或生成器(详见第 21 章)。

18.11　解构赋值

兼容性：FF2+，Opera 9.5 (部分)

通过解构赋值(在 JavaScript 1.7 中引入，在 1.8 版本中更正)，更便于从一个数组、对象或函数中提取另一个数组。可以把它看成将一个数组设置为等价于另一个数组：

```
[a, b, c] = [1, 2, 3]
// result: a == 1, b == 2, and c == 3
```

这个语句为交换变量值提供了一种简洁的方式：

```
var x = 1;
var y = 2;

[x, y] = [y, x]
// result: x == 2 and y == 1
```

替换了通过第三个临时变量来交换两个值的传统技术：

```
temp = x;
x = y;
y = temp;
```

另一个例子是从函数中接受一组值：

```
function fun()
{
   return [3, 'aardvark', 0];
}

[a, b, c] = fun()
// result: a == 3, b == 'aardvark', and c == 0
```


更多资源请访问稀酷客(www.ckook.com)

在撰写本书时，解构赋值得到 Firefox 2.0+的支持和 Opera 9.5 的部分支持，但 Internet Explorer 8 和其他浏览器都不支持它。不能使用 JavaScript 测试浏览器能否执行这些代码，因为旧版本在编译它时就会崩溃。可以使用如下的特殊脚本标记把这些代码与其他代码隔离开：

```
<script type="application/javascript;version=1.8" src="scriptname.js"></script>
```

但是，如果还必须为其他浏览器提供较传统的语法，就可以检查这些代码是否可用。除非在特殊的情况下可以确保代码在支持 JavaScript 1.8 的浏览器上运行，否则就可以等到其他浏览器厂商普遍采用解构赋值语法后，再执行它们。

18.12　与旧浏览器的兼容性

本章介绍的一些功能是在 JavaScript 1.6 及其以后版本中引入的，它们运行在 Firefox 2.0+或 3.0+上，但许多其他浏览器都不支持它们，包括 Internet Explorer 8。新的数组对象方法，例如 every()和 filter()，添加到对象原型后(如果它们不包含在其中)，就可以用于跨浏览器的脚本中，如本章所述。但语法的革新，包括数组包含和解构赋值，不能包含在由旧浏览器运行的脚本块中，因为 JavaScript 的旧版本在编译过程中会停止运行，也不能运行页面上的其他脚本。

在新语法得到普遍应用之前，是否应避免使用新语法呢？

如第 4 章所述，有一种方式可以识别仅运行在 JavaScript 给定版本(或更高版本)上的代码块，例如：

```
<script type="text/javascript" src="for-everyone.js"></script>
<script type="application/javascript;version=1.7" src="new-stuff.js"></script>
```

还可以把只有某些浏览器能执行的代码段与旧用户代理安全地隔离开。

问题是，如果浏览器较旧，应运行什么替代代码？如果必须编写代码，来执行与数组包含相同的操作，以支持旧浏览器，为什么不简单地将替代代码用于所有的浏览器？

与 Web 开发的其他方面一样，尤其是 JavaScript，即使没有 JavaScript 或该语言的某个特殊功能，也要确保页面上的所有基本功能都是有效的，再使用可用的高级功能在编写脚本时进行改进。这个逐步改进的原则确保页面可以用这些更卓越的工具更好、更快地工作，同时页面还可以为每个人实现基本的功能。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第19章

JSON — Native JavaScript Object Notation

JSON (JavaScript Object Notation)是表示便于许多编程语言读写的数据结构的一种轻型方式。这使JSON(类似于XML)成为各种语言编写的应用程序之间(例如服务器端PHP和客户端JavaScript)一种有效的数据交换格式。根据ECMAScript第5版标准，JSON现在是该语言的一个正式组成部分，未来几年会获得越来越多的浏览器的支持。而且，还有免费的代码来帮助旧式浏览器。

JSON由Douglas Crockford开发，于2006年向Internet社区推出。在2010年4月定稿的ECMAScript第5版中(JavaScript就基于ECMAScript)，它是ECMAScript标准的一个组成部分。

本章包含哪些内容？

- JSON的工作原理
- 收发JSON数据
- JSON对象方法
- 安全限制

19.1 JSON的工作原理

JSON合并了JavaScript语法中表示数组和对象的两个方面，来表示数据结构——使用方括号[ ]表示简单数组：

```
[ "item1", "item2", "item3" ]
```

使用花括号{}表示对象和复合数组：

```
{ "key": "value" }
```

在第一种情况下，可通过数字索引访问嵌入的值(注意第一项的索引是0)：

```
var aItems = [ "item1", "item2", "item3" ];
// e.g., aItems[0] == "item1"
```

PDG

更多资源请访问稀酷客(www.ckook.com)

在第二种情况下，可以通过键名访问嵌入的值：

```
var oExample = { "charlie": "horse" };
// oExample["charlie"] == "horse"
// oExample.charlie == "horse"
```

使用这两种方式，可以用子记录(带命名的或数值的索引键)描述许多数据结构：

```
var oNovelist =
{
   "firstName": "Joanna",
   "lastName": "Russ",
   "novels":
   [
      {
         "title": "And Choas Died",
         "year": "1970"
      },
      {
         "title": "The Female Man",
         "year": "1975"
      }
   ]
};
```

在这个例子中，firstName 和 lastName 是简单的字符串值，而 novels 是一个数值索引的数组。下面是两种表示这些值的方式：

```
var sMsg = oNovelist['firstName'] + ' ' + oNovelist['lastName'] + "'s novel " +
 oNovelist['novels'][0][ 'title'] + ' was published in ' +
 oNovelist['novels'][0][ 'year'] + '.';

var sMsg = oNovelist.firstName + ' ' + oNovelist.lastName + "'s novel " +
 oNovelist.novels[0].title + ' was published in ' +
 oNovelist.novels[0].year + '.';

// result: Joanna Russ's novel And Chaos Died was published in 1970.
```

第一个例子把结构处理为一个复合数组(键放在方括号中)。第二个例子使用句点记号检索对象的属性。第一个语法总是有效的，因为键是字符串时，应放在引号中。第二个例子使用了句点记号，是因为这些键都不包含连字符、句点和其他合法键值的字符，不会让 JavaScript 出错。

```
var sMsg = oNovelist['first-name'];  // this works
var sMsg = oNovelist.first-name;     // oh no! JavaScript tries to subtract
```

交叉引用：
这方面可查阅第 18 章中的 18.8 一节。

更多资源请访问稀酷客(www.ckook.com)

数据类型

有 6 种类型的数据可以用作 JSON 值：

- 字符串(0 个或多个 Unicode 字符的集合，放在双引号中，双引号使用反斜杠转义)。
- 数值(整数或实数，不使用 8 进制和 16 进制格式)。
- 对象(键:值对的集合，用逗号隔开，放在花括号中)。
- 数组(一系列有序的值，用逗号隔开，放在方括号中)。
- 布尔值(true 或 false)
- null

可将空白插入到任何标记对之间。

19.2 收发 JSON 数据

JSON 语法使用 JavaScript 的正常对象和数组语法，所以不需要语言的任何插件就可以在单个脚本中建立和读取 JSON 数据结构。只有在脚本之间共享这些结构时，会出现复杂问题。

为了读取、写入、发送和接收 JSON 数据对象，需要把它转换为字符串，并能从字符串转换为 JSON 数据对象。其标准方式是用与 JavaScript 相同的方式读写它们：

```
{"firstName":"Joanna","lastName":"Russ","novels":[{"title":"And Choas
Died","year":"1970"},{"title":"The Female Man","year":"1975"}]}
```

接着，这个字符串就可以导入能处理 JSON 的任何编程语言。

在 JavaScript 的对象和字符串之间执行这个转换是非常简单的。最好给旧式浏览器使用 http://json.org 上的函数，这样就不必完成重复的工作了。

因为 JSON 数据存储在字符串中，就像在 JavaScript 源代码中一样，所以初看起来，似乎把 JSON 数据从字符串转换回对象的最佳权宜之计是使用 JavaScript 方法 eval()：

```
var sStringJSON = readJSONDataFromWhateverSource();

eval("var oJSON = " + sStringJSON);
// or:
var oJSON = eval('(' + sStringJSON + ')');
```

现在，oJSON.firstName 会给上面的例子生成‘Joanna’(在第二次使用 eval()时，注意只有给 JSON 字符串加上括号，才能满足 JavaScript 解释器的要求)。

但是，eval()为 JavaScript 中的恶意攻击打开了大门，应在所有公共页面上禁止使用它。正常的JSON解析器会逐个字符地检查用于建立JSON对象的数据字符串，而不是屈服于使用eval()的诱惑。

JSON 成为标准

JSON 对象是 ECMAScript 第 5 版标准(2009 年 12 月出版)的语言的一部分。Mozilla 在 JavaScript 1.8.1 中使用 Gecko 布局引擎 1.9.1 实现了该对象；受此影响，Firefox 3.5、Thunderbird 3 和 SeaMonkey 2

更多资源请访问稀酷客(www.ckook.com)

随后也相继实现了它。其他浏览器开发团队也肯定会实现它。

JSON 对象有两个方法 parse()和 stringify()。toJSON()方法也添加到 Date 对象上。

19.3 JSON 对象

属　性	方　法
	parse()
	stringify()

方法

JSON.parse(*string [，translator]*)

返回值：JSON 对象引用

兼容性：Moz1.9.1+，Firefox3.5+

使用 JSON.parse()方法可以把 JSON 字符串转换为 JSON 对象。

```
object = JSON.parse(string, translator)
```

string 参数是要转换为对象的 JSON 字符串。

相关主题：JSON.stringify() 方法

JSON.stringify(*object [，replacer [，space]]*)

返回值：JSON 字符串

兼容性：Moz1.9.1+，Firefox3.5+

JSON.stringify() 可以把 JSON 对象转换为 JSON 字符串。

```
string = JSON.stringify(object, replacer, space)
```

object 参数是要转换为字符串的 JSON 对象。

replacer 是一个可选参数，它可以是改变字符串转换过程的函数，也可以是一组 String 和 Number 对象，这些对象用作一个白名单，用于选择要转换为字符串的对象的属性。如果这个值是空或没有提供，则在所得的 JSON 字符串中包含对象的所有属性。

space 是一个 String 或 Number 对象，用于把空白插入输出的 JSON 字符串，以提高可读性。如果这是一个数值，就表示用作空白的空格字符数；如果该数值大于 10，就取其值为 10。小于 1 的值表示不应使用空格。如果这是一个字符串(如果该字符串多于 10 个字符，就取其前 10 个字符)，就把该字符串用作空白。如果没有提供这个参数(或者为空)，就不使用空白。

相关主题：JSON.parse()方法

更多资源请访问稀酷客(www.ckook.com)

19.4　安全限制

有些人关注 JSON 在 Web 安全性方面的漏洞。我们认为，只要足够谨慎地使用该技术，就可以解决这些问题。

使用 eval()把 JSON 字符串转换为对象(或在公共页面上完成其他任务)是很容易避免的一个陷阱。如果一个方法(函数)嵌入串行化的 JSON 中，并使用 eval()转换它，就可以在没有得到页面作者的脚本许可的情况下执行 JavaScript 代码。明显的解决方案是使用非 eval()解析算法，例如 Douglas Crockford 发布的、在老式浏览器上使用的非 eval()解析算法，以及在新浏览器上使用的新 JSON 对象方法 parse()。

DOM 方法 XMLHttpRequest()常用于在服务器和客户端之间互换信息。它默认只允许与生成当前页面的服务器互换信息。但是，插入一个 script 标记，来链接站外的 JavaScript 文件，就可以绕过这层保护。其解决方案是允许在自己的页面上执行第三方的 JavaScript 时，要保持警惕。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第20章

E4X — Native XML Processing

近年来，随着 Ajax (Asynchronous JavaScript and XML)连接的流行，Web 开发人员使用 XML 越来越常见，否则就无法越出 HTML 的范围。实际上，XML 对于每天的 Web 开发是必不可少的，以至于 ECMA 把它合并到 JavaScript 的句法结构中。现在可以在 JavaScript 语句中编写不带引号的 XML 标记——大多数现代浏览器都允许这样。

这个 JavaScript 语法的新特性使 XML 在核心 JavaScript 语言和 DOM 中都非常显眼。脚本编写的其他每个方面都是核心 JavaScript 语言或 DOM 的一部分，而 XML 在核心 JavaScript 语言和 DOM 中都存在。

配书光盘上的第 39 章将讨论如何在客户端和服务器之间传输 XML。

本章包含哪些内容？

- 使用 XML 对象
- 添加元素和特性
- 选择记录
- 把 XML 对象串行化为文本

20.1 XML

简言之，XML 是一种标记语言，用于包含和传输文本数据。它用三种标记包含嵌入的元素，这三种标记是开始标记和结束标记：

```
<hobby>music</hobby>
```

以及空元素标记：

```
<breathe/>
```

元素可以包含特性，特性是位于开始标记或空标记中的名称/值对：

```
<fax location="home" call-first="true">123-456-7890</fax>;
```

在 HTML 和 XHTML 中，元素和特性名称受 Document Type Declaration 或 XMLSchema 的限制，而 XML 元素和特性名称是完全随意的，由数据的作者定义，以满足当前的目标。

XML 的完整描述可参阅 http://w3.org/TR/ REC-xml/上的 W3C 规范。

PDG

更多资源请访问稀酷客(www.ckook.com)

20.2 ECMAScript for XML (E4X)

在 E4X 推出之前，用 JavaScript 访问 XML 文档的唯一方式是在对象级别上使用 DOM。随着 E4X 的出现，XML 可以处理为基本类型，与字符串、数值和布尔值相同。这意味着什么？简言之，XML 访问对于 Web 开发人员而言更简单了，对于终端用户而言更迅捷了。用 XML 基本类型创建的 XML 对象不是 DOM 的一部分，也不是 XML 的 DOM 表示。因为它是一种基本类型，所以可以像使用其他 JavaScript 对象那样使用它。

20.2.1 使用 XML 对象

有了 E4X 后，就可以像声明字符串、数值或布尔对象那样，通过 XML 构造函数声明 XML 对象：

```
var oCopyright = new XML();
```

可以用文本初始化对象：

```
var sXMLText = "<copyright>"
               + "<date>2010</date>"
               + "<author>Danny Goodman</author>"
               + "</copyright>";

var oCopyright = new XML(sXMLText);
```

显然，还可在赋值语句中使用不带引号的 XML 标记，隐式地创建 XML 对象：

```
var oCopyright = <copyright>
                    <date>2010</date>
                    <author>Danny Goodman</author>
                 </copyright>;
```

这类语法，尤其是所构建的文本可以不加引号就赋予对象，可能令人回想起 JavaScript 内部的正则表达式语法，如第 45 章所述。

1. XML 元素

XML 对象中的 XML 元素是其属性。使用熟悉的句点记号 object.property 和方括号记号 object[property] 都可以引用元素及其值：

```
var sText = "Copyright " + oCopyright.date + " by " + oCopyright.author;
var sText = "Copyright " + oCopyright['date'] + " by " + oCopyright['author'];
```

最外层的 XML 元素是对象的根，所以上述 oCopyright 对象的第一个顶级属性是 date 元素：

```
alert(oCopyright.name());              // result = 'copyright'
alert(oCopyright.child(0).name());     // result = 'date'
alert(oCopyright.date);                // result = '2010'
```

当然，与其他 JavaScript 对象一样，元素标记名称包含使 JavaScript 混淆的任何字符时，就

更多资源请访问稀酷客(www.ckook.com)

不能使用对象句点记号了，例如：

```
var oXML = <author>
               <first-name>Danny</first-name>
           </author>;

var sText = oXML['first-name'];      // result: Danny
var sText = oXML.first-name;         // result: JavaScript stops cold!
```

在上面的例子中，JavaScript 尝试从 XML 元素 oXML.first 的值中减去 name 变量的值。如果 oXML.first 或 name 不存在，或者它们存在，但其数值不能相减，JavaScript 就会失败。其寓意是，如果希望使用句点记号，且保证能完全控制元素的标记名称，就可以把该名称的字符集限制为字母数字和下划线。如果所处理的数据来自于无法控制的源，就使用方括号记号，它总是能成功。

2. 添加元素

XML 对象的 appendChild()方法与同名的 DOM 方法相似，都是把子元素插入指定的对象中：

```
var oBook = <book/>;
var oAuthor = <author/>;
oBook.appendChild(oAuthor);

result: <book>
            <author/>
        </book>
```

通过字符串通常使用的连接语法，也可以把元素追加到 XML 元素上：

```
oCopyright += <addendum>All rights reserved.</addendum>;
```

其结果是：

```
<copyright>
  <date>2010</date>
  <author>Danny Goodman</author>
</copyright>;
<addendum>All rights reserved.<addendum>
```

为在 copyright 元素中插入新元素，需要编写如下代码：

```
oCopyright.author += <addendum>All rights reserved.</addendum>;
```

其结果是：

```
<copyright>
   <date>2010</date>
   <author>Danny Goodman</author>
   <addendum>All rights reserved.<addendum>
</copyright>
```


更多资源请访问稀酷客(www.ckook.com)

3. 嵌入的 JavaScript 表达式

希望 XML 对象与 JavaScript 表达式动态交互操作时，E4X 就非常有效。为此需要使用花括号：

```
var oDate = new Date();

var oCopyright = <copyright>
                    <date>{oDate.getFullYear()}</date>
                    <author>Danny Goodman</author>
                 </copyright>;
```

只要花括号中的内容是 JavaScript 表达式，其结果就会插入 XML 流。上述 JavaScript 表达式的结果是 2020 年，所以上面的例子会生成：

```
<copyright>
   <date>2020</date>
   <author>Danny Goodman</author>
</copyright>;
```

4. XML 元素的特性

与 XML 文档一样，XML 对象中的元素可以有特性。这些特性值也可以用花括号动态设置。使用句点记号，并在特性名前加上@操作符，就可以提取元素的特性值。如下面对 location 特性的操作：

```
var oPhonebook = <phonebook>
                    <entry>
                       <name>Janis Joplin</name>
                       <phone location="home">123-456-7890</phone>
                    </entry>
                 </phonebook>;

var sText = oPhonebook.entry.name + "'s "
          + oPhonebook.entry.phone.@location + " phone is "
          + oPhonebook.entry.phone + ".";
// or:
var sText = oPhonebook['entry'][ 'name'] + "'s "
          + oPhonebook['entry'][ 'phone'][ '@type'] + " phone is "
          + oPhonebook['entry'][ 'phone'] + ".";

// result == Janis Joplin's home phone is 123-456-7890.
```

还可以使用@记号来创建和修改特性值：

```
oPhonebook.entry.phone.@location = "work";
```

5. XMLList 对象

大多数 XML 文档都包含重复的元素组。在 XML 对象中创建这种重复的元素组时，可以给父元素相同的元素重复一个标记名：

```
var oBook = <book>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>The JavaScript Bible 7th Edition</title>
        <author>Danny Goodman</author>
        <coauthor>Michael Morrison</coauthor>
        <coauthor>Paul Novitski</coauthor>
        <coauthor>Cynthia Gustaff Rayl</coauthor>
    </book>;
```

在引用重复的组时，就会返回一个 XMLList 对象，它可以像数组那样处理。但有两个主要区别：length()是一个方法，而不是属性，因此需要括号();以及 XMLList 不支持的其他 Array 对象方法。

```
var sText = "The number of co-authors is " + oBook.coauthor.length() + ".";
```

在当前的例子中，oBook.coauthor 会在整个数据结构中包含一个列表，因为有多个同级元素的标记名称相同。

```
// extract the XMLList of coauthor elements
var oCoauthors = oBook.coauthor;

var sText = oBook.title + ' was written by ' + oBook.author + ' with ';

// loop through co-authors
var sConj = '';
var iCoauthors = oCoauthors.length();
var sLastAuthor = oCoauthors[iCoauthors-1];

for (var i=0; i < iCoauthors; i++)
{
      // comma before item if more than two items
      if (i > 0 && iCoauthors > 2)
      {
         sConj = ', ';
      }
      // 'and' before item if last in a series
      if (i == iCoauthors-1 && iCoauthors > 1)
      {
         sConj += ' and ';
      }

   // add item to list
   sText += sConj + oCoauthors[i];
}
sText += '.';

// result == The JavaScript Bible 7th Edition was written by Danny Goodman
             with Michael Morrison, Paul Novitski, and Cynthia Gustaff Rayl.
```

星号(*)可用于表示集合中的所有项。使用前面的 XML：

```
var iCount = oBook.*.length();
```

结果是 5——根父元素有一个 title、一个 author 和三个 coauthor 元素。

更多资源请访问稀酷客(www.ckook.com)

6. 添加列表项

为了给 XMLList 添加新项，可使用前面的连接语法，并用其重复的元素给列表命名：

```
oBook.coauthor += <coauthor>John Lennon</coauthor>;
```

得到：

```
<book>
   <title>The JavaScript Bible 7th Edition</title>
   <author>Danny Goodman</author>
   <coauthor>Michael Morrison</coauthor>
   <coauthor>Paul Novitski</coauthor>
   <coauthor>Cynthia Gustaff Rayl</coauthor>
   <coauthor>John Lennon</coauthor>
</book>
```

注意：
如果用前面的方法从 XML 对象中提取 XMLList 对象：

```
// extract the XMLList of coauthor elements
var oCoauthors = oBook.c oauthor;
```

该列表就是 XML 元素的静态副本，没有动态链接。以后对 XML 对象或 XMLList 对象的任何修改都不会反映出来。在这个例子中，给 oCoauthors 列表添加一个新的 coauthor 元素，不会改变最初的 XML 对象。

7. 选择列表项

E4X 可以根据特性和值选择 XMLList 项。考虑下面的例子：

```
var oPhonebook = <phonebook>
                    <entry>
                       <name>Janis Joplin</name>
                       <phone location="home">123-456-7890</phone>
                       <phone location="work">234-567-8901</phone>
                       <phone location="cell">345-678-9012</phone>
                    </entry>
                    <entry>
                       <name>John Lennon</name>
                       <phone location="home">987-654-3210</phone>
                       <phone location="work">876-543-2109</phone>
                    </entry>
                 </phonebook>;
```

前面提到，很容易选择所有名称匹配的元素：

```
// extract a list of all the phone elements
var oPhones = oPhonebook.entry.phone;
```

```
result:
```


更多资源请访问稀酷客(www.ckook.com)

```
<phone location="home">123-456-7890</phone>
<phone location="work">234-567-8901</phone>
<phone location="cell">345-678-9012</phone>
<phone location="home">987-654-3210</phone>
<phone location="work">876-543-2109</phone>
```

注意，这会在多条 entry 记录中查找到 phone 元素。选择语句其实表示：“找出父元素是根元素 entry 的所有 phone 元素”。如果只希望搜索一个 entry 元素，只需指定其索引：

```
var oPhones = oPhonebook.entry[0].phone;
```

同样，可根据特性值来选择元素：

```
var oWorkPhones = oPhonebook.entry.phone.(@location == 'work');

result:
<phone location="work">234-567-8901</phone>
<phone location="work">876-543-2109</phone>
```

这就把选择范围缩小为 location 特性为 work 的 phone 元素——圆括号中的表达式等于 true。注意这里给相等比较使用了两个等于号==，而一个等于号=用于赋值。

这个语法的一个方便之处是，在搜索同级元素后，可以提取其中一个元素的值：

```
var oPhone = oPhonebook.entry.(name == 'Janis Joplin').phone.
  (@location == 'home');

result:
<phone location="home">123-456-7890</phone>
```

选择器表示，“给出名为 Janis Joplin 的项的家庭电话号码。”该语句定位有 Janis Joplin 子元素的 entry，但返回其 phone 元素(name 元素的一个同级元素)。这很不错吧？当然，与许多已去世的摇滚歌星一样，Janis 的电话号码目前没有列入电话簿，但我们总是可以这样梦想。

XML 选择器表达式可以包含 JavaScript 函数，且不必使用花括号语法：

```
function checkExchange(sExchange, sPhone)
{
    sExchange = '-' + sExchange + '-';
    return (sPhone.indexOf(sExchange) > 0);
}
var sPhone = oPhonebook.entry.(checkExchange('543', phone.
  (@location=='work'))).name;
```

这会返回 John Lennon，因为我们选择了 checkExchange()函数返回 true 的项。

8. 串行化 XML 对象

XML 是跟踪脚本中信息的一种简便格式，但其主要用途是在程序之间交流结构化数据。为了把 XML 结果发送给服务器，需要把 JavaScript 对象转换为文本字符串。XML 对象有两个可以互换的方法来完成这个工作：toXMLString()和 toString()。这两个方法可以处理任意对象子集：

```
var sXMLText = oPhonebook.toXMLString();
```


更多资源请访问稀酷客(www.ckook.com)

```
var sXMLText = oPhonebook.entry.(name == 'John Lennon').toString();
```

20.2.2　在 HTML 中嵌入 E4X

尽管可以通过 E4X 使用标准的 MIME 类型：

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

但可能会出现费解的语法错误。这通常有两个原因：使用 HTML 注释对老式浏览器隐藏脚本；或者把脚本放在 XML CDATA 段中。如果有这两种情形，可以把 E4X 参数添加到标准 MIME 类型中：

```
<script type="text/javascript;e4x=1">
```

20.2.3　方法

child()

返回值： XML 对象
兼容性： WinIE5-，MacIE-，NN-，FF3.5+，Safari-，Opera-，Chrome-
child()方法返回一个 XML 元素，它是所指定对象的直接子对象：

```
var oPhonebook = <phonebook>
                      <person>
                          <name>Janis Joplin</name>
                          <phone location="home">123-456-7890</phone>
                      </person>
                  </phonebook>;

alert(oPhonebook.person.child(1));        // result = '123-456-7890'
```

注意 child()是一个方法，但我们常常像数组那样使用它，所以要给索引使用圆括号，而不是方括号。与数组一样，child()集合是一个基于 0 的列表，所以 child(1)会找到第二个子元素。与数组不同的是，不能使用 forEach 语法。

相关主题： childNodes() DOM 属性

copy()

返回值： XML 对象
兼容性： WinIE5-、 MacIE-、 NN-、FF3.5+、Safari-、Opera-、Chrome-
copy()方法返回 XML 对象中所选部分的深度副本。例如：

```
var oPhonebook = <phonebook>
                      <person>
                          <name>Janis Joplin</name>
                          <phone location="home">123-456-7890</phone>
                      </person>
                  </phonebook>;
```


更多资源请访问稀酷客(www.ckook.com)

```
var oPerson = oPhonebook.person.copy();
```

这个操作的结果是如下 XML 对象：

```
<person>
   <name>Janis Joplin</name>
   <phone location="home">123-456-7890</phone>
</person>
```

注意这也可以使用简单的赋值语句得到：

```
var oPerson = oPhonebook.person;
```

name()

返回值：字符串

兼容性：WinIE5-，MacIE-，NN-，FF3.5+，Safari-，Opera-，Chrome-

name()方法返回一个表示 XML 元素标记名称的字符串。

```
var oPhonebook = <phonebook>
                    <person>
                       <name>Janis Joplin</name>
                       <phone location="home">123-456-7890</phone>
                    </person>
                 </phonebook>;

alert(oPhonebook.child(0).name());              // result = 'person'
alert(oPhonebook.person.child(1).name());       // result = 'phone'
alert(oPhonebook.person.child(1));              // result = '123-456-7890'
```

注意 child()是一个方法，但我们常像数组那样使用它，所以要给索引使用圆括号，而不是方括号。与数组一样，child()集合是一个基于 0 的列表，所以 child(1)会找到第二个子元素。

相关主题：childNodes() DOM 属性

toString()，toXMLString()

返回值：字符串

兼容性：WinIE5-，MacIE-，NN-，FF1.5+，Safari-，Opera-，Chrome-

toString()和 toXMLString()方法返回一个字符串，它连接了 XML 对象的所有文本节点。

相关主题：*string*.toString()

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第21章

控制结构和异常处理

早上起床之后，人们开始处理一天的事务，下班后遛狗，换上拖鞋放松放松。这与程序从头执行到尾没有什么不同，在这个过程中，程序需要执行许多小步骤，并不是所有的步骤都按顺序处理，有时需要做出决定，控制下一步做什么，或者重复任务，直到所有工作全部完成。控制结构就是 JavaScript 完成这些任务的工具。

本章包含哪些内容？
脚本执行的多路径分支
在有序的数据集中循环
使用异常处理技巧

JavaScript 控制结构遵循与其他许多编程语言同样的规则，基本的决策和循环结构能完成所有编程任务。

另一个重要的程序控制机制是错误(异常)处理，它是 ECMA-262 语言标准的正式组成部分。异常处理概念在 IE5.5 和 NN6 的 JavaScript 中引入，但其他环境的程序员非常熟悉它。在代码中采用异常处理技术可以极大地提高从错误中恢复的能力，例如从错误的用户输入或网络问题中恢复。

21.1 if 和 if...else 判定语句

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

JavaScript 程序经常需要根据变量和对象属性的当前值做出判定。这种判定每次都只能有两个可能的输出。决定程序执行哪条路径的因素是语句的真假。例如，晚上遛狗后回家，要测试的语句是“我的拖鞋在客厅吗？”，若语句为假，就寻找拖鞋；如果语句为真，就穿上拖鞋放松一下。

21.1.1 简单判定

JavaScript 中简单的判定语法总是用关键字 if 开始，后跟测试条件，以及条件为真时执行的语句。JavaScript 没有 then 关键字，该关键字隐含在包括该结构各个成员的圆括号和花括号中。正式语法如下：

更多资源请访问稀酷客(www.ckook.com)

```
if (condition)
{
   statementsIfTrue
}
```

此结构表示，如果条件为真，程序就执行花括号中的语句。之后继续执行右花括号(})后面的语句。如果寻找拖鞋是脚本语言的一部分，代码如下：

```
if (comfyFootwearInLivingRoom == false)
{
   look for them
}
```

若不熟悉 C/C++，可能会注意到上述代码中的双等号，下一章将介绍此类操作符。这个操作符可以比较等号两边的部分是否相等。也就是说，if 结构的 condition 语句必须产生 Boolean 值(true 或 false)。许多对象属性都是 Boolean 值，因此可以将这类属性的引用用于 condition 语句。另外，condition 语句包含两个用比较符(如==(等于)或!=(不等于))分开的值。

在下面的 JavaScript 例子中，函数接收一个文本输入对象：

```
function notTooHigh(oInput)
{
      if (parseInt(oInput.value) > 100)
      {
         alert("Sorry, the value you entered is too high. Try again.");
         return false;
      }
   return true;
}
```

括号中的条件比较输入域中的内容和值 100。若输入值大于 100，函数就会提醒用户，并给在脚本中其他地方调用该函数的语句返回 false 值；若值小于 100，就跳过所有代码，函数返回 true。

21.1.2　(*condition*)表达式

控制结构的条件会比较具体的条件与值，例如字符串是否为空，或值是否为 null。可以使用两个简单的方法处理许多不同的情况。表 21-1 是求值为(或等价于)true 或 false 的值，这个值用于控制结构的条件表达式。

表 21-1　条件的值

True	False	True	False	True	False
非空字符串	空字符串	非空值	空值	属性已定义	属性未定义
非 0 数字	0	对象存在	对象不存在		

不需要在涉及这些值的条件中写出这些等价表达式，只需要提供要检测的值即可。例如，如果 if 结构中变量 myVal 的值可能是 null、空字符串或需要进一步处理的字符串，就可以使用

更多资源请访问稀酷客(www.ckook.com)

下面的语句：

```
if (myVal)
{
   // do processing on myVal
}
```

所有的 null 或空字符串都等价于 false，这样，只有 myVal 是某个可处理的值才能执行到 if 结构的内部。该机制和本书其他部分中为浏览器使用对象检测机制来分支是相同的。例如，只有 document 对象有 images 数组属性，才执行嵌套在下面代码段内的内容：

```
if (document.images)
{
   // do processing on image objects
}
```

21.1.3 复杂判定语句

若返回主路径之前只需要做出简单的判定，就可以使用前述的简单 if 结构。但程序和生活中的判定不全是这样。为了演示 JavaScript 判定中的另外两条路径，可在判定结构中加上一个组件。语法如下：

```
if (condition)
{
   // statementsIfTrue
}
else
{
   // statementsIfFalse
}
```

增加 else 关键字，就为 if 结构指定了条件为假时的处理路径。statementsIfTrue 和 statementsIfFalse 未必相同；一个分支可能只有一行代码，而另一个分支可能有上百行代码。任一个分支执行完毕后，就继续执行右花括号后面的代码。为了说明如何使用该结构，下面的脚本段示例根据当年是否是闰年(有关 Date 对象的内容，请参见第 17 章)，指定二月份的天数(使用第 22 章介绍的取模运算，确定年数能否被 4 整除，但这里省略了闰年计算的其他细节)。

```
var howMany = 0;
var currentTime = new Date();
var theYear = currentTime.getFullYear();
   if (theYear % 4 == 0)
   {
      howMany = 29;
   }
   else
   {
      howMany = 28;
   }
```


更多资源请访问稀酷客(www.ckook.com)

下面的代码只能在两条可能的路径中执行一条路径，将天数赋给 howMany 变量，如果未使用 else 部分，如下：

```
var howMany = 0;
var currentTime = new Date();
var theYear = currentTime.getFullYear();
   if (theYear % 4 == 0)
   {
      howMany = 29;
   }
howMany = 28;
```

变量在暂时设置为 29 之后，便永远设置为 28，所以 else 结构是必需的。但是，把变量初始化为 28，就可以在没有 else 部分的情况下得到正确结果，如下：

```
var howMany = 28;
var currentTime = new Date();
var theYear = currentTime.getFullYear();
   if (theYear % 4 == 0)
   {
       howMany = 29;
   }
```

21.1.4　嵌套的 if...else 语句

设计复杂的判定过程需要注意脚本必须处理的判定逻辑，以及必须在给定条件下执行的语句。在版本 4 的浏览器中引入 switch 结构(详见本章后面)后，就不需要这种重复的结构了，但有些情况下，仍需要将一系列嵌套的 if...else 结构组合成复杂的判定结构。如果没有 JavaScript 文本编辑器的帮助来缩进语句，并且正确结束(使用右花括号)语句块，就必须小心地监控编写处理过程。另外，在错误发生时，JavaScript 提供的错误信息(参见配书光盘中的第 48 章)可能不直接指出有问题的行，只是指出有问题的区域。

为了演示嵌套的 if...else 结构，程序清单 21-1 给出了一个处理复杂问题的简单用户界面。一个文本对象要求用户输入三个字母 A、B 或 C 中的一个，该输入域后面的脚本为每种可能的输入处理不同的信息：

- 用户没有输入值。
- 用户输入 A。
- 用户输入 B。
- 用户输入 C。
- 用户输入其他值。

注意：

本章和本书许多章节的代码中指定事件处理程序的函数是 addEvent()。有关这个跨浏览器的事件处理程序的详情，见第 32 章。

更多资源请访问稀酷客(www.ckook.com)

格式问题

JavaScript 并不需要缩进 if 结构，也不需要进一步缩进条件为“真”时执行的语句，但如本书所示，这是大多数 JavaScript 脚本编写者都遵循的惯例。在文本编辑器中编写代码时，可以用 Tab 键确定缩进等级，一些开发人员喜欢使用编辑器的设置，将制表符转换为空格，这可以在不同的编辑器之间保持一致。当加载脚本时，浏览器会忽略这些制表符(和/或空格)。

缩进有许多样式，本书最常用的样式称为 Allman：左右花括号各单独占一行，而许多程序员喜欢 1TBS(The One True Brace Style)：将左花括号与控制语句放在同一行上。

程序清单 21-1　嵌套的 if ...else 结构

HTML: jsb-21-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Nested if...else</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-21-01.js"></script>
   </head>
   <body>
      <h1>Nested if...else</h1>
      <form id="theForm" action="validate.php" method="get">
         <fieldset>
            <label for="theText">Please enter A, B, or C: </label>
            <input id="theText" type="text">
            <input type="submit">
         </fieldset>
      </form>
   </body>
</html>
```

JavaScript: jsb-21-01.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // add an event to the form
   addEvent(document.getElementById('theForm'), 'submit',
      suppressFormSubmission);

   // add an event to the input box
   addEvent(document.getElementById('theText'), 'change', testLetter);
}

function suppressFormSubmission(evt)
{
      // consolidate event handling
```


更多资源请访问稀酷客(www.ckook.com)

```
        if (!evt) evt = window.event;

    // cancel default behavior
        // W3C DOM method (hide from IE)
        if (evt.preventDefault) evt.preventDefault();

    // IE method
    return false;
}

function testLetter()
{
    // assign to shorter variable name
    var inpVal = this.value;

        if (inpVal != "")
        { // if entry is not empty then dive in...
            if (inpVal == "A")
            { // Is it an "A"?
              alert("Thanks for the A.");
            }
            else if (inpVal == "B")
            { // No. Is it a "B"?
              alert("Thanks for the B.");
            }
            else if (inpVal == "C")
            { // No. Is it a "C"?
              alert("Thanks for the C.");
            }
            else
            { // Nope. None of the above
              alert("Sorry, wrong character or case.");
            }
        }
        else
        {    // value was empty, so skipped all other stuff above
            alert("You did not enter anything.");
        }
}
```

每个条件只执行满足这个特定条件的语句，但需要多次查询才能确定入口。找到 true 后，就执行相应的语句，执行路径上没有需要运行的其他语句，这样就不需要跳出嵌套结构。

form 元素中的 action 表示 PHP 脚本 validate.php。第 7 章建议，最好总是指定一个服务器端程序，以防用户关闭 JavaScript。

21.2　条件表达式

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

前面介绍了 JavaScript 的判定结构，现在介绍一种可以代替 if...else 控制结构的特殊表达式，

更多资源请访问稀酷客(www.ckook.com)

它也是一个通用的判定类型：当希望根据某个条件的输出，决定将两个值的一个赋给变量时，就可以使用这个结构。条件表达式的正式定义如下：

```
variable = (condition) ? val1 : val2;
```

这个表达式的含义是：如果 condition 语句为真，JavaScript 将 val1 赋予变量；否则，将 val2 赋予变量。与其他条件表达式相同，这个条件表达式也必须放在括号里，问号是其关键，而冒号将两个可能的值分开。

条件表达式虽然不符合人们的直觉，其代码也不容易读取，但非常紧凑。比较下面赋值判定的 if...else 版本：

```
var collectorStatus;
   if (CDCount > 500)
   {
      collectorStatus = "fanatic";
   }
   else
   {
      collectorStatus = "normal";
   }
```

和条件表达式版本：

```
var collectorStatus = (CDCount > 500) ? "fanatic" : "normal";
```

后一种少了许多代码行，但其内部的处理和 if...else 结构相同。当然，若判定路径包含多个语句，而不是只用一个语句设置变量的值，那么 if...else 或 switch 结构较好。但如果需要处理二元动作，例如，在脚本中设置真/假标志，这种快捷方式非常方便。

21.3　switch 语句

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

在一些情况下，二元结构(真或假)判定不能满足脚本的处理需求。object 属性或 variable 值可能包含几个值中的一个，每个值都需要一个单独的执行路径。建立这类判定路径的显而易见的方式是，使用一系列 if...else 结构。然而，嵌套代码处理起来比较笨拙，而且要检测的条件越多，处理效率就越低，因为必须测试每个条件。最终结果是大量子句和括号会令人感到迷惑不解。

从版本 4 的浏览器开始，JavaScript 引入了一种其他许多语言都使用的控制结构，其实现方式和 PHP、Java 以及 C/C++相同：使用 switch 和 case 关键字。基本前提是可以根据某个表达式的值创建任意数量的执行路径。在该结构的开头指定表达式，并为每个执行路径指定匹配特定值的标记。

switch 语句的正式语法是：

```
switch (expression)
{
   case label1:
```


更多资源请访问稀酷客(www.ckook.com)

```
        [statements]
        [break;]
    case label2:
        [statements]
        [break;]
    ...
    [default:
        statements]
}
```

例如：

```
switch (TimeOfDay)
{
    case 'morning':
        meal = 'breakfast';
        break;

    case 'midday':
        meal = 'lunch';
        break;

    case 'afternoon':
    case 'evening':
        meal = 'dinner';
        break;

    default:
        meal = 'fast';
}
```

switch 语句的 expression 参数可以等于任何字符串或数值，标签表示表达式的字符串值时要加引号。注意 break 语句是可选的，break 语句迫使 switch 表达式不根据表达式的值检测后面的标签。注意，在每个 case 的末端不使用 break 语句就会执行 switch 表达式中的每行代码。但 switch 语句不允许使用布尔操作符(每个标签都只能用于一个值)，所以要小心使用 break 语句。在前面的例子中，如果当前时间是下午或晚上，meal 就是 dinner。另一个选项是 default 语句，它提供了表达式的值和任何一个 case 语句的标签都不匹配时的执行路径。如果表达式不匹配任何 case 语句时不执行任何语句，可以省略结构的 default 部分。

为说明 switch 语句的语法，程序清单 21-2 提供了使用这种控制结构的应用程序架构。页面包含两个不同产品类别的数组，每种产品的名称和价格分别存储在各自的数组中，select 列表显示产品名称。用户选择一种产品后，脚本就会在对应的数组中查找产品名称，并显示价格。

这个应用程序的技巧在于选择列表中赋给每种产品的值。显示产品名称时，<option>标记的 value 特性是产品的数组类别，其值是决定执行哪个分支的表达式。另外还要为整个 switch 结构分配一个标签，目的是一旦找到匹配，就可以从 case 的深度嵌套循环中完全跳出 switch 结构(使用 break 语句)。可使用附加的 case 语句，把这个例子扩展到任意数目的产品类别数组。

更多资源请访问稀酷客(www.ckook.com)

程序清单 21-2　switch 结构

HTML: jsb-21-02.html

```html
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Switch Statement and Labeled Break</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-21-02.js"></script>
   </head>
   <body>
      <h1>Switch Statement and Labeled Break</h1>
      <p>Select a chip for lookup in the chip price table:</p>
      <form id="theForm" action="validate.php" method="get">
         <p>
            <label for="chips">Chip:</label>
            <select id="chips">
               <option></option>
               <option value="ICs">Septium 900MHz</option>
               <option value="ICs">Septium Pro 1.0GHz</option>
               <option value="ICs">Octium BFD 750MHz</option>
               <option value="snacks">Rays Potato Chips</option>
               <option value="snacks">Cheezey-ettes</option>
               <option value="snacks">Tortilla Flats</option>
               <option>Poker Chipset</option>
            </select>
            <label for="cost">  Price:</label>
            <input type="text" id="cost" size="10">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-21-02.js

```javascript
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // add an event to the drop down list
   addEvent(document.getElementById('chips'), 'change', getPrice);
}

// build two product arrays with a custom object, simulating two database tables
function product(name, price)
{
   this.name = name;
   this.price = price;
```


更多资源请访问稀酷客(www.ckook.com)

```
}

var ICs = new Array();
ICs[0] = new product("Septium 900MHz", "$149");
ICs[1] = new product("Septium Pro 1.0GHz", "$249");
ICs[2] = new product("Octium BFD 750MHz", "$329");

var snacks = new Array();
snacks[0] = new product("Rays Potato Chips", "$1.79");
snacks[1] = new product("Cheezey-ettes", "$1.59");
snacks[2] = new product("Tortilla Flats", "$2.29");

// lookup in the 'table' the cost associated with the product
function getPrice()
{
   var chipName = this.options[this.selectedIndex].text;
   var chipType = this.options[this.selectedIndex].value;
   var outField = document.getElementById('cost');

   master:
   switch(chipType)
   {
      case "ICs":
         for (var i = 0; i < ICs.length; i++)
         {
            if (ICs[i].name == chipName)
            {
               outField.value = ICs[i].price;
               break master;
            }
         }
         break;
      case "snacks":
         for (var i = 0; i < snacks.length; i++)
         {
            if (snacks[i].name == chipName)
            {
               outField.value = snacks[i].price;
               break master;
            }
         }
         break;
      default:
         outField.value = "Not Found";
   }
}
```

21.4 重复(for)循环

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

如其他章节的大量例子所示，对于很多 JavaScript 脚本而言，能够遍历数组中的每一项或表单中的每个元素，是非常重要的。最典型的操作是检查多个相同项的属性来查找某个值，如确定组中哪个单选按钮被选中。允许执行这种重复操作的一个 JavaScript 结构是 for 循环，它由该结构开头的关键字来命名。另外两种结构是 while 循环和 do-while 循环，参见后面的章节。

JavaScript 的 for 循环允许重复执行一系列语句任意多次，并包含在语句执行时使用的一个可选的循环计数器。下面是正式的语法定义：

```
for ( [initial expression]; [condition]; [update expression])
{
   statements
}
```

在执行 for 循环时，括号中的三个语句(for 语句的参数)具有重要的作用。

for 循环的初始表达式只执行一次，在 for 循环开始运行时执行。初始表达式的最常见用途是为循环计数变量指定名称和开始值，所以 for 结构中的 var 语句常用于声明一个变量，并赋予初始值(一般为 0 或 1)，例如：

```
var i = 0;
```

循环计数变量可使用任意变量名，但一般习惯使用字母 i，即 index 的缩写。也可以使用 counter 或其他能表示变量含义的词。但这个语句只在 for 循环开始时执行一次。

第二个语句是条件，类似于本章前面 if 结构中的条件语句。在初始表达式中建立循环记数变量后，条件语句通常定义循环计数器的上界。所以，这里最常用的语句是将循环计数变量与某个固定值比较——循环计数器是否小于允许的最大值？如果一开始条件为假，循环体就不执行；但若执行循环体，每次执行完毕后都回到循环的顶部，JavaScript 再计算条件，决定表达式的当前结果。随着循环的执行，循环计数器逐步递增，最终循环计数器的值就会超过条件语句中的值，使条件语句产生 false 值。此时，程序就完全跳出 for 循环。

最后一个语句是更新表达式，在每次循环的结尾处执行，即嵌套在 for 结构内的所有语句执行完毕后才执行。在这里，循环计数器是一个重要因素，若想让计数器的值在下一次循环时加 1(称为递增该值)，就可以将 JavaScript 操作符++放在变量名后面。这就是为什么在 for 循环中会经常见到 i++的原因。递增值也可以不是 1，而可以是任何值，甚至可以通过递减(i--)使循环计数器减少。

现在，运用这些知识，把典型的循环变量 i 用于正式的语法定义，常见的方法如下：

```
//incrementing loop counter
for (var i = minValue; i <= maxValue; i++)
{
   // statements
}
//decrementing loop counter
for (var i = maxValue; i >= minValue; i--)
{
   // statements
}
```


更多资源请访问稀酷客(www.ckook.com)

在上面的循环计数器中，变量 i 一开始就初始化为 minValue 的值，然后和 maxValue 进行比较。如果 i 小于等于 maxValue，则继续处理循环体，在循环结束时执行更新表达式。i 值每次加 1。这样，若将 i 初始化为 0，第一次执行循环时，i 变量在循环语句第一次执行的过程中始终是 0，下次循环开始时，i 变量变成 1。

在正式的语法定义中，for 语句的每个参数都是可选的。例如，在循环体内部执行的语句能根据处理过程中操作的数据控制循环计数器的值，所以 update 语句可能会干扰循环的运行。建议在熟悉 for 循环的每个部分之前使用全部的三个参数。例如，若忽略 condition 语句不编写退出循环的语句，脚本就可能进入死循环，这对用户没有好处。

21.4.1　使用循环计数器

i 循环计数器看起来微不足道，却是循环内部处理数据的一个强大工具。例如，下面是一个传统的 JavaScript 函数版本，它创建数组，将数组项初始化为 0：

```
// initialize array with n entries
function MakeArray(n)
{
   this.length = n;
   for (var i = 0; i < n; i++)
   {
      this[i] = 0;
   }
   return this;
}
```

循环计数器 i 初始化为值 0，因为 JavaScript 数组的第一个元素是 0。在条件语句中，只要计数器的值小于创建的数组项个数(n)，循环就继续执行。例如，如果 n 是 10，数组就初始化为包含 10 个元素 0~9。每次循环后，计数器加 1。在循环中执行的嵌套语句中，使用 i 变量的值替换赋值语句的索引值：

```
this[i] = 0;
```

第一次循环执行时，值的表达式等于：

```
this[0] = 0;
```

第二次循环执行时，值的表达式等于：

```
this[1] = 0;
```

如此循环，直到创建了所有数组项，并赋值 0 为止。

在程序清单 18-2 的 HTML 页面中，用户从 select 列表中选择区域办公室(这会让脚本查找该地区经理的姓名和销售额)。由于区域办公室名称存储在一个数组中，所以页面可能随时变化，并使脚本通过数组填充 select 元素的选项。这样，若改变区域办公室的排列，只需要改变办公室数组就可以了，不必改变 HTML。下面是区域办公室的数组定义，在页面加载时创建：

```
var aRegionalOffices = ["New York", "Chicago", "Houston", "Portland"];
```


更多资源请访问稀酷客(www.ckook.com)

HTML 表单内的脚本用来动态生成如下的 select 列表：

```
for (var i = 0; i < aRegionalOffices.length; i++)
{
   oSelect.options[i] = new Option(aRegionalOffices[i]);
}

// pre-select the first option
oSelect.options[0].selected = true;
```

注意，对于 for 循环的条件语句，一个要点是：JavaScript 从数组中取出 length 属性，用作循环计数器的上界。从代码的维护和格式的角度来看，该方法适用于固定的上界。若公司增加新的区域办公室，就把它增加到 database 数组中，相应的代码部分会自动改变，包括创建更长的弹出式菜单。

还要注意，条件语句的操作符是小于号(<)：数组的索引值从 0 开始，所以可使用的最大下标值比数组中实际的元素个数小 1。这是非常重要的，因为每次循环时，都把索引计数器变量(i)用作 regionalOffices 数组的索引，为每项数据读取字符串。还可以使用第一个选项的索引(0)将 selected 特性添加到第一个选项的定义中。

for 循环计数器的使用方式经常影响数据结构的设计方式，如用作数据库的二维数组(详见第 18 章)。在根据文档中包含的数据集编写 JavaScript 脚本时，要时刻想到循环计数器机制。

21.4.2　跳出循环

一旦满足了某个条件，一些循环结构就会执行操作，之后，就可能不再需要继续执行循环计数器范围内的其余循环。这种情况最常见的例子是遍历整个数组，查找满足某个条件的数组项。该条件测试建立为循环中的 if 结构。若满足条件，就跳出循环，脚本在主流程中执行其他更有意义的处理。要从循环中跳出，需要使用 break 语句。下面的语句说明了如何在 for 循环中使用 break 语句：

```
for (var i = 0; i < array.length; i++)
{
   if (array[i].property == magicValue)
   {
      // statements that act on entry array[i]
      break;
   }
}
```

break 语句告诉 JavaScript 跳出最里面的 for 循环(如果有嵌套的 for 循环)，脚本立即执行 for 语句结束括号后面的语句，变量 i 保持循环中断时的值，这样以后就可以在同样的脚本中使用该变量访问同一个数组项。

第 35 章使用了一个与此类似的结构，在程序清单 35-6 中讨论单选按钮时，演示了这个结构，其中有一个单选按钮集，其 value 特性包含屏幕大小，以像素为单位。函数使用 for 循环查找所选择的按钮，在跳出 for 循环后，使用这个选项的索引值警告用户。程序清单 21-3 显示了相关的函数。

更多资源请访问稀酷客(www.ckook.com)

程序清单 21-3　跳出 for 循环

JavaScript: jsb-21-03.js

```
function showMegapixels()
{
   var theForm = document.getElementById('sizesForm');
      if (theForm)
      {
         var sizeChoices = theForm.choices;
         for (var i = 0; i < sizeChoices.length; i++)
         {
            if (sizeChoices[i].checked)
            {
               break;
            }
         }
         alert("That image size requires " + sizeChoices[i].value +
              " megapixels.");
      }
}
```

在这个例子中，跳出 for 循环不仅仅是提高效率；循环计数器的值(在中止点被冻结)用于在循环外调用另一个属性。从版本 4 的浏览器开始，break 语句和新的 label 控制语句一起使用，可以获得许多附加功能，参见本章后面的讨论。

21.4.3　使用 continue 继续循环

for 循环的另一个用法是根据某条件中止嵌套语句的执行。也就是说，循环结构要为循环计数器的每个值执行循环体中的语句，但当计数器包含某个值时，不执行循环体中的语句。要完成这个功能，嵌套语句要包含 if 结构来检测计数器是否包含要跳出循环的值。计数器包含该值时，continue 命令告诉 JavaScript 立即跳出剩下的循环体，执行更新语句，并回到循环顶部。

为说明这个结构，创建下面的例子，当计数器变量是数字 13 时，跳过语句：

```
for (var i = 0; i <= 20; i++)
{
   if (i == 13)
   {
      continue;
   }
   // statements
}
```

在本例中，对除 13 外的所有 i 值执行循环的 statements 部分。continue 语句使执行流跳到循环结构的 i++部分，为下次循环递增 i 的值。在嵌套的 for 循环中，continue 语句会影响 if 结构求值为 false 的 for 循环。在版本 4 的浏览器中，continue 语句与新的 label 控制结构一起使用时，功能有所增强，详见本章后面的讨论。

更多资源请访问稀酷客(www.ckook.com)

21.5 while 循环

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

for 循环不是在 JavaScript 中构建的唯一循环结构，另一个语句以略有不同的格式创建循环，称为 while 语句。while 循环没有提供更改循环计数器的机制，而是假设脚本语句会满足一个条件，从而强制退出循环。

while 循环的基本语法是：

```
while (condition)
{
   statements
}
```

condition 表达式和 if 结构以及 for 循环的中间参数相同。若在执行到这个循环之前的代码中存在等于 true 的条件，就进入该循环。接着循环开始执行，反复修改条件的结果，直到条件为 false 为止。这时，循环退出，脚本继续执行循环括号外的语句。如果 while 循环中的语句不影响 condition 语句中测试的值，脚本就不会终止，程序进入死循环。

许多循环可以使用 for 结构，也可以使用 while 结构。事实上，程序清单 21-4 就是程序清单 21-3 中 for 循环的 while 循环版本。

程序清单 21-4 程序清单 21-3 的 while 循环版本

JavaScript: jsb-21-04.js

```
function showMegapixels()
{
   var theForm = document.getElementById(' sizesForm' );
       if (theForm)
       {
         var sizeChoices = theForm.choices;
         var i = 0;
         while (i < sizeChoices.length && !sizeChoices[i].checked)
            {
               i++;
            }
         }
         alert("That image size requires " + sizeChoices[i].value +
             " megapixels.");
       }
}
```

注意，若 while 循环的条件取决于循环计数器的值，脚本编写人员就应在创建 while 循环结构前初始化计数器，并在 while 循环内处理计数器的值。

如有必要，break 和 continue 控制语句可在 while 循环内部使用，与 for 循环一样。但因为这两种循环处理循环计数器和条件的方式不同，所以在这些循环中使用 break 和 continue 语句时要小心(多做一些测试)。

更多资源请访问稀酷客(www.ckook.com)

在脚本中使用哪种循环结构并没有硬性的规定。遍历的数据或对象在循环前已在脚本中时，也就是说，条件或循环计数器已经通过脚本中以前的语句初始化，通常使用 while 循环。但如果需要遍历对象的属性或数组项来提取脚本以后要使用的数据，则应使用 for 循环。在循环的计数器只是从一个值变化到另一个值时，也首选 for 循环。

对于 for 循环，注意脚本编写者在何处声明 i 变量。有些程序员喜欢将所有变量(如果知道初始值，则进行初始化)在脚本和函数的开始语句中声明，所以在脚本的这些地方有许多 var 语句。若函数中只有一个 for 循环，在 for 循环的初始化表达式部分声明和初始化循环计数器 i 不会出现任何错误。但如果在函数中使用多个重用计数器变量 i 的 for 循环(即循环的运行完全独立于其他循环)，就只需要在函数开始处对变量 i 声明一次，然后在每个 for 结构中将新的初始值赋给 i。

21.6 do-while 循环

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

JavaScript 还有一种循环结构，即 do-while 循环。这种结构的正式语法如下：

```
do
{
   statements
} while (condition)
```

do-while 循环和 while 循环的一个重要区别在于：在 do-while 循环中，结构中的语句在检测条件之前至少执行一次；而在 while 循环中，若一开始条件表达式为 false，结构中的语句就不会执行。所以，可以将 do-while 循环视为 while 循环，只是不管怎样，其语句至少执行一次。

若循环语句肯定至少执行一次，就使用 do-while 循环。如果条件在开始时可能不会满足，就使用 while 循环。在许多情况下，虽然只有 while 循环与旧浏览器兼容，但两种结构还是可以互换使用。

21.7 遍历属性(for-in)

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

JavaScript 的 for 循环有一个变体，称为 for-in 循环，可以提取当前浏览器内存中的对象属性名和属性值。语法结构如下：

```
for (varName in object)
{
   // statements
}
```

object 参数不是对象的字符串名，而是对象本身的引用。如果把对象名提供为不带引号的字符串，如 window 或 document，或通过 getElementById 提供对象，JavaScript 就传送对象引用。尽管 HTML 中的合法名称不区分大小写，但 JavaScript 中的合法名称区分大小写，所以即使在

更多资源请访问稀酷客(www.ckook.com)

HTML 代码中也应使用 camelCase 约定。使用 varName 变量就能够创建脚本，从给定对象中提取并显示各种属性。

程序清单 21-5 显示的页面包含一个实用函数，可在 JavaScript 增强页面的创作和调试阶段将它插入外部 JavaScript 文件中。本例检查当前的 window 对象，并将其属性显示在页面上。注意，Safari 1.0 不会显示 window 对象属性。

程序清单 21-5　属性检查函数

HTML: jsb-21-05.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Property Inspector Function</title>
    <script type="text/javascript" src="../jsb-global.js"></script>
    <script type="text/javascript" src="jsb-21-05.js"></script>
    <link rel="stylesheet" href="jsb-21-05.css" type="text/css">
  </head>
  <body>
    <h1>Property Inspector Function</h1>
    <p>Here are the properties of the current window:</p>
    <ul id="propertyList"></ul>
  </body>
</html>
```

JavaScript: jsb-21-05.js

```
// initialize when the page has loaded
addEvent(window, 'load', showProps);

// display all the properties of the object
function showProps()
{
   var oOutput = document.getElementById('propertyList');
      if (oOutput)
      {
         var newElem;
         var newText;
         for (var i in window)
         {
            newElem = document.createElement('li');
            newElem.className = 'objProperty';
            newText = document.createTextNode('window.' + i + ' = ' + window[i]);

            // insert the text node into the new paragraph
            newElem.appendChild(newText);

            // insert the completed paragraph into propertyList
            oOutput.appendChild(newElem);
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
}
```

Stylesheet: jsb-21-05.css

```
ul#propertyList li
{
   white-space: pre;
}
```

为了进行调试，可对函数稍加修改，在警告对话框或 textarea 元素中显示结果。在每个属性的后面添加\n 回车符，以更美观的格式显示结果。showProps()函数看起来可能很熟悉，因为它非常类似于 The Evaluator(详见第 4 章)中的属性检测例程。在配书光盘的第 48 章中，介绍了如何在页面中嵌入 The Evaluator 的功能，以便在调试脚本期间查看属性值。

21.8 with 语句

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

with 语句可以放在一些语句的前面告诉 JavaScript：脚本正在使用哪个对象。这样后面的语句就不必使用完全正式的引用，来访问同一对象的属性或者调用同一对象的方法。with 语句的正式语法定义如下：

```
with (object)
{
   statements
}
```

对象引用指向当前浏览器内存中使用的有效对象，第 16 章讨论 Math 对象时举了一个 with 语句的例子。通过把几个 Math 语句放在 with 结构中，脚本就可以调用 Math 对象的属性和方法，而不必在每个属性和方法的引用中加上 Math 对象。例子如下：

```
with (Math)
{
   var randInt = round(random() * 100);
}
```

这个例子使用 Math 对象的 round()和 random()方法，获得 0～100 之间的随机数。这段代码说明，使用 with 语句可以完整地表示 Math.round()和 Math.random()。

使用 with 结构的优点是使对象相关的语句更便于读取和理解。考虑下面的函数，它需要多次访问同一个对象(不过是不同的属性)：

```
function seeColor(form)
{
   var newColor = (form.colorsList.options[form.colorsList.selectedIndex]
   .text);
   return newColor;
}
```


PDG

更多资源请访问稀酷客(www.ckook.com)

使用 with 结构就可以使长语句变短：

```
function seeColor(form)
{
   with (form.colorsList)
   {
      newColor = (options[selectedIndex].text);
   }
   return newColor;
}
```

JavaScript 在 with 语句中遇到不认识的标识符时，将试图从指定为参数的对象和这个陌生的标识符中建立引用，但 with 语句不能嵌套。例如，在前面的例子中，不能将 with(colorsList)嵌套在 with(form)语句中，JavaScript 不会从两个对象名中建立引用。

with 语句看起来很聪明，但使用它时要小心，它会给脚本带来性能问题(因为 JavaScript 解释器必须手工生成引用)。如果只是偶尔使用这个结构，可能不会注意到这个缺陷，但如果在一个重复多次的循环内部使用这个结构，会极大地影响处理速度。另外，一些人认为，with 语句会给代码带来歧义。

21.9　标签语句

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

脚本查找的最后一个条件位于多个嵌套中时，多个嵌套循环的编程就非常困难。问题是，break 或 continue 语句只能用于最近的一级循环，即使跳出了内部循环，外部循环也会继续执行。如果只希望在满足条件后终止函数，使用简单的 return 语句即可，就像其他语言中的退出命令一样。但如果条件满足后，需要继续在函数内部处理，就需要使用现代浏览器提供的 JavaScript 工具，为语句块指定标签。这样就可以改变 break 和 continue 语句的范围，使它们用于带标签的语句块，而不是包含语句的循环。

标签可以是任何标识符(也就是说，标签的名称以字母开头，不包含空格或除了下划线之外的奇怪符号)，后跟冒号以及执行语句逻辑块，如 if...then 结构或循环结构。正式语法如下：

```
labelID:
   statements
```

对于用于标签块的 break 或 continue 语句，标签作为参数，加在每个语句的后面。

```
break labelID;
continue labelID;
```

注意：

如果以前使用其他结构化程度更高的编程语言，比如 C++或 Java，现在转而使用 JavaScript，就会担心使用标签代码可能会创建难以管理的代码。这种担心可能植根于一些语言中(如 BASIC)的 goto 语句，在现代的结构化语言中不推荐使用它。但与其他语言中声名狼藉的 goto 语句相比，JavaScript 中的标签有更多的限制：只能使用 break 语句跳到嵌套了代码的标签上。所以，

更多资源请访问稀酷客(www.ckook.com)

尽管不鼓励大量使用标签，但这里的标签肯定与 goto 语句不同。

为了说明如何在适当情况下使用标签，程序清单 21-6 包括相同嵌套循环结构的两个版本。每个版本都会遍历两个不同的索引变量，直到变量值等于循环外部设置的目标值。之后，整个嵌套循环结构终止，继续处理后面的语句。为了弄清循环执行过程，脚本向文本区域输出了中间和最终结果。

在没有标签的版本中，达到目标值时，只使用简单的 break 语句，终止内部循环，但外部循环继续执行，直到整个循环结构结束，这会浪费大量处理资源。此外，由于达到目标值后，循环还要从头到尾再执行几次，因此记数变量的值将超过其最大值。

但在有标签的版本中，一旦达到目标值，就中止内部循环，跳到标签指定的外部循环。这样需要执行的代码行会少许多，循环记数变量也等于目标值。试着将程序清单 21-6 中的 break 改为 continue 语句，然后仔细分析 Result 文本区域中的结果，查看一下这两个版本的执行情况。

程序清单 21-6　标签语句

HTML: jsb-21-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Breaking Out of Nested Labeled Loops</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-21-06.js"></script>
   </head>
   <body>
      <h1>Breaking Out of Nested Labeled Loops</h1>
      <p>Look in the Results field for traces of these button scripts:</p>
      <form id="theForm" action="validate.php" method="get">
        <p><input id="withoutLabelButton" type="button" value="Execute WITHOUT↵
              Label"></p>
        <p><input id="withLabelButton" type="button" value="Execute WITH↵
              Label"></p>
        <p>
           <label for="output">Results:</label>
           <textarea id="output" rows="43" cols="60"></textarea>
        </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-21-06.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // locate output field
```


更多资源请访问稀酷客(www.ckook.com)

```
   out = document.getElementById('output');
      // if found...
      if (out)
      {
         // add an event to the button button labeled Execute WITHOUT Label
         addEvent(document.getElementById('withoutLabelButton'), 'click', run1);

         // add an event to the button button labeled Execute WITH Label
         addEvent(document.getElementById('withLabelButton'), 'click', run2);
      }
}
var out;   // global variable for output field
var targetA = 2;
var targetB = 2;
var range = 5;

function run1()
{
   out.value = "Running WITHOUT labeled break\n";
   for (var i = 0; i <= range; i++)
   {
      out.value += "Outer loop #" + i + "\n";
      for (var j = 0; j <= range; j++)
      {
         out.value += " Inner loop #" + j + "\n";
         if (i == targetA && j == targetB)
         {
            out.value += "**BREAKING OUT OF INNER LOOP**\n";
            break;
         }
      }
   }
   out.value += "After looping, i = " + i + ", j = " + j + "\n";
}

function run2()
{
   out.value = "Running WITH labeled break\n";
   outerLoop:
   for (var i = 0; i <= range; i++)
   {
      out.value += "Outer loop #" + i + "\n";
      innerLoop:
      for (var j = 0; j <= range; j++)
      {
         out.value += " Inner loop #" + j + "\n";
         if (i == targetA && j == targetB)
         {
            out.value += "**BREAKING OUT OF OUTER LOOP**\n";
            break outerLoop;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
    }
    out.value += "After looping, i = " + i + ", j = " + j + "\n";
}
```

21.10　异常处理

在 JavaScript 中，异常处理是一个较新的主题。异常处理在 ECMA-262 第 3 版中正式定义，IE5 实现了其部分官方机制，在 IE6、NN6、Mozilla、Firefox、Camino、Safari、Opera 和 Chrome 中实现了其完整机制。

21.10.1　异常及错误

只要编写过脚本，就曾遇到过 JavaScript 错误，可能是代码中的语法错误，或者所谓的运行错误(脚本处理信息时发生的错误)。理想情况下，程序知道何时产生错误，并尽可能处理它们，这种自我诊断功能可以防止数据丢失，并防止用户看到错误信息。第 27 章介绍了 onerror 事件处理程序和 window.onerror 属性，它们使脚本具有某种控制运行错误的能力。这种事件驱动机制作用于全局层次(也就是 window 对象)，可以处理页面中发生的任何错误。这个事件处理程序主要用来防止向用户显示错误信息，但它和程序员认为的异常处理有很大区别。

在英语中，exception(异常)表示不平常的或不正常的事物，这个定义和 error(错误)区别很大。但在编程语言中，这两个词可以交替使用，其差别主要取决于用户的视角。

例如，用户在页面上的两个文本域中输入数值，一个简单的脚本将这两个数相乘，结果显示在第三个文本域内。若脚本没有验证输入的数据，JavaScript 就尝试将输入文本框的任何数据相乘；如果用户输入两个数值，即使两个文本输入域的 value 属性是字符串，JavaScript 也可以识别，并将含有数值的字符串转换成用于乘法的数值类型。两个数值进行乘法计算后，结果显示在结果域中。

但是，如果用户在其中一个文本框中输入了一个字母，会怎样呢？由于脚本中没有输入验证，JavaScript 就采用一种固定方式加以响应：乘法操作的结果是 NaN(不是数值)常量。没有经验的用户不知道 NaN 的意思，但显然发生了错误。这时，用户可能会对计算机或自己抱怨不止，而正确的响应是通知 JavaScript 开发人员。

但从程序员的角度看，把脚本设计成由从来不犯输入错误的用户来运行，当然不是很好的编程方法。用户会犯输入错误，而程序员没有验证输入，此时输入对于程序的操作规则来说就是异常。必须在程序中增加额外的代码来处理异常情况，以免用户对莫名其妙的输出感到疑惑，还有助于用户修正输入，以生成正确的结果。这些额外的代码用以处理意外的输入和错误输入，使脚本更加友好健壮。

在用于处理异常的词汇中，JavaScript 和 W3C 文档对象模型混合使用异常和错误的许多术语。如后面所述，异常会创建 error 对象，该对象包含异常信息，所以可以说异常和错误是相同的。

更多资源请访问稀酷客(www.ckook.com)

21.10.2　异常机制

JavaScript(或者任何编程环境)的初学者一开始可能无法想象出，这些异常元素如何在浏览器中运行，但较容易理解页面如何加载、如何创建对象模型以及事件处理程序(在 W3C DOM 术语中就是监听器)如何运行脚本函数。许多动作似乎都在后台运行，例如在事件处理函数运行时，每个事件动作会自动生成 event 对象(参见第 32 章)，这些 event 对象似乎存在于“某个地方”，才能得到事件的细节。函数处理完毕后，event 对象就会消失，不留下任何痕迹。

具有异常处理功能的浏览器有许多在后台处理的“东西”，它们在需要时为脚本服务。用户肯定看到过至少一个脚本错误的细节，也就见过内置于浏览器的某种异常处理机制。脚本发生错误时，浏览器在内存中创建一个 error 对象，其属性包含错误的细节。精确的细节(后面讨论)因浏览器而异，但显示出来的错误细节都是浏览器处理异常/错误的默认内容。随着浏览器技术的日益成熟，浏览器厂商大大降低了脚本错误的侵扰。例如，在 NN4+中，错误出现在一个单独的 JavaScript 控制台窗口中(在 NN4 中需要将 javascript: 输入 Location 域，才能调用该窗口；在 NN6+和基于 Mozilla 的浏览器中，包括 Firefox 和 Camino，可以用 Tools 菜单直接打开该窗口)。在 Windows 的 IE4+中，左下角的图标变成警告图标时，双击该图标，会在状态栏中显示更多的错误信息。MacIE 用户可以关闭脚本错误警告。Safari 1.0 不会显示任何脚本错误，但 JavaScript 控制台在 1.3 版本中添加了此项功能。

但真正的异常处理不止显示错误信息，还让脚本通过统一的方式预防异常的发生。理想情况下，该机制保证用同一机制筛选所有的运行错误，简化了脚本的异常处理。这种机制还可以让自己的代码按照统一的方式生成错误，这样脚本的其他部分就可以自动地正确处理错误。也就是说，可将异常处理机制作为一种“秘密通道”，使脚本的一部分与其他部分进行通信。

JavaScript 异常处理机制有两组程序执行语句，第一组是 try-catch-finally 语句结构；第二组是单个的 throw 语句。

21.11　使用 try-catch-finally 结构

try-catch-finally 语句组的作用是：为可能遇到运行错误的脚本语句提供可控环境，若发生异常，脚本可以不触发其他浏览器错误机制，直接处理异常。三个语句后面都有一个代码块，语法如下：

```
try
{
   statements to run
}

catch (errorInfo)
{
   statements to run if exception occurs in try block
}
finally
{
   statements to run whether or not an exception occurred [optional]
}
```


更多资源请访问稀酷客(www.ckook.com)

每个 try 块都必须和 catch、finally 块在同一个嵌套层上，中间不能有其他语句。例如，函数有一个一级 try-catch 结构，如下：

```
function myFunc()
{
   try
   {
      // statements
   }
   catch (e)
   {
      // statements
   }
}
```

但如果有其他 try 块嵌套在更深一级，也必须在这一级包含对应的 catch 或 finally 块，如下：

```
function myFunc()
{
   try
   {
      // statements
      try
      {
         // statements
      }
      catch (e)
      {
         // statements
      }
   }
   catch (e)
   {
      // statements
   }
}
```

try 块内的语句包括可能产生运行错误的语句，错误原因一般是用户输入错误、页面组件加载失败或其他异常。catch 块防止错误显示在浏览器的常规脚本错误报告系统中(例如 Safaril.3+、NN6+和基于 Mozilla 的浏览器的 JavaScript 控制台)。

这类异常处理的一个重要术语是 throw。按照约定，若一个操作或调用的方法激活了异常，就称为“抛出异常”。例如，如果一个脚本语句试图调用 string 对象的方法，但该方法并不存在(可能是因为输入的方法名有误)，JavaScript 就会抛出异常。异常一般有名称，该名称有时(并不总是)会透露异常的重要信息。在上述方法名错误的例子中，异常名为 TypeError(还有更多的异常和错误交叉使用的例子)。

现代浏览器中的 JavaScript 语言并不是唯一可以抛出异常的实体，W3C DOM 也为 DOM 对象定义了一类异常。例如，根据 Level 2 规范，appendChild()方法(参见第 26 章)可以抛出(用 W3C 的术语是“引发”)表 21-2 中的 3 个异常。

更多资源请访问稀酷客(www.ckook.com)

表 21-2 3 个异常

异常名	抛出异常的时机
HIERARCHY_REQUEST_ERR	当前节点的类型不允许有 newChild 节点类型的子节点，或者附加的节点是这个节点的祖先节点
WRONG_DOCUMENT_ERR	创建 newChild 的文档不是创建当前节点的文档
NO_MODIFICATION_ALLOWED_ERR	当前节点是只读的

因为 appendChild()方法可以抛出异常，所以调用该方法的 JavaScript 语句应该嵌套在 try 块内。若抛出异常，脚本执行流马上跳到与 try 相关的 catch 或 finally 块中，之后并不返回 try 块。

catch 块的行为比较特殊，其格式在某个方面类似于函数，因为 catch 关键字后跟一对括号和一个任意变量，该变量的值是 error 对象的引用，当发生异常时，浏览器就给该 error 对象的属性赋值。error 对象的一个属性是错误名称，所以 catch 块中的代码能够检查错误名称，并包含一些分支代码，来处理捕获的各种错误。

为了理解这种代码结构，下面设计一个假想的通用函数，其作用是创建一个新元素，并添加在其他节点的后面。所创建的元素类型和父节点的引用都作为参数传递到函数中。注意，这个函数可能因为传递不正确的参数值而误用，所以包含额外的错误处理代码，来处理两个 DOM 方法 createElement()和 appendChild()抛出的异常。该函数见程序清单 21-7。

程序清单 21-7 一个假想的 try-catch 程序

```
// generic appender
function attachToEnd(theNode, newTag)
{
   try
   {
      var newElem = document.createElement(newTag);
      theNode.appendChild(newElem);
   }
   catch (e)
   {
      switch (e.name)
      {
         case "INVALID_CHARACTER_ERR" :
            // statements to handle this createElement() error
            break;
         case "HIERARCHY_REQUEST_ERR" :
            // statements to handle this appendChild() error
            break;
         case "WRONG_DOCUMENT_ERR" :
            // statements to handle this appendChild() error
            break;
         case "NO_MODIFICATION_ALLOWED_ERR" :
            // statements to handle this appendChild() error
            break;
         default:
```


更多资源请访问稀酷客(www.ckook.com)

```
                // statements to handle any other error
            }
            return false;
        }
        return true;
    }
```

只有 try 块中的一个语句抛出异常，才执行程序清单 21-7 中的 catch 块。异常可以是 catch 块中的 4 个指定错误之一，也可以是 try 块中的语法或其他错误，所以要使用最后的 default 来处理意外的错误。脚本编写者不仅要预测错误，还要提供处理异常的方法，这些方法可能通过明显的警告对话框来显示错误，也可能执行一些自我修复操作。例如，如果 createElement()抛出了无效字符错误，脚本就试图修复传递给 attachToEnd()函数的数据，重新调用该方法，原样传递 theNode 值，将修复值传给 newTag。如果修复成功，try 块就会顺利执行，不发生错误，用户不会意识到已经解决了一个危险的问题。这就是异常处理的目标：在出现异常情况时，不要让用户感到困惑迷茫。

finally 块包含的代码总是在 try 块执行完毕后执行，而不管 try 是否抛出异常。与 catch 块不同，finally 块不将 error 对象作为参数，所以 finally 块执行时并不知道 try 块中发生了什么问题。若 try 块的后面既有 catch 块又有 finally 块，执行路径就取决于是否抛出了异常。若没有抛出异常，执行完 try 块的最后一个语句后，就执行 finally 块；但若 try 块抛出异常，程序就先运行 catch 块；catch 内的处理全部结束后，执行 finally 块。在程序员完全控制资源的开发环境中，如内存分配，finally 块用于删除 try 块生成的临时项，不管在 try 中是否抛出了异常。现在，JavaScript 的自动内存管理系统减少了这种维护需求，但要注意，try-catch-finally 中的 finally 块可能会执行。

现实的异常

程序清单 21-7 中的例子有点理想化，它假设浏览器能报告在正式规范中定义的每个 W3C DOM 异常。可惜，即使是最新的浏览器，在异常报告方面，也未完全遵循 DOM。在实际的 DOM 异常和异常发生时 error 对象报告的内容之间，大多数浏览器都增加了额外的错误命名约定和层。

这些差异使跨浏览器的异常处理更加困难，即使是最简单的错误，两个主要的浏览器品牌 (IE 和 Mozilla) 的报告都不同于 W3C DOM 规范。在浏览器的异常报告标准统一之前，脚本编写者最通畅的开发之路是只为一个浏览器平台编写代码，如 Windows 的 IE 或 Mac 的 Safari。

但是，异常处理的一个方面仍可以用在所有现代浏览器中：利用 try-catch 结构抛出自己的异常，这在高级编程环境中比较常见。

21.12　抛出异常

最后一个异常处理关键字是 throw，它可将异常处理工具用于过程管理，例如数据输入验证。在 try 块内的任何一点，都可以手工抛出异常，让相关的 catch 块捕获它，具体的异常细节取决于用户。

更多资源请访问稀酷客(www.ckook.com)

throw 语句的语法如下：

```
throw value;
```

抛出的异常可以是任意类型，但实践证明，该值最好是 error 对象(详见本章后面)，所抛出的值会作为参数赋给 catch 块。在下面的两个例子中，第一个例子的值是字符串消息；第二个例子的值是 error 对象类型。

程序清单 21-8 有一个输入文本框，允许输入 1～5 之间的值，单击一个按钮，查找数组中的相应字母，并在第二个文本框中显示它。查找脚本有两个简单的数据验证例程，可以确保输入的是数值，并在允许范围之内。

这里的错误检查是用脚本手工完成的。只要发生错误，throw 语句就强制执行流跳到 catch 块，catch 块把输入的 string 参数赋给变量 e。这里假设传递的消息是警告对话框的文本。单个 catch 块不仅要处理这两个错误(以后还可能添加其他错误)，而且其变量作用域与函数相同，这样就可以使用输入文本框的引用，在错误发生时使输入文本框具有焦点，并选择输入文本。

程序清单 21-8　抛出字符串异常

HTML: jsb-21-08.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Throwing a String Exception</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-21-08.js"></script>
   </head>
   <body>
      <h1>Throwing a String Exception</h1>
      <form id="theForm" action="validate.php" method="get">
         <p>
            <label for="visitorInput">Enter a number from 1 to 5:</label>
            <input type="text" id="visitorInput" size="5">
            <input type="button" id="getLetterButton" value="Get Letter">
            <label for="output">Matching Letter is:</label>
            <input type="text" id="output" size="5">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-21-08.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // add an event to the Get Letter button
```


更多资源请访问稀酷客(www.ckook.com)

```
    addEvent(document.getElementById('getLetterButton'), 'click', getLetter);
}

var letters = new Array("A","B","C","D","E");

function getLetter()
{
    try
    {
        var inputField = document.getElementById('visitorInput');
        var inp = parseInt(inputField.value, 10);
        var outputField = document.getElementById('output');

            if (isNaN(inp))
            {
                throw "Entry was not a number.";
            }

            if (inp < 1 || inp > 5)
            {
                throw "Enter only 1 through 5.";
            }

        outputField.value = letters[inp - 1];
    }
    catch (e)
    {
        alert(e);
        outputField.value = "";
        inputField.focus();
        inputField.select();
    }
}
```

程序清单 21-8 的缺陷在于，如果在 try 块内抛出其他异常，传递给 catch 块的值将是 error 对象，而不是字符串。给用户显示警告对话框也没有任何意义。所以最好在 throw-catch 结构中统一传递 error 对象。

程序清单 21-9 是程序清单 21-8 的修正版本，说明如何创建 error 对象，并将它通过 throw 语句传递给 catch 块。HTML 代码与程序清单 21-8 相同，这里不再重复。

程序清单 21-9　抛出一个 error 对象异常

JavaScript: jsb-21-09.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
    // add an event to the Get Letter button
    addEvent(document.getElementById('getLetterButton'), 'click', getLetter);
}
```


更多资源请访问稀酷客(www.ckook.com)

```
var letters = new Array("A","B","C","D","E");

function getErrorObj(msg)
{
   var err = new Error(msg);
   return err;
}

function getLetter()
{
   try
   {
      var inputField = document.getElementById('visitorInput');
      var inp = parseInt(inputField.value, 10);
      var outputField = document.getElementById('output');

      if (isNaN(inp))
      {
         throw getErrorObj("Entry was not a number.");
      }
      if (inp < 1 || inp > 5)
      {
         throw getErrorObj("Enter only 1 through 5.");
      }
      outputField.value = letters[inp - 1];
   }
   catch (e)
   {
      alert(e.message);
      outputField.value = "";
      inputField.focus();
      inputField.select();
   }
}
```

唯一的区别是现在 catch 块读取传入的 error 对象的 message 属性，这就是说，如果在 try 块内抛出其他异常，浏览器生成的信息就显示在警告对话框中。

但事实上，工作尚未完成。在各种可能的情况下，若没有抛出浏览器生成的异常，警告对话框中的信息就对用户没有任何用处，因为其错误信息可能是语法或类型错误，正如在一些自己喜欢的操作系统中可能显示没有任何意义的信息一样。较好的设计是将 catch 块进行分支处理，代码“蓄意”产生的异常由警告对话框处理，其他类型的异常则执行不同的处理。为此，需要控制 error 对象的 name 属性，这样 catch 块就可以单独处理自定义消息。

在程序清单 21-10 中，getErrorObj()函数给新建 error 对象的 name 属性添加了一个自定义值。可以指定任何名称，但要避免由 JavaScript 或 DOM 使用的异常名。即使不知道这些异常名，也应该为错误指定唯一的、合适的名称。在 catch 块中，switch 结构要处理两类不同的错误。在这个简化的例子中，除了 try 块明确抛出的错误之外，唯一可能的异常是在下载页面过程中出现的数据错误，所以该错误的处理只是让用户重新加载页面。但要点是，程序可以有任意多种自定义错误和系统错误，它们应在相应的单个 catch 块中分别处理。HTML 代码与程序

更多资源请访问稀酷客(www.ckook.com)

清单 21-8 相同，这里不再重复。

程序清单 21-10　自定义对象异常

JavaScript: jsb-21-10.js

```
// initialize when the page has loaded
addEvent(window, 'load', initialize);

function initialize()
{
   // add an event to the Get Letter button
   addEvent(document.getElementById('getLetterButton'), 'click', getLetter);
}

var letters = new Array("A","B","C","D","E");

function getErrorObj(msg)
{
   var err = new Error(msg);
   err.name = "MY_ERROR";
   return err;
}

function getLetter()
{
   try
   {
      var inputField = document.getElementById('visitorInput');
      var inp = parseInt(inputField.value, 10);
      var outputField = document.getElementById('output');

      if (isNaN(inp))
      {
         throw getErrorObj("Entry was not a number.");
      }

      if (inp < 1 || inp > 5)
      {
         throw getErrorObj("Enter only 1 through 5.");
      }

      outputField.value = letters[inp - 1];
   }
   catch (e)
   {
      switch (e.name)
      {
         case 'MY_ERROR':
            alert(e.message);
            outputField.value = "";
            inputField.focus();
            inputField.select();
```


更多资源请访问稀酷客(www.ckook.com)

```
                break;

            default :
                alert('Reload the page and try again.');
        }
    }
}
```

如果想看看程序清单 21-10 中的另一个分支，可将光盘中的程序文件复制到硬盘上，更改 try 块的最后一行，删除数组名中的一个字符：

```
outputField.value = letter[inp - 1];
```

这可以模仿页面的错误加载。若输入允许值，会出现重载警告，而不是实际的 error 对象消息：letter is undefined(字母未定义)。

最后将详细介绍 error 对象。

21.13　error 对象

属　　性	方　　法
errorObject.prototype	errorObject.toString()
errorObject.constructor	
errorObject.description	
errorObject.Filename	
errorObject.lineNumber	
errorObject.message	
errorObject.name	
errorObject.number	

21.13.1　语法

创建 error 对象：

```
var myError = new Error("message");
var myError = Error("message");
```

访问静态 Error 对象的属性：

```
Error.property
```

访问 error 对象的方法和属性：

```
errorObject.property | method([parameters])
```

兼容性：WinIE5+，MacIE−，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

21.13.2　关于 error 对象

只要抛出异常，或者调用任何一种创建 error 对象的构造函数，就会创建 error 对象实例。这个 error 对象实例的属性包括错误的性质信息，以便 catch 块检查并处理相应的错误。

IE5 在 ECMA-262 正式引入 error 对象之前就实现了 error 对象，IE5 版本的 error 对象有自己的属性集，它们并不是 ECMA 标准的一部分。IE5.5+仍支持这些属性，还包括 ECMA 属性。另一方面，NN6 一开始就拥有 ECMA 属性，还增加了两个专用属性，该浏览器在脚本错误报告中使用这些新增的属性。然而，对于跨浏览器的开发人员而言，error 对象没有一个所有浏览器都支持的属性，但是 IE5.5+和其他浏览器有两个公用属性(name 和 message)。

如本章所述，只要使用 throw 语句执行错误控制，就会创建 error 对象。

21.13.3　属性

constructor

参见第 15 章。

description

值：字符串，　　读/写

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

description 属性是一个说明字符串，提供了错误的详细内容。对于浏览器抛出的错误，这个说明与 IE 脚本错误对话框中的文本相同。尽管浏览器继续支持该属性，但应优先使用 message 属性。

相关主题：message 属性。

fileName，lineNumber

值：字符串，　　读/写

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

NN6 浏览器使用 error 对象的 fileName 和 lineNumber 属性处理内部的脚本错误，这些值列在 JavaScript 控制台中，作为错误消息的一部分。fileName 是抛出错误的文档的 URL；lineNumber 是抛出异常的语句的源代码行号。这些属性可用于 JavaScript，所以，若这些信息对应用程序有用，就可以使用这些信息处理错误。

请参见第 27 章关于 window.error 属性的讨论，以便了解如何使用这些信息报告错误。

相关主题：window.error 属性。

message

值：字符串，　　读/写

兼容性：WinIE5.5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

message 属性是一个描述字符串，提供了错误的细节。对于浏览器抛出的错误，这个消息与 IE 脚本错误对话框中的文本相同，也与 Mozilla 的 JavaScript 控制台中的文本相同。一般来说，这些信息对脚本开发人员比对用户更有用。但对于给定的错误，这种消息没有统一的标准，所以，在 catch 块中使用这些消息内容作为分支方式，来处理特殊类型的错误是十分危险的。

更多资源请访问稀酷客(www.ckook.com)

若为一个浏览器平台开发程序，可以使用这种方法，但不能确保某个异常的消息文本在将来的浏览器版本中不变。

如果希望向用户显示这些消息，对于由代码明确抛出的错误的自定义消息，可以使用对用户友好的语言。这些用法请参见程序清单 21-8 一直到程序清单 21-10。

相关主题：description 属性。

name

值：字符串，　　读/写

兼容性：WinIE5.5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

name 属性一般包括一个词来表示抛出的错误类型，最普通的错误(以及用 new Error()构造函数创建的错误)称为 Error。但 JavaScript 错误有几个种类：EvalError、RangeError、ReferenceError、SyntaxError、TypeError 和 URIError。其中一些错误类型不一定提供给脚本编写者，它们主要用于 JavaScript 引擎的内部工作，但一些浏览器会提供它们。对于脚本错误，这个属性的名称存在一些差异。

JavaScript 用于 W3C DOM 兼容浏览器时，一些 DOM 异常类型用 name 属性返回。但浏览器常常给这个属性插入自己的错误类型，这种情况很常见，所以浏览器品牌很少保持一致。

对于代码明确抛出的自定义异常，可以指定合适的名称。如程序清单 21-9 和程序清单 21-10 所示，这些信息有助于 catch 块处理多种类型的错误。

相关主题：message 属性。

number

值：数值，　　读/写

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE5+为每种错误描述和消息都分配了唯一的数值，但 number 属性值必须进行一些处理，才能获得有意义的错误描述信息。下面的例子说明了如何对错误号应用二进制运算，得到有意义的结果：

```
var errNum = errorObj.number & 0xFFFF;
```

相关主题：description 属性。

21.13.4　方法

toString()

返回值：字符串(见正文)

兼容性：WinIE5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

error 对象的 toString()方法返回错误的字符串描述信息。但在 IE5+中，此方法返回同一个 error 对象的引用；在基于 Mozilla 的浏览器中，此方法返回 message 属性字符串，其前面是字符串“Error:”(冒号后有一个空格)。通常，如果要得到 error 对象的可读表达式，可以读取其 message(在 IE5+中是 description)属性。

相关主题：message 属性。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第22章

JavaScript 操作符

本章包含哪些内容？
理解操作符的类别
理解操作符在脚本语句中的作用
操作符的优先级

JavaScript 具有丰富的操作符。操作符是表达式中的单词或符号，用来对一个或两个值进行操作，从而得到另一个值。操作符处理的值称为操作数。表达式可以包括一个操作数和一个操作符(称为单目运算)，或者由一个操作符分开的两个操作数(称为双目运算)，例如 a+b。同一符号可表示多种操作符。将这些符号组合排序，就可以区分其功能。

注意：

大部分 JavaScript 操作符一开始就存在于该语言中。但随着该语言的成熟和发展，加入了许多新的操作符。本章的兼容性列表指出了操作符的类别。若某类别中有特殊的操作符，文中会提及。现代浏览器支持所有 JavaScript 操作符。

22.1 操作符的类别

为帮助理解 JavaScript 操作符，这里将它们分为 7 类。第二组使用了一个非传统的名称，但这个名称能正确表示它在该语言中的作用。表 22-1 列出了操作符的类型。

包含一个操作符的表达式总是计算为某种类型的值，这个值总是一个操作的结果。有时，操作符会改变某个操作数的值；有时，结果是一个新值。下面简单的表达式：

```
5 + 5
```

表示两个整数操作数用加号操作符连接起来，这个表达式的值为 10。操作符(+)提供了命令，使 JavaScript 计算脚本中的每个表达式。

更多资源请访问稀酷客(www.ckook.com)

表 22-1　JavaScript 操作符的类别

类　　型	功　　能
比较操作符	比较两个操作数的值，结果为 true 或 false(主要在条件语句 if...else 和 for 循环结构中使用) == != === !== > >= < <=
结合操作符	结合两个操作数，生成一个值，作为算术或其他操作的结果 + - * / % ++ -- +val -val
赋值操作符	将右操作数的表达式值赋给左边的变量。有时需要稍加修改，这由操作符符号来确定 = += -= *= /= %= <<= >= >>= >>>= &= |= ^= []
布尔操作符	对一个或两个布尔操作数执行布尔计算 && || !
位操作符	对两个操作数的二进制表示(以 2 为基)执行算术或移位操作 & | ^ ~ << >> >>>
对象操作符	在调用对象及其属性或方法之前，帮助脚本检查该对象的继承权和能力 . [] () delete in instanceOf new this
其他操作符	一些具有特殊作用的操作符 , ? : typeof void

对两个操作数作相等比较时，操作数常常看起来大不相同。JavaScript 不关心操作数的形式，只关心它们求值的方式。两个看似完全不同的值在计算时可能相等。下面的表达式比较两个值是否相等：

```
fred == 25
```

若变量 fred 的值等于 25，该表达式的值则为 true。

22.2　比较操作符

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在 JavaScript 中，只要比较两个值，结果就是 Boolean 值 true 或 false。根据要对两个操作数执行的测试类型，可从大量的比较操作符中选择一个。表 22-2 列出了所有的比较操作符。

表 22-2　JavaScript 比较操作符

语　　法	名　　称	操作数类型	结　　果
==	等于	所有	Boolean
!=	不等	所有	Boolean
===	严格相等	所有	Boolean (IE4+、NN4+、Moz+、W3C)
!==	严格不等	所有	Boolean (IE4+、NN4+、Moz+、W3C)
>	大于	所有	Boolean

更多资源请访问稀酷客(www.ckook.com)

(续表)

语　法	名　称	操作数类型	结　果
>=	大于或等于	所有	Boolean
<	小于	所有	Boolean
<=	小于或等于	所有	Boolean

对于数值型，比较操作的结果和在高中代数课中学到的相同。参看下面的例子，有些例子的结果可能不是很明显。

```
10 == 10                 // true
10 == 10.0               // true
9 != 10                  // true
9 > 10                   // false
9.99 <= 9.98             // false
```

字符串也可以进行此类比较：

```
"Fred" == "Fred"         // true
"Fred" == "fred"         // false
"Fred" > "fred"          // false
"Fran" < "Fred"          // true
```

为了计算比较字符串的结果，JavaScript 将字符串中的每个字符都转换成相应的 ASCII 值，从左操作数的第一个字母开始，与右操作数的相应字母进行比较。因为大写字母的 ASCII 值小于其小写字母，所以大写字母小于相应的小写字母。JavaScript 严格区分大小写。

用于比较的值还可以来自对象属性，或给事件处理程序调用的函数以及其他函数传递的值。在验证数据输入的有效性时，最常见的字符串比较是查看字符串是否包含内容：

```
form.entry.value != ""       // true if something is in the field
```

22.3 不同数据类型的相等比较

在 JavaScript 1.2 以前的版本中(旧浏览器)，当脚本比较包含数值和实数(例如，"123" ==123 或 "123" !=123)的字符串时，JavaScript 希望比较相同类型的值。所以，它在内部进行了一些不影响原始值(例如，值存储在变量中)的数据类型转换。但这种情况较为复杂，因为需要处理其他数据类型，例如对象。JavaScript 1.2 以前的比较规则如表 22-3 所示。

表 22-3 JavaScript 1.0 和 1.1 的相等比较

操作数 A	操作数 B	比较操作的内部处理
对象引用	对象引用	比较对象引用
任何数据类型	Null	将非空值转换为其对象类型，并与空值进行比较
对象引用	字符串	将对象转换为字符串，并比较字符串
字符串	数值	将字符串转换为数值，并比较数值

更多资源请访问稀酷客(www.ckook.com)

为了理解表 22-3 中的相等比较逻辑，需要从脚本编写者的角度进行思考，因为必须弄清楚在进行相等比较时可能或不可能进行的数据类型转换(即使数据本身不进行转换)。最好根据需要在比较语句中提供正确的转换命令，以保证比较需要比较的内容，例如，比较两个值的字符串版本，或比较两个值的数值版本，而不是让 JavaScript 执行转换。

从数值到字符串的向后兼容转换，需要将一个空字符串与该数值连接起来：

```
var a = "09";
var b = 9;
a == "" + b; // result: false, because "09" does not equal "9"
```

将字符串转换成数值有许多可能性，最简单的是从数值字符串中去掉 0：

```
var a = "09";
var b = 9;
a-0 == b; // result: true because number 9 equals number 9
```

还可以用 parseInt()和 parseFloat()函数将字符串转换成数值：

```
var a = "09";
var b = 9;
parseInt(a, 10) == b; // result: true because number 9 equals number 9
```

当然，另一个方法是，假设用户有支持 JavaScript 1.2+的现代 Web 浏览器。为消除 JavaScript 内部相等转换的模糊性，JavaScript 1.2 增加了两个操作符，将相等比较强制为严格比较。严格相等(===)和严格不相等(！==)操作符同时对数据类型和值进行比较。只有两个操作数的数据类型(如都是数值型)和值都相同，===操作符才返回真。所以，数值不会自动等于其字符串版本，数据和对象类型必须匹配，才能比较值。

JavaScript 1.2+还提供了方便的全局函数 String()和 Number()，来转换字符串和数值。在下面的例子中，typeof 操作符使用这些函数来显示表达式的数据类型：

```
typeof 9;              // result: "number"
typeof String(9);      // result: "string"
typeof "9";            // result: "string"
typeof Number("9");    // result: "number"
```

这两个函数都不能改变待转换值的数据类型，而其返回值是在相等比较中进行比较的值：

```
var a = "09";
var b = 9;
a == String(b);        // result: false, because "09" does not equal "9"
typeof b;              // result: still "number"
Number(a) == b;        // result: true, because 9 equals 9
typeof a;              // result: still "string"
```

这说明，在测试两个值是否相等时，十分有必要考虑数据类型。

更多资源请访问稀酷客(www.ckook.com)

22.4　结合操作符

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+

结合操作符将两个操作数结合起来，产生一个与操作数相关的值。表 22-4 列出了 JavaScript 中的结合操作符。

表 22-4　JavaScript 结合操作符

语　法	名　称	操作数类型	结　果
+	加	Integer，float，string	Integer，float，string
-	减	Integer，float	Integer，float
*	乘	Integer，float	Integer，float
/	除	Integer，float	Integer，float
%	取模	Integer，float	Integer，float
++	增量	Integer，float	Integer，float
--	减量	Integer，float	Integer，float
+*val*	取正	Integer，float，string	Integer，float
-*val*	取负	Integer，float，string	Integer，float

数值的四则基本运算比较简单，加号还可用于连接字符串，如：

```
"Scooby " + "Doo" // result = "Scooby Doo"
```

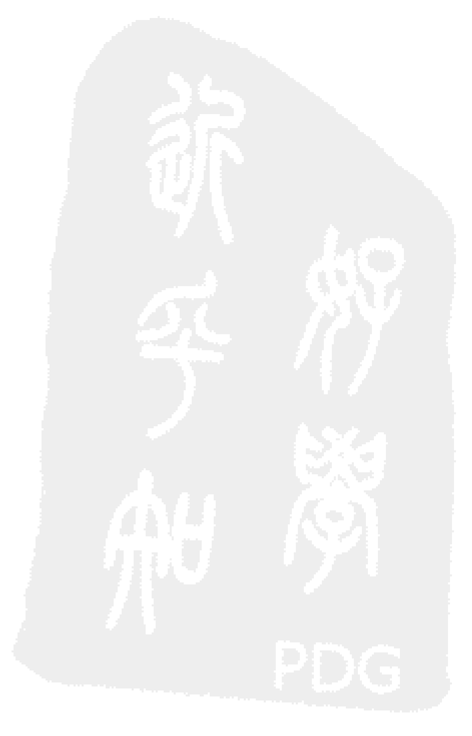
在面向对象的编程术语中，加号是重载的，也就是说它根据上下文执行不同的动作。另外，字符串连接本身不能监控各个字之间的空格，或者插入空格。在前面的例子中，名称之间的空格是第一个字符串的一部分。

取模运算可用于确定一个数是否被另一个数整除。第 21 章中的一个例子就使用它确定某年是否为闰年，虽然需要考虑整百年份(如 1900 年、2000 年)是否是闰年，但该例中的数学计算只检查某年能否被 4 整除。取模运算的结果是两个值相除后的余数：当余数为 0 时，一个数就能被另一个数整除。下面是一些可被 4 整除的年份：

```
2002 % 4 // result = 2
2003 % 4 // result = 3
2004 % 4 // result = 0 (Bingo! Leap year!)
```

可在 if...else 结构的条件语句中使用取模操作符：

```
var howMany = 0;
today = new Date();
var theYear = today.getYear();
   if (theYear % 4 == 0)
   {
       howMany = 29;
   }
```



更多资源请访问稀酷客(www.ckook.com)

```
    else
    {
        howMany = 28;
    }
```

有时需要在循环中按特定间隔执行某个动作，例如每 3 次循环执行一个操作，此时也可以使用取模操作符。下面是一个循环的例子，每循环 3 次，计数器就加 1，共循环 100 次。

```
for (var i = 1; i < 100; i++)
{
   if (i % 3 == 0)
      threeCounter++;
}
```

取模操作符可获得除法操作的余数，但其他一些语言还提供了整除操作符 div，来获得除法操作的结果的整数部分。虽然 JavaScript 没有处理这种操作的操作符，但若知道操作数总是正数，就一定可以实现这个操作符的操作：使用 Math.floor()或 Math.ceil()方法和除法操作符，如下：

```
Math.floor(4 / 3);      //result=1
```

在这个例子中，Math.floor()只在除法操作的结果大于或等于 0 时，才执行整除操作；而 Math.ceil()只在除法操作的结果小于 0 时，才执行整除操作。

增量操作符(++)是单目操作符(只有一个操作数)，它根据操作数在操作符的前面还是后面，执行两个不同的操作。增量和减量(--)操作符都能与赋值操作符结合使用，后面将会提到这一点。

顾名思义，增量操作符将操作数的值加 1。但在赋值语句中，要注意何时才会加 1。赋值语句将右操作数的值赋给左边的变量，如果++操作符位于右操作数的前边(前缀)，则先给右操作数加 1，再将其值赋给变量；如果++操作符位于右操作数的后边(后缀)，则先把右操作数的值赋给变量，再给该值加 1。分析以下语句，来了解这两个行为：

```
var a = 10;         // initialize a to 10
var z = 0;          // initialize z to zero
z = a;              // a = 10, so z = 10
z = ++a;           // a becomes 11 before assignment, so a = 11 and z becomes 11
z = a++;            // a is still 11 before assignment, so z = 11; then a becomes 12
z = a++;            // a is still 12 before assignment, so z = 12; then a becomes 13
```

减量操作符的用法和增量操作符相同，只是操作数减 1。

增量和减量操作符常在 for 和 while 循环中用作循环计数器。较简单的++或--符号比加 1 后再赋值(如 z=z+1 或 z+=1)更加简单明了。因为它们是单目操作符，所以可在循环中使用增量和减量操作符来调整计数变量的值，而不必使用赋值语句：

```
function doNothing()
{
   var i = 1;
   while (i < 20)
   {
      ++i;
   }
```


更多资源请访问稀酷客(www.ckook.com)

```
    alert(i); // loop ends with i = 20
}
```

最后一对结合操作符也是单目操作符(只有一个操作数)，取正和取负操作符可用作 Number()全局函数的快捷方式，将包含数值的字符串操作数转换为数值数据类型。字符串操作数没有改变，但该操作返回一个数值类型的值，如下面的语句所示：

```
var a = "123";
var b = +a;          // b is now 123
typeof a;            // result: "string"
typeof b;            // result: "number"
```

取负操作符(-val)还有一个功能：在数值前面(没有空格)加上减号，就可以使 JavaScript 将一个正值变成相应的负值，反之亦然。该操作符不改变操作数的值，但表达式返回修改后的值。下面例子中的几个语句说明了这种情况：

```
var x = 2;
var y = 8;
var z = -x;           // z equals -2, but x still equals 2
z = -(x + y);         // z equals -10, but x still equals 2 and y equals 8
z = -x + y;           // z equals 6, but x still equals 2 and y equals 8
```

要将 Boolean 值取负，请参阅 Boolean 操作符中关于 Not(!)操作符的讨论。

22.5　赋值操作符

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

赋值语句在 JavaScript 脚本中最常用。只要将一个值或表达式的结果复制给变量，用于进一步处理，就要使用这些语句。

为变量赋值有许多原因，也可在脚本中多次使用原始的值或表达式。下面列举了为变量赋值的几个原因。

- 变量名一般较短。
- 变量名更具描述性。
- 需要存储原始值，用于后面的脚本。
- 属性的原始值不能改变。
- 在脚本中多次调用同一方法的效率不高。

脚本编写新手常忽略最后一个原因。例如，如果脚本将一个长段落写入新文档中，则先将该段落组合成一个长句子字符串，再调用 document.createTextNode()方法把该长字符串添加到文档中，这比一次添加一个句子更高效。

表 22-5 列出了 JavaScript 中的赋值操作符。

更多资源请访问稀酷客(www.ckook.com)

表 22-5　JavaScript 赋值操作符

<table>
<tr><th>语　　法</th><th>名　　称</th><th>示　　例</th><th>含　　义</th></tr>
<tr><td>=</td><td>赋值</td><td>x=y</td><td>x =y</td></tr>
<tr><td>+=</td><td>相加并赋值</td><td>x+=y</td><td>x =x+y</td></tr>
<tr><td>-=</td><td>相减并赋值</td><td>x -= y</td><td>x = x -y</td></tr>
<tr><td>*=</td><td>相乘并赋值</td><td>x *= y</td><td>x = x *y</td></tr>
<tr><td>/ =</td><td>相除并赋值</td><td>x / =y</td><td>x =x / y</td></tr>
<tr><td>%=</td><td>取模并赋值</td><td>x %=y</td><td>x =x%y</td></tr>
<tr><td><<=</td><td>左移并赋值</td><td>x<<=y</td><td>x = x<<y</td></tr>
<tr><td>>=</td><td>右移并赋值</td><td>x>=y</td><td>x =x>y</td></tr>
<tr><td>>>=</td><td>填充 0 并赋值</td><td>x>>= y</td><td>x = x >>y</td></tr>
<tr><td>>>>=</td><td>右移并赋值</td><td>x>>>=y</td><td>x = x >>>y</td></tr>
<tr><td>&=</td><td>按位“与”并赋值</td><td>x&=y</td><td>x = x&y</td></tr>
<tr><td>|=</td><td>按位“或”并赋值</td><td>x| =y</td><td>x = x|y</td></tr>
<tr><td>^=</td><td>按位“异或”并赋值</td><td>x^=y</td><td>x = x ^y</td></tr>
<tr><td>[]=</td><td>解构并赋值</td><td>[a,b]=[c,d]</td><td>a=c, b=d</td></tr>
</table>

如上所述(表中后面几个操作符参见本章后面的 22.7 一节)，除了简单的等号外，使用其他赋值操作符可以少输入一些字符，特别是许多值要在一系列语句中结合在一起时。前面有许多这样的例子，如使用“相加并赋值”操作符(+=)组合一个长字符串。下面的脚本建立了一个段落，列出了区域办公室：

```
var aRegionalOffices = ['New York', 'Paris', 'Istanbul', 'Lusaka'];
var sText = 'Our offices are located in ';
var sSuffix = ',';

for (var i = 0; i < aRegionalOffices.length; i++)
{
      // doing penultimate office?
      if (i == aRegionalOffices.length-2)
      {
         sSuffix += 'and ';
      }
       // doing final office?
      else if (i == aRegionalOffices.length-1)
      {
         sSuffix = '.';
      }

   // add office to paragraph
   sText += aRegionalOffices[i] + sSuffix;
}

// create text node and add to paragraph element
```


PDG

更多资源请访问稀酷客(www.ckook.com)

```
var oNewText = document.createTextNode(sText);
var oParagraph = document.getElementById("p1");
oParagraph.appendChild(oNewText);
```

结果是：

```
<p id="p1">Our offices are located in New York, Paris, Istanbul, and Lusaka.</p>
```

该脚本段首先是一个普通的等号赋值操作符，将 sText 变量初始化为段落的开头。在添加办公室的循环中，使用"相加并赋值"操作符向 sText 变量加入其他字符串值。没有加值操作符，就必须在每行代码中使用普通的等号赋值操作符，将新的字符串值连接到已有的字符串数据上，如下：

```
var sText = 'Our offices are located in ';
...
   sText = sText + aRegionalOffices[i] + sSuffix;
```

这些增强的操作符都是极佳的快捷方式。

JavaScript 1.7 还添加了一个赋值语法：

[ ] (*方括号表示解构赋值*)

兼容性： FF2+，Opera 9.5 (部分)

方括号可将一组值赋予一系列变量，包括从函数中返回的数组：

```
[a, b, c] = [12, 23, 34];

function moveSoutheast(x, y)
{
   return [x+1, y+1];
}
[x, y] = moveSoutheast(x, y);
```

参见： 对象操作符[ ]，方括号可枚举一个对象成员。

22.6　布尔操作符

兼容性： WinIE4+，MacIE4+，NN3+，Moz+，Safari+

许多程序都涉及逻辑，所以逻辑算术非常重要。前面的许多例子都根据语句或表达式是布尔值 true 或 false 来进行决策。尽管现在还没有介绍如何组合多个布尔值和表达式，这是略微复杂的脚本必须具备的能力。

在 JavaScript 需要的各种条件表达式(如 if 结构)中，程序需要测试的条件比较复杂，并不仅仅是某个变量值是否大于一个固定值，或域是否为空。在验证文本输入域时，要检查该输入域是否只包含脚本所需要的数字。JavaScript 函数不能确定字符串是否只包含数字，所以必须把输入字符串逐个拆开，验证每个字符是否在 0~9 的范围内。其实这个检验包含两部分测试：检查每个字符的 ASCII 值是否小于 0 或大于 9(该字符不是数字)；或者检验字符的 ASCII 值是否

更多资源请访问稀酷客(www.ckook.com)

大于等于 0 且小于等于 9(该字符是数字)。接着要对两个测试的结果进行布尔运算。

22.6.1　布尔运算

此时就需要运用布尔运算了。只要使用两个值 true 和 false，就可以组合结果为布尔值的字符串表达式，通过布尔算术确定结果是 true 还是 false。

不能像数值那样加减布尔值，但可使用三个 JavaScript 布尔操作符。表 22-6 列出了这三个操作符。注意，OR 操作符是通过输入 Shift+\(反斜杠)得到的。

表 22-6　JavaScript 布尔操作符

语　法	名　称	操　作　数	结　果
&&	AND	Boolean	Boolean
||	OR	Boolean	Boolean
!	NOT	一个 Boolean	Boolean

如果不习惯，则布尔操作数和布尔操作符比较难使用，下面从最简单的布尔操作符 NOT 开始。这个操作符只需要一个操作数，并将其后的布尔值转换成相反的值，从 true 变为 false，或从 false 变为 true。例如：

```
!true          // result = false
!(10 > 5)      // result = false
!(10 < 5)      // result = true
!("cat" == "bat")     // result = true
```

如上所示，最好把 NOT 表达式的操作数放在括号中，它强制 JavaScript 在用 NOT 操作符取反之前，先计算括号内的表达式。否则，可能会意外对表达式的一部分执行取反操作，导致意外的结果。

AND(&&)操作符会结合两个布尔值，根据这两个值得到 true 或 false 值，这就引出了真值表的概念，它有助于理解操作符的所有可能结果。表 22-7 是 AND 操作符的真值表。

表 22-7　AND 操作符的真值表

左 操 作 符	AND 操作符	右 操 作 数	结　果
True	&&	True	True
True	&&	False	False
False	&&	True	False
False	&&	False	False

只有两个操作数都是 true，才会得到 true 结果。操作符左边还是右边的值是 true 或 false 并不重要，下面是 AND 操作符的例子：

```
5 > 1 && 50 > 10    // result = true
5 > 1 && 50 < 10    // result = false
5 < 1 && 50 > 10    // result = false
```


更多资源请访问稀酷客(www.ckook.com)

```
5 < 1 && 50 < 10    // result = false
```

注意：

在这段代码中，为什么没有使用括号来分隔比较和布尔表达式？原因是操作符具有优先级，详见本章后面的讨论。简言之，比较运算在布尔运算之前完成。即便如此，也最好使用括号将子表达式进行分组，这样可确保获得需要的结果，而且代码更便于阅读。

对照而言，OR(||)操作结果为 true 的情况比较多，原因是，只要一个操作数为 true，OR 操作就返回 true。OR 操作的真值表如表 22-8 所示。

表 22-8 OR 操作符的真值表

左 操 作 数	OR 操作符	右 操 作 数	结 果
True	||	True	True
True	||	False	True
False	||	True	True
False	||	False	False

所以，只要任何一边的操作数为 true 值，结果就是 true。将前面例子的 AND 操作符换成 OR 操作符，就可以看出 OR 操作符对结果的影响：

```
5 > 1 || 50 > 10    // result = true
5 > 1 || 50 < 10    // result = true
5 < 1 || 50 > 10    // result = true
5 < 1 || 50 < 10    // result = false
```

只有两个操作数都为 false，OR 操作符才返回 false。

22.6.2 使用布尔操作符

学习在 JavaScript 中使用布尔操作符需要花费一点时间，在纸上画一些草图有助于理解表达式的逻辑。前面在验证数据输入时使用了布尔操作符，确定字符是否在 ASCII 值的范围内。程序清单 22-1 中的函数将在配书光盘中的第 46 章详细论述，这个函数接收字符串，确定字符串中每个字符的 ASCII 值是否小于 0 或大于 9，即输入的字符串不是数值。

程序清单 22-1 输入字符串是否为数值

```
function isNumber(inputStr)
{
   for (var i = 0; i < inputStr.length; i++)
   {
      var oneChar = inputStr.substring(i, i + 1);
         if (oneChar < "0" || oneChar > "9")
         {
           alert("Please make sure entries are numerals only.");
           return false;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
    }
    return true;
}
```

这段代码运用了 JavaScript 的多种能力，在 for 循环中读取 string 对象中的单个字符(子字符串)。这里要注意 if 结构的条件：

```
(oneChar < "0" || oneChar > "9")
```

可以使用 OR 操作符在一个条件语句中执行两个测试。在 OR 真值表(表 22-8)中，只要一个测试返回 true，这个表达式就返回 true。之后，函数其余部分就警告用户出现了问题，并给调用语句返回 false。对于字符串中的每个字符，只有条件中的两个测试都是 false，函数才返回 true。

下面的函数在一个条件语句中使用 OR 操作符检查某数是否在几个取值范围内。程序清单 22-2 中的脚本来自配书光盘的第 53 章中的一个数组查找应用程序，该程序让用户输入美国社会保险号的头三位数。

程序清单 22-2　数值是否在不连续的范围内？

```
// function to determine if value is in acceptable range for this application
function inRange(inputStr)
{
  num = parseInt(inputStr)
     if (num < 1 || (num > 586 && num < 596) || (num > 599 && num < 700) ||
        num > 728)
     {
        alert("Sorry, the number you entered is not part of our database. Try
           another three-digit number.");
        return false;
     }
  return true;
}
```

调用该函数时，JavaScript 已经验证了用户输入的数据，了解到该输入是一个数值。现在，函数必须检查该数是否不在应用程序保存匹配数据的区域内。函数检查的条件是：数值是否

- 小于 1。
- 大于 586 且小于 596(使用 AND 操作符)。
- 大于 599 且小于 700(使用 AND 操作符)。
- 大于 728。

这些测试使用 OR 操作符连接在一起。所以，如果其中的一个测试是 true，整个 if 条件就是 true，就会警告用户。

如果不在一个条件语句中组合多个布尔表达式，另一个可选方法是嵌套一系列 if 结构。但这种结构不仅需要编写大量的代码，而且要为每个条件重复警告对话框信息，这可能导致失败，使用组合的布尔条件是最好的选择。

更多资源请访问稀酷客(www.ckook.com)

22.7　按位操作符

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

对于脚本编写者而言，按位操作符是一个高级主题。除非在服务器端应用程序或 Java applet 连接中做外部处理，否则很少使用按位操作符。熟悉某些数据类型(例如长整型)的资深程序员在这个领域中游刃有余，所以这里简单解释一下 JavaScript 按位操作符，表 22-9 列出了 JavaScript 的按位操作符。

表 22-9　JavaScript 的按位操作符

操　作　符	名　　称	左 操 作 数	右 操 作 数
&	按位与	整数值	整数值
		按位或	整数值	整数值
^	按位异或	整数值	整数值
~	按位非	无	整数值
<<	左移	整数值	移动数
>>	右移	整数值	移动数
>>>	右移，使用 0 填充	整数值	移动数

数值操作数可以是 JavaScript 语言的十进制、八进制和十六进制数，只要操作符有一个操作数，该值就转换成二进制(32 位)。在前三个按位操作中，一个操作数的每个二进制位和另一个操作数的相应位比较，每位的结果值取决于具体的操作符。

- **按位与**：两个位都是 1，则结果为 1。
- **按位或**：只要有一位是 1，则结果为 1。
- **按位异或**：只有一位是 1，则结果为 1。

按位非是单目操作符，将单个操作数的每位取反。按位移动操作符也只操作单个操作数，其第二个操作数指定要移位的二进制位按照操作符箭头方向移动的位数。

示例

例如，左移(<<)操作符具有以下作用：

```
4 << 2 // result = 16
```

这样移位的原因是十进制 4 的二进制形式为 00000100(8 位)。左移操作符让 JavaScript 将所有的位左移两位，得到二进制结果 00010000，转换为十进制是 16。若对这些操作符感兴趣，可使用第 4 章的 The Evaluator 来测试示例表达式，高级 C 和 C++编程的书也很有帮助。

22.8　对象操作符

下一组操作符处理对象(包括 JavaScript 内部对象、DOM 和自定义对象)以及数据类型。大

更多资源请访问稀酷客(www.ckook.com)

多数操作符在最早的 JavaScript 浏览器中就已实现，因此每个操作符都有自己的兼容等级。

.(*句点*)

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

句点操作符表示其左边的对象拥有或包含其右边的资源，例如 object.property 和 object.method()。

JavaScript 内部对象

```
var s = new String('syzygy');
var len = s.length;
var pos = s.indexOf('zyg'); // result: pos == 2
```

JavaScript 自定义对象

```
function myMethod()
{
   // (custom code goes here)
}

function myCustomObject()
{
   this.myProperty = 'something';
   this.myMethod = myMethod;
}

var o = new myCustomObject();
var p = o.newProperty;
o.myMethod();
```

DOM 对象：

```
var sTitle = document.title;
var oThing = document.getElementById("elementID");
```

[ ] (*方括号用于枚举对象成员*)

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

方括号枚举对象的成员，例如数组的一个元素：

定义数组：

```
var a = ['homo erectus', , 'homo sapiens'];
```

枚举一个数组元素：

```
a[5] = 'sixth element';
```

枚举一个对象属性：

```
a['color'] = 'puce';
```

参见：赋值操作符：[]方括号表示解构赋值

更多资源请访问稀酷客(www.ckook.com)

Delete

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

数组对象没有从集合中删除元素的方法，自定义对象也没有提供删除属性的方法。但将值设为空字符串或 null，就可以将数组项或属性置空。但数组元素或属性仍然保留在对象中，可以使用 delete 操作符完全删除这些元素或属性。

删除数组项比较特殊，应特别予以注意。如果数组使用数值索引，则删除给定的索引，就从数组中删除了这个索引值，但整个数组没有受到破坏，只是高于删除项的数组项改变了其索引值。

示例

例如下面简单的紧凑数组：

```
var oceans = new Array("Atlantic", "Pacific", "Indian","Arctic");
```

这类数组自动为数组项指定数值索引，以便以后使用，例如在 for 循环中：

```
for (var i = 0; i < oceans.length; i++)
{
   if (oceans[i] == form.destination.value)
   {
      // statements
   }
}
```

若有下面的语句：

```
delete oceans[2];
```

数组就有比较大的改变：第三个元素从数组中删除，但数组长度不变，只是索引值(2)从数组中删除。于是数组变成：

```
oceans[0] = "Atlantic";
oceans[1] = "Pacific";
oceans[3] = "Arctic";
```

若再引用集合中的 oceans [2]，结果将是 undefined。

delete 操作符对指定索引的数组非常有效，因为删除数字索引时很少会引起混淆。脚本可对余下的项及其值施加更多的控制，因为它们不依赖被删除的索引项。

对于这种删除行为，JavaScript 没有提供控制对相关内存的使用，所有的垃圾收集都由 JavaScript 解释器引擎管理，它试图确定何时不再使用占用内存的数据项，何时恢复浏览器应用程序不使用的内存。但是，不能强制浏览器完成垃圾收集任务。所以，从数组中删除一项不能保证相关内存立即释放。

in

兼容性： WinIE5.5+，MacIE−，NN6+，Moz+，Safari+，Opera+，Chrome+

in 操作符允许脚本语句检查对象，确定它是否有指定的属性或方法。该操作符左边的操作

更多资源请访问稀酷客(www.ckook.com)

数是属性或方法的字符串引用(只有方法名没有括号)；右边的操作数是要检查的对象。若对象知道属性或方法，表达式就返回 true，因此可将 in 操作符用在条件表达式中。

示例

可在 The Evaluator(参见第 4 章)中测试 in 操作符。例如，为证明 document 对象实现了 write() 方法，在 The Evaluator 程序的顶端文本框中输入表达式：

```
"write" in document
```

但比较 IE5.5+和现代 W3C 浏览器中 W3C DOM document.defaultView 属性的实现方式：

```
"defaultView" in document
```

在 NN6+、Mozilla(包括 Firefox 和 Camino)和 Safari 中，结果是 true，而在 IE5.5 和 IE6 中，结果是 false。

将这个操作符用在条件表达式中，不仅可在分支代码中检查简单的对象。例如，如果准备在脚本中使用 document.defaultView，就可在使用它之前确保支持该属性(假设用户的浏览器都支持 in 操作符)。

instanceof

兼容性： WinIE5+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

instanceof 操作符允许脚本测试一个对象是否是特定的 JavaScript 内部对象或 DOM 对象的实例。该操作符的左操作数是要测试的值；右操作数是根类的引用，要检测的值可能是由此根类构建而来的。

对于内部的 JavaScript 类，操作符右边的对象引用类型可以是静态对象，如 Date、String、Number、Boolean、Object、Array 和 RegExp。有时，需要知道内部 JavaScript 类是否是其他内部类的子类，即一个值可能是两个不同静态对象的实例。

示例

考虑下面的语句(可在 The Evaluator 程序中使用)：

```
a = new Array(1,2,3);
a instanceof Array;
```

第二个语句的结果是 true，因为 Array 构造函数可以用于生成对象。但 JavaScript Array 本身是根 Object 对象的一个实例，所以下面的语句会得到 true：

```
a instanceof Object;
Array instanceof Object;
```

new

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

大多数 JavaScript 核心对象都有内置于语言的构造函数。要访问这些函数，可使用 new 操作符和构造函数名。该函数返回指向对象实例的引用，接着脚本就可以用它来得到或设置对象

更多资源请访问稀酷客(www.ckook.com)

的属性，也可以调用对象的方法。例如，创建一个新日期对象，需调用 Date 对象的构造函数，如下：

```
var today = new Date();
```

有的对象构造函数需要参数来帮助定义对象，有的构造函数(例如 Date 对象的构造函数)可以接收许多不同的参数，具体取决于设置初始对象的日期信息格式。new 操作符用在表 22-10 所示的核心语言对象中，每个对象都指定了 JavaScript 版本：

表 22-10　核心语言对象

JavaScript 1.0	JavaScript1.1	JavaScript 1.2	JavaScript 1.5
Date	Array	RegExp	Error
Ob j ect	Boolean		
(自定义对象)	Function		
	Image		
	Number		
	String		

this

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

JavaScript 有一个自我引用操作符 this，允许脚本语句指向脚本所在的对象。

this 操作符常用在事件处理程序中，它将其本身的引用传递给函数，做进一步处理。

接收值的函数将它赋给一个变量，用来引用发送者、发送者的属性和方法。

示例

this 操作符引用一个对象，所以它可以用于访问对象的属性。例如：

```
oInputField.onchange = validateInput;

function validateInput(evt)
{
   var sInputvalue = this.value;
}
```

当访问者改变输入域的内容时——这会触发 onchange 事件，该事件会执行 validateInput() 函数——该函数中的代码就把 this 看成它所属的输入域对象。这里，this.value 表示访问者输入的值。

在内联 JavaScript 中，代码如下：

```
<input type="text" name="entry" onchange="validateInput(this)" />
```

this 操作符还可以用于其他对象，如自定义对象。为自定义对象定义构造函数时，常可以使用 this 操作符定义对象的属性，为对象的属性赋值。分析如下创建对象的例子：

```
function bottledWater(brand, ozSize, flavor)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   this.brand = brand;
   this.ozSize = ozSize;
   this.flavor = flavor;
}
var myWater = new bottledWater("Eau de Odio", 16, "original");
```

使用构造函数创建新的对象后，this 操作符就定义对象的每个属性，并为属性分配相应的输入值。这里为属性和参数变量使用同一个名字，使构造函数便于维护。

另外，如果将函数赋予对象的一个属性(其行为类似于对象的一个方法)，函数内的 this 操作符就指向调用函数的对象，提供访问对象属性的路径。例如，在上面创建的 myWater 对象中添加下面的函数定义和语句，函数就能直接访问对象的 brand 属性：

```
function adSlogan()
{
   return "Drink " + this.brand + ", it's wet and wild!";
}
myWater.getSlogan = adSlogan;
```

上述语句调用 myWater.getSlogan()方法时，myWater 对象调用 adSlogan()函数，但它们都在 myWater 对象中。因此，this 操作符用于指向 myWater 对象，通过 this 操作符可使用 brand 属性(this.brand)。

22.9 其他操作符

最后一组操作符和前面的所有类型都不一样，但它们同样非常重要。

,(逗号)

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

逗号操作符表示一系列从左至右求值的表达式，一般用来初始化多个变量。例如，可将几个变量声明组合在一个 var 语句中，如下：

```
var name, address, serialNumber;
```

这个操作符还可用在 for 循环结构的表达式内。在下面的例子中，初始化了两个不同的计数变量，并以不同幅度递增。当循环开始时，两个变量都初始化为 0(不一定要这样，但本示例以这种方式开始循环)；每次循环后，一个变量加 1，另一个变量加 10：

```
for (var i=0, j=0; i < someLength; i++, j+10)
{
   ...
}
```

不要把逗号操作符和语句之间的分号分隔符混淆。

更多资源请访问稀酷客(www.ckook.com)

?:

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

条件操作符是第 21 章中 if ... else 条件结构的快捷方式，该操作符一般和赋值操作符一起使用，根据条件表达式的结果将两个值中的一个赋给变量。条件操作符的语法如下：

```
condition ? expressionIfTrue : expressionIfFalse
```

若与赋值操作符一起使用，语法为：

```
var = condition ? expressionIfTrue : expressionIfFalse;
```

使用这个操作符的要点是：含有这个操作符的表达式等于问号后面两个表达式中的一个。事实上，这两个表达式都可以包含任何 JavaScript 语句，可以调用其他任何函数，甚至可将条件操作符嵌套在一个表达式中，达到与 if... else 嵌套结构等价的效果。为了保证嵌套条件的正确性，用括号将内部表达式括起，确保它们在外部表达式之前求值。例如，下面的语句根据月份的日期，将三个字符串中的一个赋给变量：

```
var monthPart = (dateNum <= 10) ? "early" : ((dateNum <= 20) ?
   "middle" : "late");
```

执行该语句时，先计算第一个冒号右边的内部条件表达式，返回 middle 或 late 值；然后计算外部的条件表达式，返回 early 或内部条件表达式的结果。

typeof

兼容性：WinIE3+，MacIE3+，NN3+，Moz+，Safari+，Opera+，Chrome+

与处理逻辑和算术的大多数其他操作符不同，单目操作符 typeof 定义变量或表达式的值的类型。一般情况下，这个操作符用于确定变量值是下面的哪个类型：number、string、boolean、object、function 或 undefined。

示例

因为变量不仅能包含上述任一种数据类型，还可在运行期间改变数据类型，所以研究 JavaScript 的数据类型很有帮助。根据值的类型，脚本可能需要采用不同方式处理该值。typeof 属性常用作条件的一部分。例如：

```
if (typeof myVal == "number")
{
   myVal = parseInt(myVal);
}
```

typeof 操作的值是一个字符串。

void

兼容性：WinIE3+，MacIE3+，NN3+，Moz+，Safari+，Opera+，Chrome+

在所有脚本浏览器中，都可以使用 javascript:伪协议，为 HTML 标记的 href 和 src 特性提

更多资源请访问稀酷客(www.ckook.com)

供参数，如链接。在处理过程中，要小心 URL 调用的函数或语句，它们可能不返回或计算任何值。若这样的表达式返回一个值，这个值(有时是客户硬盘上的目录)常会替换页面的内容。为避免发生这种情况，应在 javascript:URL 调用的函数或表达式前面使用 void 操作符。

示例

使用这个操作符的最佳方式是将操作符放在表达式或函数的前面，用空格分开，如：

```
javascript: void doSomething();
```

有时需要将 void 操作符后面的表达式用括号括起，不过只有表达式含有优先级比 void 低的操作符(参见下一节)，才有必要使用括号。不要用括号括起所有的表达式，因为一些浏览器可能会出问题。通常将下面的 URL 赋给 a 链接的 href 特性，a 链接的 onclick 事件处理程序会完成所有操作：

```
href="javascript: void (0)"
```

void 操作符保证函数或表达式不返回 HTML 特性使用的值，这样一个链接的 onclick 事件处理程序还阻止了被单击链接的固有行为(例如执行 return false)。

22.10　操作符的优先级

开始使用含有多个操作符的复杂表达式时，如程序清单 22-2 所示，必须知道 JavaScript 计算这些表达式的先后顺序。JavaScript 为每类操作符指定了不同的优先级或权重，以统一的方式计算复杂的表达式。

在下面的表达式中：

```
10 + 4 * 5 // result = 30
```

JavaScript 根据优先级机制，在执行加法前，先执行乘法运算，而不管操作符在语句中的位置。也就是说，JavaScript 首先将 4 乘以 5，再将结果加 10，得到结果 30。这可能不是我们希望的方式，我们可能想先给 10 加上 4，再将所得的和乘以 5，为此，必须重写 JavaScript 固有的操作符优先级，用括号把低优先级的操作符括起。下面的语句调整了原来的表达式，使之与前面不同：

```
(10 + 4) * 5 // result = 70
```

这对括号对结果有很大的影响。在 JavaScript 中，括号的优先级最高，若在表达式中嵌套括号，首先计算最里面的括号。

为构建复杂的表达式，可参看表 22-11 中 JavaScript 的操作符优先级。通常的规则是：如果对复杂表达式的优先级感到怀疑，就根据所希望的表达式求值顺序在表达式中使用括号。

这个优先级机制有助于避免在同一个表达式中出现优先级相同的运算符，而若在同一个表达式中出现优先级相同的运算符，如加法和减法，JavaScript 按从左向右的顺序计算。

对于条件语句中的 Boolean 字符串表达式(程序清单 22-2)，JavaScript 使用最短路径来计算其值。由于嵌套表达式是从左向右计算，有时在扫描完整个表达式前，就已经计算出其值。只

更多资源请访问稀酷客(www.ckook.com)

要 JavaScript 遇到 AND 操作符，如果左操作数等于 false，整个表达式就等于 false，而不对右操作数进行求值；对于 OR 操作符，如果左操作数为 true，JavaScript 就使用最短路径计算出整个表达式等于 true。如果不对脚本做充分检查，这个功能很容易出错：当右操作数存在语法错误或其他错误时，不对右操作数求值，就不能测试出右操作数的错误，就不知道代码中有错误。当然，用户会很快发现这个错误。要仔细检查测试，消除错误。

表 22-11　JavaScript 操作符优先级

优 先 级	操 作 符	说 明
1	()	从内向外
	[]	数组索引值
	function()	任何远端函数调用
2	!	布尔非
	~	按位非
	-	取反
	++	增量
	--	减量
	new	
	typeof	
	void	
	delete	删除数组或对象项
3	*	乘
	/	除
	%	取模
4	+	加
	-	减
5	<<	按位移
	>	
	>>	
6	<	比较操作符
	<=	
	>	
	>=	
7	==	相等
	!=	
8	&	按位与
9	^	按位异或
10	|	按位或

更多资源请访问稀酷客(www.ckook.com)

(续表)

优 先 级	操 作 符	说 明
11	&&	布尔与
12	||	布尔或
13	?	条件表达式
14	= += -= *= /= %= <<= >= >>= &= ^= |=	赋值操作符
15	,	逗号(参数分隔符)

注意：

所有算术和字符串连接操作符的优先级都高于任何比较操作符，这就保证了先计算作为操作数的表达式，再进行比较。

处理复杂表达式的关键是将表达式分割为多个部分，分别计算。参见配书光盘中第 48 章介绍的其他调试技巧。

更多资源请访问稀酷客(www.ckook.com)

第23章

函数和自定义对象

本书前面已介绍了数十个函数，并简单讲解了其工作方理。本章将介绍 function 对象的规范，并深入讨论如何在 JavaScript 代码中创建对象。如果你以前没有学过面向对象编程，现在就可以学习它。JavaScript 全面支持面向对象编程，允许开发极大地依赖于自定义对象的脚本。

本章包含哪些内容？
创建函数块
向函数传递参数
创建自己的对象
Object 对象

23.1 Function 对象

属　性	方　法	事件处理程序
arguments	apply()	
arity	call()	
caller	toString()	
constructor	valueOf()	
length		
prototype		

23.1.1 语法

创建 Function 对象：

```
function functionName([arg1,...[,argN]])
{
   statement(s)
}

var funcName = new Function(["argName1",...[,"argNameN"],
          "statement1;...[;statementN]"])object.eventHandlerName =
```

更多资源请访问稀酷客(www.ckook.com)

```
function([arg1,...[,argN]]) {statement(s)}
```

访问函数对象的方法和属性：

```
functionObject.property | method([parameters])
```

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+

23.1.2　关于 Function 对象

JavaScript 用一种结构表示其他语言中的过程、子例程和函数，即自定义函数。自定义函数可以返回值(在程序中可用 return 关键字返回值)，也可以不返回值。在 JavaScript 中，除了文档载入时执行的代码外，其他延迟处理都在函数中进行。

尽管可编写长达数百行的函数，但最好把过长的处理过程分解为多个简短的函数。因为代码块越小，就越容易编写和调试；小程序块更便于查看整个脚本，还可以把函数全局化，在其他脚本中重用；脚本的其他部分和其他开放框架也可以使用它们。

编写正确的、可重用的函数需要时间和经验，但越早了解这个概念的重要性，就越能留心 Web 上其他脚本里的好例子。

23.1.3　创建函数

在脚本中定义函数的标准方法是遵循下面的简单模式，然后加入详细内容。定义函数的正式语法为：

```
function functionName( [arg1] ... [, argN])
{
   statement(s)
}
```

为函数指定名称有助于了解函数的准确活动范围。如果不能将函数要完成的任务用一到三个词来描述(应把这些词缩写为一个字符序列，作为函数名)，函数要完成的任务就可能太多了。最好将这些工作放在两个或更多的函数中。开始定义函数时，要考虑以前编写的函数有无可借鉴之处，若可以从函数中的一部分复制代码，并粘贴到另一个部分，就说明在函数的不同地方执行相同的操作，此时可将该部分提取出来，作为一个独立的函数。

下面列举一个简单的函数例子，它接受一个名称参数，返回一个包括该名称的问候字符串：

```
function sayHello(name)
{
   return ("Hello, " + name + ".");
}
```

还可用 new Function ()构造函数创建匿名函数，并为该匿名函数指定名称，如下：

```
var funcName = new Function(["argName1",...[,"argNameN"],
               "statement1;...[;statementN]"]);
```

脚本若需要在文档载入后创建函数，这种方法将非常有用。函数的所有组成都在定义中表

更多资源请访问稀酷客(www.ckook.com)

示出来，函数的每个参数名称是用逗号分隔的字符串值，最后一个参数字符串包含函数调用时执行的语句(放在引号中的字符串不能分解到多个代码行上，在下面的代码段中，第二行末尾使用了续行符，这只是因为本书的页面宽度有限)。将每个 JavaScript 语句用分号隔开，将整个语句序列放在引号中，如下所示：

```
var willItFit = new Function("width","height",
                "var sx = screen.availWidth; var sy = screen.availHeight;↵
                return (sx >= width && sy >= height)");
```

willItFit()函数有两个参数，函数体定义了两个局部变量(sx 和 sy)，如果输入参数比局部变量小，就返回布尔值 true。该函数使用传统格式的定义，如下：

```
function willItFit(width, height)
{
   var sx = screen.availWidth;
   var sy = screen.availHeight;
   return (sx >= width && sy >= height);
}
```

一旦函数在浏览器的内存中，就可以像其他任意函数一样调用它：

```
if (willItFit(400,500))
{
   // statements to load image
}
```

IE4+、NN4+、Moz 和其他 W3C DOM 浏览器可使用后一种函数创建格式，这种高级技术称为 Lambda 表达式，是创建匿名函数引用的快捷方式(匿名函数是真正的匿名，没有以后可以引用的函数名)。该技术的常见应用是在传递事件对象时，为事件处理程序赋予函数引用。下面的例子说明了如何为表单控件的 onchange 事件处理程序指定匿名函数：

```
document.forms[0].age.onchange = function(event)
     {isNumber(document.forms[0].age)}
```

因为匿名函数执行为函数对象的引用，所以在需要函数引用时，可使用匿名函数的形式，包括方法或其他函数的参数。

23.1.4　嵌套函数

现代浏览器还可在函数中嵌套函数。如果没有嵌套函数，每个函数都在全局级别定义，脚本的所有其他代码都可以访问每个函数。而使用嵌套函数，可将一个函数封装在另一个函数中，这样被嵌套函数就变成容器函数的私有函数。当然，此时最好不要重用名称，但可在多个全局函数中创建同名的嵌套函数，如下面的函数构架：

```
function outerA()
{
   // statements
   function innerA()
```


更多资源请访问稀酷客(www.ckook.com)

```
    {
        // statements
    }
    // statements
}
function outerB()
{
    // statements
    function innerA()
    {
        // statements
    }
    function innerB()
    {
        // statements
    }
    // statements
}
```

当一组语句需要在大函数中的多个地方调用，且这些语句仅在大函数中才有意义时，就应使用嵌套函数。也就是说，不要把这些语句提取出来，作为一个单独的全局函数，而是将它们放在大函数内。

注意：
嵌套函数的前提是隔离函数，使其对于整个脚本而言是私有的。这样会得到更整齐的脚本代码，因为如果没有合适的理由，对象就不应设置为全局的。这里做一个类比，热水器虽然可以放在屋外(全局)，但放在屋内(局部)通常更安全、更合适。邮箱必须与外部世界(全局)交互，而热水器是在内部(局部)起作用，因此放在屋内。毕竟，我们并不希望自己的热水供外人使用。

只有包含函数的语句才能访问嵌套函数(其顺序可任意)。内部函数可以访问外部函数定义的所有变量(包括参数变量)，但外部函数不能访问内部函数定义的变量。请参见本章的 23.2.2 一节，了解脚本的各个部分能否访问变量的更多内容。

23.1.5　函数的参数

函数定义要求在函数名 functionName 后加一对括号。若函数在调用时不需要给它传递任何参数，该括号就为空；但调用函数时需要某种数据，就要为每个参数指定名称。参数可以是任何类型的值：数值、字符串、布尔操作符甚至是完整的对象引用，如表单或表单元素。给这些变量选择有助于回忆起其内容，并避免重用已有的对象名作为变量名，因为同名的变量和对象出现在同一个语句中时，容易引起混淆。还要避免使用 JavaScript 关键字(包括光盘中的附录 C 列出的保留字)，以及脚本中定义的全局变量名(参见后面关于全局变量的更多介绍)。

JavaScript 对函数定义的参数数量和调用语句传递的参数数量之间的匹配没有严格的要求。如果函数的参数定义为三个，而调用语句只指定了两个参数，第三个参数变量就指定为 null，例如：

```
function saveWinners(first, second, third)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   // statements
}
oneFunction("George","Gracie");
```

在前面的例子中，函数内部 first 和 second 的值分别是 George 和 Gracie，third 的值为 null。

相反，如果调用语句传递的参数比函数指定的参数变量多，JavaScript 也不会予以阻止。事实上，该语言包含 arguments 属性机制，可将这个属性添加到函数中，收集传递给函数的外部参数。

23.1.6 属性

arguments(已废弃)

值：自变量数组，　　只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Chrome+

当函数得到调用函数的语句传来的参数值时，这些参数值会赋给函数对象的 arguments 属性。该属性是参数值的数组，参数值赋给数组中从 0 开始计数的索引项，而不管是否为它定义了参数。通过 functionName.arguments.length 可确定传递的参数个数。例如，传递了 4 个参数后，functionName.arguments.length 返回 4。于是，可使用数组符号(functionName.arguments[i])来提取需要的任何参数值。

理论上，不必为函数定义参数变量，因为我们可以提取想要的任何参数数组项。仔细选择的参数变量名会更易读，所以在大多数情况下最好使用这些参数变量名，而不要使用 arguments 属性。但有时，一个函数定义可能需要处理多个函数调用，并且每个调用都有不同数量的参数。除了给定名称的参数变量外，函数知道如何处理其他参数。

注意：
有些情况下，需要创建一个函数，来接受数量可变的参数，此时 arguments 属性是访问和处理参数的唯一方式。例如，创建一个函数，来计算测验分数的平均值，且测验分数的个数可能会变化，此时应编写一个函数，使其遍历 arguments 数组，将分数相加，再计算平均值。

示例

参看程序清单 23-1 和 23-2 中 arguments 和 caller 属性的例子。

arity(已废弃)

值：整型，　　只读

兼容性：WinIE−，MacIE−，NN4+，Moz−，Safari−，Chrome+

参见本章后面对 length 属性的讨论。

caller

值：function 对象引用，　　只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari−，Chrome+

更多资源请访问稀酷客(www.ckook.com)

当一个函数调用另一个函数时，两个函数之间就建立起一个链，这样返回值就知道该往哪里去。所以，被调用的函数包含一个调用它的函数引用，这些信息自动存储在 function 对象的 caller 属性中。这种关系和子窗口的 opener 属性相似，opener 属性指向创建子窗口的窗口或框架。只有被调用的函数在调用它的函数中运行时，caller 属性值才是有效的。函数不运行时，caller 属性为 null。

caller 属性值是 function 对象的引用，所以可以检查该对象的 arguments 和 caller 属性(它可能被别的函数调用)，因此，函数可以查看调用它的函数给它传递了什么值。

如果当前函数被另一个函数(包括事件处理程序)调用，functionName.caller 属性就显示整个函数定义的内容。如果函数的调用来自常规的 JavaScript 语句，而非来自函数内部，functionName.caller 属性就是 null。

示例

为了掌握这两个属性的作用，请看程序清单 23-1。

注意：

本章和本书许多章节中指定事件处理程序的函数是 addEvent()，它是一个跨浏览器的事件处理程序，详见第 32 章。

程序清单 23-1　函数的 arguments 和 caller 属性

HTML: jsb-23-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Function call information</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-23-01.js"></script>
   </head>
   <body>
      <h1>Function call information</h1>
      <div id="placeHolder"></div>
   </body>
</html>
```

JavaScript: jsb-23-01.js

```
// initialize when the page has loaded
addEvent(window, "load", showArgCaller);

var newElem;
var newText;

function showArgCaller ()
{
   // Call the hansel function twice: once directly,
   // second indirectly from the gretel function.
```


更多资源请访问稀酷客(www.ckook.com)

```
    hansel(1, "two", 3);
    gretel(4, "five", 6, "seven");
}

function hansel(x,y)
{
    var args = hansel.arguments;
    var placeHolderElement = document.getElementById("placeHolder");

        if (placeHolderElement)
        {
            // a header line
            newElem = document.createElement("h3");
            newElem.className = "objProperty";
            newText = document.createTextNode("The caller and the arguments");
            // insert the 1st line (text node) into the new h3
            newElem.appendChild(newText);
            // insert the completed h3 into placeholder
            placeHolderElement.appendChild(newElem);

            // the next line is the caller
            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode("hansel.caller is " + hansel.caller);
            newElem.appendChild(newText);
            placeHolderElement.appendChild(newElem);

            // the next line is the number or arguments
            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode("hansel.arguments.length is " +
                                              hansel.arguments.length);
            newElem.appendChild(newText);
            placeHolderElement.appendChild(newElem);

            // the rest of the lines are the arguments
            for (var i = 0; i < args.length; i++) {
                newElem = document.createElement("div");
                newElem.className = "objProperty";
                newText = document.createTextNode("argument " + i + " is " + args[i]);
                newElem.appendChild(newText);
                placeHolderElement.appendChild(newElem);
            }
        }
}

function gretel(x,y,z)
{
    today = new Date();
    thisYear = today.getFullYear();
    hansel(x,y,z,thisYear);
}
```


更多资源请访问稀酷客(www.ckook.com)

载入页面时，浏览器窗口会显示以下结果：

```
The caller and the arguments
hansel.caller is function showArgCaller()
{
  hansel(1, "two", 3);
  gretel(4, "five", 6, "seven");
}
hansel.arguments.length is 3
argument 0 is 1
argument 1 is two
argument 2 is 3
The caller and the arguments
hansel.caller is function gretel(x, y, z)
{
  today = new Date();
  thisYear = today.getFullYear();
  hansel(x,y,z,thisYear);
}
hansel.arguments.length is 4
argument 0 is 4
argument 1 is five
argument 2 is 6
argument 3 is 2010 (or whatever the current year is)
```

载入文档时，外部脚本的 onload 事件处理程序直接调用了 hansel()函数，并传递了三个参数，尽管 hansel()函数只定义了两个参数。hansel.arguments 属性也接收到三个参数。然后，onload 事件处理程序调用 gretel()函数，gretel()函数再次调用 hansel()。但 gretel()调用 hansel()时，传递了 4 个参数，而 gretel()函数只提取了调用语句传递的 3 个参数，并从自己的计算中插入另一个值，作为额外的参数传给 hansel()。hansel.caller 属性显示了 gretel()函数的全部内容，而 hansel.arguments 得到了所有 4 个参数，包括 gretel()函数引入的年值。

constructor

参见本章后面的 object.constructor。

length

值：整型，　　　　只读

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+

函数的 arguments 属性证明，JavaScript 对函数定义的参数数目是否匹配传递给函数的参数数目，并没有严格的要求。但脚本可以检查 Function 对象的 length 属性，查看函数实际定义的参数变量数目。属性的引用以表示对象的函数名开头，例如下面的函数定义：

```
function identify(name, rank, serialNum)
{
   // ...
}
```


更多资源请访问稀酷客(www.ckook.com)

函数外面的任何脚本语句都能使用如下引用来读取参数数目：

```
identify.length
```

在前面的例子中，该属性的值为 3。length 属性代替了只在 NN 中使用的 arity 属性。

prototype

详见第 18 章。

23.1.7　方法

apply(*[thisObj[, argumentsArray]]*)，call(*[thisObj[, arg1[, arg2[, . . . argN]]]]*)

返回值： 无

兼容性： WinIE5.5+，MacIE−，NN6+，Moz+，Safari+，Chrome+

function 对象的 apply()和 call()方法可以调用函数，这两个方法似乎是多余的，因为在脚本语句中，调用函数的一般方式是写出函数名，后跟括号，在括号中传递参数，等等。但这些方法的不同之处在于，通过 apply()和 call()方法，可以只使用函数的引用来调用函数。例如，如果脚本使用 new Function()构造函数(或浏览器支持的其他快捷方式)定义了一个函数，该构造函数的结果就是函数的一个引用。为了在以后只使用该引用(假设保存为全局变量)调用函数，就可以使用 apply()或 call()方法。这两个方法的结果相同，但选择哪个方法取决于传递函数参数的方式(稍后讨论)。

这两个方法的第一个参数是对象(函数将它作为当前对象)的引用。对于脚本中定义的普通函数，使用关键字 this，表示函数就是当前对象(与普通函数相同)。实际上，若没有给函数传递参数，可在这两个方法中忽略参数。

被调用的函数定义为自定义对象的方法时，对象引用才起作用(本章后面介绍了相关概念，可以在熟悉自定义对象后再回来看看)。

示例

下面的代码生成了一个自定义对象，并为该对象赋予一个方法，以便显示对象的属性：

```
// function to be invoked as a method from a 'car' object
function showCar()
{
   alert(this.make + " : " + this.color);
}
// 'car' object constructor function
function Car(make, color)
{
   this.make = make;
   this.color = color;
   this.show = showCar;
}
// create instance of a 'car' object
var myCar = new Car("Ford", "blue");
```


更多资源请访问稀酷客(www.ckook.com)

myCar 对象显示属性警告的一般方法是：

```
myCar.show();
```

此时运行 showCar()函数，将当前的 Car 对象作为函数中 this 引用的上下文。也就是说，当 showCar()运行为对象的方法时，函数将对象作为"当前对象"。

但使用 call()或 apply ()方法，就不必将 showCar()函数与 myCar 对象绑定，可以忽略 Car () 构造函数中将 showCar 函数指定为对象方法名的语句。相反，脚本可调用 showCar()方法，将 myCar 作为当前对象：

```
showCar.call(myCar);
```

showCar()函数像前面一样执行，call()方法的第一个参数中的对象引用作为 showCar()函数的当前对象。

至于后面的参数，apply()方法的第二个参数是参数值的数组，这些值传递给当前函数，值的顺序必须和函数定义中参数变量的顺序匹配。另一方面，call()方法可以用逗号隔开的列表传递各个参数。选择哪个方法取决于脚本所带的参数，若参数采用数组的形式，就使用 apply()方法；否则使用 call()方法。ECMA 建议，如果没有参数需要传递，就通过 call()方法调用函数。

注意：
ECMA 是一个标准组织，它审查正式的 JavaScript 语言标准，即 ECMAScript。ECMAScript 是语言规范，而 JavaScript 是实际的语言实现方案。

toString()，valueOf()

返回值： 字符串
兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+

脚本很少调用 Function 对象的 toString ()和 valueOf ()方法；这些方法允许调试脚本，显示函数定义的字符串版本。例如，在 The Evaluator(参见第 4 章)的顶部文本框中输入定义的函数名时，JavaScript 将函数自动转换成字符串，使它的"值"显示在 Results 框中。一般很少使用这些方法，或者解析它们返回的文本。

23.2　函数应用的注意事项

理解 JavaScript 函数的细节是脚本成功的关键，对于复杂的应用程序而言尤其如此。本节的其他主题包括调用函数的方式、函数变量的作用域、递归和重用函数的设计。

23.2.1　调用函数

只有脚本通过函数名或引用调用函数，函数才开始工作。脚本可以通过 4 种方法调用函数(也就是使函数开始工作)：文档对象事件处理程序、JavaScript 语句、指向 javascript:URL 的 href 特性以及 Function 对象的 call()和 apply ()方法。本书前面尚未讨论 javascript:URL 方法(有时称为伪 URL)。

PDG

更多资源请访问稀酷客(www.ckook.com)

几个 HTML 标记有 href 特性，它们一般指向 Internet URL，这些 URL 用于导航到另一个页面，或者载入需要辅助应用程序或插件程序的 MIME 文件。这些 HTML 标记一般是可单击对象的标记，如链接和客户端的图像映射区域。

支持 JavaScript 的浏览器有一个特殊的内置 URL 伪协议——javascript:，可以通过 href 特性指向 JavaScript 函数或方法，而不是网上的 URL。例如，一般使用 javascript:URL 改变单个链接的两个框架内容，因为 href 特性只指向单个 URL，JavaScript 没有提供便于多框架导航的方式。为了实现多框架导航，应编写一个函数，设置两个框架的 location.href 属性，然后从 href 特性中调用该函数，下面的例子显示了这个函数：

```
function loadPages()
{
   parent.frames[1].location.href = "page2.html";
   parent.frames[2].location.href = "instrux2.html";
}
// ...
<a href="javascript:loadPages()">Next</a>
```

此类函数调用可以包括参数，完成各种功能。注意当函数返回值时，可能具有一个潜在的副作用(也许，函数从脚本中需要函数返回值的其他位置上调用)：因为 href 特性将 target 窗口设置为显示该特性的值，所以把函数返回的值赋给 target 窗口，这可能不是所需要的效果。

为防止将返回值赋给 href 特性，应在函数调用前加上 void 操作符：

```
<a href="javascript:void loadPages()">
```

若不想让 href 特性执行任何操作，而让 onclick 事件处理程序完成所有工作，就在操作符后面指定空函数：

```
<a href="javascript:void (0)">
```

许多其他编程语言的资深程序员认为，这个操作符表示函数或过程没有返回值，该操作符的确具有这个功能，但它不放在传统的位置上。

23.2.2　变量的作用域：全局作用域还是局部作用域

在 JavaScript 中，变量有两个作用域。任何在脚本的主流程中(不是在函数中)初始化的变量都是全局变量，因为同一文档脚本的任何语句都能使用名称访问这些变量。另外，还可在函数内部使用 var 语句来初始化变量，这样该变量名只能用在函数内的语句中，其作用域是函数内部，所以称为局部变量。通过将变量的作用域限制在单个函数中，就可以在多个函数中重用这些变量名称，每个函数的变量都带有不同的信息。注意在函数内部声明变量时，若没有使用 var 关键字，就是在创建全局变量，而不是局部变量。程序清单 23-2 演示了各种作用域。

程序清单 23-2　工作台页面的变量作用域

HTML：jsb-23-02.html

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Variable scope</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-23-02.js"></script>
   </head>
   <body>
      <h1>Variable scope</h1>
      <h3>Charlie Brown has a global dog (Snoopy), a local dog (Gromit),
          a global toy (Gumby), and a local toy (Pokey).</h3>
      <div id="placeHolder"></div>
   </body>
</html>
```

JavaScript: jsb-23-02.js

```
// initialize when the page has loaded
addEvent(window, "load", testValues);

var newElem;
var newText;

var toyGlobal = "Gumby";
var aBoy = "Charlie Brown";
var hisDog = "Snoopy";

function showLocal()
{
   var toyLocal = "Pokey";
   return toyLocal;
}

function showGlobal()
{
   newElem = document.createElement("div");
   newElem.className = "objProperty";
   newText = document.createTextNode("Global version of hisDog is intact: "
             + hisDog);
      // just picked up a global variable value instead
      // of the local variable in the calling function
   newElem.appendChild(newText);
   placeHolderElement.appendChild(newElem);
}

function testValues()
{
   // Dangerous ground here -- declaring a global variable in a function
   placeHolderElement = document.getElementById("placeHolder");

   // Now declare the local variable
   var hisDog = "Gromit"; // initializes local version of "hisDog"
```


更多资源请访问稀酷客(www.ckook.com)

```
        if (placeHolderElement)
        {
            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode("aBoy is: " + aBoy);
               // just picked up a global variable value
            newElem.appendChild(newText);
            placeHolderElement.appendChild(newElem);

            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode("His toyGlobal is " + toyGlobal);
            newElem.appendChild(newText);
               // just picked up another global variable value
            placeHolderElement.appendChild(newElem);

            // Do not bother with toyLocal here because it will throw undefined

            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode(
                      "toyLocal value returned from the showLocal function is: "
                      + showLocal() );
            newElem.appendChild(newText);
            placeHolderElement.appendChild(newElem);

            newElem = document.createElement("div");
            newElem.className = "objProperty";
            newText = document.createTextNode("Local version of hisDog is: " + hisDog);
            newElem.appendChild(newText);
               // just picked up another global variable value
            placeHolderElement.appendChild(newElem);

            // now call another function that does not set the variable hisDog
            // and display the value. we'll see that the global value is intact
            showGlobal();
        }
    }
```

这个页面定义了许多变量，有全局变量和局部变量，它们分布在 JavaScript 文件中。当载入页面时，会运行 testValues()函数，此函数记录了所有变量名的当前值。脚本接着提取函数中的一个值或多个值，JavaScript 的结果如下：

```
aBoy is: Charlie Brown
His toyGlobal is Gumby
toyLocal value returned from the showLocal function is: Pokey
Local version of hisDog is: Gromit
Global version of hisDog is intact: Snoopy
```

检查 JavaScript 文件中变量的初始化。在脚本的开头有几行代码没有包含在函数中，它们会立即执行。在这个部分，将第一个变量 toyGlobal 定义为函数定义之外的全局变量。全局变量的 var 关键字是可选的，但使用它有利于迅速找到初始化变量的位置。还定义了另外两个全

更多资源请访问稀酷客(www.ckook.com)

局变量：aBoy 和 hisDog。

接着创建两个小函数，showLocal()函数定义了 toyLocal 变量，只有该函数中的语句才能使用它。showGlobal()函数定义了另一个显示变量值的函数(稍后解释其中的变量)。

还有一个 testValues()函数，它是 onload 事件处理程序。首先注意，该函数声明 placeHolderElement 变量时没有使用 var 关键字。尽管 placeHolderElement 是在函数中声明的，但它仍是一个全局变量。testValues()函数调用 showGlobal()函数，来使用 placeHolderElement 变量。为什么这个变量声明没有放在脚本的立即执行部分？placeHolderElement 变量是使用这些函数的 HTML 页面上的一个元素引用。只有页面加载完毕，placeHolderElement 变量才能成功引用该元素。确保在声明变量前加载页面的唯一方式是，把变量声明放在 onload 事件处理程序中。因为 testValues()函数是 onload 事件处理程序，所以使这个变量可用于其他函数的唯一方式是在此将其声明为全局变量。

另外，在 testValues()函数中重用了 hisDog 变量名(用于演示)。在函数中用 var 语句初始化 hisDog，就告诉 JavaScript 创建一个单独的局部变量，其作用域是函数内部。这种初始化不干扰同名的全局变量，但可能会使脚本编写者感到迷惑。

testValues()函数中的语句试图收集分散在 JavaScript 文件中的变量值，即使在该函数内部，JavaScript 也可以直接提取出全局变量值，包括在立即执行部分定义的变量。但 JavaScript 不能提取在 showLocal()函数中定义的局部变量，toyLocal 变量是该函数的私有变量。如果试图在 showGlobal()函数中引用 showLocal()函数的变量值，就会得到一个错误信息，说明该变量名未定义。从 showLocal()函数的外部看，这是正确的。若需要该变量值，可让函数给调用语句返回一个值，就像 testValues()函数一样。

在 testValues()函数的开头，脚本顺利读取 aBoy 全局变量的值。但由于在该函数内部初始化了另一个 hisDog 变量，所以这个局部版本只能用于 testValues()函数。如果在函数内部重新分配一个全局变量，就不能在函数内部访问该全局变量。

为了证明全局变量并未改变(在 testValues()函数内重用了该变量名)，testValues()函数调用了 showGlobal()函数。showGlobal()函数会显示 hisDog 变量的值。因为目前不在 testValues()函数内，所以该函数显示的是全局变量 hisDog 的值。Charlie Brown 和他的狗 Snoopy 组合起来了。

这种变量作用域机制的好处在于，可在任何函数中重用“可丢弃”的变量名。例如，可以在每个有循环的函数内部使用 i 循环计数变量(事实上，可以在同一个函数的多个 for 循环中重用它，因为每次循环开始时，都会重新初始化该值)。如果向函数传递参数，就可以为这些参数变量指定相同的名称，以保持一致。例如，一般将整个 form 对象引用作为参数传给函数(在事件处理程序中使用 this.form 参数)。而得到这些对象的任何函数都可以在参数中使用变量名 form：

```
function doSomething(form)
{
   // statements
}
// ...
<input type="button" value="Do Something" onclick="doSomething(this.form)">
```

若页面的 5 个按钮将其 form 对象作为参数传递给 5 个不同的函数，每个函数都将 form(或其他值)赋给参数值。

更多资源请访问稀酷客(www.ckook.com)

最好只对“可丢弃”的变量重用变量名。此时，变量是函数的局部变量，不存在与全局变量混淆的可能性。但重用全局变量名是函数中的特殊情况，肯定会导致错误，引起混乱。

警告：
局部重用全局变量名很复杂，会在 JavaScript 代码中带来难以查找的错误。局部变量会临时隐藏全局变量，且不发出任何警告。必须确保不在函数中把全局变量名重用为局部变量。同样，在函数中声明全局变量也会在 JavaScript 代码中带来难以查找的错误。

一些程序员设计了命名规范，以避免将全局变量用作局部变量。最常用的机制是在全局变量名前加小写字母 g。在程序清单 23-2 的例子中，可将全局变量命名为：

```
gToyGlobal
gABoy
gHisDog
gPlaceHolderElement
```

如果定义局部变量，就不要使用字母 g。另一个类似的机制是使用下划线字符(_)，而不是在全局变量名前加一个 g。在程序清单 23-2 的例子中，可将局部变量命名为：

```
_ToyLocal
_HisDog
```

任何用于防止在不同的作用域内重用变量名的机制都很好，只要该机制有效即可。能清楚地标识全局变量的机制也是这样。

在多框架或多窗口环境中，脚本还可以从当前载入浏览器的任何其他文档中访问全局变量，这个级别的访问详见第 27 章。

变量作用域规则同样适用于嵌套函数。所有嵌套的函数都能使用在外部函数中定义的任何变量(包括参数变量)。但如果在嵌套函数内定义新的局部变量，外部函数就不能使用这些变量。另外，嵌套函数能够将值返回到外部函数中调用嵌套函数的语句中。

23.2.3　参数变量

函数以参数形式接收数据时，其值可以是数据的副本(普通的数据值)或实际对象的引用(如 form 对象)。后一种情况下，函数把对象接收为参数时，可以改变对象的可更改属性，如下面的例子所示：

```
function validateCountry(form)
{
   if (form.country.value == "")
   {
      form.country.value = "USA";
   }
}
```

所以，只要把对象引用传递为函数参数，就要注意，改变传递给函数的对象会影响实际的对象。

更多资源请访问稀酷客(www.ckook.com)

事实上，若函数需要从传递的数据中提取属性或方法的结果，如对象属性或子字符串，最好在函数开始时进行。在函数的开头为函数以后要使用的所有数据初始化足够多的变量，可以为数据块指定有意义的名称，这样在函数进行处理时就不需要使用长引用，如使用 inputStr 替代 form.entry.value。目前的计算机速度很快，这么做时不需要考虑计算机的速度，而只需考虑人的速度——使代码的可读性和可维护性更高。下面列举一个简单的例子：

```
function updateContactInfo(form)
{
   var firstName = form.firstname.value;
   // or the preferred way:
   var lastName = document.getElementById("lastname").value;
   var address1 = document.getElementById("addr1").value;
   var address2 = document.getElementById("addr2").value;
   var phone = document.getElementById("phone").value;
   var email = document.getElementById("email").value;

   // Process contact info using local variables
}
```

注意在这个例子中，表单域信息先存储在局部变量中，然后，局部变量传送联系人的假想更新信息。函数的其余部分可以使用局部变量，代替冗长难用的表单域。

23.2.4　递归函数

函数可以调用自己，这个过程称为递归。比较经典的递归程序是计算阶乘，如 4 的阶乘是 4*3*2*1，参见程序清单 23-3。

这个函数的第三行语句调用自身，把下一个较小的 n 值传递为参数。函数执行时，会进一步调用自身。JavaScript 观察中间值，处理嵌套表达式的最后结果。如果设计得当，递归函数将最终停止调用自身，程序流最终返回到初始函数调用。

在某种意义上，递归函数很危险，很容易陷入死循环。因此，仔细测试非常重要。实际上，要确保递归循环是有限的：即递归次数是有限的。在程序清单 23-3 中，n 的初始值就限制了递归循环次数。如果没有指定这个限制，递归就可能超出浏览器内存的限制，甚至导致浏览器崩溃。

程序清单 23-3　使用递归的 JavaScript 函数

```
function factorial(n)
{
   if (n > 0)
   {
      return n * (factorial(n-1));
   } else
   {
      return 1;
   }
}
```


更多资源请访问稀酷客(www.ckook.com)

23.2.5　创建函数库

开始编写脚本函数时，要注意使函数通用化的方式(使函数可以在其他例子中重用，而不管页面的对象结构如何)，最有可能进行这种处理的函数是处理某种有效性检查(请参阅配书光盘中的第 46 章)、数据转换或迭代数学问题的函数。

要使函数通用化，不要通过名称引用具体的对象，因为对象名称一般因文档而异。另外，最好编写将对象名称作为参数的函数。例如，要编写一个将文本对象作为参数的函数，函数不必知道对象的形式或名称，就可以提取对象的数据或调用对象的方法。例如 factovial 函数，它现在是程序清单 23-4 中整个文档的一部分。

程序清单 23-4　调用通用函数

HTML: jsb-23-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Variable Scope Trials</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-23-04.js"></script>
   </head>
   <body>
      <h1>Variable Scope Trials</h1>
      <form id="theForm" action="calc-factorial.php" method="get">
         <p>
            <label for="visitorInput">Enter an input value:</label>
            <input type="text" id="input" name="input" value="0">

            <input type="submit" value="Calc Factorial">

            <label for="output">Results:</label>
            <input type="text" id="output" value="">
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb-23-04.js

```
// initialize when the page has loaded
addEvent(window, "load", initialize);

function initialize()
{
   // intercept form submission
   addEvent(document.getElementById("theForm"), "submit", calcFactorial);
}

function calcFactorial(evt)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    // consolidate event handling
    if (!evt) evt = window.event;

    // plug transformed input to output field
    var oInput = document.getElementById("input");
    var oOutput = document.getElementById("output");
      // if they both exist
      if (oInput && oOutput)
      {
        oOutput.value = factorial(oInput.value);
      }

    // cancel form submission
      // W3C DOM method (hide from IE)
      if (evt.preventDefault) evt.preventDefault();
      // IE method
      return false;
}

function factorial(n)
{
      if (n > 0)
      {
        return n * (factorial(n - 1));
      }
      else
      {
        return 1;
      }
}
```

该函数被设计成通用函数，只接收输入参数 n。在表单中，onsubmit 事件处理程序把一个表单域的输入值发送给 factorial()函数，返回值赋给表单的输出域。factorial()函数忽略文档中的表单、域或按钮。若其他脚本需要这个函数，就可以复制它，然后粘贴到相应脚本中，因为这个函数已经测试过。任何通用函数都是个人脚本库的一部分，可以从中借用代码，来节省将来编写脚本的时间。

不必总是通用化函数。有时在脚本代码中，必须引用 JavaScript 或自定义对象。但若经常编写实现同样任务的函数，可将代码通用化，并把结果放到准备好的函数库中。还要考虑将这些可重用的库函数放在外部的.js 库文件中，如本章和其他章节所述。请参阅第 4 章在多个文档中共享实用函数的简便方法。

23.2.6　封闭区间

许多脚本编写人员都感到困惑的一个主题是封闭区间(closure)，在一个函数中声明另一个函数时，就可能会遇到这个问题。在 JavaScript 的核心，封闭区间表示，可使函数定义的局部变量是活动的，即使函数已经执行完毕(这通常表示局部变量的生命期结束了)也同样如此。函数返回后，其局部变量仍是活动的，这似乎很奇怪，这怎么可能呢？请看下面的例子：

更多资源请访问稀酷客(www.ckook.com)

```
function countMe()
{
  var count = 1;
  var showCount = function() { alert(count); }
  count++;
  return showCount;
}
```

在这段代码中，注意函数如何返回赋给 showCount 变量的内部函数。在调用 countMe()函数时，会接收内部函数的一个引用，该函数显示 count 变量值。这没有问题，但内部函数作用于 countMe()函数的局部变量(count)。

为了查看封闭区间的作用，看调用 countMe()函数的代码：

```
var countamatic = countMe();
countamatic();
```

第一行调用 countMe()函数，从而创建局部变量 count，传送内部函数的引用，并存储在 countamatic 变量中。在 countMe ()函数中，还给局部变量 count 加 1。如果不考虑封闭区间，就显然有问题，因为第 1 行定义的 countamatis()函数现在显示明显超出作用域的变量值。但是，JavaScript 很聪明，它使 count 变量在封闭区间中是活动的，仍然允许 countamatic()函数访问它。这样，第二条语句就显示包含数字 2 的警告。

如果还是不理解封闭区间，就应记住，只要在一个函数内指定另一个函数，就会创建一个潜在的封闭区间。实际上，在内部函数的作用域之外传递内部函数的引用时，就是在利用封闭区间。

当然，封闭区间揭示了在 JavaScript 中操纵作用域的一条秘密通道。它们有什么好处呢？在 Ajax 应用程序中大量使用了封闭区间，因为 Ajax 编程通常使用封闭区间，来消除在使用 this 关键字时固有的局限性。

交叉引用：
Ajax 详见第 39 章。

尽管可以把与封闭区间相关的所有内容都放到 Ajax 一章中，但仍可以演示一个简单而实用的应用程序，说明封闭区间如何帮助执行看起来不可能完成的任务。假设要设置一个计数器，在某个时间间隔后调用一个函数。这似乎没问题，只要创建一个函数，将其引用传递给 setTimeout()函数即可。其实，要给这个函数传递两个参数，为什么？

除非使用匿名函数进行设计，否则，给一个函数传递另一个函数的引用时，例如在调用 setTimeout()函数时指定计数器事件处理程序，将无法使用参数。现在看一下这段代码，它使用封闭区间来解决这个问题：

```
function wakeupCaller(name, roomnum)
{
   return
   (
      function()
      {
```


PDG
更多资源请访问稀酷客(www.ckook.com)

```
            alert("Call " + name + " in room #" + roomnum + ".");
        }
    );
}
```

在参数化的函数中放置一个 0 参数函数，就可以创建接收参数的计数器处理程序了。下一步创建使用封闭区间的实际函数引用：

```
var wakeWilson = wakeupCaller("Mr. Wilson", 515);
```

现在，就给函数传递了参数，并接收到 0 参数函数的引用，随后，可以将其传递给 setTimeout() 函数：

```
setTimeout(wakeWilson, 600000);
```

感谢封闭区间，Wilson 先生现在有了叫醒程序了。

警告：

在深入研究封闭区间，在代码中使用它之前，先要注意，如果使用不当，它们会带来处理起来十分棘手的错误。大量使用涉及 DOM 对象引用的封闭区间，如果在不需要这些对象时没有释放它们(设置为 null)，也可能导致内存泄漏(浏览器占用的内存不断增加)。在使用封闭区间之前，一定要花时间仔细研究它们。

23.3 使用面向对象的 JavaScript 创建自定义对象

本书前面的章节提到，浏览器文档对象模型可以方便地组织浏览器窗口及其文档的所有信息。下面具体学习如何使用标准对象，来访问浏览器窗口和文档的不同方面。在以前进行的脚本编程中，还有一个方面不大明确：JavaScript 可在内存中创建自己的对象，这些对象具有用户自己定义的属性和方法。这些对象不是页面上的用户界面元素，而是可能包含数据和脚本函数(像方法那样)的对象类型，用户可在浏览器窗口中查看其结果。

实际上，第 18 章讨论的数组已经显示了这种能力。数组是有序的数据集合，而对象一般包含不同类型的数据，它不一定是有序的数据集合——虽然脚本能在非常类似于数组的结构中使用对象。对象还能加入任意数量的自定义函数，作为对象的方法。你还可以完全控制对象的结构、数据和行为。

在 JavaScript 代码中使用自定义对象来工作就称为面向对象编程(object-oriented program ming，OOP)。OOP 已经推出很长时间了，在其他编程语言中应用得很成功，比如 C++和 Java。然而，JavaScript 的脚本特性使 OOP 在 JavaScript 中的应用有些缓慢。即便如此，支持自定义对象是现代 JavaScript 浏览器的标准组成，如果可能，可以谨慎地使用它。

注意：

严格来讲，在技术上，JavaScript 并非真正的面向对象的语言，而是一种基于对象的语言。面向对象与基于对象之间的不同是很大的，与对象的扩展方式相关。即便如此，在概念上，JavaScript 对对象的支持与真正的 OOP 语言类似，使用 OO 术语讨论 JavaScript 也是可行的。本章后面将讲解一

更多资源请访问稀酷客(www.ckook.com)

些具体的对象特征，它们让 JavaScript 更接近 OO 语言。更有趣的是，ECMAScript 现在支持类、接口等的概念，它们更接近于传统的 OO 方式，这意味着，JavaScript 的以后版本也开始支持这些概念。

在应用程序中，知道什么时候使用自定义对象(而不是数组)并没有什么秘密。使用的自定义对象越多，对自定义对象的工作方式理解得就越多，在传送数据的脚本中就会越多地考虑它们，当对象有一个或多个与其关联的方法时尤其如此。这种方式当然不适合初学者，但是，在有了一定的 JavaScript 经验后，建议仔细研究一下自定义对象。

23.3.1　对象的具体细节

实际上，JavaScript 中的对象只是属性的集合。属性可以采用数据类型、函数(方法)或其他对象的形式。其实，更容易把自定义对象看作一个值的数组，该数组中的每个元素都映射到一个属性上(数据类型、方法或对象)。方法可以是对象的属性吗？稍后讨论这个问题，现在仅接受这个结论。包含在对象中的函数称为方法。方法与其他函数没有什么不同，只是方法要用在对象中，因此方法可以访问对象的属性。属性和方法之间的这种关系是 OO 的一个核心概念。

对象使用特殊的函数创建，即构造函数。它确定对象的名称，构造函数与对象同名。下面是构造函数的例子：

```
function Alien()
{
}
```

尽管这个函数不包含任何代码，但它是创建 Alien 对象的基础。构造函数可以视为一幅蓝图，用来创建实际的对象。构造函数的命名规则是：在传统上，构造函数名采用 TitleCase 的形式，引用对象的每个实例变量采用 camelCase 形式。下面是使用构造函数创建对象的例子：

```
var myAlien = new Alien();
```

这里使用 new 关键字和构造函数来创建 JavaScript 对象。

1. 创建自定义对象的属性

如前所述，属性是对象的关键，那么为什么要给自定义对象创建属性？自定义对象的属性在构造函数中用 this 关键字创建，如下所示：

```
function Alien()
{
   this.name = "Clyde";
   this.aggressive = true;
}
```

this 关键字用来引用当前对象，在这里，当前对象就是构造函数创建的对象。所以，要使用 this 关键字来标识对象的新属性。这个例子的唯一问题是，它使用 Alien()构造函数创建的所有 alien 对象都有相同的名字和行为。修正方法是，把属性值传递给构造函数，使每个 alien 都可以在创建时自定义：

```
function Alien(name, aggressive)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
    this.name = name;
    this.aggressive = aggressive;
}
```

现在，可以创建不同的 alien，每个 alien 都有唯一的属性值：

```
var alien1 = new Alien("Ernest", false);
var alien2 = new Alien("Wilhelm", true);
```

要获得对象的属性(在创建对象后读写它们)，可使用 DOM 对象的句点语法。要更改对象的 name 属性，语句如下：

```
alien1.name = "Julius";
```

2. 创建自定义对象的方法

实际上，属性只是 JavaScript OO 的一半内容，另一半是方法，方法是与对象相关的函数，通过它们可以访问对象的属性。下面列举一个方法的例子，它可以用于 alien 类：

```
function attack()
{
    if (this.aggressive)
    {
        // Do some attacking and return true to indicate that the attack
        // commenced
        return true;
    }
    else
    {
        // Don't attack and return false to indicate that the attack didn't
        // happen
        return false;
    }
}
```

注意，attack()方法引用 this.aggressive 属性，来确定是否发生攻击。现在，唯一没有讲解的内容就是 attack()方法和 alien 对象之间的连接。如果没有连接，this 关键字就没有意义，因为没有相关的对象。下面是连接的方式：

```
function Alien(name, aggressive)
{
    // properties
    this.name = name;
    this.aggressive = aggressive;
    // methods
    this.attack = attack;
}
```

这段代码清楚地说明，方法其实就是属性，方法的声明方式与属性相同，所以它们假装成属性。这段代码创建了新属性 attack，对 attack()方法指定引用。注意，attack()函数通过引用指

更多资源请访问稀酷客(www.ckook.com)

定，且没有括号，这非常重要。接着创建 alien 对象，调用其 attack()方法：

```
var alien1 = new Alien("Ernest", false);
alien1.attack();
```

上面为 alien 对象定义 attack()方法的方式，使其位于全局名称空间中，根据 OO 最佳实践方式(参见本章后面对封装的讨论)，这不是件好事。更好的方法是把它内嵌在构造函数中：

```
function Alien(name, aggressive)
{
   // properties
   this.name = name;
   this.aggressive = aggressive;
   // methods
   this.attack = function(name, aggressive)
   {
      if (this.aggressive)
      {
         // Do some attacking and return true to indicate that
         // the attack commenced
         return true;
      }
      else
      {
         // Don't attack and return false to indicate that
         // the attack didn't happen
         return false;
      }
   };
}
```

前面讲解了 JavaScript 对象的基础知识，现在可以学习更完整的例子了，参见下一节。这个例子在本章的后面会逐步改进，读者也在这个过程中逐步提高自定义对象的建立技巧。

23.3.2　OO 例子：行星对象

熟悉了第 18 章建立的行星数组后，本章介绍如何使用建立为自定义对象的数据。本节扩展例子的目的是建立太阳系中九大行星的弹出列表，并显示所选择行星的属性。

本章没有建立数组来存放数据，而是为每个行星建立一个对象。每个行星设计了 5 个属性和一个方法，这些属性是名称、直径、与太阳的距离、年的长度和天的长度。为了给这些对象赋予智能，程序清单 23-5 为每个对象增加了在下拉列表的下面显示其数据的功能。可以方便地定义一个函数，来确定如何处理这些行星对象，而不是分别定义 9 个不同的函数。在对象中使用函数时，函数其实就是方法。

程序清单 23-5　OO 行星数据的显示

HTML: jsb-23-05.html

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>OO Planetary Data Presentation</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-23-05.js"></script>
   </head>
   <body>
      <!-- Display the Planet choices to the visitor -->
      <h1>The Daily Planet</h1>
      <form>
         Select a planet to view its planetary data:
         <select name="planetsList" onchange="doDisplay(this)">
            <option>Mercury</option>
            <option>Venus</option>
            <option selected="selected">Earth</option>
            <option>Mars</option>
            <option>Jupiter</option>
            <option>Saturn</option>
            <option>Uranus</option>
            <option>Neptune</option>
            <option>Pluto</option>
         </select>
      </form>
   </body>
</html>
```

JavaScript: jsb-23-05.js

```
// definition of Planet object constructor
function Planet(name, diameter, distance, year, day)
{
   // properties
   this.name = name;
   this.diameter = diameter;
   this.distance = distance;
   this.year = year;
   this.day = day;
   // methods
   this.showPlanet = function()
   {
      var result = "<html><body><center><table border='2' cellpadding ='2'>";
      result += "<caption align='top'>Planetary data for <b>"
            + this.name + "</b></caption>";
      result += "<tr><td align='right'>Diameter</td><td>"
            + this.diameter + "</td></tr>";
      result += "<tr><td align='right'>Distance from Sun</td><td>"
            + this.distance + "</td></tr>";
      result += "<tr><td align='right'>One Orbit Around Sun</td><td>"
            + this.year + "</td></tr>";
      result += "<tr><td align='right'>One Revolution (Earth Time)</td><td>"
```


更多资源请访问稀酷客(www.ckook.com)

```
                + this.day + "</td></tr>";
        result += "</table></center></body></html>";
        // display results
        document.write(result);
        document.close();
    };
}

// create new Planet objects, and store in a series of reference variables
// 'new' will create a new Planet instance and insert parameter data into object
var mercury = new Planet("Mercury","3100 miles", "36 million miles",
                         "88 days", "59 days");
var venus = new Planet("Venus", "7700 miles", "67 million miles",
                       "225 days", "244 days");
var earth = new Planet("Earth", "7920 miles", "93 million miles",
                       "365.25 days","24 hours");
var mars = new Planet("Mars", "4200 miles", "141 million miles",
                      "687 days", "24 hours, 24 minutes");
var jupiter = new Planet("Jupiter","88,640 miles","483 million miles",
                         "11.9 years", "9 hours, 50 minutes");
var saturn = new Planet("Saturn", "74,500 miles","886 million miles",
                        "29.5 years", "10 hours, 39 minutes");
var uranus = new Planet("Uranus", "32,000 miles","1.782 billion miles",
                        "84 years", "23 hours");
var neptune = new Planet("Neptune","31,000 miles","2.793 billion miles",
                         "165 years", "15 hours, 48 minutes");
var pluto = new Planet("Pluto", "1500 miles", "3.67 billion miles",
                       "248 years", "6 days, 7 hours");

// invoke Planet object method corresponding to the
// index of the visitor's choice
function doDisplay(popup)
{
   i = popup.selectedIndex;
   eval(popup.options[i].text.toLowerCase() + ".showPlanet()");
}
```

第一个任务是定义对象的构造函数，它执行几项重要任务。一方面，这个函数中的所有代码都用于建立自定义对象的结构：用于存储和提取数据的属性，以及对象可以调用的方法。构造函数的名称就是以后用于创建对象新实例的名称。因此，选择能真正反映对象本质的名称非常重要。另外，因为可能需要将一些数据放在函数的属性中，以加载一个或多个对象实例，并准备供页面的用户使用，所以函数定义还给在这个对象中定义的每个属性包含一个参数。

在对象中，使用 this 关键字将用作参数的数据赋给命名的属性。这个例子对传入的参数变量和属性使用相同的名称。这主要是为了方便(也很常用)，也可为变量和属性指定任何名称，并使用任何方式连接它们。在 planet()构造函数中，为每个对象实例设置了 5 个属性，而不管每个属性是否包含数据(未分配数据的参数值为 null)。

planet()构造函数中的最后一项是 showPlanet()方法的定义。与以前的每个 JavaScript 方法相同，调用方法必须使用对象的引用、句点和后跟一对括号的方法名，稍后介绍该语法。

更多资源请访问稀酷客(www.ckook.com)

下一个语句块根据 planet()构造函数中建立的定义来创建各个对象。与数组相同，使用赋值语句和关键字 new 来创建对象，名称指定为真实的行星名(Mercury 对象引用名是 Mercury 行星对象)。

创建新对象时，并没有考虑对象及其属性的内存空间(与当前文档有关)。在内存中创建的对象称为实例。在这个脚本中，创建了 9 个对象实例，每个实例都有不同的属性集合。注意没有传送与 showPlanet()方法相关的参数，这里省略参数是可行的，因为该方法在对象定义中的规范指出，脚本会自动将该方法与它创建的行星对象的每个实例相关联。这很重要，因为它把函数(方法)链接到对象上，以便访问对象的属性。

最后一个函数定义是 doDisplay()，它是一个事件处理程序，只要用户在行星列表中进行选择，就会调用该函数。它的第一个语句提取所选项的索引值，脚本使用索引值获取文本，但这里需要一点儿技巧，因为需要使用该文本字符串作为变量名——行星名，才能将它追加到 showPlanet()方法调用上。为了把不同的数据类型合在一起，可使用 eval()函数(其完整论述参见第 24 章)。在圆括号内部提取行星名称的字符串，将它改为小写形式，并连接一个字符串，来完成对象的 showPlanet()方法的引用。eval()函数计算该字符串，将其转换为有效的方法调用。所以，如果用户从下拉列表中选择 Jupiter，方法调用就变成 Jupiter.showPlanet()。

现在来分析 showPlanet()方法的定义。当用户选择行星 Jupiter 时，该方法运行，它只处理 Jupiter 对象。所以，showPlanet()中对 this.propertyName 的所有引用只指向 Jupiter。在 Jupiter 对象中，this.name 唯一可能的值是赋给 Jupiter 的 name 属性值，这同样适用于从方法中提取的其他属性。

23.3.3 进一步的封装

在脚本编程中使用对象的一个优点在于，对象内部的所有“连接”(属性和方法)都在这个对象的局部变量作用域内定义。这个概念就是“封装”。这样，在构造函数内部，使用 this 关键字创建对象的属性，并对它们赋值时，属性名以及与对象相关的任何代码都是那个对象私有的，不会与全局变量或对象名冲突。

在大型脚本编程项目中，尤其是涉及多个程序员的项目中，全局名很容易发生冲突。例如，在 Internet Explorer 中，所有 DOM 对象的 ID 都是全局变量空间的一部分，因为浏览器允许脚本通过 ID 引用元素对象(而 W3C DOM 使用 document.getElementById()方法)。随着项目规模和复杂性的增长，在全局变量的命名空间中，避免对象名称的冲突(包括函数定义)显得越来越重要。

例如，在程序清单 23-5 中，showPlanet()方法在局部名称空间中定义。它是对象的局部方法，因为该方法在 planet()构造函数中定义。如果在 showPlanet()方法在 planet()构造函数外部定义(如本章前面的一个例子所示)，showPlanet()方法就可能与别处使用的全局对象名冲突。

23.3.4 创建对象数组

在程序清单 23-5 中，每个行星对象都赋给一个全局变量，其名称是行星名。如果不熟悉自定义对象，就可能以为这很不错，因为很容易看出每个变量所代表的对象。但是，如 doDisplay()函数所示，要通过名称访问对象，需要使用 eval()函数，将字符串转换为有效的对象引用。尽管在这个简单例子中这不太重要，但 eval()函数在 JavaScript 中不是特别有效。若使用了 eval()函数，最好寻找提高效率的方式，以直接引用对象。给这个应用程序完成这个任务的方式是将

更多资源请访问稀酷客(www.ckook.com)

对象放在数组中。

在程序清单 23-5 中，为了将自定义对象赋给数组，首先建立空数组，然后把调用每个对象构造函数的结果赋给一个数组项。修改后的代码部分如程序清单 23-6 所示。外部 JavaScript 文件中的这部分代码只列出了代码中有变化的内容。程序清单 23-5 和 23-6 中的 HTML 是完全相同的，所以这里只列出外部 JavaScript 文件中的一部分。

程序清单 23-6 行星对象数组

JavaScript: partial of jsb-23-06.js

```
// create new Planet objects, and store in an array of reference variables
// 'new' will create a new Planet instance and insert parameter data into object
var planets = new Array
(
   new Planet("Mercury","3100 miles", "36 million miles",
              "88 days", "59 days"),
   new Planet("Venus", "7700 miles", "67 million miles",
              "225 days", "244 days"),
   new Planet("Earth", "7920 miles", "93 million miles",
              "365.25 days","24 hours"),
   new Planet("Mars", "4200 miles", "141 million miles",
              "687 days", "24 hours, 24 minutes"),
   new Planet("Jupiter","88,640 miles","483 million miles",
              "11.9 years", "9 hours, 50 minutes"),
   new Planet("Saturn", "74,500 miles","886 million miles",
              "29.5 years", "10 hours, 39 minutes"),
   new Planet("Uranus", "32,000 miles","1.782 billion miles",
              "84 years", "23 hours"),
   new Planet("Neptune","31,000 miles","2.793 billion miles",
              "165 years", "15 hours, 48 minutes"),
   new Planet("Pluto", "1500 miles", "3.67 billion miles",
              "248 years", "6 days, 7 hours")
);
}
```

这种方法最大的优势是修改过的 doDisplay()函数，它可以直接使用 select 元素的 selected Index 索引值：

```
function doDisplay(popup)
{
   i = popup.selectedIndex;
   planets[i].showPlanet();
}
```

有这么多相似的对象，它们需要存储为数组。这里使用下拉列表的 selectedIndex 值定位相应的数组元素。

更多资源请访问稀酷客(www.ckook.com)

23.3.5 利用嵌套对象

使用自定义对象的一个强大技术是在一个对象中嵌入另一个对象。此时，放入其中的对象称为嵌入的对象或对象属性。下面扩展行星的例子，以帮助你理解使用自定义对象属性的含义。

假设要在行星页面中加入每个行星的照片，每张照片有一个照片文件的 URL；每张照片还包含其他信息，如版权标记和引用号，这些信息都显示在页面上。处理这种额外信息的一种方式是，为照片数据库单独创建一个对象定义，如下所示：

```
function Photo(name, URL, copyright, refNum)
{
   this.name = name;
   this.URL = URL;
   this.copyright = copyright;
   this.refNum = refNum;
}
```

接着为每个图片创建 photo 对象，其定义如下所示：

```
mercuryPhoto = new Photo("Planet Mercury", "/images/merc44.gif",
                         "(c)1990 NASA", 28372);
```

为将 photo 对象与 planet 对象关联，需要改变 planet 构造函数，使其再包含一个对象属性，新的 planet 构造函数如下：

```
function Planet(name, diameter, distance, year, day, photo)
{
   this.name = name;
   this.diameter = diameter;
   this.distance = distance;
   this.year = year;
   this.day = day;
   this.showPlanet = showPlanet;
   this.photo = photo; // add photo property
}
```

创建 photo 对象后，就可以传递与 planet 对象有关的 photo 对象，来创建每个 planet 对象：

```
// create new planet objects, and store in a series of variables
Mercury = new Planet("Mercury", "3100 miles", "36 million miles",
                     "88 days", "59 days", mercuryPhoto);
```

要访问 photo 对象的属性，脚本必须建立一个引用，使其与 planet 对象连接起来：

```
copyrightData=Mercury.photo.copyright;
```

这种内嵌对象的潜能很大。例如，可在单个文档中给在线分类嵌入所有的复制元素和图像 URL。当用户选择要浏览的项(或按顺序遍历它们)时，新写的 JavaScript 页面会立即显示有关信息，这只需要下载图像——除非图像进行了预缓存，参见第 29 章讨论的 image 对象。这种情况下，每项工作都立刻完成，而不用一页一页的等待。

更多资源请访问稀酷客(www.ckook.com)

这种对象内包含对象的结构与关系数据库有类似之处。多个对象可以拥有相同的子对象作为其属性，就像多个商业合同具有相同的公司对象属性。

23.3.6　创建对象的最新方法

现代浏览器还可以从创建新对象的快捷语法中受益，可在一对花括号内设置属性名和它们的值，并将整个结构赋给一个变量，该变量名就是对象名。下面的脚本显示了组织这种对象构造函数的方法：

```
var Earth = {diameter:"7920 miles", distance:"93 million miles", year:"365.25",
          day:"24 hours", showPlanet:showPlanet};
```

用冒号连接“名/值”对，用逗号分开多个“名/值”对。“名/值”对的值还可以是数组(使用 [...] 构造函数)或嵌套对象(使用另一对花括号)。事实上，可将数组和对象嵌套在内容中，来创建复杂的对象。总之，对于脚本控制来说，这是在页面内嵌入数据的一种简洁方式。若任务需要使用 CGI、XML 和数据库技术，就应考虑使用服务器程序，将 XML 数据转换为这种简洁的 JavaScript 版本，每个 XML 记录都是一个 JavaScript 对象。对于多个记录，将带花括号的对象定义赋给数组项，然后，客户端的脚本就可以遍历数据，生成 HTML 文件，以不同的形式来显示数据，按照不同的标准进行排序(JavaScript 数组具有排序的功能)。

23.3.7　定义对象属性的提取器和设置器

给对象创建提取器和设置器(在 JavaScript 中一般称为访问器)参见第 25 章，它们在 W3C DOM 对象中描述。下面列举一个在自定义对象中使用提取器和设置器的例子。

设置器的作用是在对象中为原型分配一个新值，同时根据需要检查其有效性。设置器指定，赋予属性的新值如何应用于对象。我们为要从对象中读取的每个属性定义一个提取器。在对象中，一般要为对象的每个属性定义提取器和设置器。如果需要只读属性，就应只定义提取器方法。

如程序清单 23-7 所示，TitleCaseName 对象属性中的提取器和设置器都编写为方法，并使用对应的属性名。读写对象的属性值都包含复杂的处理。这称为对象字面记号。Firefox 2.0+、Safari 3.0+、Chrome+和 Opera 9.5+支持这个集合。但 IE8 不支持自定义对象的提取器和设置器。这个程序清单只是外部 JavaScript 文件中的一部分，它只列出了对程序清单 23-6 中代码的修改。程序清单 23-5 和 23-7 中的 HTML 是相同的，所以这里只包含外部 JavaScript 文件中的一部分。

程序清单 23-7　使用提取器和设置器的 Planet 对象构造函数

JavaScript: partial of jsb-23-07.js

```
// object property with accessors using object literal notation
// its property 'name', is a String value that is TitleCased when set
var TitleCaseName =
{
   _name: null,

   get name()
   {
```


更多资源请访问稀酷客(www.ckook.com)

```
    return this._name;
  },

  set name(value)
  {
    this._name = value.substring(0,1).toUpperCase() + value.substring(1);
  }
};

// definition of Planet object constructor
function Planet(name, diameter, distance, year, day)
{
  // properties
  TitleCaseName.name = name;
  this.name = TitleCaseName.name;
  this.diameter = diameter;
  this.distance = distance;
  this.year = year;
  this.day = day;
  // methods
  this.showPlanet = function()
  {
    // . . .
  };
}
```

在行星例子中，要确保创建 Planet 对象时，其 name 属性采用 TitleCase 形式，而不管开发人员如何编写代码。此时就应使用设置器：提取 name 属性的当前值，转换其形式，再把新值应用于 name 属性。

程序清单首先定义了要在 TitleCaseName 对象中使用的提取器和设置器方法。注意没有使用 function 关键字。定义了局部属性后，就定义提取器和/或设置器。为了定义提取器，只需使用 get 关键字，后跟要读取的属性名。这一般会返回相应属性的值。这个例子，只定义了提取器，它读取并返回对象的_name 属性的当前值。在_name 中使用了下划线前缀，以表示这个属性是内部属性，不应在其包含对象的外部引用。

同样，用 set 关键字定义每个设置器。这里，设置器把参数转换为 TitleCase 形式，再把它赋给_name 属性。

最后是 Planet 对象的构造函数。注意这个函数的前两个属性。在这个例子中，对象实例化时为行星的 name 属性提供的值会“调用”提取器和设置器函数。在第一行中，把名称赋予 TitleCaseName，这会调用设置器，把值转换为 TitleCase 形式。第二行调用提取器，把 TitleCased 名称赋予 Planet 名称。

23.4　面向对象的概念

如前所述，JavaScript 是基于对象而不是面向对象的，JavaScript 没有使用 Java 等面向对象语言的类、子类和继承机制，而使用所谓的原型继承。该机制不仅用于内部对象和 DOM 对象，

更多资源请访问稀酷客(www.ckook.com)

还用于自定义对象。

23.4.1　增加原型

自定义对象通常用构造函数来定义，构造函数通常包含对象属性的初值，如下所示：

```
function Car(plate, model, color)
{
   this.plate = plate;
   this.model = model;
   this.color = color;
}
var car1 = new Car("AB 123", "Ford", "blue");
```

现代浏览器还提供了一个捷径：若没有为属性提供任何值，就设置其默认值(默认值为 null、0 或空字符串)。OR 操作符(||)可以让属性赋值语句给属性应用传递进来的值(有的话)，或者应用硬编码到构造函数中的默认值，所以可修改前面的函数，为属性提供默认值：

```
function Car(plate, model, colcr)
{
   this.plate = plate || "missing";
   this.model = model || "unknown";
   this.color = color || "unknown";
}
var car1 = new Car("AB 123", "Ford", "");
```

前面的语句运行后，对象 car1 的属性如下：

```
car1.plate      // value = "AB 123"
car1.model      // value = "Ford"
car1.color      // value = "unknown"
```

接着为构造函数的 prototype 属性添加新的属性，如：

```
Car.prototype.companyOwned = true;
```

任何已创建或将要创建的 Car 对象都会自动继承新的 companyOwned 属性及其值，还可以为每个 Car 对象重写 companyOwned 属性的值。但若不重写 Car 对象实例的 companyOwned 属性，Car 对象将自动继承 prototype.companyOwned 值的变化，这与 JavaScript 查找 prototype 属性值的方法有关。同样，程序清单 23-8 使用原型继续改进 Planet 对象。外部 JavaScript 文件中的这部分代码程序清单仅显示了程序清单 23-7 中代码的修改。程序清单 23-5 和 23-8 的 HTML 是相同的，所以这里只包含部分外部 JavaScript 文件中的代码。

程序清单 23-8　使用原型的 Planet 对象构造函数

JavaScript: partial of jsb-23-08.js

```
// definition of Planet object constructor
function Planet(name, diameter, distance, year, day)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    // properties
    TitleCaseName.name = name;
    this.name = TitleCaseName.name;
    this.diameter = diameter;
    this.distance = distance;
    this.year = year;
    this.day = day;
}

Planet.prototype =
{
    constructor : Planet,
    // methods
    showPlanet : function()
    {
        // . . .
    }
};
```

23.4.2　原型继承

脚本每次读写对象的属性时，JavaScript 都按指定的顺序来查找匹配的属性名，该顺序如下：

(1) 若给当前对象赋予了属性的局部值，就使用该值。

(2) 若没有局部值，就检查对象构造函数的属性原型值。

(3) 继续检查原型链，直到发现匹配的属性(包含一个值)或找到内部的 Object 对象为止。

所以，如果改变构造函数的 prototype 属性值，并没有重写构造函数实例的属性值，则 JavaScript 返回构造函数 prototype 属性的当前值。

23.4.3　嵌套对象和原型继承

开始嵌套对象时，特别是一个对象调用另一个对象的构造函数时，在原型继承链上就会增加一个节点。继续上面定义 Car 对象的例子，这里 Car 对象类似于根对象，它的两个属性由另外两类对象共享，一个对象是公司车辆，它需要根 Car 对象的属性(plate、model、color)，还增加了其他属性；另一个共享 Car 对象的对象用来表示停在公司车库的汽车，并有一个表示停泊车辆的属性，所以 Car 对象需要单独定义。

现在，看看停车记录的构造函数和基本 Car 对象的构造函数：

```
function Car(plate, model, color)
{
    this.plate = plate || "missing";
    this.model = model || "unknown";
    this.color = color || "unknown";
}
function CarInLot(plate, model, color, timeIn, spaceNum)
{
    this.timeIn = timeIn;
    this.spaceNum = spaceNum;
```


更多资源请访问稀酷客(www.ckook.com)

```
    this.carInfo = Car;
    this.carInfo(plate, model, color);
}
```

CarInLot 构造函数不仅为自己的每个属性(timeIn 和 spaceNum)赋值，还包含对 Car 构造函数的引用，该构造函数赋给 CarInfo 属性。这个属性赋值仅是允许 Car 构造函数的属性值在 CarInLot 构造函数中传递的一种方式。要创建 CarInLot 对象，应该使用下面的语句：

```
var car1 = new CarInLot("AA 123", "Ford", "blue", "10:02AM", "31");
```

语句执行后，carl 对象具有下面的属性和值：

```
car1.timeIn         // value = "10:02AM"
car1.spaceNum       // value = "31"
car1.carInfo        // value = reference to car object constructor function
car1.plate          // value = "AA 123"
car1.model          // value = "Ford"
car1.color          // value = "blue"
```

脚本创建了 5 个 CarInLot 对象(carl～car5)。若为 Car 构造函数原型指定一个新属性，原型继承就会起作用：

```
Car.prototype.companyOwned = true;
```

即使 CarInLot 对象使用 Car 构造函数，CarInLot 对象的实例也没有返回 Car 对象的原型链。如前面的代码所示，即使为 Car 构造函数添加了 companyOwned 属性，CarInLot 对象也不会继承该属性(即使在定义了 Car 的 prototype 新属性后，创建新的 CarInLot 对象)。要使 CarInLot 实例继承 prototype.companyOwned 属性，必须在创建 CarInLot 对象的实例前，明确地将 CarInLot 构造函数的原型连接到 Car 构造函数中：

```
CarInLot.prototype=new Car();
```

完整序列如下所示：

```
function Car(plate, model, color)
{
    this.plate = plate || "missing";
    this.model = model || "unknown";
    this.color = color || "unknown";
}
function CarsInLot(plate, model, color, timeIn, spaceNum)
{
    this.timeIn = timeIn;
    this.spaceNum = spaceNum;
    this.carInfo = Car;
    this.carInfo(plate, model, color);
}
CarsInLot.prototype = new Car();
var car1 = new CarsInLot("123ABC", "Ford","blue","10:02AM", "32");
Car.prototype.companyOwned = true;
```


更多资源请访问稀酷客(www.ckook.com)

这段代码运行后，carl 对象具有如下属性和值：

```
car1.timeIn          // value = "10:02AM"
car1.spaceNum        // value = "31"
car1.carInfo         // value = reference to car object constructor function
car1.plate           // value = "AA 123"
car1.model           // value = "Ford"
car1.color           // value = "blue"
car1.companyOwned    // value = true
```

NN4+/Moz 在原型领域里还提供了一个专有语法：__proto__属性(在单词 proto 前后有两个下划线)返回原型链中下一个对象的引用。例如，运行前面的代码后，如果检查 carl.__proto__属性，就会看到原型链中下一个对象的属性：

```
car1.__proto__.plate           // value = "AA 123"
car1.__proto__.model           // value = "Ford"
car1.__proto__.color           // value = "blue"
car1.__proto__.companyOwned    // value = true
```

这个属性在调试自定义对象和原型继承链时很有帮助，但它并非 ECMA 标准的一部分，所以不要在商业脚本中使用该属性，因为它不能用于非 Mozilla 浏览器。

23.5 Object 对象

属　性	方　法
constructor	hasOwnProperty()
prototype	isPrototypeOf()
	propertyIsEnumerable()
	toSource()
	toString()
	unwatch()
	valueOf()
	watch()

23.5.1 语法

创建 Object 对象：

```
function constructorName([arg1,...[,argN]])
{
    statement(s)
}
var objName = new constructorName(["argName1",...[,"argNameN"]);
var objName = new Object();
```


更多资源请访问稀酷客(www.ckook.com)

```
var objName = {propName1:propVal1[, propName2:propVal2[,...N]]}
```

访问 Object 对象的属性和方法：

```
objectReference.property | method([parameters])
```

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Chrome+

23.5.2　关于该对象

Object 对象是 JavaScript 环境中至关重要的内部对象，如图 23-1 所示，它是其他内部对象如 Date、Array、String 等的根对象，是创建自定义对象的基础，详见本章前面的内容。

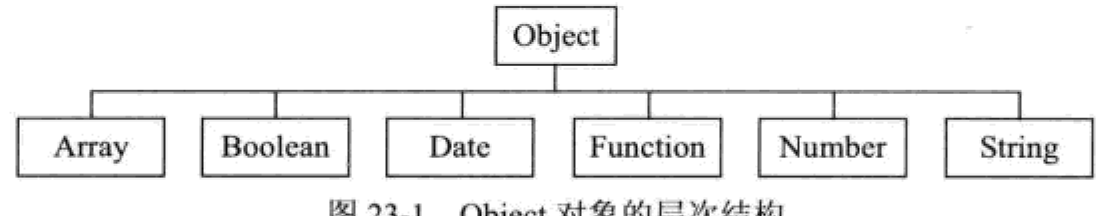


图 23-1　Object 对象的层次结构

脚本一般不会访问内部 Object 对象的属性，也不会访问它的许多方法，如 toString()和 valueof()，当指向对象或其构造函数时，这些方法允许在内部调试警告对话框(和 The Evaluator)，显示其信息。像 toString()和 valueOf()这样的方法，一般在子对象中重写它们。同样，toSource()方法为对象返回源代码的字符串表示，通常不在脚本中使用。

比较实用的是 Object 对象的 watch()和 unwatch()方法，它们提供了在属性值改变时采取行动的机制。可调用 watch()方法，并指定一个在某个属性改变时调用的函数。这样，就可以把内部属性的变化处理为事件。在观察完属性后，用 unwatch()方法停止观察。

程序清单 23-9 列举一个例子，说明了如何使用 watch()和 unwatch()方法跟踪属性值。

程序清单 23-9　观察对象属性

HTML：jsb-23-09.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Object Watching</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-23-09.js"></script>
   </head>
   <body>
      <h1>Watching Over You</h1>
      <form id="theForm" action="validate.php" method="get">
         <div>
         <label for="theText">Enter text here: </label>
         <input id="theText" type="text" name="entry" size="50"
                value="Default Value">
         <p><input id="fourScoreButton" type="button" value="Set to Phrase I"
                   onclick="setIt('Four score and seven years ago... ')"></p>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <p><input id="whenInButton" type="button" value="Set to Phrase 2"
                      onclick="setIt('When in the course of human events... ')"></p>
            <p><input id="resetButton" type="reset"
                      onclick="setIt('Default value')"></p>
            <p><input id="watchButton" type="button" value="Watch It"
                      onclick="watchIt(true)">
               <input id="unWatchButton" type="button" value="Don't Watch It"
                      onclick="watchIt(false)"></p>
            </div>
        </form>
    </body>
</html>
```

JavaScript: jsb-23-09.js

```
function setIt(msg)
{
   document.forms[0].entry.value = msg;
}

function watchIt(on)
{
   var obj = document.forms[0].entry;
      if (on)
      {
         obj.watch("value",report);
      } else
      {
         obj.unwatch("value");
      }
}

function report(id, oldval, newval)
{
   alert("The field's " + id + " property on its way from \n'" +
         oldval + "'\n to \n'" + newval + "'.");
   return newval;
}
```

使用 hasOwnProperty()、isPrototypeOf()和 propertyIsEnumerable()等方法，还可以检查一下对象实例的原型环境。编写扩展、模拟、面向对象应用程序的高级脚本编写者会对此非常感兴趣。

23.5.3　属性

constructor

值：Object 类型，　　　　只读

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Chrome+

自定义对象、Array、String 或其他对象在实例化时，会继承这个属性。它是用来创建对象的构造函数的引用。但注意，如果使用了原型，这个属性可能没有包含有效的值。以前面的 car

更多资源请访问稀酷客(www.ckook.com)

例子来说明：如果要使用 Car 构造函数创建一个 car 对象，constructor 属性就会指向 Car：

```
var car1 = new Car("AA 123", "Ford", "blue");
   if (car1.constructor == Car) // would evaluate to true
   {
      // statements dealing with the car object
   }
```

23.5.4　方法

hasOwnProperty("*propName*")

返回值： Boolean
兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

若当前对象实例在构造函数或相关联的构造函数中定义了属性，hasOwnProperty()方法就返回 true。但如果属性在外部定义，即使是通过为对象的 prototype 属性赋值来定义，该方法也返回 false。

对于本章前面的 Car 和 CarInLot 对象例子，下面的表达式等于 true：

```
car1.hasOwnProperty("spaceNum");
car1.hasOwnProperty("model");
```

即使 model 属性在被其他构造函数调用的构造函数中定义，该属性也属于 car1 对象，但下面的语句会得到 false：

```
car1.hasOwnProperty("companyOwned");
```

该属性通过某个构造函数的原型定义，不是对象实例的内嵌属性。

isPrototypeOf(*objRef*)

返回值： Boolean
兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

isPrototypeOf()方法表示当前对象是否和作为参数传递的对象具有原型关系。事实上，这个方法的 IE 和 NN/Moz 版本不仅操作不同，返回的结果也不同。

propertyIsEnumerable("*propName*")

返回值： Boolean
兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

在 ECMA-262 语言规范的术语中，如果 for...in 属性检测循环(参见第 21 章)等结构能够检查到某个值，该值就是可以枚举的。可枚举的属性包括数组、字符串和几乎每类对象。根据 ECMA 规范，该方法不能在原型链上工作。

toSource()

返回值： 字符串

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

toSource()方法为对象获得源代码的字符串表示。toString()方法返回对象值的等效字符串，而 toSource()方法把对象的底层代码返回为字符串。这是一种非常专业的底层功能，通常只由 JavaScript 在内部用于一些调试工作。

toString()

返回值： 字符串

兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

toString()方法用来获得对象值的字符串表示。在需要检查对象的原始文本时，通常使用此方法。在创建自定义对象时，可以创建自己的 toString()方法，来显示需要查看的对象信息。

unwatch("*propName*")

返回值： Boolean

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari-，Chrome+

与 watch()方法相反，unwatch()方法终止对特定属性设置的观察。

valueOf()

返回值： 对象

兼容性： WinIE5.5+，MacIE-，NN6+，Moz+，Safari-，Chrome+

valueOf()方法把对象解析为基本数据类型。这不见得能够实现，如果不能解析，该方法就只返回对象本身。否则，此方法返回表示对象的基本值。

watch("*propName*")

返回值： Boolean

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari-，Chrome+

watch()方法是 JavaScript 调试功能(称为观察点)的关键。观察点允许指定一个函数，只要设置属性值，就调用该函数。接着，就可以仔细跟踪属性的状态，在重要的属性值发生变化时采取行动。

设置观察点的方法是调用 watch()方法，传递要观察的属性名，如下：

```
obj.watch("count",
   function(prop, oldval, newval)
   {
      document.writeln(prop + " changed from " + oldval + " to " + newval);
      return newval;
   }
);
```

这个例子创建了一个函数，输出属性改变的通知信息，然后返回新的属性值。每个观察点处理程序都必须遵循一个约定：使用三个参数，分别指定属性、旧值和新值。

更多资源请访问稀酷客(www.ckook.com)

全局函数和语句

除了本书前面章节描述的对象和其他语言结构外，还有几个语言项需要进行全局处理。这些项不属于某个对象(或任何对象)，可供在脚本的任何地方使用。前面的章节介绍了许多这样的函数和语句。本章将重点说明这些容易忘记、但很重要的主题。本章最后将简要介绍几个仅用于 Internet Explorer 的 Window 版本的对象。

本章首先介绍表 24-1 中列出的全局函数和语句，它们是 JavaScript 核心语言的一部分。

本章包含哪些内容？

将字符串转换成对象引用

创建 URL 友好的字符串

给脚本添加注释

表 24-1 本章介绍的全局函数和语句

函 数	语 句
decodeURI()	//和/*...*/(注释)
decodeURIComponent()	const
encodeURI()	var
encodeURIComponent()	
escape()	
eval()	
isFinite()	
isNaN()	
Number()	
parseFloat()	
parseInt()	
toString()	
unescape()	
unwatch()	
watch()	

更多资源请访问稀酷客(www.ckook.com)

全局函数不属于文档对象模型。它们通常能把数据从一种类型转换成另一种类型。全局语句很少，但它们在脚本中使用广泛。

24.1　函数

decodeURI("*encodedURI* ")，decodeURIComponent("*encodedURIComponent* ")
encodeURI("*URIString* ")，encodeURIComponent("*URIComponentString*")

返回值： 字符串

兼容性： WinIE5.5+，MacIE−，NN6+，Moz+，Safari+，Opera+，Chrome+

IE5.5+、NN6+和基于 Mozilla 浏览器实现了 ECMA-262 标准，该标准提供了实用函数，可在字符串与有效的 URI 字符串之间进行转换，它比早期通过 escape()和 unescape ()函数(本章后面介绍)进行的转换更严格。实际上，现代函数 encodeURI()、encodeURIComponent()、decodeURI()和 decodeURIComponent()替代了现在已废弃的 escape()和 unescape()函数。

编码函数的作用是把字符串转换成能用作统一资源标识符(Uniform Resource Identifier)的版本，例如网页地址或服务器 CGI 脚本的调用。拉丁字母数字字符组在编码处理后不会改变，但某些符号和其他 Unicode 字符必须使用编码函数转换成 Internet 能传输的格式(字符数字的十六进制表示)。例如，空格字符必须编码成十六进制形式：%20。

encodeURI()和 escape()函数(还有 decodeURI()和 unescape())的最大区别在于，现代版本的浏览器不对大量符号进行编码，因为根据 RFC2396(http://www.ietf.org/rfc/rfc2396.txt)推荐的语法，它们可用作 URI 字符。因此，不通过 encodeURI()函数来编码下面的字符：

```
; / ? : @ & = + $ , - _ . ! ~ * ' ( ) #
```

encodeURI()和 decodeURI()函数只能在完整的 URI 中使用。可用的 URI 可以是相对地址或绝对地址，但这两个函数是连接在一起的，所以不编码 URI 中的协议(://)、搜索字符串(例如?和=)和目录层次分隔符(/)。decodeURI()函数能处理从服务器传来的页面地址 URI，但注意，有些服务器的 CGI 程序把空格编码成加号(+)，但 JavaScript 函数不能把加号解码回空格。如果脚本需要解码的 URI 用加号代替空格，则需要通过一个字符串替代方法来运行解码的 URI，来完成这项工作(这里使用正则表达式比较方便)。如果要解码的 URI 字符串是用脚本编码的，则只有通过相应的编码函数进行编码的 URI，才使用解码函数。不要试图对使用旧 escape()函数创建的 URI 进行解码，因为转换过程根据不同的规则来进行。

URI 和 URI 组件之间的区别在于，组件是 URI 的一个片段，一般不含有分隔符。例如，如果在完整的 URI 上使用 encodeURIComponent()函数，则几乎所有符号(句点除外)都会编码成十六进制版本，包括目录分隔符。因此，应该在 URI 的最小单位上使用组件级的转换函数。例如，如果组合一个包含“名/值”对的查询字符串，就可以在名称和值上分别使用 encodeURIComponent()函数。但如果在 name=value 形式的名/值对上使用这个函数，这个函数就会把等号编码成对应的十六进制。

escape()和 unescape()函数以前有时用于不是 URL(URI)的字符串，所以，在把使用 escape()和 unescape()的代码转换为现代代码时，通常使用 encodeURIComponent()和 decodeURIComponent()函数。

PDG

更多资源请访问稀酷客(www.ckook.com)

示例

使用 The Evaluator(参见第 4 章)来了解编码完整的 URI 和编码 URI 组件之间的区别，以及对 URI 字符串进行编码和转义之间的区别。例如，比较以下三条语句的结果：

```
escape("http://www.giantco.com/index.htm?code=42")
encodeURI("http://www.giantco.com/index.htm?code=42")
encodeURIComponent("http://www.giantco.com/index.htm?code=42")
```

因为示例的 URI 字符串是有效的，所以 encodeURI()函数不会改变它。在查询的字符串值中插入一个空格，再试一次，看看每个函数如何处理这个字符。

escape("*URIString*" [,1])，unescape("*escapedURIString*")

返回值： 字符串

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

查看浏览器的 Location 域，偶尔会看到 URL 含有许多%字符和一些数字。这个格式是 URL (Uniform Resource Locator，统一资源定位符)编码(更确切地说是 URI 编码)。这种格式甚至允许把多字字符串和非字母数字字符发送为一个公共字符集的连续字符串。这种编码把一个字符(例如一个空格)转换成十六进制，并在十六进制的前面加一个百分号。例如，空格字符(ASCII 值是 32)的十六进制为 20，因此空格的编码版本为%20。

所有字符(包括制表符和回车)都能用这种方法编码，并发送为一个简单的字符串，这个字符串能在接收端解码并重构。这种编码方式还可以用来预处理多行文本，这些文本行必须在数据库中存储为字符串。可以使用 escape()方法把纯语言字符串转换成其编码形式，它返回一个由编码组成的字符串。例如：

```
var theCode = escape("Hello there");
    // result: Hello%20there
```

大多数(但不是全部)非字母数字字符都使用 escape()函数来转换成编码版本。但加号除外，因为 URL 使用加号来分隔查询字符串的各个部分。如果必须对加号进行编码，就应在函数中添加第二个可选参数，把加号转换成它的十六进制值(2B)：

```
var a = escape("Adding 2+2");
    // result: Adding%202+2
var a = escape("Adding 2+2",1);
    // result: Adding%202%2B2
```

注意，浏览器对第二个可选参数的处理并不一致；但无论是否包含第二个参数，都可以返回相同的结果。

unescape()函数以前用于把已编码的字符串转换回纯语言形式。之所以说“以前”，是因为有了 decodeURI()和 decodeURIComponent()，现在这个函数已废弃，不应当使用。

escape()函数的执行方式处于新函数 encodeURI()和 encodeComponentURI()之间。然而，因为有了新函数，现在这个函数已废弃，不应当使用。

更多资源请访问稀酷客(www.ckook.com)

eval("*string*")

返回值：对象引用

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

现在我们知道，表达式求值是编写 JavaScript 脚本(和常规编程)时需要掌握的一个重要概念。表达式总是计算为某个值，但有时只有在表达式上强制执行另一次计算，才能得到希望的结果。eval ()函数则可以在字符串值上强制计算该字符串表达式。eval()函数常用于把对象引用的字符串版本转换成真正的对象引用。

示例

eval()函数能计算任何存储为字符串的 JavaScript 语句或表达式，包括算术表达式、对象赋值和对象方法调用的字符串形式。然而，这里不推荐使用 eval()函数，因为这个函数的效率极低(从性能角度来看)。幸好，不用 eval()函数也可将对象名的字符串版本转换成有效的对象引用。例如，如果脚本遍历一系列的对象，而这些对象的名称包含连续的数字，可以将对象名用作数组索引，而不是使用 eval()函数来组合对象引用。

下面是设置一系列域 data0、datal 等的值的低效方式：

```
function fillFields()
{
   var theObj;
   for (var i = 0; i < 10; i++)
   {
      theObj = eval("document.forms[0].data" + i);
      theObj.value = i;
   }
}
```

更高效的方式是在对象引用的索引方括号中进行连接：

```
function fillFields()
{
   for (var i = 0; i < 10; i++)
   {
      document.getElementById("myForm").elements["data" + i].value = i;
   }
}
```

提示：

只要打算使用 eval()函数，就要看看能否使用对象数组的字符串索引值来替代 eval()函数。W3C DOM 使用 document.getElementByld()方法，使这个任务更容易完成，这个方法把字符串作为参数，返回指定对象的引用。

isFinite(*number*)

返回值：Boolean

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

isFinite()函数一般很少使用，它用于检查数字是否超出了 JavaScript 能处理的最大值或最小值。如果数字超出了这个范围，该函数就返回 false。这个函数的参数必须是数字数据类型。

isNaN(*expression*)

返回值： Boolean

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

有时，计算过程依赖从文本域或者其他面向字符串的源中获得的数据，此时经常需要检查这个值是否是数字。如果这个值不是数字，计算操作将可能导致脚本错误。

示例

把值传递给计算操作之前，需要使用 isNaN()函数来检查这个值是不是数字。这个函数常常用于检查 parseInt()或 parseFloat()函数的结果。如果提交给这些函数的字符串不能转换成数字，结果就为 NaN(一个特别的符号，表示“不是数字”)。如果这个值不是数字，isNaN()函数就返回 true。

这个函数的一个用法是在非法数据产生破坏前截取它。如下所示：

```
function calc(form)
{
   var inputValue = parseInt(form.entry.value);
      if (isNaN(inputValue))
      {
         alert("You must enter a number to continue.");
      }
      else
      {
         // statements for calculation
      }
}
```

脚本设计者在使用这个函数时，可能犯下的最大错误是，没有注意到这个函数名的大小写形式。尾部的大写字母 N 很容易遗漏。

Number("*string*")，parseFloat("*string*")，parseInt("*string*" [,*radix*])

返回值： 数值

兼容性： WinIE3+，Mac1E3+，NN2+，Moz+，Safari+，Opera+，Chrome+

这三个函数的作用是把字符串转换成数字。parseInt()和 parseFloat()函数在所有浏览器的版本中都兼容，包括非常旧的浏览器；但 Number()函数只用在版本 4 的浏览器中。

如果脚本不在意值的精度，并允许通过原字符串决定返回值是浮点型还是整型，就可以使用 Number()函数。这个函数只有一个参数：要转换成数字的字符串。

parseFloat()函数也能通过原字符串决定返回值是浮点型还是整型。如果原字符串在小数点右边有非零值，返回值就是浮点数。但如果原字符串是如“3.00”之类的数字，返回值就是整型。

在 parseInt()中，另一个可选参数可以决定转换时使用的数字基数。如果不指定基数参数，JavaScript 会试着找一个基数，但这样可能使 JavaScript 出问题。当 parseInt()的字符串参数以 0

开头时(文本框输入项或数据库域有可能以 0 开头)，就会出现一个重要问题：在 JavaScript 中，以 0 开头的数字处理为八进制(基数为 8)，因此，parseInt("010")返回的十进制值为 8。

使用 parseInt()函数时，如果处理基数为 10 的数字，就应总是指定基数为 10。基数也可以指定为 2～36 之间的数字。例如，指定基数为 2，把二进制数字符串转换成十进制形式，如下所示：

```
var n = parseInt("011",2);
    // result: 3
```

同样，也可以指定基数为 16，把十六进制字符串转换成十进制：

```
var n = parseInt("4F",16);
    // result: 79
```

示例

parseInt()和 parseFloat()函数有一个非常有用的功能。如果字符串参数以至少一位数字开头，后跟字母，这些函数将只处理字符串前面的数字部分，并忽略剩余的部分。因此，可在 navigator.appVersion 字符串上使用 parseFloat()函数，来提取报告的版本号，而不必分析字符串的剩余部分。例如，Windows 的 Firefox 1.0 报告 navigator.appVersion 值为：

```
5.0 (Windows; en-US)
```

也可通过 parseFloat()函数来获取字符串的数字部分：

```
var ver = parseFloat(navigator.appVersion);
```

注意：

navigator.appVersion 属性中存储的数字是底层浏览器引擎的版本号。所以，在这个例子中，即使 Firefox 浏览器应用程序是 1.0 版，报告的版本也是 5.0。

因为结果是一个数字，所以可进行数字比较，例如，看一下版本是否大于或等于 4。

toString([*radix*])

返回值： 字符串

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

每个 JavaScript 核心语言对象和每个 DOM document 对象都有与其相关的 toString()方法。这个方法采用尽可能有意义的方式，将对象的内容显示为一串文本。表 24-2 显示了在每个可转换的 language 核心对象类型上使用 toString()方法得到的结果。

表 24-2　对象类型的 toString()方法结果

对　　象	类型结果
String	相同的字符串
Number	等效的字符串(但不能转换为数字字面量)
Bollean	true 或 false

更多资源请访问稀酷客(www.ckook.com)

(续表)

对 象	类型结果
Array	数组内容列表，用逗号分隔，逗号后面没有空格
Function	反编译函数定义的字符串版本

许多 DOM 对象都能转换成字符串。例如，location 对象返回它的 URL。但当对象不能把合适的内容返回为字符串时，它通常返回下列格式的字符串：

```
[object objectType]
```

示例

toString()方法在所有浏览器的所有版本中都可用。将可选的 radix 参数设置在 2～16 之间，可用不同的数字基数把数字转换成字符串。程序清单 24-1 计算了 0～20 之间的数字的十进制、十六进制和二进制，并绘制了一个转换表。在这个例子中，源值是每次执行 for 循环语句时索引计数变量的值。

注意：

本章和本书其他许多地方使用的事件处理程序分配函数是 addEvent()，这是一个跨浏览器的事件处理程序，详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，该文件位于配书光盘中。

程序清单 24-1 使用基数参数的 toString()

HTML: jsp-24-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Using toString() to convert to other number bases:</title>
      <link rel="stylesheet" type="text/css" href="jsp-24-01.css">
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-24-01.js"></script>
   </head>
   <body>
      <h1>Using toString() to convert to other number bases:</h1>
      <table id="numberConversions">
         <tr>
            <th>Decimal</th>
            <th>Hexadecimal</th>
            <th>Octal</th>
            <th>Binary</th>
         </tr>
      </table>
   </body>
</html>
```

CSS: jsp-24-01.css

更多资源请访问稀酷客(www.ckook.com)

```
th, td
{
   border: 5px groove black;
   text-align: center;
   padding-left: 2px;
   padding-right: 2px;
}

th
{
   font-weight: bold;
}
```

JavaScript: jsb-24-01.js

```
// initialize when the page has loaded
addEvent(window, "load", initialize);

function initialize()
{
   var theTable = document.getElementById("numberConversions");

      if (theTable)
      {
         var newRowElem;
         var newCellElem;
         var newText;
         // Insert the rows after the existing row
         var newRowPosition = 1;

         // put the converted number data into the table
         for (var i = 0; i <= 20; i++)
         {

            // create a new row
            newRowElem = theTable.insertRow(newRowPosition);

            // create the 1st new cell in the new row and its text
            newCellElem = newRowElem.insertCell(0);
            newText = document.createTextNode(i.toString(10));
            newCellElem.appendChild(newText);

            // create the next new cell in the new row and its text
            newCellElem = newRowElem.insertCell(1);
            newText = document.createTextNode(i.toString(8));
            newCellElem.appendChild(newText);

            // create the next new cell in the new row and its text
            newCellElem = newRowElem.insertCell(1);
            newText = document.createTextNode(i.toString(16));
            newCellElem.appendChild(newText);

            // create the next new cell in the new row and its text
            newCellElem = newRowElem.insertCell(2);
```


更多资源请访问稀酷客(www.ckook.com)

```
                newText = document.createTextNode(i.toString(2));
                newCellElem.appendChild(newText);

                newRowPosition += 1;
            }
        }
}
```

用户自定义对象的 toString()方法不能把该对象转换成有意义的字符串，但可创建自己的方法来实现这个功能。例如，如果想使自定义对象的 toString()方法像数组的 toString()方法那样，就可以定义这个方法的行为，并把该功能赋给这个对象的一个属性(如程序清单 24-2 所示)。

程序清单 24-2　创建自定义的 toString()方法

HTML: jsp-24-02.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>A custom object defined toString() result:</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-24-02.js"></script>
   </head>
   <body>
      <h1>A custom object defined toString() result:</h1>
      <!-- Hard code what's in the object so we can validate
           the JavaScript visually on the page. -->
      <h3>The custom object is book and its properties are</h3>
      <h4>title:"The Aeneid", author:"Virgil", pageCount:543</h4>
      <h3>Now let's look at the results of the custom object's toString() in
          the JavaScript:</h3>
      <div id="placeHolder"></div>
   </body>
</html>
```

JavaScript: jsb-24-02.js

```
// initialize when the page has loaded
addEvent(window, "load", initialize);

var newElem;
var newText;

// define the function that will be the method for the custom object
function customToString()
{
   var dataArray = new Array();
   var count = 0;
   for (var i in this)
   {
      dataArray[count++] = this[i];
```


更多资源请访问稀酷客(www.ckook.com)

```
            if (count > 2)
            {
               break;
            }
      }
      return dataArray.join(",");
   }

   // define a custom object
   var book = { title:"The Aeneid", author:"Virgil", pageCount:543 };

   // declare a method for the custom object
   book.toString = customToString;

   // the onload event we identified earlier
   function initialize()
   {

      var placeHolderElement = document.getElementById("placeHolder");

         if (placeHolderElement)
         {
            //book.toString();

            newElem = document.createElement("h4");
            newText = document.createTextNode(book.toString());
            // insert the text into the new h4
            newElem.appendChild(newText);
            // insert the completed h4 into placeholder
            placeHolderElement.appendChild(newElem);
         }
   }
```

运行程序清单 24-2，custom 对象的 toString()处理程序就会提取这个对象所有元素的值。可自定义这些数据的表示方式和格式化方式。

可对用户创建的任何对象提供自定义的 toString()方法，而不仅仅是数组。这是调试时快速查看对象内容的方便方式。例如，可以用 toString()方法把对象的所有属性格式化为便于读取的文本字符串。如果出现问题，就使用警告框或者浏览器调试控制台来查看对象的内容。

unwatch(*property*)，watch(*property*, *handler*)

返回值：无

兼容性：WinIE-，MacIE-，NN4+，Moz-，Safari-，Opera-，Chrome-

为给外部调试器提供正确信息，NN4+中的 JavaScript 实现了两个全局函数，每个对象都可以使用这两个全局函数，包括用户自定义对象。watch()函数可以密切观察对象及其属性。如果通过赋值语句来设置属性，该函数就会调用另一个用户自定义函数，来接收属性名的信息、其旧值和新值。unwatch()函数可以关闭特定属性的观察功能。

更多资源请访问稀酷客(www.ckook.com)

24.2 语句

//，/*...*/

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

注释是 JavaScript 解释器(或服务器端编译器)忽略的语句。然而，程序设计者可使用这些语句说明脚本是怎样工作的。虽然在脚本的创建和维护期间，足够的注释对程序设计者是有用的，但所有注释内容都要和文档一并下载到客户端，所以下载页面中的这些非操作内容要花费更长的时间。然而，创建脚本时，最好使用注释。

JavaScript 提供了两种风格的注释。一种风格的注释由两根斜线(它们之间没有空格)组成，用于创建单行注释。JavaScript 将忽略双斜线右边的任意字符，即使双斜线在行的中间也是如此。可使用任意多个单行注释来表达想法。通常在第二根斜线和注释的开头之间加一个空格。下面是有效的单行注释例子：

```
// this is a comment line usually about what's to come
var a = "Fred";  // a comment about this line
// You may want to capitalize the first word of a comment
// sentence if it runs across multiple lines.
//
// And you can leave a completely blank line, like the one above.
```

对于更长的注释，更简便的方法是将整个注释语句放在另一种风格的注释中，这些注释可以放在多个代码行上。下面的注释以一根斜线和一个星号(/*)开头，以一个星号和一根斜线(*/)结束，JavaScript 将忽略它们之间的所有语句，包括多行语句。如果为了进行调试，临时注释脚本中的一大段语句，最简便的方式是用这些注释字符把这一段括起来。为了更便于找到注释块，通常使这些注释字符单独占一行，如下所示：

```
/*
some
  commented-out
     statements
*/
```

如果开发非常复杂的文本，使用注释就可以非常方便地组织脚本段，使其更容易查找。例如，可在每个函数上定义一个注释块，说明这个函数的作用，如下例所示：

```
/*-----------------------------------------------
   calculate()
   Performs a mortgage calculation based on
   parameters blah, blah, blah. Called by blah
   blah blah.
-----------------------------------------------*/
function calculate(form)
{
   // statements
}
```


更多资源请访问稀酷客(www.ckook.com)

const

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

const 关键字用来初始化常量。载入页面后，变量的数据可以改变，而一旦指定常量值，就不能改变。在许多编程语言中，常量标识符通常使用全部大写字母来定义，使用下划线字符来分隔多个单词。这便于在代码中快速找到常量，常量的值是固定不变的。

示例

程序清单 24-3 说明了如何在非 IE 的浏览器中使用常量。网页显示了几个城市的温度数据(假定数据在服务器上更新，当用户请求这个页面时，页面会显示一组数据)。温度低于冰点时，就显示为另一种文本样式。因为冰点是不变的参考点，所以把它指定为常量。

程序清单 24-3　使用 const 关键字

HTML: jsp-24-03.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>The const keyword</title>
      <link rel="stylesheet" type="text/css" href="jsp-24-03.css">
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb-24-03.js"></script>
   </head>
   <body>
      <h1>The const keyword</h1>
      <table id="temps">
         <tr>
            <th>City</th>
            <th><div>Temperature</div><div>(Fahrenheit)</div></th>
         </tr>
      </table>
   </body>
</html>
```

CSS: jsp-24-03.css

```
.cold
{
   font-weight: bold;
   color: blue;
}

td
{
   text-align: center;
}
```

JavaScript: jsb-24-03.js

更多资源请访问稀酷客(www.ckook.com)

```
// initialize when the page has loaded
addEvent(window, "load", showData);

const FREEZING_F = 32;

var cities = ["London", "Moscow", "New York", "Tokyo", "Sydney"];

var cityTempsF = [33, 12, 20, 40, 75];

function showData()
{
   var theTable = document.getElementById("temps");

      if (theTable)
      {
         var newRowElem;
         var newCellElem;
         var newText;
         // Insert the rows after the existing row
         var newRowPosition = 1;
           // put the city temperature data in the table
           for (var i = 0; i < cities.length; i++)
           {
              // create a new row
              newRowElem = theTable.insertRow(newRowPosition);

              // create the new city cell in the new row and its text
              newCellElem = newRowElem.insertCell(0);
              newText = document.createTextNode(cities[i]);
              newCellElem.appendChild(newText);

              // create the new temp cell in the new row and its text
              newCellElem = newRowElem.insertCell(1);
              newText = document.createTextNode(cityTempsF[i]);
              newCellElem.appendChild(newText);

              // style the cells with cold data
                 if (cityTempsF[i] < FREEZING_F)
                 {
                    newCellElem.className = "cold";
                 }
              newRowPosition += 1;
           }
      }
}
```

const 关键字可能在 ECMA-262 标准的下一个版本中采用，并在将来的浏览器中成为 JavaScript 的一部分。它还得到基于 Mozilla 的浏览器的完全支持。

Var

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

变量在使用之前，应该通过 var 语句声明(还可用一个值初始化它)。如果遗漏了 var 关键字，

更多资源请访问稀酷客(www.ckook.com)

这个变量在当前文档中将自动指定为全局变量。为使变量成为函数的局部变量，必须在这个函数的花括号内使用 var 关键字来声明或初始化它。

如果不给变量指定值，它就是 null。因为 JavaScript 变量在其生命周期内不只存储一种类型的数据，所以不需要将变量初始化为空字符串或 0，除非那个初始值对编写脚本有帮助。例如，如果将变量初始化为空字符串，就可以在文档后面的语句中使用+=操作符给变量添加字符串值。

为了减少语句行，可使用一个 var 语句来声明并/或初始化多个变量。用逗号来分隔每个 varName=value 对，如：

```
var name, age, height;  // declare as undefined
var color = "green", temperature = 85.6; // initialize
```

变量名(也称为标识符)必须是一个连续的字符串，且第一个字符必须是字母。变量名中的字符不能使用标点符号，但可以使用下划线字符，它经常用来分隔长变量名中的多个单词。所有变量名(JavaScript 中的大多数标识符)都区分大小写，因此必须在变量的作用域内以相同的方式命名变量。

根据约定，变量名采用 camelCase 形式，如下所示：

```
var colorCode = "green";
```

24.3 WinIE 对象

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

不管如何，Microsoft 都对 Web 浏览器功能和 Windows 操作系统的集成非常自豪，这种浏览器与操作系统之间的链接在访问 ActiveX 对象的 IE 功能中体现得非常明显。Microsoft 提供了几个这样的对象供脚本开发人员访问——假定程序仅在 Internet Explorer 的 Windows 版本上部署。一些对象还用来为 JavaScript 提供某种 Visual Basic Script(VBScript)功能。因为这些对象主要用于 Windows 和 ActiveX 编程领域，所以在 WinIE 中使用这些对象的细节最好在其他地方讨论。但由于读者可能不熟悉这些功能，下面的讨论将介绍基本的 WinIE 对象集，更多信息可参见 Microsoft Developer Network(MSDN)网站(http:// msdn.microsoft.com/)。

这里介绍 ActiveXObject、Dictionary、Enumerator 和 VBArray 对象。Microsoft 说明了这些对象，就像它们是 JScript 语言的一部分。然而，可以确保它们仍是 Internet Explorer 的专有对象，尽管它们并非仅用于 Window 版本。

注意：

JScript 是 JavaScript 的 Microsoft 专用语言，得到 Internet Explorer 的支持。JScript 其实与 JavaScript 相同，只是增加了几个 Windows 特定的功能，比如对 Activex 对象的支持。

24.3.1 ActiveXObject

ActiveXObject 是一个通用对象，它允许脚本打开并访问 Microsoft 所谓的自动化对象。自

更多资源请访问稀酷客(www.ckook.com)

动化对象是可执行的程序，可在客户端运行或从服务器上获取服务。它包括本地应用程序，例如 Microsoft Office 组中的应用程序以及可执行的 DLL(动态链接库)等。

根据下面的语法，可以用 ActiveXObject 的构造函数来获得这个对象的引用：

```
var objRef = new ActiveXObject(appName.className[, remoteServerName]);
```

这个 JScript 语法与 VBScript 中的 CreateObject()方法相同。为了确定应用程序名和应用程序可用的类或者类型，需要了解 Window 编程的一些内容。例如，为了获得 Excel 工作表的引用，可使用这个构造函数：

```
var mySheet = new ActiveXObject("Excel.Sheet");
```

有了所需对象的引用后，还必须了解这个对象的属性和方法名。通过 Microsoft 的开发工具可访问许多这样的信息，例如 Visual Studio.NET 或 Visual Basic.NET 附带的工具。这些工具可以查询对象，得到它的属性和方法。但是，通过 JavaScript 的 for...in 属性检查工具，并不能列举出 ActiveXObject 的属性。

特别是在客户端，访问 ActiveXObject 涉及许多严重的安全问题。IE 客户端的典型安全设置是阻止脚本访问客户应用程序，至少在没有询问用户是否可以访问之前，脚本是不能访问应用程序的。尽管不经用户许可，就不能偷偷地检查或破坏客户程序(电脑黑客有时会找到漏洞)，但这是可能的。在公司环境中，如果需要对所有客户进行某个级别的访问，就可以设置客户端，使其从可信的源中接收指令，来使用 ActiveXObject。其前提是，除非精通 Windows 编程，否则 ActiveXObject 不会成为一种非法侵入用户的隐私和安全的方法。

24.3.2　Dictionary

Dictionary 对象对 VBScript 编程人员非常有帮助，而 JavaScript 也提供了同样的功能。Dictionary 对象的操作非常类似于具有字符串索引值的 JavaScript 数组(和 Java 哈希表类似)，在 Dictionary 中也可以使用数值索引。索引在该环境中称为 keys(键)，VBScript 数组没有这种功能，所以 Dictionary 对象给 VBScript 语言填补了这个空白。与 JavaScript 数组不同，只有使用 Dictionary 对象的各种属性和方法，才能增加、访问或删除该对象中的项。

使用 ActiveXObject 创建 Dictionary 对象：

```
var dict = new ActiveXObject("Scripting.Dictionary");
```

必须为每个数组创建各自的 Dictionary 对象，表 24-3 列出了 Dictionary 对象的属性和方法。创建空 Dictionary 对象后，使用 Add()方法添加每个数组项。例如，下面的语句创建 Dictionary 对象，来存储美国的州府：

```
var stateCaps = new ActiveXObject("Scripting.Dictionary");
stateCaps.Add("Illinois", "Springfield");
```

然后，就可以通过 Key 属性访问单个项(该对象继承了 VBScript 功能，这个属性就像一个 JavaScript 方法)。Dictionary 对象的一个方法是 Keys()，它返回字典中的所有键——使用字符串索引的 JavaScript 数组可以使用这些键。

更多资源请访问稀酷客(www.ckook.com)

表 24-3　Dictionary 对象的属性和方法

属　　性	说　　明
Count	字典中的项的个数(整数，只读)
Item("key")	读写名为 key 的数组项的值
Key("key")	为数组项指定新的键名
方　　法	**说　　明**
Add("key",value)	添加与唯一键名相关的值
Exists("key")	如果字典中存在 key，则返回 true
Items()	返回字典中值的 VBArray
Keys()	返回字典中键的 VBArray
Remove("key")	删除 key 及其值
RemoveAll()	删除所有项

24.3.3　Enumerator

Enumerator 对象允许 JavaScript 访问集合，但不允许直接使用索引值或名称访问集合项。在处理 DOM 集合(如 document.all)时，不见得使用该对象，因为可以使用 item()方法得到任何一个集合成员的引用。但如果编写 ActiveX 对象的脚本，这些对象的方法或属性可以返回集合，但不能使用这种机制或 JavaScript 的 for…in 属性检查技术进行访问。另外，必须将集合放在一个 Enumerator 对象中。

要在 Enumerator 中包含集合，应调用对象的构造函数，将集合作为参数：

```
var myEnum = new Enumerator(someCollection);
```

要访问 enumerator 实例，可通过 4 个方法来定位指向某一项的“指针”，然后提取该项的副本。也就是说，不能直接访问集合的成员(即进入集合，得到项的编号)，而可以将指针移到指定的位置，然后读取该项的值。从表 24-4 的方法列表可以看出，该对象可以用于遍历集合，指针控制只限于将指针放在集合开始处，再将指针位置值加 1，逐个访问集合项：

```
var val;
myEnum.moveFirst();
for (; !myEnum.atEnd(); myEnum.moveNext())
{
    val = myEnum.item();
    // more statements that work on value
}
```

表 24-4　Enumerator 对象的方法

方　　法	说　　明
atEnd()	如果指针到达集合末端，返回 true
item()	返回当前指针位置上的值
moveFirst()	将指针移到集合的第一个位置
moveNext()	将指针移到集合中的下一个位置

更多资源请访问稀酷客(www.ckook.com)

24.3.4 VBArray

VBArray 对象允许 JavaScript 访问 Visual Basic 安全数组。这种数组是只读的，通常通过 ActiveX 对象返回。这种数组可在客户端脚本的 VBScript 部分进行组合。Visual Basic 数组本质上是多维的。例如，下面的代码创建一个 3 行 2 列的 VB 数组：

```
<script type="text/vbscript">
   Dim myArray(2, 1)
   myArray(0, 0) = "A"
   myArray(0, 1) = "a"
   myArray(1, 0) = "B"
   myArray(1, 1) = "b"
   myArray(2, 1) = "C"
   myArray(2, 2) = "c"
</script>
```

一旦有了有效的 VB 数组，就可以将它转换为 JScript 解释器不能阻塞的对象：

```
<script type="text/javascript">
   var theVBArray = new VBArray(myArray);
</script>
```

一个脚本语言块中的全局变量可由另一个脚本块访问，甚至是使用不同语言的脚本块。但是，该数组现在还不是 JavaScript 数组。通过 VBArray.toArray()方法可将它转换为 JavaScript 数组，通过其他方法可以访问 VBArray 对象的信息(参见表 24-5)。将 VBArray 转换为 JavaScript 数组后，就可以像 JavaScript 数组一样遍历数组的值。

表 24-5 VBArray 对象的方法

方　法	说　明
dimensions()	返回初始数组的维数
getItem(diml [，dim2 [，...dimN]])	返回维地址定义的数组位置值
ibound(dim)	返回给定维的最低索引值
toArray()	返回 VBArray 的 JavaScript 数组版本
ubound(dim)	返回给定维的最高索引值

使用 toArray()方法，且源数组是多维时，第一行后的维值会加在 JavaScript 数组的后面，不采用嵌套结构。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第 IV 部分

文档对象参考

本部分内容

第 25 章　文档对象模型基础

第 26 章　通用 HTML 元素对象

第 27 章　window 对象和 frame 对象

第 28 章　location 对象和 history 对象

第 29 章　document 对象和 body 对象

第 30 章　link 和 anchor 对象

第 31 章　image、area、map 和 canvas 对象

第 32 章　event 对象

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第25章

文档对象模型基础

毫无疑问，客户端 Web 脚本开发人员面临的最大困难是文档对象模型中有时令人备感困惑的数组，在脚本浏览器的短暂发展过程中，DOM 在不断地相互竞争，以吸引人们的眼球。Netscape 在 Navigator 2 中包含了第一个对象模型，开始了这场竞争。其第 4 版问世后，原始对象模型不仅增加了一些有用的跨浏览器功能，而且获得了 Navigator 或 Internet Explorer 的一些独有功能。对象模型的不统一，给页面设计者带来了无休无止的痛苦，因为他们的脚本必须运行在尽可能多的浏览器上。因此 World Wide Web Consortium(W3C)以文档对象模型(DOM)标准的形式，开始标准化过程，这为目前的危机带来了希望。DOM 继承了许多原始的对象模型，提供了在文档中确定每个对象地址的新方法。本章将深入分析每个对象模型，并帮助理解现代浏览器如何解决大多数对象模型的兼容性问题。但在介绍这些细节之前，先看看对象模型在设计脚本应用程序方面的作用。

本章包含哪些内容？

对象模型与浏览器版本

专用模型扩展

W3C DOM 的结构

脚本编程的趋势

25.1 对象模型层次结构

第II部分介绍了脚本浏览器中文档对象层次结构的基本概念。在其他面向对象的环境中，对象层次结构的作用比在支持 JavaScript 的浏览器中更大，因为在 JavaScript 中，不必考虑相关的术语，例如类、继承和实例。不过，即使这样也不能忽略层次结构概念，因为许多代码都需要编写对象的引用，而对象引用依赖它们在层次结构中的位置。

将这些对象称为“JavaScript 对象”并不正确，它们其实是浏览器文档对象：只是碰巧在 JavaScript 语言中使用它们而已。一些 Microsoft Internet Explorer 脚本开发人员还使用 VBScript 语言对这些文档对象编程。在技术方面，JavaScript 对象使用与文档分离的数据类型和其他核心语言对象。

更多资源请访问稀酷客(www.ckook.com)

25.1.1　作为路径图的层次结构

对于程序员而言，文档对象层次结构的基本作用是提供一种方式，让脚本在浏览器窗口包含的所有对象中引用某个对象。层次结构就是路径图，脚本可用它来精确定位某个对象。

考虑一个场景，你和朋友 Tony 在中学教室里。下午，太阳透过房子西侧的玻璃墙照射进来，教室越来越闷热。你对 Tony 说："可以打开窗户吗？"并把头转向房间里的某扇窗户。在编程术语中，这就是给对象发出了一个命令(不管 Tony 是否同意)，这种人类交互有许多优于编程的地方。首先，在说话之前可与 Tony 进行眼神的交流，让他知道他是命令的接收者。其次，身体语言也为该命令传递了一些参数，微妙地指出了特定墙上的特定窗户。

然而，假如使用公用地址系统的校长办公室通过广播发出这个命令："可以打开窗户吗？"，就没有人知道你的意思。没有指定对象就发布了这个命令，就是浪费时间，因为每个人都会认为，"那不可能是给我发出的命令"。为了达到类似一对一命令的目的，通过广播发出的命令应该是："312 房间的 Tony Jeffries，可以打开西墙中间的窗户吗？"

下面把最后这个命令转换为 JavaScript 句点格式(参阅第 6 章)。如前所述，对象的引用一般从全局视图出发，然后缩小到具体的视图。从校长办公室的角度来看，目标对象的位置为：

```
room312.Jeffries.Tony
```

假设 Tony 知道如何打开窗户，这是 Tony 的一个方法，Tony 及其方法的完整引用是：

```
room312.Jeffries.Tony.openWindow()
```

至此，工作尚未完成。该方法需要一个参数，来指定要打开哪一扇窗户。在这个例子中，要打开房间 312 西墙中间的窗户。从校长办公室的角度来看，它是：

```
room312.westWall.middleWindow
```

该对象路径图是 Tony 的 openWindow()方法的参数。因此，传送给 PA 系统的完整命令是：

```
room312.Jeffries.Tony.openWindow(room312.westWall.middleWindow)
```

假如不在校长办公室发出命令，而是在轨道航天飞机上，向地球上的所有居民通过广播发出相同的命令，这个对象层次结构就会非常复杂。从航天飞机的角度来看，建立对 Tony 的 openWindow()方法以及待打开窗户的完全引用，可能要花很长时间从数以亿计的对象中找出想要的对象。

需要指出的重要一点是，面向对象编程的范围越小，对对象位置的假设就越多。对于客户端 JavaScript，其范围不会超过浏览器，换言之，JavaScript 使用的每个对象都可在浏览器应用程序中使用。这极少有例外，脚本一般不会访问计算机硬件、操作系统、其他应用程序、桌面或除浏览器程序之外的其他东西。

25.1.2　第一个浏览器文档对象路径图

图 25-1 显示了最低公用标准的文档对象层次结构，它可在所有脚本浏览器中实现，包括可编写脚本的旧浏览器，如 IE3 和 NN2。注意，window 对象位于整个结构的最顶部，在 JavaScript 中编写的任何东西都在浏览器的窗口中。

更多资源请访问稀酷客(www.ckook.com)

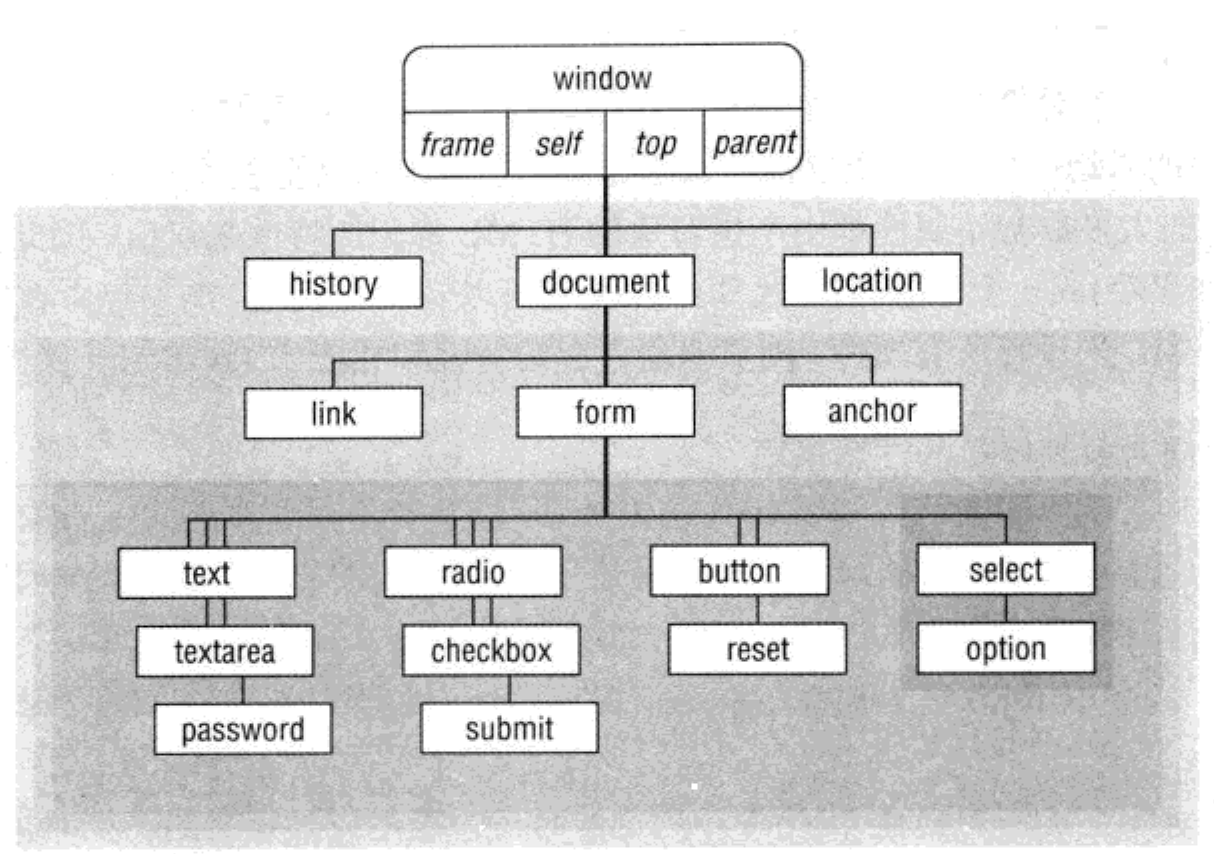


图 25-1　最低公用标准的浏览器文档对象层次结构

注意同心矩形的阴影部分，同一个阴影区域的每个对象都在相对于 window 对象的同一层上。若有一个线条从一个对象指向下一个更暗的阴影矩形，则那个对象包含暗区中的所有对象，在层与层之间最多有一条这样的线。window 对象包含 document 对象；document 对象包含 form 对象；form 对象包含许多不同类型的表单控制元素。

25.2　产生文档对象的过程

浏览器创建的多数对象都是在 HTML 文档载入浏览器时建立的。HTML 代码会告诉 JavaScript 增强浏览器，需要在内存中创建链接、锚点和输入元素等对象。不管脚本是否调用它们，这些对象都存在于内存中。

对于定义这些对象的 HTML 代码来说，唯一可见的差别是一个或多个专用于 JavaScript 的可选特性。这些特性一般指定了用户界面元素要响应的事件，以及用户执行该动作时 JavaScript 应该做什么。通过文档的 HTML 代码来创建对象，就可以有更多的时间来决定如何使用这些对象或让它们为自己服务。

对象是以它们的载入顺序创建的。假如创建一个多框架环境，则直到两个框架都载入后，一个框架中的脚本才能和另一个框架中的对象通信。许多创建多框架和多窗口站点的脚本开发人员最容易犯这方面的错误(详见第 27 章)。

但有趣的是，网站不仅仅拥有文档的 HTML 代码生成的对象，如本章后面所述，W3C DOM 允许在对象层次结构上添加和删除对象。实际上，如果要创建不包含 JavaScript 的 HTML 代码，就可以通过这种方式增删对象。

更多资源请访问稀酷客(www.ckook.com)

25.3　对象的属性

属性一般定义了对象当前的某个设置，该设置可能反映了对象的一个可见特性，例如复选框的状态(选中或未选中)；也可能包含不太明显的信息，例如提交表单的动作和方法。

文档对象的大多数初始属性都由产生对象的 HTML 标记的特性来赋值。因此，属性可以是一个单词(例如名字)或一个数值(例如尺寸)。属性也可以是一个数组，例如包含在文档中的图像数组。假如 HTML 未设置所有属性，浏览器通常会给特性和相应的 JavaScript 属性填充默认值。

富有经验的面向对象程序员的注意事项

虽然基本对象模型层次结构似乎有类/子类关系，但它并不具备真正面向对象环境的许多传统特征。原始的 JavaScript 文档对象层次结构是一个包含层次结构，而不是继承层次结构。对象不会继承高层对象的属性或方法，也不会自动将消息从一个对象传递到另一个对象。因此，不能通过 document 或 form 对象把消息发送给窗口，来调用窗口的方法。所有对象引用都必须是明确的。

只有包含对象定义的 HTML 代码载入浏览器时，才能生成预定义的文档对象。第 23 章学习了如何创建自己的对象，但是这些对象不代表页面上的新可见元素(超过了 HTML、Java applet 和插件程序可描述范围的元素)。

继承在 W3C 定义的对象模型中发挥了作用，如本章后面所述。这个新的层次结构更加通用，满足了 XML 和 HTML 的要求，但是 HTML 对象的包含层次结构(如本节所述)在 W3C DOM 兼容的浏览器中仍然有效。

无论使用什么 DOM，都可能在脚本中改变属性值；区别是自己编写的代码。本章后面将给出如下例子的说明。只要可行，我们都将给出 DOM0 和 W3C DOM 的例子。

在脚本语句中，属性名是区分大小写的。因此，假如属性名是 bgColor，在脚本语句中就必须用精确匹配的大小写字母组合来使用它。但用 HTML 特性设置属性的初始值时，特性名(就像所有的 HTML 一样)是不区分大小写的，因此，<BODY BGCOLOR="white">和<body bgColor="white">会设置相同的 bgColor 属性值。如果标记和特性名只使用小写字母，XHTML 的验证结果将不正确，但多数浏览器都会加载该页面，因为它们仍旧认为标记是不区分大小写的，而不管对页面的 DOCTYPE 指定了什么 HTML 或 XHTML 版本。特性名的大小写不会受到标记属性名大小写的影响。HTML 可在 http://validator.w3.org/上验证。

每个属性都能确定自己的读/写状态，一些属性是只读的，然而可以给它们赋新值，而改变其值。例如，为了把一些新文本放在 textbox 对象中，可把字符串赋给对象的 value 属性：

```
document.forms[0].phone.value = "555-1212";
```

或

```
document.getElementById("myPhoneTextBox").value = "555-1212";
```

对象只要在文档中存在(即它的 HTML 载入文档中)，就可以在该对象中添加一个或多个自定义属性。假如希望在对象中添加一些附加的数据，以便以后提取，这会很有用。为了添加这

更多资源请访问稀酷客(www.ckook.com)

样的属性，只需像赋值那样指定它：

```
document.forms[0].phone.delimiter = "-";
```

或

```
document.getElementById("myPhoneTextBox").value = "-";
```

如本章后面所述，给对象添加属性还有其他方式。只要文档仍在窗口中加载，且脚本没有覆盖该对象，则给对象设置的属性就会继续存在。然而要注意，重载页面通常会破坏自定义属性。

25.4　对象的方法

对象的方法是脚本给该对象提供的命令。一些方法有返回值，但方法不一定要有返回值。也不是每个对象都定义了方法。在多数情况下，从脚本中调用一个方法会执行一些动作，其结果可能很明显(例如重置窗口的大小)，也可能很微妙(例如对内存中的数组排序)。

所有方法的后面都有括号，它们总是位于对象引用的尾部。当方法接受或需要参数时，参数值放在括号内(用逗号分开多个参数)。

对象有对象模型预定义的方法，用户也可以给已有的对象指定一个或多个额外的方法(即在它的 HTML 载入文档中后)。为此，文档中(或该文档可访问的另一个窗口或框架中)的脚本必须定义一个 JavaScript 函数，然后把函数赋给对象的新属性。在下面利用现代浏览器功能的例子中，fullScreen()函数调用了 window 对象的两个方法。把函数引用赋给新的 window.maximize 属性，就可以为 window 对象定义 maximize()方法。因此，按钮的事件处理程序可直接调用这个方法。

```
// define the function
function fullScreen()
{
   this.moveTo(0,0);
   this.resizeTo(screen.availWidth, screen.availHeight);
}
// assign the function to a custom property
window.maximize = fullScreen;
...
<!-- invoke the custom method -->
<input type="button" value="Maximize Window" onclick="window.maximize()" />
```

最后这个例子使用了 DOM0。而使用 W3C DOM 时，该函数和赋予自定义属性的代码是相同的。区别是 onclick 事件处理程序的标识方式。在 DOM0 中，事件直接在 HTML 的 input 标记中定义为对象的一个特性，如上所示。而在 W3C DOM 中，事件可以在脚本中定义，而不是在 HTML 元素上定义：

```
// initialize when the page has loaded
addEvent(window, "load", initialize);

function initialize()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   // add an event to the button
   addEvent(document.getElementById("theButton"), click);
}
```

注意：

本章和本书其他许多地方使用的事件处理程序分配函数是 addEvent()，这是一个跨浏览器的事件处理程序，详见第 32 章。

addEvent()函数在 jsb-global.js 脚本文件中，该文件位于配书光盘中。

25.5　对象事件处理程序

事件处理程序指定对象如何响应由用户动作(例如单击按钮)或浏览器动作(例如完成文档的载入)触发的事件。在最早的 JavaScript 浏览器上，事件处理程序在 HTML 标记中定义为附加的特性。它们包括特性名，后跟一个等号(用作赋值操作符)和一个字符串，该字符串包含事件发生时执行的脚本语句或函数(参阅第 7 章)。

事件处理程序通常在对象的 HTML 标记中定义，也可以指定或更改事件处理程序，就像指定或更改对象的属性一样。事件处理属性的值看起来像是一个函数定义。例如，假设有以下 HTML 定义：

```
<input type="text" name="entry" onfocus="doIt()" />
```

对象的 onfocus(全部为小写)属性值为：

```
function onfocus()
{
   doIt();
}
```

不过，将一个函数引用赋给属性，也可为事件处理程序分配一个完全不同的函数。这类引用不包括函数定义中的括号(参见第 23 章给对象属性指定函数的讨论)。

使用上述的文本域定义，可为事件处理程序分配另一个函数，根据用户在文档其他位置的输入，该域在得到焦点时的行为会有所不同。如果将函数定义如下：

```
function doSomethingElse()
{
   //statements
}
```

然后就可以将函数分配给该域：

```
document.formName.entry.onfocus = doSomethingElse;
```

或

```
document.getElementById("theEntryID").onfocus = doSomethingElse;
```


更多资源请访问稀酷客(www.ckook.com)

由于新函数引用是用 JavaScript 编写的，所以必须遵循函数名的大小写。另外，坚持像属性那样，给事件处理程序名使用全小写字母，这适用于所有浏览器。

如果脚本动态创建了新的元素对象，就可以通过事件处理属性给这些对象分配事件处理程序。例如，下面的代码使用 W3C DOM 语法生成了一个新的按钮输入元素，并为其指定了一个 onclick 事件处理程序，该事件处理程序调用在脚本其他位置定义的函数：

```
var newElem = document.createElement("input");
newElem.type = "button";
newElem.value = "Click Here";
newElem.onclick = doIt;
document.getElementById("theFormId").appendChild(newElem);
```

25.6　对象模型概述

在脚本浏览器从 NN2 和 IE3 开始的整个发展过程中，总共有 6 个不同的 DOM 系列。即使仅为一个当前的浏览器版本开发内容，也会吃惊地发现，在自己的设计空间中有多个 DOM 系列的成员。

研究对象模型的发展历程对脚本编程新手非常有用。学习当前浏览器中最新的对象模型技巧是很简单的，但使用更早期的浏览器访问自己的页面时，脚本将在很多地方不起作用。即使只准备支持现代浏览器，对象模型的历史也是 JavaScript 知识库的一个有用部分。因此，下面看看 6 种主要的对象模型及其诞生过程。表 25-1 列出了各个对象模型系列和支持它们的浏览器版本。注意如果对某种对象模型的支持程度在提高，就只列出版本号(例如从“部分”支持到“多数”支持)。本章的后面还罗列一些指南，可以根据它们选择最适合用户的对象模型。

表 25-1　对象模型系列

模　　型	浏览支持
基本对象模型	NN2+，IE3+，Moz1+，FF2+，Safari1+，Opera+，Chrome+
基本附加图像对象模型	NN3+，MacIE3.01+，IE4+，Moz1+，FF2+，Safari1+，-，Opera+，Chrome+
NN4 扩展	NN4
IE4 扩展	IE4+ (在所有版本中，一些功能需要 Win32 OS)
IE5 扩展	IE5+(在所有版本中，一些功能需要 Win32 OS)，Opera 9.62+ (非常有限)
W3C DOM (I 和 II)	IE5+ (部分)，Moz1 (多数)，FF2+ (多数)，Safari1 (部分)，Safari1.3+ (多数)，Opera9.62+ (多数)，Chrome1+ (多数)

尽管浏览器在为 Web 标准提供统一支持方面有了很大进步，但尚未完全达到目标，认识到这一点很重要。在编写本书时，目前的浏览器都未能完全精确地支持 W3C DOM 的 Level I 和 II 标准。Firefox 2、Safari 1.3/2、Opera 9 和 Chrome 1 已经大大地缩小了兼容性上的差距，但仍存在一些不太影响编写 HTML 的问题。

更多资源请访问稀酷客(www.ckook.com)

25.7　基本对象模型

第一个脚本浏览器 Netscape Navigator 2 实现了一个非常基本的 DOM。本章前面的图 25-1 提供了一个可用于脚本对象的可视向导。层次结构开始于窗口，其下是文档、表单和表单控件元素。文档是屏幕上几乎不可改变的页面，只有交互性元素(链接和表单元素，如文本域、按钮等)处理为有属性、方法和事件处理程序的对象。

对表单控件的重视引出了很多想法，这些想法在当时是很新颖的。因为脚本能检查表单控件的值，所以表单在客户端可以预先验证。假如页面包括执行一些运算的脚本，数据项和运算结果通过可编辑的文本域显示出来。

浏览器加载了页面后，文档外的其他对象(window、history 和 location 对象)允许通过脚本访问浏览器的简单而实用的属性。该环境的全局观察点是 navigator 对象，它包括关于浏览器品牌和版本的属性。

Internet Explorer 3 推出后，Navigator 2 的短暂生命就终结了。尽管 NN3 以预发行的形式广泛使用，但 Internet Explorer 3 实现了 NN2 中的基本对象模型(加上 NN3 中的一个 window 对象属性)。因此，尽管浏览器的版本号有矛盾，但 NN2 和 Internet Explorer 3 的 DOM 本质上是相同的。在 Internet 时代的一段短暂时间，Microsoft 和 Netscape 的 DOM 完全相同，虽然这是在非常简单的层次上。

25.8　附加图像的基本对象模型

在 Internet Explorer 3 推出后不久，Netscape 就推出了 Navigator 3，它在原始版本上构建了对象模型。几个已有的对象，特别是 window 对象，有了新的属性、方法和事件处理程序。脚本也可以和作为对象的 Java applet 通信，但当时最新的对象是 Image 对象和用于 document 对象的 image 对象数组。

Navigator 3 image 对象的多数属性只能读取在<img>标记中赋予特性的值，但在页面载入后，可修改一个属性——src 属性。脚本还可在固定的图像矩形内转换图像。虽然这些新 image 对象没有与鼠标相关的事件处理程序，但可以在链接(它有 onmouseover 和新 onmouseout 事件处理程序)中嵌入图像，这样脚本编程人员就可以实现“图像滚动”功能，使页面更加生动。

越来越多的新脚本开发人员开始研究在页面中添加 JavaScript 的可能性，所以，当他们为 Navigator 3 设计的图像转换功能不能用于 Internet Explorer 3 时，就备感失望。即使没有 image 对象，也很容易编写脚本，来防止 Internet Explorer 3 中的脚本错误，可惜缺乏这么酷的页面功能会使许多人失望。安装了 Navigator 2 的用户群也会对此感到失望。后来 Internet Explorer 3.01 的 Macintosh 版本(用于 Mac 浏览器的 Internet Explorer 第二版)实现了可编写脚本的 image 对象，这使事情变得更为复杂。

尽管存在这些兼容性问题，Navigator 3 实现的对象模型还是最终成为后来 DOM 的基本参考。为这个对象模型编写的代码可运行在 Navigator 3 和 Inernet Explorer 4 及其以后的所有浏览器上，包括这两个品牌的最新版本和其他现代浏览器，很少有例外。

PDG

更多资源请访问稀酷客(www.ckook.com)

25.9　仅用于 Navigator 4 的扩展

下一个推出的浏览器是 Netscape Navigator 4，它给现有对象添加了许多功能，这使脚本开发人员拥有了更强大的力量。利用脚本可检测的 screen 对象属性(例如，用户的显示器屏幕有多大)，可移动和改变浏览器窗口的大小。两个代表对象模型新思想的概念是增强的事件模型和 layer 对象。

25.9.1　事件捕获模型

Navigator 4 在指令集中添加了许多新事件。键盘事件和更多的鼠标事件(onmousedown 和 onmouseup)允许脚本响应表单控件元素和链接上的更多用户动作。所有这些事件的工作方式与先前的对象模型一样，事件处理程序一般作为特性赋予元素标记(也可将事件处理程序指定为脚本语句中的属性)。为了利用 Navigator 4 对象模型中的一些动态 HTML 功能，还大大增强了事件模型。

系统最基础的观点是，用户在一个支持事件的对象上执行某个物理动作时(例如，单击表单按钮)，事件就通过文档对象层次结构从顶层到达按钮。假如有多个对象共享一个事件处理程序，在一个地方捕获事件就可能更简便(window 或 document 对象层)，而不是把这个事件处理程序分配给所有元素。Navigator 4 默认允许事件到达目标对象，就像早期的浏览器那样。还可以启用 window、document 或 layer 对象中的事件捕获功能，来捕获事件。之后，该事件在传递给原始目标前，可在上层处理，也可重定向到另一个对象中。

不管是否捕获事件，Navigator 4 事件模型都为每个事件创建一个 event 对象(首字母 e 小写，以便与静态 Event 对象区分开)，这个对象的属性包含了特定事件的更多信息，例如为键盘事件按下的键盘字符或页面上单击事件的位置。任何事件处理程序都可检查 event 对象属性，来了解事件的更多信息，并相应地处理事件。

25.9.2　层

也许 Navigator 4 对象模型增加的最激进的内容是一个新对象，它表示一个全新的 HTML 元素(layer 元素)。层是一个容器，可包含自己的 HTML 文档，然而它在主文档之前的平面上。通过脚本可改变层的大小、移动和隐藏层。这个新元素首次允许遮挡 HTML 页面中的其他元素。

为在文档对象层次结构中容纳 layer 对象，Netscape 定义了一个嵌套的层次结构，使层包含在文档中。因此，document 对象添加了一个属性(document.layers)，它是文档中 layer 对象的数组，这个数组只显示当前 document 对象中的第一级 layer 对象。

每个层都有自己的 document 对象，因为每个层都可根据需要载入一个外部 HTML 文档。layer 对象作为一个可定位的元素，有许多属性和方法，允许脚本移动、隐藏、显示和改变其堆栈顺序。

可惜，W3C 不同意把 Netscape 的<layer>标记作为 HTML 4 规范的一部分。所以，这个元素仅存在于 Navigator 4 中(在 Moz1 或后续版本中未实现)，使用 layer 对象和其嵌套引用的脚本也只能用于 Navigator 4。

更多资源请访问稀酷客(www.ckook.com)

25.10 Internet Explorer 4+扩展

在 Navigator 4 发行的几个月后，Microsoft 推出了 IE4，开辟了重要的新领域。其主要改进是提供了所有的 HTML 元素、CSS 的脚本支持和新的事件模型，其他一些新增内容只用于 Windows 32 位操作系统平台。

25.10.1 HTML 元素对象

对象模型中最大的变化是，每个 HTML 元素都变成了脚本对象，同时仍支持原始对象模型。Microsoft 发明了 document.all 数组(也叫做集)，不管元素如何嵌套，该数组都包含文档中每个元素的引用。假如把唯一标识符(名称)赋给元素的 id 特性，可通过下列语法引用元素：

```
document.all.elementID
```

多数情况下，可省略引用的 document.all.部分，从元素 ID 开始。

每个元素对象都有一套全新的属性和方法，允许脚本开发人员以前所未有的方式控制文档内容。这些属性和方法详见第 26 章，但这里要讨论几组属性。

innerHTML，innerText、outerHTML 和 outerText 这 4 个属性提供了对文档主体中实际内容的读/写访问。这意味着不再需要使用文本框来显示脚本的计算结果，可随意修改段落内、表单元或其他地方的内容。当元素的内容改变时，浏览器的显示引擎会即刻回显文档。这个功能实现了“动态 HTML”中的“动态”。对在早期浏览器上编写静态页面的人来说，这个功能无异于一个新发现，现在人们已习惯使用它了。

有一系列“偏移”属性与元素在页面中的位置相关，它们与 CSS 执行的定位操作不同。通过这些属性可得到页面中任何元素的大小和位置，更便于把可定位的内容移动到文档中元素的上面，并根据浏览器窗口的当前大小显示在不同位置。

最后，Style 属性是为元素定义的 CSS 规范的入口。重要的是，脚本可修改 sty1e 对象的许多属性。因此，在页面载入后，可以修改字体规范、颜色、边框和位置属性，页面的动态回显功能会执行必要的布局修改(例如，调整到更大的字体)。

25.10.2 元素包含层次结构

Internet Explorer 4 能识别原始对象模型的元素层次结构(如图 25-1 所示)，但 Internet Explorer 4 的 DOM 没有把这种层次结构完全扩展到其他元素中。否则，表中的 td 元素就必须根据包含它的 tr 或 table 元素来定位(就像表单控件元素必须通过包含它的 form 元素来定位)。图 25-2 显示了所有的 HTML 元素如何在 document 对象下组合在一起。Document.all 数组摊平了包含层次结构，以便引用对象。对嵌套层次最深的 TD 元素的引用仍然是 document.all.cellID。window 对象中的突出路径是使用 Explorer 4 文档对象层次结构时的主要引用路径。

更多资源请访问稀酷客(www.ckook.com)

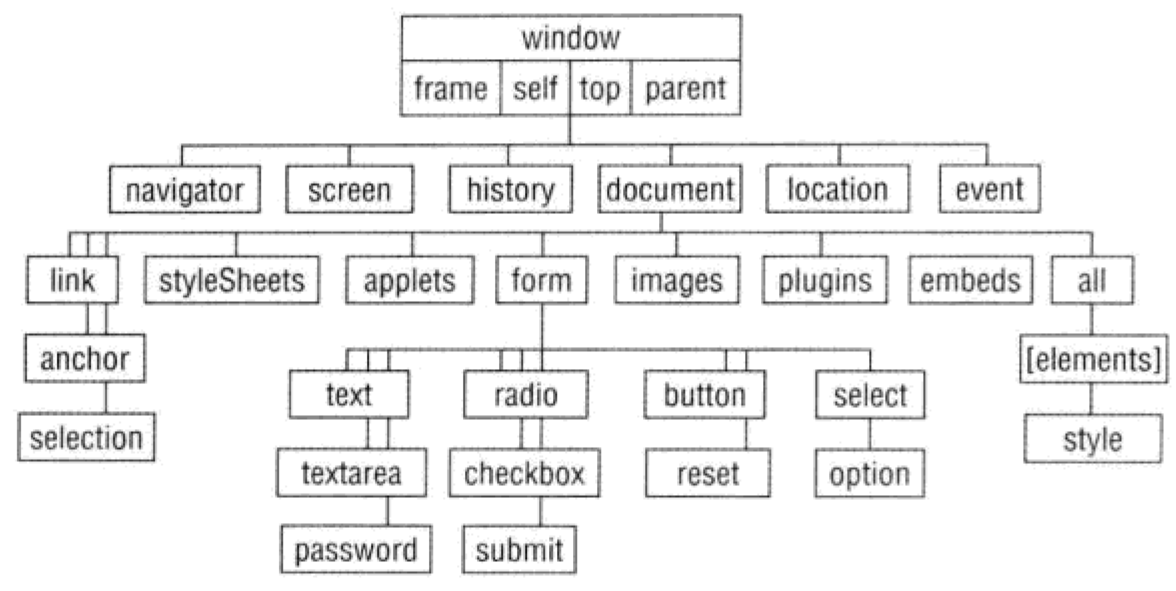


图 25-2 IE4 的文档对象层次结构

然而，Internet Explorer 4 中的元素包含层次结构非常重要还有其他原因。因为一个元素可从包含它的元素中继承一些样式表特性，所以在设计文档的 HTML 时，应将每条内容嵌入容器。段落元素是文本容器(有首尾标记)，而不是文本块之间的行间隔。Internet Explorer 4 在容器和嵌入其中的元素之间引入了父子关系的概念。同时，元素的位置可相对于其最近的外层定位元素来计算。

这里的前提是元素包含层次结构与对象引用不相关(类似于原始对象模型)，它和元素相对于页面其他内容的上下文有关。

25.10.3 层叠样式表

虽然 Microsoft 的版本 4 浏览器进入浏览器市场的时间比 Netscape 晚，但在该浏览器面世前，CSS Level 1 规范有了更全面的发展，Microsoft 因此获益良多。它其实远比 Navigator 4 更完整(但它和标准不是 100%兼容)。

和第一代 CSS 规范相比，样式表属性的脚本编写功能显得有点奇怪，它忽略了用 JavaScript 编写样式的功能。许多 CSS 属性名是带有连字符的词(例如 text-align、z-index)。但在 JavaScript 中，标识符名不允许使用连字符，所以需要把多字 CSS 属性名转换为 camelCase 样式的 JavaScript 属性名。因此，text-align 变成 textAlign，z-index 变成 zIndex。通过元素的 style 属性可以访问所有的这些属性：

```
document.all.elementID.style.stylePropertyName
```

或

```
document.getElementById("myElement").style.stylePropertyName
```

在 Internet Explorer 4 和后续版本中，样式表的脚本编程功能会带来所谓的“幻影页面效果(phantom page syndrome)”。页面的基本 HTML 下载到浏览器中后，浏览器会处理页面的布局，这时就会出现这种效果。页面载入时，不是所有的内容都可见，否则页面会显得非常凌乱。然后，页面中的 onload 事件处理程序令脚本为页面设置样式或内容，元素就会显示在它们最后的静止位置。这可能使一些用户不安，因为他们刚开始看到一个可单击的链接，但把光标移到单

更多资源请访问稀酷客(www.ckook.com)

击的位置上时，会重新显示页面，会将链接移动到页面的其他地方。

注意：

对于使用 32 位 Windows 操作系统的 Internet Explorer 用户，Internet Explorer 4 在对象模型中包括一些增强显示效果的额外功能。滤镜是在 body 文本上提供各种可视效果的样式表附加功能。例如，将滤镜样式应用于文本，就可以为文本添加阴影或发光效果，或将每张幻灯片的内容放在定位的 div 元素中，就可以创建与幻灯片演示相同的效果。尽管滤镜遵循 CSS 语法，但它们并非 W3C 规范的组成部分。

25.10.4　事件冒泡

Netscape 为 Navigator 4 引入了事件模型，Microsoft 也为 Internet Explorer 4 引入了事件模型。然而，对跨浏览器的脚本开发人员来说，这两个事件模型相径庭。IE 事件不是沿层次结构向下传递到目标元素，而是相反，事件从目标元素开始，沿元素包含层次结构向上“冒泡”，最终到达 window 对象。途中任何对象的事件处理程序都可在此事件上执行额外的处理操作。因此，假如只用一个事件处理程序处理页面上所有的单击事件，就应把事件处理程序赋给 body 或 window 对象，因为这些事件最终会到达 body 或 window 对象(假如包含层次结构中的其他对象没有取消事件冒泡)。

Internet Explorer 也有一个 event 对象(window 对象的一个属性)，它包含事件的细节，如为键盘事件按下的键盘键和鼠标事件的位置，这些属性的名称与 Navigator 4 的 event 对象属性完全不同。

尽管 Navigator 4 和 IE4 看起来不兼容，但假如它们的事件模型不完全对立，则可以在两个浏览器中用相同的脚本处理事件(例子见第 32 章和第 59 章)。IE 直到 7 都是支持 Internet Explorer 4 事件模型是唯一模型。

25.11　Internet Explorer 5+扩展

随着 Internet Explorer 5 的发行，Microsoft 在 Internet Explorer 4 开始使用的专用对象模型中内置了更多功能。虽然对象的范围没有改变，对象的属性、方法和事件处理程序的数目却大大增加了。其中一些新增内容是为了满足 W3C DOM 规范的要求(在下一节讨论)，这偶尔会和 Internet Explorer 4 不兼容。Microsoft 也推出了一些仅用于 Windows 用户的功能，它们不一定会成为业界标准，例如 DHTMI 行为和 HTML 应用程序。

DHTML 行为是一些保存为外部文件的脚本，它定义了一些可应用于任何元素的动作，通常是改变一个或多个样式属性。其目标是创建一个可重用的组件，以载入需要这个行为的文档中。DHTML 行为的一个例子是，定义一个行为，只要光标指向文本，就将元素的文本转换为红色，光标离开文本，文本就再转换回黑色。通过类似 CSS 的语法规则把该行为赋给文档中的一个元素时，元素就会执行这个行为，来响应用户。这个行为可应用于文档的任何元素中。

交叉引用：

第 26 章描述 addBehavior()方法时就有一个 DHTML 行为的例子，配书光盘的第 50 章对此

更多资源请访问稀酷客(www.ckook.com)

进行了更多的讨论。

HTML 应用程序(以 Microsoft 的术语是 HTA)是 HTML 文件，它包含一个名为 hta:application 的 XML 元素。HTA 可从服务器上下载到 Internet Explorer 5 或更新版本中，就像它是一个 Web 页面(但它的文件扩展名为.hta，而不是.htm 或.html)一样。也可在客户机上安装 HTA，其行为非常像一个有桌面图标的应用程序，该程序还可明确控制窗口的外观。HTA 在客户端被赋予了更大的安全特权，使其能像常规程序一样工作。实际上，可关闭系统菜单栏，使用 DHTML 技术为应用程序构建自己的菜单栏。HTA 的实现细节超出了本书的讨论范围，但是应该注意它们的存在，更多信息可访问 http://msdn.microsoft.com。

25.12 W3C DOM

Netscape 和 Microsoft 之间的浏览器对象模型冲突使开发工作越来越难进行，脚本开发人员渴望有个公用标准，就像针对内容和样式的 HTML 和 CSS 标准一样。W3C 承担了制订 DOM 标准的任务，该标准就是 W3C DOM。

W3C DOM 工作组的任务是创建能应用到 HTML 和 XML 文档中的 DOM。XML 文档的标记名可以任意(由 Document Type Definition 或 XML Schema 定义)，它不像 HTML 文档那样拥有内在的结构或固定的元素词汇。因此，DOM 规范必须适用于 HTML 的已知结构(在 HTML 4 规范中定义)和 XML 文档的未知结构。

为了有效地完成这项工作，工作组把 DOM 规范分为两个部分，第一部分叫做核心 DOM，它定义了 HTML 和 XML 文档共享的基本文档结构规范。这个规范包含文档，文档包括元素，元素包含标记名和属性，一个元素能包含 0 个或更多其他元素。DOM 规范的第二部分规定了只应用在 HTML 中的元素以及其他特性。该 HTML 部分继承了核心 DOM 的所有功能，给旧浏览器中已实现的对象模型提供了向后兼容的方法，还提供了具有新功能的框架结构。

对于富有经验的脚本开发人员而言，认识到 W3C DOM 并没有从现有浏览器的对象模型中继承所有的功能是很重要的。Internet Explorer 4 和更新版本中对象模型的许多功能没有包含在 W3C DOM 规范中。假如用户熟悉 IE 环境，并希望进行 W3C DOM 规范编程，就必须改变一些做法，如本章所述。在许多方面，特别是 DHTML 应用程序，W3C DOM 是一个具有新概念的全新 DOM，要在此环境中成功地编写脚本，就必须掌握这些概念。

基于 Mozilla 的浏览器基本实现了所有的 DOM Level 1 和大多数 Level 2，而 Microsoft 从 Internet Explorer 5.5 开始只实现了部分 W3C DOM 规范。尽管 IE6 和 IE7 实现了 W3C 的更多功能，但没有实现一些重要特性，比如 W3C DOM 事件。其他现代浏览器，比如 Chrome 1+、Safari 1.3+和 Opera 9+，提供了全面的 W3C DOM 支持，就支持 W3C DOM 而言，大致可以与 Mozilla 相媲美。

25.12.1 DOM 层

就像多数 W3C 规范一样，仅一个 DOM 版本肯定不够。DOM 工作组的工作量太大了，不可能在一个 DOM 版本中包含所有的内容，因此，DOM 是一个不断演进的规范。规范通常很少和

更多资源请访问稀酷客(www.ckook.com)

浏览器的发行时间一致，因此，任何给定浏览器的发行版本常常只包括某个最近的 W3C 版本。

DOM 第一个正式的规范 DOM Level 1，在 NN4 和 IE4 面世后才发行。Level 1 的 HTML 部分包括 DOM level 0(没有按这个名称发布标准)，它本质上是 Navigator 3 实现的对象模型(大多在 Internet Explorer 3 中实现，再加上 image 对象)。可能 Level 1 最大的疏忽就是事件模型，它甚至忽略了 NN2 和 IE3 实现的简单事件模型。

DOM Level 2 在 Level 1 的基础上构建。除了 Level l 中 Core 和 HTML 部分的几个增强功能外，Level 2 还在事件模型、检查文档层次结构的方法、XML 命名空间、文本范围、样式表和样式属性中添加了很多重要的新内容。Level 3 DOM 的一些模块已经进入“推荐”状态，尽管 IE 实现这些功能还有很长的路要走，但其他主流浏览器至少部分实现了一些模块，包括仍在设计的一些内容。

25.12.2 规范中恒定不变的部分

W3C 采用 DOM Level 0 作为 DOM 中 HTML 部分的起点，允许许多现有脚本代码在 W3C DOM 兼容的浏览器上工作。原始对象模型中的每个对象，从 document 对象开始(图 25-1)，以及 image 对象都在 DOM Level 0 中，几乎所有对象属性和方法都是可用的。

更重要的是，当改变 W3C DOM 中其他元素的引用时，引用对象(如表单、表单控件元素和图像)的旧方法仍是有效的。假如工作组从无到有地进行设计，document 对象就不可能包含由表单、链接和图像数组组成的属性。

现有代码可能遇到的唯一潜在问题与 document 对象以前的几个属性有关。在新的 DOM 中，document 对象的 4 个样式相关属性(alinkColor、bgColor、linkColor 和 vlinkColor)变成了 body 对象的属性(引用为 document.body)。此外，这三个链接颜色属性在处理中使用了新名称(aLink、link 和 vLink)，然而，目前，IE6 和 Moz1 仍向后兼容旧的文档对象颜色属性。

还要注意，DOM 规范只与文档及其内容相关。Window、navigator 和 screen 等对象并不是 DOM Level 2 规范的一部分。脚本开发人员仍希望浏览器厂商考虑这些领域的兼容性。

25.12.3 W3C DOM 不具备的特性

前面提到，尽管 W3C DOM 并不只是重申了现有浏览器的规范，但 Internet Explorer 和 Netscape Navigator 对象模型的许多方便功能并没有出现在 W3C DOM 中。假如在 Internet Explorer 4 及后续版本或 Navigator 4 中开发 DHTML 内容，就必须学会不用这些特性进行开发。

Navigator 4 的<layer>标记并未出现在 W3C 中。因此，有关它的标记和脚本编写约定都不在 W3C DOM 中。为了减轻一些脚本开发人员的负担，document.layerName 引用(甚至是带有嵌套层的更复杂引用)从对象模型中消失了。定位元素只处理为另一个元素，它有一些特殊的样式表属性，这些属性可以把它移动到页面上的任何位置，堆在其他定位元素之间或者在视图中隐藏起来。

在许多流行的 Internet Explorer 4+功能中，未出现 W3C DOM 中的一个是 HTML 元素的 document.all 集合和 4 个用于动态内容的元素属性(innerHTML、innerText、outerHTML 和 outerText)。W3C 提供了一个新方法，可从文档中得到所有元素的数组，但只有使用一个繁杂的语句序列，才能生成 HTML 内容来替换现有的内容，或将其插入文档中(见本章后面的 25.12.5

更多资源请访问稀酷客(www.ckook.com)

一节)。然而，所有新浏览器都为 HTML 元素对象实现了 innerHTML 属性，只是 Firefox 例外，它实现了其他 3 个属性。

25.12.4　新的 HTML 惯例

在 W3C DOM 中开发 DHTML 的可能性依赖于目前众多 HTML 作者采用的一些 HTML 惯例。这些惯例(由 HTML 4 规范支持)的核心是确保所有内容都在某种 HTML 容器内。因此，不需要用<p>标记作为文本块之间的分隔符，而用<p>...< / p>标记对包围每段文本即可。否则，浏览器就把每个<p>标记作为一段的开始，在下一个<p>标记或其他块级元素前结束段元素。

虽然浏览器可以忽略某些结束标记(例如 td、tr 和 li 元素的结束标记)，以便达到向后兼容的目的，但最好养成提供这些结束标记的习惯，以帮助理解元素的影响范围。把这作为一个最佳实践方式，也意味着代码是有效的。

要编写的任何元素(不管是否改变它的内容或样式)都应该有一个标识符，它通常赋给元素的 id 特性，标识符在当前网页中应是唯一的。假如将表单内容提交给服务器，表单控件元素仍需要 name 特性，但可给控件的 id 特性赋予相同或不同的标识符。脚本可使用 id 或 document.formReference.elementName 引用来获取一个控件对象。标识符本质上与赋给表单和表单输入元素的 name 特性值是一样的。根据与 name 特性值相同的规则，id 标识符必须是一个单词(没有空格)，它不能以数值开头(避免在 JavaScript 中出现冲突)，且不应使用标点符号(下划线除外)。

25.12.5　新 DOM 概念

W3C DOM 引入了几个概念，除非用户已经广泛地使用树层次结构术语，否则这些概念就可能是全新的。对编写脚本影响最大的概念是引用元素和节点的新方法。

1. 元素引用

DOM Level 0 中对象的脚本引用仍存在于 W3C DOM 中，以保证向后兼容性。因此，把一个表单输入元素的 name 特性赋值为 userName，则该元素的引用总是下列形式：

```
document.forms[0].userName
```

或

```
document.formName.userName
```

但因为文档的所有元素都可由 document 对象访问，所以可使用 document 对象的 document.getElementById()方法来访问指定 ID 的任何元素。该方法唯一的参数是用户要得到的对象标识符的字符串版本。为将其用于 Internet Explorer 4 对象模型，可使用下列 HTML 段标记：

```
<p id="myParagraph">...</p>
```

在 Internet Explorer 4 及更新版本中，可用下面的方式引用这个元素：

```
var elem = document.all.myParagraph;
```

虽然 document.all 集合在 W3C DOM 中没有实现，但 document 对象的 getElementById()方

更多资源请访问稀酷客(www.ckook.com)

法(在 Internet Explorer 5 及后续版本、Mozilla、Safari 及其他浏览器中可用)允许使用 ID 来访问任何元素：

```
var elem = document.getElementById("myParagraph");
```

这种方法适用于根据 ID 来引用元素。遗憾的是，对于脚本开发人员，该方法很难输入，因为它是区分大小写的，注意结束字符是小写的 d。

2. 节点层次结构

容器在 W3C DOM 的底层结构上起作用。HTML(或 XML)文档中的每个元素或独立文本块都是一个对象，它包含在其外层容器中。下面通过一个简单的 HTML 文档来说明这个系统是如何工作的。程序清单 25-1 用来显示元素和字符串块的包含层次结构。

程序清单 25-1　一个简单的 HTML 文档

```
<html>
   <head>
      <title>
         A Simple Page
      </title>
   </head>

   <body>
      <p id="paragraph1">
         This is the
         <em id="emphasis1">
            one and only
         </em>
         paragraph on the page.
      </p>
   </body>
</html>
```

在程序清单中看不到 document 对象。这个页面载入浏览器时，会自动创建 document 对象。document 对象包含了程序清单 25-1 中的所有内容。因此，document 对象有一个嵌套元素：html 元素。html 元素又有两个嵌套元素：head 和 body。head 元素包含 title 元素，title 元素又包含一个文本块。在 body 元素的下面，p 元素包含三块：一个字符串块、em 元素和另一个字符串块。

根据 W3C DOM 的术语，每个容器、独立元素(例如 br 元素)或文本块都称为节点，节点是 W3C DOM 的基本构建块。一个容器包含另一个容器时，节点之间具有父子关系。就像在现实生活中一样，父子关系只存在于相邻的代之间，因此一个节点可有 0 个或多个子节点。然而，节点树中第 3 代节点的数目不影响与父节点相关的子节点数目。因此，在程序清单 25-1 中，html 节点有两个子节点 head 和 body，它们是共有一个父节点的兄弟节点。body 元素有一个子节点(p)，该子节点又包含三个子节点(两个文本节点和一个 em 元素节点)。

可绘制出程序清单 25-1 中文档的层次结构树图，如图 25-3 所示。

更多资源请访问稀酷客(www.ckook.com)

```
document
+--<html>
    +--<head>
    |  +--<title>
    |      +--"A Simple Page"
    +--<body>
       +--<p ID="paragraph1">
           +--"This is the  "
           +--<em ID="emphasis1">
           |   +--"one and only"
           +--" paragraph on the page."
```

图 25-3　程序清单 25-1 中文档的节点树图

注意：

假如文档的源代码在<html>标记之前包含一个文档类型定义(Document Type Definition，在 DOCTYPE 元素中)，浏览器就把该 DOCTYPE 节点看做 HTML 元素节点的兄弟节点。此时，根 document 节点包含两个子节点。

W3C DOM(直到 Level 2)定义了 12 个不同类型的节点，其中 7 个能在 HTML 文档中直接应用。这 7 个节点类型列在表 25-2 中(其他节点类型应用于 XML)。在这 7 种类型中，最常用的是 document(文档)、element(元素)和 text(文本) 三个类型，所有 W3C DOM 浏览器(包括 Internet Explorer 5 及更新版本、Mozilla、Safari 和其他浏览器)都实现了这 3 个常用的节点类型，而 Mozilla 实现了所有类型。IE6 只有一个没有实现，Safari 1 有两个没有实现。

表 25-2　W3C DOM 中与 HTML 相关的节点类型

类　型	编　号	节　点　名	节　点　值	说　明	IE	Moz	Safari
Element	1	标记名	Null	任何 HTML 或 XML 标记元素	6+	1+	1+
Attribute	2	特性名	特性值	元素中的名-值特性对	6+	1+	1+
Text	3	# text	文本内容	元素包含的文本段	6+	1+	1+
Comment	8	# comment	注释文字	HTML 注释	6+	1+	4+
Document	9	#document	Null	根 document 对象	6+	1+	1+
DocumentType	10	DOCTYPE	Null	DTD 说明	否	1+	4+
Fragment	11	# document-fragment	Null	文档外的一个或多个节点系列	6+	1+	1+

把表 25-2 中的节点类型应用于图 25-3 中的节点图，则该简单的页面包含 1 个文档节点、6 个元素节点和 4 个文本节点。

3. 节点属性

节点具有许多属性，其中大多数都是与当前节点相关的其他节点的引用。表 25-3 列出了 DOM Level 2 中所有节点类型共享的所有属性。

更多资源请访问稀酷客(www.ckook.com)

注意：
表 25-3 中的所有属性也在 Internet Explorer 6+以及 Moz1+中实现了，请参见第 26 章中所有 HTML 元素对象共享的属性列表。这是因为，HTML 元素作为一种节点类型，继承了原型节点的所有属性。

表 25-3　节点对象属性(W3C DOM Level 2)

属　性	值	说　明	IE6Win+	IE5Mac+	Moz1	Safaril
nodeName	String	节点类型有变化，参见表 25-2	是	是	是	是
nodeValue	String	节点类型有变化，参见表 25-2	是	是	是	是
nodeType	Integer	表示每个类型的常量	是	是	是	是
parentNode	Object	引用相邻的外层容器	是	是	是	是
childNodes	Array	按源代码次序排列的所有子节点	是	是	是	是
firstChild	Object	引用第 1 个子节点	是	是	是	是
lastChild	Object	引用最后一个子节点	是	是	是	是
previousSibling	Object	按源代码次序引用兄弟节点	是	是	是	是
nextSibling	Object	按源代码次序引用下一个兄弟节点	是	是	是	是
attributes	NodeMap	特性节点的数组	是	一些	是	是
ownerDocument	Object	包含文档对象	是	是	是	是
namespaceURI	String	命名空间定义的 URI(仅用于元素和特性节点)	是	否	是	是
Prefix	String	命名空间前缀(仅用于元素和特性节点)	是	否	是	是
localName	String	可应用到受命名空间影响的节点	是	否	是	是

为了帮助理解关键节点属性的意义，表 25-4 提供了程序清单 25-1 的简单页面中几个节点的属性值。对于每个节点列，在图 25-3 中找到这个节点，然后找到这个节点的属性值列表，将这些值与图 25-3 中的实际节点结构进行对比。

表 25-4　简单 HTML 文档的所选节点的属性

属　性	节　点			
	document	html	p	"one and only"
nodeType	9	1	1	3
nodeName	#document	html	p	#text
nodeValue	Null	null	null	"one and only"
parentNode	Null	document	body	em

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	节　点			
previousSibling	Null	null	null	null
nextSibling	Null	null	null	null
childNodes	Html	head body	"This is the" em paragraph On the page.	(无)
firstChild	Html	head	"This is the"	null
lastChild	Html	body	"paragraph on the page．"	null

nodeType 属性是一个整数，有助于脚本遍历未知的节点集。HTML 文档的多数内容是类型 1(HTML 元素)或类型 3(文本节点)，最外层容器是类型 9，也就是文档。节点的 nodeName 属性可以是节点标记的名称(用于 HTML 元素)或常量(如表 25-2 所示，前面有＃前缀)。除了文本节点类型外，nodeValue 属性也是 null，文本段的值是节点的实际文本字符串，换言之，对于 HTML 元素，W3C DOM 不把容器的 HTML 当做字符串。

面向对象的 W3C DOM

假如你熟悉面向对象(OO)编程的概念，肯定会赞赏 W3C 定义 DOM 的 OO 方式。Node 对象包括属性(表 25-3)和方法(表 25-5)集合，它们由基于 Node 的每个对象继承。继承 Node 行为的大多数对象都有自己的属性，和/或定义其特殊行为的方法。图 25-4 显示了(用 W3C DOM 术语)Node 根对象的继承树，其中多数项都在 Core DOM 中定义，而黑体字显示的项来自于 HTML DOM 部分。

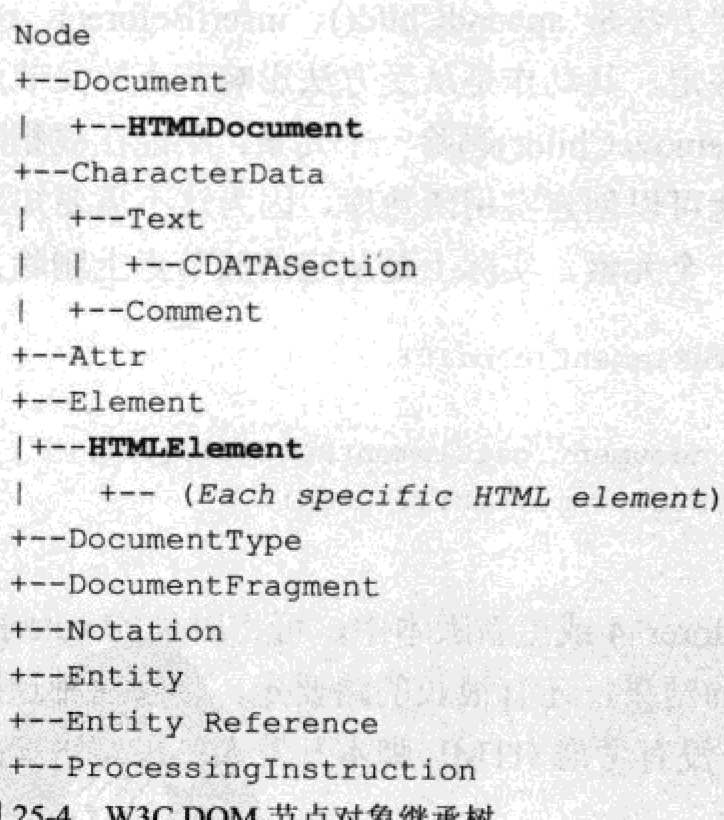


图 25-4　W3C DOM 节点对象继承树

从上图可以看出，各个 HTML 元素继承了通用 HTML 元素的属性和方法，通用 HTML 元素继承了核心 Element 对象，而核心 Element 对象又继承了基本 Node 元素。

了解 Node 对象的继承关系对于编写 DOM 并不重要，但它有助于解释 W3C DOM 的 ECMA Script Language Binding 附录，也可解释简单元素对象为什么拥有如此多与其相关的属性和方法。

更多资源请访问稀酷客(www.ckook.com)

能否使用节点面向关系的全部属性是令人置疑的，主要是因为从任何其他节点到达某个节点的路径有一些重叠的部分。parentNode 属性是当前节点的直接容器的引用，所以很重要。虽然 firstChild 和 lastChild 属性直接指向容器内的第一个和最后一个子节点，但多数脚本一般使用 childNodes 属性和数组记号，通过 for 循环遍历子节点。假如没有子节点，childNodes 数组的长度为 0。

4. 节点方法

要在 W3C DOM 领域中修改节点的 HTML 内容，涉及为原型 Node 定义的方法，表 25-5 列出了节点方法以及 W3C DOM 浏览器对它们的支持。

表 25-5　节点对象方法(W3C DOM Level 2)

方　　法	说　　明	IE5+	Moz 1	Safari 1
appendChild(newChild)	将子节点添加到当前节点的末端	是	是	是
cloneNode(deep)	获取当前节点的副本(可以选择获取子节点的副本)	是	是	是
hasChildNodes()	确定当前节点是否具有子节点(Boolean)	是	是	是
insertBefore(new, ref)	在一个子节点前面插入一个新的子节点	是	是	是
removeChild(old)	删除一个子节点	是	是	是
replaceChild(new, old)	使用新的子节点替换旧的子节点	是	是	是
isSupported(feature, version)	确定节点是否支持特定的功能	否	是	是

修改内容的重要方法是 appendChild()、insertBefore()、removeChild()和 replaceChild()。但注意，这些方法都假定，其动作是从受方法影响节点的父节点的角度来考虑的。

例如，要使用 removeChild()删除一个元素，不能在被删除元素上调用该方法，而应在其父元素上调用。这样就可以创建实用函数库，因为这不需要知道元素的包含层次结构信息。下面的简单函数可删除一个元素，实际上是从它的父节点上删除：

```
function removeElement(elemID)
{
   var elem = document.getElementById(elemID);
   elem.parentNode.removeChild(elem);
}
```

在 Internet Explorer 4 或更新版本中，可将对象的 outerHTML 属性设置为空，而要在其他浏览器中实现相同的结果，还有很长的路要走。尽管这种迂回方式对 XML 是有部分意义的，但 W3C 工作组似乎没有考虑 HTML 脚本开发人员最感兴趣的内容。不过如本章后面所述，还有希望。

5. 生成新节点内容

W3C DOM 节点结构的最后一个要点是，脚本开发人员采用类似的方式，创建要在页面上添加或替换的内容。如果仅改变文本(例如表单元格内的文本)，有简单的方法，也有复杂的方

更多资源请访问稀酷客(www.ckook.com)

法。而要改变 HTML，就只有复杂的方法(后面会讨论一个便捷的方法)。首先看看修改文本的复杂方法，然后讨论简单的方法。

为在 DOM 中生成新节点，可以考虑使用为核心 DOM 对象 document 定义的各种方法(由 HTML document 对象继承)。DOM 中的每类节点都定义了创建节点的方法，用于 HTML 文档的两个重要方法是 createElement()和 createTextNode()。createElement()方法生成一个元素，其参数是元素的标记名(字符串)；createTextNode()方法用所传递的文本参数生成一个文本节点。

第一次创建新元素时，该元素仅存在于浏览器内存，不是文档包含层次结构的一部分；此外，createElement()方法的结果是空元素的引用(只包含标记名)。例如，要创建一个新 p 元素，可使用如下方法：

```
var newElem = document.createElement("p");
```

新元素没有 ID、特性或其他内容。要把一些属性赋给这个元素，可使用 setAttribute ()方法(每个元素对象都有这个方法)，或给对象的相应属性赋值。例如，要把标识符赋给新元素，可使用如下方法：

```
newElem.setAttribute("id", "newP");
```

或者：

```
newElem.id = "newP";
```

这两种方式都是完全有效的。即使元素有 ID，它也不是文档的一部分，所以不能用 document.getElementById()方法得到它。

要在段落中添加一些内容，可将文本节点创建为独立的对象：

```
var newText = document.createTextNode("This is the second paragraph.");
```

另外，该节点不会一直保存在内存中，等待作为其他节点的子节点。要使它成为新段落的内容，可添加这个节点，作为仍在内存中的段元素的子节点：

```
newElem.appendChild(newText);
```

代表新段元素的 HTML 如下：

```
<p id="newP">This is the second paragraph.</p>
```

此时，可以把新的段元素插入文档了。使用程序清单 25-1 中的文档，可以把它添加为 body 元素的子节点：

```
document.body.appendChild(newElem);
```

最后，新元素是文档包含层次结构的一部分，现在可以引用它了，就像引用文档中的其他元素一样。

6. 替换节点内容

上一节添加的段落需要改变原始段落的一部分文本(因为第一段不再是页面上的唯一一

更多资源请访问稀酷客(www.ckook.com)

段)。如前所述，要改变文本，可使用 replaceChild()方法，或把新文本赋给文本节点的 nodeValue 属性。下面看看这两种方式如何把第一段 em 元素的文本从 one and only 改为 first。

要使用 replaceChild()，脚本必须先用新文本生成一个有效的文本节点：

```
var newText = document.createTextNode("first");
```

接着使用 replaceChild()方法，但这个方法要在被替换节点的父节点上调用。这里的子节点是 em 元素内的文本节点，因此必须在 em 元素上调用 replaceChild()方法。replaceChild()方法需要两个参数，第一个是新节点，第二个是要替换的节点引用。因为脚本语句可很好地使用 getElementByld()方法，所以需要一个中间步骤来得到 em 元素内文本节点的引用：

```
var oldChild = document.getElementById("emphasis1").childNodes[0];
```

现在脚本就可以在 em 元素上调用 replaceChild()方法，用新的文本节点替换旧节点：

```
document.getElementById("emphasis1").replaceChild(newText, oldChild);
```

注意 replaceChild()方法返回被替换节点的引用(目前它仅在内存中，不是文档节点层次结构的一部分)。假如想在旧节点消失前捕获它，只需把该方法的结果赋给一个变量，就可以在其他需要的地方使用旧节点。

似乎还有很多工作要做，所创建的 HTML 十分复杂时尤其如此。幸好，替换文本节点还可以使用一个更简单的方式，它只需被替换的文本节点的引用，给该节点的 nodeValue 属性赋予一个新字符串值：

```
document.getElementById("emphasis1").childNodes[0].nodeValue = "first";
```

当元素的内容只包含文本时(例如，已有一个文本节点的表单元格)，这是使用 W3C DOM 语法交换文本的最直接方式。但这不适用于本章前面第二段文本的创建，因为根本不存在文本节点，必须使用 createTextNode()方法明确地创建文本节点。

另外，文本节点没有任何内在的样式，HTML 容器元素的样式控制着文本的样式。假如想改变文本节点的文本及其外观，就必须修改文本节点的父元素的 style 属性。浏览器执行这些内容替换和样式改变后，将自动回显这个页面，以适应内容大小的变化。

程序清单 25-2 修改了程序清单 25-1 中的原始文档，新版本包括一个按钮和一个脚本，该脚本包含了前面有关节点的所有修改。重载该页面以重新开始。

程序清单 25-2　加入/替换 DOM 内容

```
<html>
<head>
   <title>A Simple Page</title>
   <script type="text/javascript">
   function modify()
   {
      var newElem = document.createElement("p");
      newElem.id = "newP";
      var newText = document.createTextNode("This is the second paragraph.");
      newElem.appendChild(newText);
```


更多资源请访问稀酷客(www.ckook.com)

```
        document.body.appendChild(newElem);
        document.getElementById("emphasis1").childNodes[0].nodeValue = "first";
    }
    </script>
</head>

<body>
    <button onclick="modify()">Add/Replace Text</button>

    <p id="paragraph1">This is the <em id="emphasis1">one and
    only</em> paragraph on the page.</p>
</body>
</html>
```

交叉引用：

第 26 章详细介绍了所有 HTML 元素都继承了的节点属性和方法，它们大多都在所有现代 W3C DOM 浏览器中实现了。参阅第 29 章中 document 对象的内容，可获得其他有价值的 W3C DOM 方法的信息。

7. 事实标准：innerHTML

从 Internet Explorer 4 开始，Microsoft 率先实现了所有元素对象的 innerHTML 属性。尽管 W3C DOM 还不支持此属性，但脚本开发人员发现，与创建和组合元素及文本节点相比，通过包含 HTML 标记的字符串来动态修改内容常常更加方便。因此，大多数现代 W3C DOM 浏览器，包括 Moz1 和 Safari 1，都支持读/写所有元素对象的 innerHTML 属性，并作为一个事实标准。

将包含 HTML 标记的字符串分配给现有元素的 innerHTML 属性时，浏览器会自动将新呈现的元素插入到文档节点树中。还可使用 innerHTML 和未标记文本，来实现与 Internet Explorer 的 innerText 属性相同的功能。

尽管与按部就班地操作元素和文本节点对象的过程相比，innerHTML 属性显然更方便，但浏览器对节点的操作比组合长字符串要高效得多。这是不一定要转换少量 JavaScript 代码，以提高效率的一个例子。

25.12.6　W3C DOM 的静态 HTML 对象

对于对象模型，Moz1+ DOM(可惜不是 Internet Explorer 5 或更新版本)支持“原型继承”这个核心 JavaScript 概念。当页面载入 Moz1+浏览器时，浏览器根据 W3C DOM 定义的每个对象原型，来创建 HTML 对象。比如，使用 The Evaluator Sr.(第 4 章)来确定 mrP 段落对象的类型(在顶部文本框中输入 document.getElementById("mrP")，并单击 Evaluate 按钮)，The Evaluator Sr. 就会显示，该对象基于 DOM 的 HTMLParagraphElement 对象。页面上 p 元素对象的每个实例都会从 HTMLParagraphElement 中继承它的默认属性和方法(HTMLParagraphElement 又从 HTMLElement、Element 和 Node 对象中继承，详见 W3C DOM 规范的 JavaScript 附录)。

可以使用脚本在一些静态对象的原型中添加属性。为此，必须使用 Moz1+的新特性，它们是 JavaScript 1.5 和 ECMAScript 5 的一部分。两个新方法__defineGetter__()和__defineSetter__()允许把函数赋给对象的自定义属性。

更多资源请访问稀酷客(www.ckook.com)

注意：
这些方法是 Mozilla 专用的。为了防止它们与这些特性在 ECMAScript 中的标准化实现发生冲突，方法名两侧的下划线字符是成对出现的。

只要读取(通过__defineGetter__()方法指定的函数)或修改(通过__defineSetter__()方法指定的函数)属性，就执行这些函数。这些函数常以匿名函数的形式定义(参见第 23 章)，下面两条语句把这些行为赋给一个对象原型：

```
object.prototype.__defineGetter__("propName",
      function([param1[,...[,paramN]]]) {
  // statements
  return returnValue;
})
object.prototype.__defineSetter__("propName",
      function([param1[,...[,paramN]]]) {
  // statements
  return returnValue;
})
```

程序清单 25-3 说明了如何把只读属性添加到当前文档的每个 HTML 元素对象中。childNodeDetail 属性返回一个对象，该对象有两个属性：一个是元素子节点的数目，另一个是文本子节点的数目。注意函数定义中的 this 关键字是对象(其属性被执行)的引用。因为每次脚本语句读取属性，都会运行函数，所以页面载入后，脚本对内容的任何改变都反映在返回的属性值中。

程序清单 25-3　为所有 HTML 元素对象添加一个只读属性

```
<script type="text/javascript">
if (HTMLElement)
{
   HTMLElement.prototype.__defineGetter__("childNodeDetail", function()
   {
      var result = {elementNodes:0, textNodes:0 }
      for (var i = 0; i < this.childNodes.length; i++)
      {
         switch (this.childNodes[i].nodeType)
         {
            case 1:
               result.elementNodes++;
               break;
            case 3:
               result.textNodes++;
               break;
         }
      }
      return result;
   })
}
</script>
```


更多资源请访问稀酷客(www.ckook.com)

要访问这个属性，只需像对象的其他属性一样使用它。例如：

```
var BodyNodeDetail = document.body.childNodeDetail;
```

本例的返回值是一个对象，因此可使用常规的 JavaScript 语法来访问它的一个属性值：

```
var BodyElemNodesCount = document.body.childNodeDetail.elementNodes;
```

25.12.7 双向事件模型

尽管 NN4 的向下扩散事件模型和 Internet Explorer 4 的向上冒泡事件模型表面上存在冲突，但 Level 2 中定义的 W3C DOM 事件模型仍能同时使用这两个事件传播模型。因此脚本开发人员可以在事件的传播路径上选择在何处处理事件。为了防止和现有事件模型术语相互冲突，W3C 模型为事件的属性和方法使用了新术语。代码可能需要在页面上使用 W3C DOM 中针对多对象模型的特殊处理。

W3C 事件模型也引入了一个新概念：事件监听器。事件监听器其实是一个机制，它指导对象响应特定类型的事件，这非常像 HTML 标记的事件处理属性对事件的响应方式。但 DOM 建议，最好使用更面向脚本的方式指定事件监听器：使用文档层次结构中每个节点都可用的 addEventListener()方法。通过这个方法，浏览器可迫使事件沿着层次结构向上冒泡(假如使用 HTML 特性类型的事件处理程序，则默认行为仍起作用)，或者在更高的层次上捕获事件。

事件监听器调用的函数会接收一个 event 对象参数，该对象的属性包含事件的相关细节(如鼠标单击的位置、键盘键的字符代码或目标对象的引用)。例如，假如表单包括一个按钮，单击该按钮会调用一个计算函数，W3C DOM 就会采用下列方法指定事件处理程序：

```
document.getElementById("calcButton").addEventListener("click",
doCalc, false);
```

addEventListener()方法有三个参数：第一个是要监听的事件的字符串；第二个是触发事件时调用的函数引用；第三个是布尔值。将该布尔值设为 true 时，只要该事件指向目标，就启动事件捕获功能，然后该函数将 event 对象作为参数来完成它的任务：

```
function doCalc(evt)
{
   // get shortcut reference to input button's form
   var form = evt.target.form;
   var results = 0;
   // other statements to do the calculation //
   form.result.value = results;
}
```

为了修改事件监听器，可使用 removeEventListener()方法来去除旧的事件监听器，然后在 addEventListener()中用另一个参数来指定新的事件监听器。

防止事件执行它的默认操作也是 W3C 事件模型与 IE 不同的一个方面。在 Internet Explorer 4(以及 NN3 和 4)中，可让事件处理程序执行 return false，来取消默认动作。这种方式仍在 Internet Explorer 5 及更新版本中起作用，而 Microsoft 还包括了 window.event 对象的另一个属性 returnValue。在事件处理器调用的函数中的任何地方，将此属性设置为 false，就可取消事件的常规动作。但

更多资源请访问稀酷客(www.ckook.com)

W3C 事件模型使用一个事件对象方法 preventDefault()，来防止事件执行它的常规任务。可在触发事件后执行的函数中的任何地方，调用该方法。

事件的详细信息包含在事件对象中，该对象必须传递给事件处理函数，该函数可以读取事件的详细信息。如果通过 W3C DOM 的 addEventListener()方法或事件处理属性来指定事件处理程序，事件对象就作为唯一的参数自动传递给事件处理函数。该函数包括一个参数变量来捕获传入的参数：

```
function swap(evt)
{
   // statements here to work with W3C DOM event object
}
```

但如果通过标记特性来指定事件，就必须明确地在函数调用中传递事件对象：

```
<a href="http://www.example.com" onmouseover="swap(event)">
```

遗憾的是，Microsoft 直到 Internet Explorer 8 for Windows 和 Internet Explorer 5 for Macintosh 为止，还是不支持 W3C DOM 事件模型。但如果通过对象属性或标记特性来指定事件处理程序，并插入一小段对象检测代码(参见本章后面的内容和第 32 章)，Internet Explorer 和 W3C 事件模型就可以共同工作。

25.13　脚本编程的发展趋势

尽管浏览器脚本编程开始于给表单控件加入某种智能，但 JavaScript 语言和 DOM 的功能鼓励许多 Web 开发人员创建实质的应用程序。实现基于 Web 的电子邮件系统时，常常大量使用脚本编程，与服务器进行后台通信，以使页面快速更新，在用户每次从收件箱列表中删除消息时，不必获取并重新显示整个页面。大项目常常由多个脚本开发人员(与 CSS 专业人员、服务器编程人员、艺术家和作家一起)共同完成。融合所有代码可能会很繁琐。

25.13.1　将内容与脚本分离

如果用 CSS 控制站点的样式，则在更改整个站点的颜色或字体规格时，将样式定义与 HTML 标记分离会显著提高效率。不需要修改散布在站点中的数百个<font>标记，而只需调整某个.css 文件中的一个 CSS 规则，该修改操作就会立即应用于整个站点。

仅将 HTML 用于页面结构这一思想也影响到了脚本编程。目前专业脚本开发人员极少会在 HTML 标记中放置事件处理属性，这会混淆内容与行为。换句话说，HTML 标记应能独立存在，这样，若用户使用不支持脚本编程的浏览器(包括使用专用浏览器的有视觉或行动障碍的用户)，仍能获得页面提供的基本信息。影响页面的显示或行为的任何脚本都在载入和显示 HTML 标记后才添加到页面上。甚至向元素分配事件，也是在页面载入后由脚本完成。

脚本代码常从外部的.js 文件中链接到页面。这不仅是内容与脚本分离趋势的一部分，还提供了很多好处，如同一代码可以立即在多个页面上使用。另外，当项目涉及许多代码编写人员时，可以各司其职：作者编写 HTML，设计人员处理外部 CSS 代码，脚本开发人员编写自己的代码。

更多资源请访问稀酷客(www.ckook.com)

注意：

本书的许多例子都在标记中使用事件处理属性，在页面中嵌入脚本。采用这种方式主要是为了简化语言或 DOM 功能的演示。

25.13.2 尽量使用 W3C DOM

很长时间以来，主流浏览器都提供了对 W3C DOM 元素引用和内容操作的基本支持，可以确保为该模型编写的脚本对大多数访问者有效。这并不是说可以假定每个访问者都获得了这种支持，但是脚本开发人员过去为支持冲突的对象模型而遇到的种种麻烦大都不复存在。为 IE 和 Netscape 编写大量的分支代码并不是美好的回忆。

不过，仍需要使用对象检测技术，以提防旧浏览器偶尔出现的麻烦。此时，通过脚本指定事件处理程序就可以解决问题。

除了在页面载入时执行的一些初始化操作外，Web 页面中的大多数脚本都在触发事件时执行：用户单击按钮、在文本框中输入内容、从 select 元素中进行选择等。在事件指定的代码中进行基本的对象检测，就可以防止旧浏览器在 W3C DOM 语法上出错，如以下简化的示例所示：

```
function setUpEvents()
{
   if (document.getElementById)
   {
      // statements to bind events to elements
   }
}
window.onload = setUpEvents;
```

现在，当用户单击或输入时，即使浏览器没有提供最低的 W3C DOM 支持，也不会产生脚本错误，因为没有为这些浏览器指定那些事件。通过了对象检测的脚本也可以修改页面，如第 4 章的程序清单 4-2 所示。

25.13.3 处理事件

W3C DOM 在一些情况下仍不能满足需要。在处理事件时尤其如此，Internet Explorer(至少是到版本 8)不支持将事件信息传递到事件处理函数的 W3C DOM 方式。W3C DOM 自动将 event 对象作为参数传递给处理函数。而在 Internet Explorer 模型中，event 对象是 window 对象的一个属性。因此，函数必须在必要时考虑这一差别。例如，无论使用什么浏览器，要获得表示 event 对象的单个变量，可使用以下结构：

```
function calculate(evt)
{
   evt = (evt) ? evt : window.event;
   // more statements to process event
}
```


更多资源请访问稀酷客(www.ckook.com)

为了检查事件的重要信息，需要使用额外的分支。例如，要指向接收事件的对象，Internet Explorer 要使用事件对象属性 srcElement，而 W3C DOM 的版本称为 target。同样，事件处理函数中的一小段平衡代码可以处理这种差异。当脚本有对接收事件的元素的引用时，就可以使用元素的 W3C DOM 属性和方法，因为 Internet Explorer 支持它们。事件对象详见第 32 章。

25.14　标准兼容模式(DOCTYPE 切换)

Microsoft 和 Netscape/Mozilla 都发现，随着时间的推移，他们实现 CSS 功能的方式与后来公布的标准(通常是在工作组成员间的大量争论之后)完全不同。为弥补这些差异，并兼顾标准兼容方式，主流浏览器厂商决定让页面作者选择<!DOCTYPE>标题元素的细节，来确定文档是遵循旧的方式(有时也称为怪癖模式)，还是采用与标准兼容的方式。这一策略的非正式名称是 DOCTYPE 切换，在 Internet Explorer 6 及以后版本、Internet Explorer 5 for the Mac 和所有基于 Mozilla 的浏览器、所有基于 WebKit 的浏览器中实现。

尽管这两种模式之间的大多数差别都很细微，但在 Internet Explorer 6 及以后版本中，两种模式有一些显著的区别，尤其是当样式或 DHTML 脚本依赖用边框、边距和补白来设计的元素时。Microsoft 的原始框模型测量尺寸元素采用与最终的 CSS 标准不同的方式。

要将受影响的浏览器置于 CSS 标准兼容模式下，应在每个文档的顶部包括一个<!DOCTYPE>元素，来指定以下详细信息(第一个<!DOCTYPE>语句用于 HTML5)：

```
<!DOCTYPE html>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
        "http://www.w3.org/TR/REC-html40/loose.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/REC-html40/frameset.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/REC-html40/strict.dtd">

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

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

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

不过要注意，Internet Explorer for Windows 的旧版本，如 Internet Explorer 5 或 Internet Explorer 5.5，不支持标准兼容模式，无论<!DOCTYPE>设置是什么，都使用较旧的 Microsoft 专用模式(怪癖模式)。但使用标准兼容模式，DOCTYPE 更可能使内容和样式表在最新的浏览器上具有相似的显示效果。

更多资源请访问稀酷客(www.ckook.com)

25.15　小结

前面两章概述了使用 JavaScript 设计页面的人必须面对的核心语言和对象模型问题。其目标是鼓励读者独立思考如何提高或降低页面的兼容性，以便在创建漂亮页面和为用户服务之间平衡。从现在开始，这个艰难的选择就取决于用户了。

为了帮助选择最适合的对象、属性、方法和事件处理程序，第III部分的所有章节和第IV部分余下的章节都深入讨论了 DOM 和 JavaScript 核心语言功能。仔细观察每个语言术语的兼容性等级，有助于确定哪些功能最适合用户的浏览器。多数的程序清单都是完整的 HTML 页面，可将它们载入不同的浏览器，查看它们是如何工作的。许多其他的例子还需要通过 The Evaluator Sr.(第 4 章)来了解其工作原理。先执行这些文件，然后修改它们，来构建自己的应用程序，或在浏览器环境中扩展 JavaScript 的知识。

语言和对象模型自从诞生以来，就在逐步发展，语言词汇数目也迅猛增长。要完全吸收它们，并得心应手地应用它们，需要耗费许多时间。不要记忆这些词汇，而应熟悉这些功能，以后需要其实现细节时再回头查看。

耐心，坚持，必有回报。

更多资源请访问稀酷客(www.ckook.com)

更多资源请访问稀酷客(www.ckook.com)

第26章

通用 HTML 元素对象

> **本章包含哪些内容？**
> 使用 HTML 元素对象
> 常用属性和方法
> 所有元素对象的事件处理程序

Internet Explorer 4+和 W3C(基于 Mozilla 和基于 WebKit 的)浏览器中实现的对象模型规范定义了许多脚本对象，它们表示我们常说的“通用”HTML 元素。通用元素可分为两组：第一组元素(例如 b 和 strike 元素)定义了在封装的文本序列中使用的字体样式。随着越来越多的页面设计者使用样式表，已经不再需要这些元素和表示它们的对象。第二组元素为它们首尾标记内的内容指定上下文，上下文元素的例子包括 hl、blockquote 和常见的 P 元素。虽然在默认情况下，浏览器有时使用固有的可视化方法来显示上下文元素(例如，使用<hl>标记显示大粗体字)，但标记的主要作用不是进行这种特定的显示。正式标准没有要求，em 元素内的文本必须斜体，只是自从浏览器的初期开始，人们就习惯使用这种样式。

所有这些通用元素都共享许多脚本属性、方法和事件处理程序，这种共享不仅延伸到通用元素中，而且延伸到每个可显示的元素中(这些元素还拥有本部分其他章节中深入讨论的其他元素专用属性、方法和事件处理程序)。为了不重复介绍每个对象共享的属性、方法和事件处理程序的详情，它们只在本章详细描述(除非在其他地方描述的某个对象具有和该项相关的特殊行为、错误或技巧)。在接下来的各章节中，给每个对象都列出了对象属性、方法和事件处理程序，但没有重复列出共享的项(以免无法确定专用于特定元素的项)。对于常规 HTML 元素对象的通用内容，仅指出了包含这些内容的章节。

通用对象

表 26-1 列出了本书中的所有通用对象。这些对象都共享后面描述的属性、方法和事件处理程序，并且没有特殊的内容需要在本书其他地方详述。

PDG

更多资源请访问稀酷客(www.ckook.com)

表 26-1　通用 HTML 元素对象

格式对象	上下文对象	格式对象	上下文对象
b	acronym	sub	listing
big	address	sup	p
center	cite	tt	plaintext
i	code	u	pre
nobr	dfn	wbr	samp
rt	del		span
ruby	div		strong
s	em		var
small	ins		xmp
strike	kbd		

属　　性	方　　法	事件处理程序
accesskey	addBehavior()	onactivate
all[]	addEventListener()	onafterupdate
attributes[]	appendChild()	onbeforecopy
baseURI	applyElement()	onbeforecut
behaviorUrns[]	attachEvent()	onbeforedeactivate
canHaveChildren	blur()	onbeforeeditfocus
canHaveHTML	clearAttributes()	onbeforepaste
childNodes[]	click()	onbeforeupdate
children	cloneNode()	onblur
cite	compareDocumentPosition()	oncellchange
className	componentFromPoint()	onclick
clientHeight	contains()	oncontextmenu
clientLeft	createControlRange()	oncontrolselect
clientTop	detachEvent()	oncopy
clientWidth	dispatchEvent()	oncut
contentEditable	doScroll()	ondataavailable
currentStyle	dragDrop()	ondatasetchanged
dateTime	fireEvent()	ondatasetcomplete
dataFld	focus()	ondblclick
dataFormatAs	getAdjacentText()	ondeactivate

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	方　法	事件处理程序
dir	getAttributeNode()	ondragend
disabled	getAttributeNodeNS()	ondragenter
document	getAttributeNS()	ondragleave
filters[]	getBoundingClientRect()	ondragover
firstChild	getClientRects()	ondragstart
height	getElementsByTagName()	ondrop
hideFocus	getElementsByTagNameNS()	onerrorupdate
id	getExpression()	onfilterchange
innerHTML	getFeature()	onfocus
innerText	getUserData()	onfocusin
isContentEditable	hasAttribute()	onfocusout
isDisabled	hasAttributeNS()	onhelp
isMultiLine	hasAttributes()	onkeydown
isTextEdit	hasChildNodes()	onkeypress
lang	insertAdjacentElement()	onkeyup
language	insertAdjacentHTML()	onlayoutcomplete
lastChild	insertAdjacentText()	onlosecapture
length	insertBefore()	onmousedown
localName	isDefaultNamespace()	onmouseenter
namespaceURI	isEqualNode()	onmouseleave
nextSibling	isSameNode()	onmousemove
nodeName	isSupported()	onmouseout
nodeType	item()	onmouseover
nodeValue	lookupNamespaceURI()	onmouseup
offsetHeight	lookupPrefix()	onmousewheel
offsetLeft	mergeAttributes()	onmove
offsetParent	normalize()	onmoveend
offsetTop	releaseCapture()	onmovestart
offsetWidth	removeAttribute()	onpaste
outerHTML	removeAttributeNode()	onpropertychange
outerText	removeAttributeNS()	onreadystatechange
ownerDocument	removeBehavior()	onresize
parentElement	removeChild()	onresizeend

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	方　法	事件处理程序
parentNode	removeEventListener()	onresizestart
parentTextEdit	removeExpression()	onrowenter
prefix	removeNode()	onrowexit
previousSibling	replaceAdjacentText()	onrowsdelete
readyState	replaceChild()	onrowsinserted
recordNumber	replaceNode()	onscroll
runtimeStyle	scrollIntoView()	onselectstart
scopeName	setActive()	
scrollHeight	setAttribute()	
scrollLeft	setAttributeNode()	
scrollTop	setAttributeNodeNS()	
scrollWidth	setAttributeNS()	
sourceIndex	setCapture()	
style	setExpression()	
tabIndex	setUserData()	
tagName	swapNode()	
tagUrn	tags()	
textContent	toString()	
title	urns()	
uniqueID		
unselectable		
width		

1. 语法

要访问元素的属性或方法，可使用以下语句：

```
(IE4+)       [document.all.]objectID.property | method([parameters])
(IE5+/W3C) document.getElementById(objectID).property | method ([parameters])
```

注意：
除非需要支持 IE4(现在几乎不太可能)，否则应只使用后一种方式，通过 getElementById()方法来引用元素的属性和方法，这很重要。

2. 关于这些对象

表 26-1 中的所有对象都是 HTML 元素的 DOM 表示，这些元素会影响字体样式或 HTML 内容。与这些对象相关的大量属性、方法和事件处理程序也应用到表示 HTML 元素的其他每个

更多资源请访问稀酷客(www.ckook.com)

DOM 对象中，本章有关对象细节的讨论也适用于第 IV 部分后续章节介绍的其他许多对象。

3. 属性

accessKey

值：单字符字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN7+，Moz-，Safari+，Opera+，Chrome+

可为许多元素指定一个键盘字符(字母、数值或标点符号)，当输入 Alt+键组合(在 Win32 OS 平台上)或 Ctrl+键组合(在 Mac OS 上)或 Shift+Esc+键组合(在 Opera 上)时，该元素就会获得焦点。具有焦点的元素设置为响应键盘的动作，如果新获得焦点的元素在文档当前位置的视图之外，文档会自动滚动，把该元素显示在可视范围内(另请参见 scrollIntoView()方法)。指定的字符可以是大写或小写，但这些值是不区分大小写的。如果把同一个字母分配给多个元素，使用 accessKey 就可遍历与其相关的所有元素。

在某些情况下，Internet Explorer 给 accessKey 属性提供了一些附加功能。例如，如果把 accessKey 值赋给 label 元素对象，焦点就传递给与该标记相关的表单元素。当元素(如按钮)有焦点时，按下空格键就和用鼠标单击该元素一样。

在为 accessKey 值选择字符时应小心。如果把一个通常用于访问浏览器内置菜单(例如在 IE 中，Alt+F 用于访问 File 菜单)的字符赋给 accessKey，该 accessKey 设置就会覆盖浏览器的常规行为。对依赖键盘访问菜单的用户而言，最好不要设置这样的组合键。在其他浏览器中，accessKey 设置不会覆盖浏览器的常规行为。

示例

通过程序清单 26-1 的示例可以了解到如何使用 accessKey 属性来操作键盘，以便导航网页。载入程序清单 26-1 中的脚本后，调整浏览器窗口的高度，以便看不到第二条水平分隔线下的内容。在页面的 Settings 部分输入任意字符，并按回车键(按回车键可能使计算机发出蜂鸣声)。按下回车键不会使浏览器显示什么内容，只是把输入的字符赋予对应元素的文本框。然后在按住 Alt (Windows)或 Ctrl (Mac)键(在 Opera 上是 Shift+Esc 键)的同时按下输入的这个键。第二条水平分隔线下的元素将显示出来。

注意：

本章及本书许多章节的代码所使用的事件处理属性分配技术是有意简化了的，以使代码更容易阅读。通常最好采用更现代的方式，即用 addEventListener()(NN6+/Moz/W3C)或 attachEvent()(IE5+)方法来绑定事件。第 32 章将详细介绍现代的跨浏览器事件处理技术。

程序清单 26-1 控制 accessKey 属性

HTML: jsb26-01.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>accessKey Property</title>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <script type="text/javascript" src="../jsb-global.js"></script>
    <script type="text/javascript" src="jsb26-01.js"></script>
</head>
<body>
   <h1>accessKey Property Lab</h1>
   <hr />
   Settings:<br />
   <form name="input">
     Assign an accessKey value to the Button below and press Return:
     <input type="text" size="2" maxlength="1"
            onkeypress="assignKey('button', this)" />
     <br />
     Assign an accessKey value to the Text Box below and press Return:
     <input type="text" size="2" maxlength="1"
            onkeypress="return assignKey('text', this)" />
     <br />
     Assign an accessKey value to the Table below (IE5.5+ only)
     and press Return:
     <input type="text" size="2" maxlength="1"
            onkeypress="return assignKey('table', this)" />
   </form>
   <br />
   Then press Alt (Windows) or Control (Mac) or ShiftESC (Opera) + the key
   <br />
   <em>Size the browser window to view nothing lower than this line.</em>
   <hr />
   <form name="output" onsubmit="return false">
     <input type="button" id="access1" value="Standard Button" />
     <input type="text" id="access2" />
   </form>
   <table id="myTable" cellpadding="10" border="2">
     <tr>
        <th>Quantity</th>
        <th>Description</th>
        <th>Price</th>
     </tr>
     <tbody bgcolor="red">
        <tr>
           <td width="100">4</td>
           <td>Primary Widget</td>
           <td>$14.96</td>
        </tr>
        <tr>
           <td>10</td>
           <td>Secondary Widget</td>
           <td>$114.96</td>
        </tr>
     </tbody>
   </table>
</body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

JavaScript: jsb26-01.js

```
function assignKey(type, elem)
{
   if (window.event)          // IE, Opera, Safari, Chrome
   {
      if (window.event.keyCode == 13)
      {
         switch (type)
         {
            case "button":
               document.getElementById("access1").accessKey = elem.value;
               break;
            case "text":
               document.getElementById("access2").accessKey = elem.value;
               break;
            case "table":
               document.getElementById("myTable").accessKey = elem.value;
         }
         return false;
      }
   }
}
```

相关主题：scrollIntoView()方法。

all[ ]

值：嵌套元素对象的数组，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+ (部分)，Opera+，Chrome+ (部分)

除 IE 之外，all 属性是当前对象作用域内 HTML 元素和(在 IE5+中)XML 标记的集合(数组)。现在 Opera 完全支持该属性，而基于 WebKit 的浏览器(例如 Safari 和 Chome)仅把它支持为文档对象的一个属性。尽管如此，仍应考虑使用 document.getElementById()方法(参见第 29 章)，它是引用元素的正式 W3C 方式和跨浏览器方式。IE5+支持 document.getElementById()方法。

数组中的项以源代码顺序出现，数组并不知道每个项包含什么元素。对于 HTML 元素容器而言，源代码顺序由元素首标记的位置决定，不考虑尾标记。但对于 XML 标记，尾标记在数组中是单独的一项。

即使实际的 HTML 源代码忽略 html、head、title 和 body 元素对象，每个 document.all 集也包含这些标记。对象模型会为每个加载到窗口或框架中的文档创建这些对象。document.all 引用可能最常用，而 all 属性可用于任何容器元素，例如，document.forms[0].all 表示在页面第一个表单中定义的所有元素。

只要给任何元素的 id 特性指定一个标识符，就可通过标识符的字符串形式(也可用整数索引值)访问该元素。不必再使用性能低下的 eval()函数将字符串转换为对象引用，而可以使用名字的字符串值作为一个数组索引值：

更多资源请访问稀酷客(www.ckook.com)

```
var paragraph = document.all["myP"];
```

Internet Explorer 允许对单个集合索引值使用方括号或圆括号。因此，下列两条语句的结果相同：

```
var paragraph = document.all["myP"];
var paragraph = document.all("myP");
```

在 all 集合内很少出现两个或多个元素有相同 ID 的情况，使用字符串索引值的语法会返回同名元素的集合。但可在圆括号内使用另一个变量来表示最初元素集合的整数索引，从而挑选出指定元素的特定实例：

```
var secondRadio = document.all("group0",1);
```

更易读的替代方法是使用 item()方法(在本章后面讨论)来访问集内类型相同的项：

```
var secondRadio = document.all.item("group0",1);
```

也可以使用 tags()方法(见本章后面的内容)从匹配特定标记名字的 all 集中提取一组元素。

示例

使用 The Evaluator(参见第 4 章)来试验 all 集合。在下面的文本框中一次输入一条以下语句，在每个文本区域中查看结果：

```
document.all
myTable.all
myP.all
```

如果在集合中遇到了编号元素，可以检查该元素，看看哪个标记与其相关联。例如，若 document.all 集合的某个结果为 document.all.8=[object]，则在最上面的文本框中输入以下语句：

```
document.all[8].tagName
```

相关主题：item()、tags()、document.getElementById()方法。

attributes[]

值：attribute 对象引用数组， 只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

attributes 属性包括一个为元素指定的特性数组。在 IE5+中，attributes 数组包含浏览器为其元素定义的每个可能的属性——甚至没有在 HTML 标记中明确设置的特性。以后通过脚本工具(setAttribute()方法)添加的任何特性，都不反映在 attributes 数组中。换句话说，IE5+的 attributes 数组是固定的，除了在 HTML 标记中明确设置的特性之外，所有其他属性都使用默认值。

Mozilla 浏览器的 attributes 属性返回一个数组，该数组是一个命名的节点图(W3C DOM 术语)，这个节点图对象有自己的属性和读写特性值的方法。例如，通过 W3C DOM 语法，在从 attributes 属性返回的数组上使用 getNamedItem(attrName)和 item(index)方法，就可以访问各个 attribute 对象。

对于特性对象，IE5 和 Mozilla 的观点不同。Mozilla 依赖于 W3C DOM 节点继承模型，参

更多资源请访问稀酷客(www.ckook.com)

见第 25 章。过去，只有 Mozilla 支持 W3C DOM 节点中的许多属性，现在，IE 也支持 W3C DOM 的许多属性。表 26-2 显示了这两个对象模型定义的 attribute 对象的属性，基于 WebKit 的浏览器也支持 Mozilla 支持的属性，而 Opera 介于两者之间。

表 26-2　attribute 对象属性

属　性	IE5	Moz	说　明
attributes	支持	支持	嵌套特性对象的数组(空)
childNodes	不支持	支持	子节点数组
firstChild	不支持	支持	第一个子节点
lastChild	不支持	支持	最后一个子节点
localName	不支持	支持	在当前命名空间中的名称
name	支持	支持	特性名称
nameSpaceURI	支持	支持	XML 命名空间 URI
nextSibling	支持	支持	下一个兄弟节点
nodeName	支持	支持	特性名称
nodeType	支持	支持	节点类型(2)
nodeValue	支持	支持	赋给特性的值
ownerDocument	支持	支持	文档对象引用
ownerElement	支持	支持	元素节点引用
parentNode	支持	支持	父节点引用
prefix	支持	支持	XML 命名空间前缀
previousSibling	支持	支持	上一个兄弟节点
specified	支持	支持	是否明确指定特性(布尔值)
value	支持	支持	赋给特性的值

attribute 对象中最有用的属性是布尔型 specified 属性。在 IE 中，它指定特性是否在元素的标记中明确设置。因为 Mozilla 只返回 attributes 数组中明确指定的特性，所以 Mozilla 中的值总是 true。然而，多数情况下，都可使用元素对象的 getAttribute()和 setAttribute()方法来读写特性值。

示例

用 The Evaluator(参见第 4 章)检查该文档中某些元素的 attributes 数组值。在底部文本框中输入以下表达式，在 Results 文本区域中查看各表达式的数组内容：

```
document.body.attributes
document.getElementById("myP").attributes
document.getElementById("myTable").attributes
```

如果同时使用 IE5+和符合 W3C DOM 标准的浏览器，就可比较每个表达式的结果。要在 WinIE5+中访问 attributes 数组，来查看单个特性的值，可在顶部文本框中输入以下语句：

更多资源请访问稀酷客(www.ckook.com)

```
document.getElementById("myForm2").attributes["name"].value
```

对于 W3C 浏览器、IE6+和 MacIE5，应使用 W3C DOM 语法：

```
document.getElementById("myForm2").attributes.getNamedItem("name").value
```

相关主题：getAttribute()方法，mergeAttributes()方法，removeAttribute()方法，setAttribute()方法。

baseURI

值：完整的 URI 字符串，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

此属性给出了为元素提供服务的源的完整路径，可用于导入 XML 数据的应用程序。在这种情况下，XML 元素的源可能不同于处理该元素的 HTML 页面。

behaviorUrns[]

值：操作 URN 字符串的数组，　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

Internet Explorer 的 behaviorUrns 属性为赋给当前对象的所有操作提供一个 URN (Uniform Resource Name，统一资源名)形式的地址列表。如果没有操作，数组的长度就为 0。然而实际上，IE5+总是返回空字符串的数组。让脚本提供 URN 可能威胁到个人隐私。

相关主题：urns ()方法。

canHaveChildren

值：Boolean，　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

canHaveChildren 属性表示某个元素是否可包含子(嵌套)元素，这在一些动态内容中很有用。多数具有首尾标记的元素(特别是本章讨论的通用元素)都可以包含嵌套元素。嵌套元素称为父容器的子节点。

示例

程序清单 26-2 中列举的示例说明了如何使用 canHaveChildren 属性，来直观地标识出页面上可以有嵌套元素的元素。此示例用颜色来区分可以有子元素的元素和不能有子元素的元素。第一个按钮将页面上每个可见元素的 color 样式属性设置为红色，因此，这个颜色更改操作会影响所有元素(包括通常不能有子元素的元素，如 hr 和 input)。但如果重置页面，并单击最大的按钮，则只有能包含嵌套元素的元素才接受颜色更改。

程序清单 26-2　读取 canHaveChildren 属性

HTML: jsb26-02.html

```
<!DOCTYPE html>
<html>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>canHaveChildren Property</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-02.js"></script>
    </head>
    <body>
        <h1>canHaveChildren Property Lab</h1>
        <hr />
        <form name="input">
           <input type="button" value="Color All Elements" onclick="colorAll()" />
           <br />
           <input type="button" value="Reset" onclick="history.go(0)" />
           <br />
           <input type="button" value="Color Only Elements That Can Have Children"
                  onclick="colorChildBearing()" />
        </form>
        <br />
        <hr />
        <form name="output">
           <input type="checkbox" checked="checked" />Your basic checkbox
           <input type="text" name="access2" value="Some textbox text." />
        </form>
        <table id="myTable" cellpadding="10" border="2">
           <tr>
              <th>Quantity</th>
              <th>Description</th>
              <th>Price</th>
           </tr>
           <tbody>
              <tr>
                 <td width="100">4</td>
                 <td>Primary Widget</td>
                 <td>$14.96</td>
              </tr>
              <tr>
                 <td>10</td>
                 <td>Secondary Widget</td>
                 <td>$114.96</td>
              </tr>
           </tbody>
        </table>
    </body>
</html>
```

JavaScript: jsb26-02.js

```
function colorAll()
{
   var elems = document.getElementsByTagName("*");
   for (var i = 0; i < elems.length; i++)
```


更多资源请访问稀酷客(www.ckook.com)

```
        {
            elems[i].style.color = "red";
        }
    }
    function colorChildBearing()
    {
        var elems = document.getElementsByTagName("*");
        for (var i = 0; i < elems.length; i++)
        {
            if (elems[i].canHaveChildren)
            {
                elems[i].style.color = "red";
            }
        }
    }
```

相关主题：childNodes、firstChild、lastChild、parentElement、parentNode 属性；appendChild()、hasChildNodes()、removeChild()方法。

canHaveHTML

值：Boolean，　　只读和读/写

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

并非所有 HTML 元素都是 HTML 内容的容器。canHaveHTML 属性允许脚本确定某个对象是否可接收 HTML 内容，例如用对象方法插入或替换的 HTML 内容。例如，对于 p 元素，此属性的值为 true；对于 br 元素，该值为 false。对于除 HTML Components 外的所有元素，这个属性是只读的，对于 HTMLComponents 元素，它是读/写属性。

示例

在 WinIE5+中使用 The Evaluator(参见第 4 章)来试验 canHaveHTML 属性。在顶部文本框中输入以下语句，并查看结果：

```
document.getElementById("input").canHaveHTML
document.getElementById("myP").canHaveHTML
```

第一条语句返回 false，因为 input 元素(在这里是顶部的文本框)不能嵌套 HTML，但 myP 元素是一个可以接受 HTML 内容的 P 元素。

相关主题：appendChild()方法，insertAdjacentHTML()方法，insertBefore()方法。

childNodes[]

值：Node 对象的数组，　　只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

childNodes 属性包括当前对象中包含的 Node 对象数组。注意子节点包含元素对象和文本节点。因此，根据当前对象的内容，childNodes 和子节点集合的数目可能会不同。

更多资源请访问稀酷客(www.ckook.com)

警告：

如果要在 for 循环中使用 childNodes 数组遍历 HTML(或者 XML)元素序列，浏览器可能将源代码中的空格(元素之间的空行，甚至元素之间简单的回车)作为文本节点。这个潜在的问题会影响 MacIE5 和基于 W3C 的浏览器，这些多余的文本节点主要出现在块元素周围。

注意：

大多数使用 childNodes 数组的循环都用于检查、计算或修改集合内的元素节点。如果这是脚本的目标，就测试 childNodes 数组返回的每个节点，并在处理此节点前验证 nodeType 属性是否为 1(元素)；否则，就跳过那个节点。这个循环的大致结构如下：

```
for (var i = 0; i < myElem.childNodes.length; i++)
{
   if (myElem.childNodes[i].nodeType == 1)
   {
      statements to work on element node i
   }
}
```

这些“幻影”文本节点也影响通过 firstChild 和 lastChild 属性引用的节点，参见本章后面的内容。

示例

程序清单 26-3 中的示例说明了如何编写一个函数，来遍历指定节点的所有子节点。程序清单中的 walkChildNodes()函数在文档的 HTML 元素(默认)上调用，或在将其 ID 传递为字符串参数的元素上调用，累计并返回子节点的层次列表。此函数嵌套在 The Evaluator 中，以便检查该页面的子节点层次，或将 evaluator.js 用于调试时 (如配书光盘中的第 48 章所述)，检查正在创建的页面中的节点层次。在 The Evaluator 顶部的文本框中输入以下语句，进行试验：

```
walkChildNodes()
walkChildNodes(document.getElementById("myP"))
```

此函数的结果显示了在原始对象范围内的所有子节点之间的嵌套关系，它还检查了 childNodes 集合，显示并分类了所有的子节点。为文本节点添加了相应的标记。实际文本的前 15 个字符放在结果中，在比较结果和 HTML 源代码时，可以帮助识别节点。

程序清单 26-3　收集子节点

```
<!-- xmp tags added for display of HTML in a browser window -->
<xmp>
   function walkChildNodes(objRef, n)
   {
      var obj;
      if (objRef)
      {
         if (typeof objRef == "string")
         {
            obj = document.getElementById(objRef);
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
        else
        {
            obj = objRef;
        }
    }
    else
    {
        obj = (document.body.parentElement) ?
            document.body.parentElement : document.body.parentNode;
    }
    var output = "";
    var indent = "";
    var i, group, txt;
    if (n)
    {
        for (i = 0; i < n; i++)
        {
            indent += "+---";
        }
    }
    else
    {
        n = 0;
        output += "Child Nodes of <" + obj.tagName;
        output += ">\n=====================\n";
    }
    group = obj.childNodes;
    for (i = 0; i < group.length; i++)
    {
        output += indent;
        switch (group[i].nodeType)
        {
            case 1:
                output += "<" + group[i].tagName;
                output += (group[i].id) ? " ID=" + group[i].id : "";
                output += (group[i].name) ? " NAME=" + group[i].name : "";
                output += ">\n";
                break;
            case 3:
                txt = group[i].nodeValue.substr(0,15);
                output += "[Text:\"" + txt.replace(/[\r\n]/g,"<cr>");
                    if (group[i].nodeValue.length > 15)
                    {
                        output += "...";
                    }
                output += "\"]\n";
                break;
            case 8:
                output += "[!COMMENT!]\n";
```


更多资源请访问稀酷客(www.ckook.com)

```
                    break;
                default:
                    output += "[Node Type = " + group[i].nodeType + "]\n";
            }
            if (group[i].childNodes.length > 0)
            {
            output += walkChildNodes(group[i], n+1);
            }
        }
        return output;
    }
</xmp>
```

相关主题: nodeName、nodeType、nodeValue、parentNode 属性; cloneNode()、hasChildNodes()、removeNode()、replaceNode()、swapNode()方法。

children

值: 元素对象的数组，　　　　只读

兼容性: WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera+，Chrome+

children 属性是当前对象包含的一个元素对象数组。与 childNodes 属性不同，children 不考虑文本节点，而考虑当前对象，将重点放在 HTML(和 XML)元素的包含层次上。当前对象中显示的子节点只是直接的子节点。要得到嵌套在当前对象内的所有元素对象(不管嵌套有多深)，应改用 all 集合。

示例

程序清单 26-4 说明如何使用 children 属性来遍历指定节点的子节点。此函数在文档的 HTML 元素(默认)，或在将其 ID 传递为字符串参数的元素上调用，累计并返回子元素的层次列表。此函数嵌套在 The Evaluator 中，以便检查该页面的父/子层次，或在将 evaluator.js 用于调试时(如配书光盘上的第 48 章所述)，检查正在创建的页面中的节点层次。在 The Evaluator 顶部的文本框中输入以下语句，进行试验：

```
walkChildren("myTable")
```

注意，在此示例中用元素名来调用 walkChildren()函数，而不是对 document.getElementId()的调用，这显示了 walkChildren()函数的灵活性以及它操作对象或对象(元素)名的方式。程序清单 26-3 中的 walkChildNodes()函数具有同样的灵活性。

walkChildren ()函数的结果显示了原始对象范围内的所有父/子元素间的嵌套关系，它还检查了 children 集合，显示并分类了所有子元素。如果在 HTML 源代码中为元素指定了 id 和/或 name 特性值，元素标记也会显示这些值。

程序清单 26-4　收集子元素

```
<!-- xmp tags added for display of HTML in a browser window -->
<xmp>
    function walkChildren(objRef, n)
    {
```


更多资源请访问稀酷客(www.ckook.com)

```
        var obj;
        if (objRef)
        {
            if (typeof objRef == "string")
            {
                obj = document.getElementById(objRef);
            }
            else
            {
                obj = objRef;
            }
        }
        else
        {
            obj = document.body.parentElement;
        }
        var output = "";
        var indent = "";
        var i, group;
        if (n)
        {
            for (i = 0; i < n; i++)
            {
                indent += "+---";
            }
        }
        else
        {
           n = 0;
           output += "Children of <" + obj.tagName;
           output += ">\n====================\n";
        }
        group = obj.children;
        for (i = 0; i < group.length; i++)
        {
           output += indent + "<" + group[i].tagName;
           output += (group[i].id) ? " ID=" + group[i].id : "";
           output += (group[i].name) ? " NAME=" + group[i].name : "";
           output += ">\n";
           if (group[i].children.length > 0)
           {
               output += walkChildren(group[i], n+1);
           }
        }
        return output;
    }
</xmp>
```

相关主题：canHaveChildren、firstChild、lastChild、parentElement 属性；appendChild()、removeChild()、replaceChild()方法。

更多资源请访问稀酷客(www.ckook.com)

cite

值：URL 字符串，　　　　读/写

兼容性：WinIE6+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

cite 属性包含了一个作为元素源引用的 URL，就像引文的作者一样。此属性只应用于 blockquote、q、del 和 ins 元素对象，但 IE 在更多文本内容对象中支持它。这可能是一个错误，也可能不是，因此对除上述元素外的其他元素使用此属性可能并不安全。

className

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

类名是一个标识符，赋予元素的 class 特性。为将 CSS 规则与文档中的几个元素关联起来，可将同一个标识符赋给那些元素的 class 特性，并使用该标识符(前面加一个句点)作为 CSS 规则的选择符。在脚本控制下，元素的 className 属性将不同的 CSS 规则应用于该元素。程序清单 26-5 提供了这样一个脚本的例子。

示例

在程序清单 26-5 中，将元素的 className 属性交替设置为现有样式表类选择器名称和空字符串，就可以打开和关闭元素的样式。将 className 设置为空字符串时，hl 元素的默认行为决定了第一个标题的显示样式，单击按钮则使样式表规则覆盖第一个 hl 元素中的默认行为。

程序清单 26-5　设置 className 属性

HTML: jsb26-05.html

```
<!DOCTYPE html>
<html>
  <head>
     <meta http-equiv="content-type" content="text/html;charset=utf-8">
     <title>className Property</title>
     <style type="text/css">
        .special
        {
           font-size:16pt; color:red;
        }
     </style>
     <script type="text/javascript" src="../jsb-global.js"></script>
     <script type="text/javascript" src="jsb26-05.js"></script>
  </head>
  <body>
     <h1>className Property Lab</h1>
     <hr />
     <form name="input">
        <input type="button" value="Toggle Class Name"
               onclick="toggleSpecialStyle('head1')" />
     </form>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <br />
        <h1 id="head1">ARTICLE I</h1>
        <p>Congress shall make no law respecting an establishment of religion, or
           prohibiting the free exercise thereof; or abridging the freedom of
           speech, or of the press; or the right of the people peaceably to
           assemble, and to petition the government for a redress of grievances.
        </p>
        <h1>ARTICLE II</h1>
        <p>A well regulated militia, being necessary to the security of a free
           state, the right of the people to keep and bear arms, shall not be
           infringed.
        </p>
    </body>
</html>
```

JavaScript: jsb26-05.js

```
function toggleSpecialStyle(elemID)
{
   var elem = (document.all) ? document.all(elemID) :
              document.getElementById(elemID);
      if (elem.className == "")
      {
          elem.className = "special";
      }
      else
      {
          elem.className = "";
      }
}
```

还可用不同的类选择器标识符创建样式规则的多个版本，并将它们应用于指定的元素。

相关主题：rule、stylesheet 对象(参阅第 38 章)、id 属性。

clientHeight，clientWidth

值：Integer，　只读

兼容性：WinIE4+，MacIE4+，NN7，Moz1.0.1+，Safari+，Opera+，Chrome+

这两个属性基本上显示了元素(它的样式表规则包括高度和宽度设置)内容的高度和宽度(以像素为单位)。在理论上，这些尺寸没有考虑通过样式表添加元素的页边距、边框或补白。实际上，页边距、边框和补白的不同组合会以意料不到的方式影响这些值。例如 clientHeight 属性更可靠的一个用途是指出溢出元素的文本在何处结束。为了读取元素的显示尺寸，可以跨浏览器使用 offsetHeight 和 offsetWidth 属性。

对于 document.body 对象，clientHeight 和 clientWidth 属性返回窗口或框架的内高和内宽(加上或减去两个像素)，它们代替了需要的、但不存在的 IE 窗口属性。

与以前的版本不同，IE5+扩展了使用这些属性的对象，包括代表 HTML 元素的所有对象。在基于 Mozilla 的浏览器中，这些属性值都是 0；但 document.body 例外，它度量浏览器的当前内容区域。

更多资源请访问稀酷客(www.ckook.com)

示例

程序清单 26-6 包括了一个针对 IE 的示例，演示了如何根据客户区的宽度和高度，来动态度量页面内容的大小。此示例对包含段落元素的 div 元素调用了 clientHeight 和 clientWidth 属性。在 div 元素的样式表规则中只指定了其宽度，这意味着段落文本在该宽度内将自动换行，并向下延伸必要的深度，以显示整个段落，clientHeight 属性指定了该深度。然后不管文本的长度如何，利用 clientHeight 属性计算紧接 div 元素之后放置 logo 图像的位置。另外，clientWidth 属性帮助脚本将图像相对于段落文本水平居中。

程序清单 26-6　使用 clientHeight 和 clientWidth 属性

HTML: jsb26-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>clientHeight and clientWidth Properties</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-06.js"></script>
   </head>
   <body>
      <button onclick="showLogo()">Position and Show Logo Art</button>
      <div id="myDIV" style="width:200px">
         <p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do
            eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
            adminim veniam, quis nostrud exercitation ullamco laboris nisi ut
            aliquip ex ea commodo consequat. Duis aute irure dolor in
            reprehenderit involuptate velit esse cillum dolore eu fugiat nulla
            pariatur. Excepteur sint occaecat cupidatat non proident.
         </p>
      </div>
      <div id="logo" style="position:absolute; width:120px; visibility:hidden">
         <img alt="image" src="logo.gif" />
      </div>
   </body>
</html>
```

JavaScript: jsb26-01.js

```
function showLogo()
{
   var paragraphW = document.getElementById("myDIV").clientWidth;
   var paragraphH = document.getElementById("myDIV").clientHeight;
   // correct for Windows/Mac discrepancies
   var paragraphTop = (document.getElementById("myDIV").clientTop) ?
      document.getElementById("myDIV").clientTop :
      document.getElementById("myDIV").offsetTop;
   var logoW = document.getElementById("logo").style.pixelWidth;
```


更多资源请访问稀酷客(www.ckook.com)

```
    // center logo horizontally against paragraph
    document.getElementById("logo").style.pixelLeft = (paragraphW-logoW) / 2;
    // position image immediately below end of paragraph
    document.getElementById("logo").style.pixelTop = paragraphTop + paragraphH;
    document.getElementById("logo").style.visibility = "visible";
}
```

为帮助垂直定位 logo 图像，div 对象的 offsetTop 属性提供了 div 相对于其外部容器(body)的起始位置。但 MacIE 使用 clientTop 属性来获得所需的尺寸。该尺寸(赋给 paragraphTop 变量)加上 div 的 clientHeight，就是图像的顶部坐标。

相关主题：offsetHeight、offsetWidth 属性。

clientLeft，clientTop

值：整数，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera+，Chrome+

clientLeft 和 clientTop 属性的目的和名称是最容易混淆的。clientHeight 和 clientWidth 属性应用到元素的内容中，而 clientLeft 和 clientTop 属性只返回元素周围边框的厚度(假设元素已定位)。如果不指定边框或不定位该元素，这两个属性的值就是 0(但如果没有明确设置，document.body 对象在每个方向显示两个像素)。如果要读取元素左边和上边的坐标位置，在 WinIE 中就应使用 offsetLeft 和 offsetTop 属性。但如程序清单 26-6 所示，clientTop 属性在 MacIE 中返回一个适当的值。所有元素在 IE5+中都有 clientLeft 和 clientTop 属性，而 MacIE 中的支持不大一致。

相关主题：offsetLeft，offsetTop 属性。

contentEditable

值：Boolean，　　读/写

兼容性：WinIE5.5+，MacIE-，NN-，Moz+，Safari+，Opera+，Chrome+

Internet Explorer 5.5 引入了页面上可编辑 HTML 内容的概念。元素标记可包括 content Editable 特性，它的值通过元素的 contentEditable 属性来反映。这个属性的默认值为 inherit，它意味着这个属性继承了在 HTML 容器层次中设置的属性值。如果将 contentEditable 属性设置为 true，则继承该值的元素及其所有嵌套元素都是可编辑的；相反，false 设置将禁止编辑 HTML 内容。一些浏览器给可编辑的元素自动使用闪光的蓝色边框，说明哪些是可编辑元素。

示例

程序清单 26-7 演示了如何使用 contentEditable 属性来创建一个非常简单的诗歌编辑器。单击新载入页面的按钮时，toggleEdit()函数通过要编辑的 div 元素的 ContentEditable 属性，对当前可编辑状态取反，下一条语句将新值赋给 div 元素的 contentEditable 属性，来启用该元素的编辑状态。为了增强视觉效果，将 div 的文本变为红色，来提供额外的用户反馈，说明页面上哪些是可编辑的内容。还可以切换按钮标签，以表示下一次单击该按钮将引发的动作。

更多资源请访问稀酷客(www.ckook.com)

程序清单 26-7　使用 contentEditable 属性

HTML: jsb26-07.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Editable Content</title>
      <style type="text/css">
         .normal
         {
            color: black;
         }
         .editing
         {
            color: red;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-07.js"></script>
   </head>
   <body>
      <h1>Poetry Editor</h1>
      <hr />
      <p>Turn on editing to modify the following text:</p>
      <div id="noneditableText">
         Roses are red,
         <br />
         Violets are blue.
      </div>
      <div id="editableText">
         Line 3,
         <br />
         Line 4.
         </div>
      <p>
         <button id="editBtn" onclick="toggleEdit()"
                 onfocus="this.blur()">Enable Editing</button>
      </p>
   </body>
</html>
```

JavaScript: jsb26-07.js

```
function toggleEdit()
{
   var newState = document.getElementById("editableText").contentEditable;
   // While contentEditable is boolean, different browsers behave differently
      if (newState == false || newState == "false" || newState == "inherit")
      {
```


更多资源请访问稀酷客(www.ckook.com)

```
            newState = true;
        }
        else
        {
            newState = false;
        }
    document.getElementById("editableText").contentEditable = newState;
    document.getElementById("editableText").className = (newState) ?
            "editing" : "normal";
    document.getElementById("editBtn").innerHTML = (newState) ?
            "Disable Editing" : "Enable Editing";
}
```

相关主题：isContentEditable 属性。

currentStyle

值：style 对象，　只读

兼容性：WinIE5+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

每个元素都有应用于其上的样式特性，有时这些特性是浏览器的默认设置。因为元素的 style 对象只反映某些属性，这些属性的相应特性通过 CSS 语句明确设置，所以不能使用 element 对象的 style 属性来查看应用于元素的默认样式设置，此时就应使用 currentStyle 属性。

这个属性返回一个只读 style 对象，它包含可应用于元素的每个 style 属性的值。如果 style 属性通过 CSS 语句或脚本语句来明确设置，该属性的当前值就是可读的。这样，脚本就可检查任意属性，来确定它是否应该改变，来满足某个设计目标。

例如，若使用<em>标记包括一些文本，则浏览器会默认把这些文本变为斜体样式。这个设置并没有反映在元素的 style 对象中(fontStyle 属性)，因为斜体设置不是通过 CSS 设置的。与此形成对照的是，element 对象的 currentStyle.fontStyle 属性使元素的当前 fontStyle 属性值变成 italic。

示例

要更改 style 属性设置，可通过元素的 style 对象访问它。使用 The Evaluator(参见第 4 章)来比较元素的 currentStyle 和 style 对象的属性。例如，未修改的 The Evaluator 版本包含一个 ID 为 myEM 的 em 元素。在底部的属性列表文本框中输入 document.getElementById ("myEM").style，并按回车键，则大多数属性值都为空。现在，在属性列表文本框中输入 document.getElementById ("myEM").currentStyle，并按回车键，每个属性就都有了相关的值。

相关主题：runtimeStyle、style 对象(参阅第 38 章)；window.getComputedStyle()方法(用于 W3C DOM 浏览器)(参阅第 27 章)。

dateTime

值：Date 字符串，　只读

兼容性：WinIE6+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

dateTime 属性包含了为元素建立时间戳的日期/时间值。与 cite 属性类似，使用 dateTime 属性的元素对象(del 和 ins)个数比 IE 实际能支持的要少。这可能是一个错误，也可能不是，因此对除上述元素外的其他元素使用此属性可能并不安全。

dataFld，dataFormatAs，dataSrc

值：字符串，　　读/写

兼容性：WinIE4+，MacIE5，NN-，Moz-，Safari-，Opera-，Chrome-

dataFld、dataFormatAs 和 dataSrc 属性(以及更多的专有元素属性，如 dataPageSize 和 recordNumber)是 Internet Explorer 中基于 ActiveX 控件的数据绑定工具的一部分。IE4 的 Win32 版本有几个嵌入浏览器的 ActiveX 对象，以便 Web 页面和数据源之间进行直接通信。数据源包括文本文件、XML 数据、HTML 数据和外部数据库(MacIE 仅支持文本文件)。数据绑定是一个非常宽泛的主题，许多内容的讨论都需要了解 Microsoft Data Source Objects (DSO，数据源对象)、ODBC 和 JDBC，它们大都超出了本书的讨论范围。但数据绑定是一个功能强大的工具，即使不是数据库高手也可使用它。因此，这里讨论的三个基本属性 dataFld、dataFormatAs 和 dataSrc 主要通过 Microsoft 的 Tabular Data Control DSO 进行数据绑定，这允许任何页面访问、分类、显示和筛选(但不是更新)从外部文本文件下载到 Web 页面中的数据(通常是逗号或 tab 分隔的数据)。

利用 Tabular Data Control (TDC)，可将外部文本文件中的数据载入文档，并在<object>标记对中指定 TDC 对象和附加参数(例如文本文件的 URL 和域分隔字符)，来获得数据。object 元素可放到文档的 body 中的任何地方，最好把它放到代码的底部，使所有常规页面在控件载入前显示。获得数据只是将数据放在浏览器中，页面并不显示数据。

如果没有用过 IE 中的嵌入对象，就可能觉得 classid 特性值有点奇怪。最令人迷惑的是为对象指定全局唯一标识符(Globally Unique Identifier，GUID)格式的长数据值，这是唯一标识对象的 IE 方式。只有像下列例子那样输入值，才能正确运行 ActiveX TDC。这个对象的 HTML 语法如下：

```
<object id="objName" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
   <param name="DataURL" value="URL">
   [additional optional parameters]
</object>
```

表 26-3 列出了 TDC 的可用参数，它只需要 DataURL 参数；是否需要其他参数(例如 FieldDelim、UseHeader、RowDelim 和 EscapeChar)取决于数据源的属性。

表 26-3　Tabular Data Control 的参数

参　　数	说　　明
CharSet	数据源文件的字符集，默认为 latin1
DataURL	数据源文件的 URL(相对或绝对)
EscapeChar	用于转义数据中的定界符的字符，默认为空，常用值是"\"
FieldDelim	记录中字段间的定界符，默认为逗号(,)。对于 Tab 字符，使用值

更多资源请访问稀酷客(www.ckook.com)

(续表)

参　数	说　明
Language	源数据的 ISO 语言代码，默认为 en-us
TextQualifier	包围字段数据的可选字符，默认为空
RowDelim	记录间的定界符，默认为换行(NL)
UseHeader	如果文件中的第一行数据包含字段名，则设置为 true。默认为 false

赋给 object 元素的 id 特性的值是标识符，在页面和数据完全载入后，脚本就使用该标识符与数据通信。因此，有多少个要同时访问的数据源文件，就必须将多少个具有唯一名称的 TDC 载入页面。

将值赋给元素的 datasrc 和 datafld 特性时，数据通常先绑定到 HTML 元素中。Datasrc 特性指向 dso 标识符(与 object 元素的 id 特性匹配，前面有一个斜线符号)，而 datafld 特性指向提取其数据的域名。当与交互元素(例如表)进行数据绑定时，多个记录会显示在表的连续行中(详见后面的内容)。

如果想改变同一个 HTML 元素(除了表)显示的数据，只需调整 dataSrc 和 dataFld 属性。这些属性应用于和外部数据相关的 HTML 元素子集：a、applet、body、button、div、frame、iframe、img、input(大多数类型)、label、marquee、object、param、select、span 和 textarea 对象。

在一些情况下，数据源可在元素中存储用来显示的 HTML 格式文本块。除非有另外的设置，否则，即使内容包含 HTML 格式标记，浏览器也将数据源域显示为普通文本。但如果要在显示过程中识别 HTML，可将 dataFormatAs 属性(或标记的 dataformatas 特性)设置为 HTML，它的默认值为 text。

示例

程序清单 26-8 中的简单文档有两个相关的 TDC 对象。外部文件是不同格式的美国人权法案文件。第一个文件仅由两条记录组成，是用 Tab 分隔的传统数据文件：第一条记录是用 Tab 分隔的字段名序列(名称为"Articlel"、"Article2"等)；第二条记录是用 Tab 分隔的、以 HTML 定义的条款内容序列：

```
<h1>ARTICLE I</h1>
<p>Congress shall make...</p>
```

第二个文件是另一个法案的完整文本文件，没有附带 HTML 格式。

在载入程序清单 26-8 时，在蓝色边框的文本框中只显示另一个法案的第一个条款，单击按钮可以导航到序列中的前一个和后一个条款。由于数据源是用 Tab 分隔的传统文件，因此 nextField() 和 prevField()函数计算下一个源字段的名称，并给 dataFld 属性指定新值。在页面载入后，所有数据都已经在浏览器中了，因此浏览记录的速度就是浏览器回流页面来提供新内容的速度。

程序清单 26-8 数据绑定到页面

HTML: jsb26-08.html

```
<!DOCTYPE html>
<html>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Data Binding</title>
        <style type="text/css">
            #display
            {
                width:500px; border:10px ridge blue; padding:20px;
            }
            .hiddenControl
            {
                display:none;
            }
        </style>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-08.js"></script>
    </head>
    <body>
        <h1>U.S. Bill of Rights</h1>
        <form>
            <input type="button" value="Toggle Complete/Individual"
                   onclick="toggleComplete()" />
            <span id="buttonWrapper" class="">
                <input type="button" value="Prev" onclick="prevField()" />
                <input type="button" value="Next" onclick="nextField()" />
            </span>
        </form>
        <div id="display" datasrc="#rights_html" datafld="Article1"
             dataformatas="HTML">
        </div>
        <object id="rights_html"
                classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
            <param name="DataURL" value="Bill of Rights.txt" />
            <param name="UseHeader" value="True" />
            <param name="FieldDelim" value="&#09;" />
        </object>
        <object id="rights_raw"
                classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
            <param name="DataURL" value="Bill of Rights (no format).txt" />
            <param name="FieldDelim" value="\" />
            <param name="RowDelim" value="\" />
        </object>
    </body>
</html>
```

JavaScript: jsb26-08.js

```
function nextField()
{
    var elem = document.getElementById("display");
    var fieldName = elem.dataFld;
    var currFieldNum = parseInt(fieldName.substring(7, fieldName.length),10);
```


更多资源请访问稀酷客(www.ckook.com)

```
    currFieldNum = (currFieldNum == 10) ? 1 : ++currFieldNum;
    elem.dataFld = "Article" + currFieldNum;
}
function prevField()
{
  var elem = document.getElementById("display");
  var fieldName = elem.dataFld;
  var currFieldNum = parseInt(fieldName.substring(7, fieldName.length),10);
  currFieldNum = (currFieldNum == 1) ? 10 : --currFieldNum;
  elem.dataFld = "Article" + currFieldNum;
}

function toggleComplete()
{
  if (document.getElementById("buttonWrapper").className == "")
  {
     document.getElementById("display").dataSrc = "#rights_raw";
     document.getElementById("display").dataFld = "column1";
     document.getElementById("display").dataFormatAs = "text";
     document.getElementById("buttonWrapper").className = "hiddenControl";
  }
  else
  {
     document.getElementById("display").dataSrc = "#rights_html";
     document.getElementById("display").dataFld = "Article1";
     document.getElementById("display").dataFormatAs = "HTML";
     document.getElementById("buttonWrapper").className = "";
  }
}
```

页面上的另一个按钮可在文档最初的逐项条款格式与未格式化版本之间切换。为将整个文档载入为单个记录，第二个 object 元素的 FieldDelim 和 RowDelim 参数用根本不在文档上显示的字符替换了其默认值。由于外部文件中没有字段名，默认值(对于此文档中的唯一一列，默认值是 column1)就是数据字段。这样，在 toggleComplete()函数中，dataSrc 属性更改为所需的 object 元素 ID，dataFld 属性设置为数据源的正确值，dataFormatAs 属性更改为反映源内容的不同用途(显示为 HTML 或纯文本)。整个文档显示出来后，可向包围按钮的 span 元素指定一个 className 值，来隐藏两个单选按钮。className 值是文档样式表中的类选择器标识符。当 toggleComplete()函数将 className 属性重置为空时，默认属性(正常的内联显示样式)生效。

另一个示例展示了 TDC 在脚本控制下的这种强大功能。程序清单 26-9 用表格显示以 Tab 分隔的奥斯卡奖信息文件中的数据。该数据文件有 8 列数据，每个列标题都用作字段名：Year、Best Picture、Best Director、Best Director Film、Best Actress、Best Actress Film、Best Actor 和 Best Actor Film。为了设计页面，每条记录只显示 5 个字段：Year、Film、Director、Actress 和 Actor。注意，在程序清单中，表格及其内容的 HTML 绑定到数据源对象和数据中的字段。

此示例的动态部分表现为在数据载入浏览器后，可以对数据进行排序和筛选，而不需要访问最初的源数据。TDC 对象的 Sort 和 Filter 属性能对当前载入浏览器的数据进行操作。最简单的排序操作是指出整个数据集应按哪个字段(或哪些字段，通过用分号分隔开的字段名列表)排序。在排序字段名前有一个加号(表示升序)或减号(表示降序)。设置好 data 对象的 Sort 属性后，

更多资源请访问稀酷客(www.ckook.com)

调用其 Reset()方法，告诉对象应用新的属性。这会立即重新提取绑定表中的数据，以反映相应的变化。

同样，可让数据集显示符合特定条件的记录。在程序清单 26-9 中，两个选择列表和一对单选按钮提供了设置 Filter 属性的界面。例如，可对输出进行筛选，只显示同时获得最佳影片奖和最佳女主角奖的电影记录。简单的筛选表达式是基于字段名的：

```
dataObj.Filter = "Best Picture" = "Best Actress Film";
```

程序清单 26-9 对绑定的数据进行排序

HTML: jsb26-09.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Data Binding - Sorting</title>
      <style type="text/css">
         thead
         {
         background-color:yellow; text-align:center;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-09.js"></script>
   </head>
   <body>
      <h1>Academy Awards 1978-2005</h1>
      <form>
         <p>Sort list by year
           <a href="javascript:sortByYear('normal')">from newest to oldest</a> or
           <a href="javascript:sortByYear('reverse')">from oldest to newest</a>.
         </p>
         <p>Filter listings for records whose
           <select name="filter1" onchange="filterInCommon(this.form)">
                <option value="Best Picture">Best Picture</option>
                <option value="Best Director Film">Best Director's Film</option>
                <option value="Best Actress Film">Best Actress' Film</option>
                <option value="Best Actor Film">Best Actor's Film</option>
           </select>
           <input type="radio" name="operator" checked="checked"
                   onclick="filterInCommon(this.form)" />is
           <input type="radio" name="operator"
                   onclick="filterInCommon(this.form)" />is not
           <select name="filter2" onchange="filterInCommon(this.form)">
                <option value="Best Picture">Best Picture</option>
                <option value="Best Director Film">Best Director's Film</option>
                <option value="Best Actress Film">Best Actress' 'Film</option>
                <option value="Best Actor Film">Best Actor's Film</option>
           </select>
```


更多资源请访问稀酷客(www.ckook.com)

```
            </p>
          </form>
          <table datasrc="#oscars" border="1" align="center">
            <thead>
              <tr>
                <td>Year</td>
                <td>Film</td>
                <td>Director</td>
                <td>Actress</td>
                <td>Actor</td>
              </tr>
            </thead>
            <tr>
                <td><div id="col1" datafld="Year"></div></td>
                <td><div id="col2" datafld="Best Picture"></div></td>
                <td><div id="col3" datafld="Best Director"></div></td>
                <td><div id="col4" datafld="Best Actress"></div></td>
                <td><div id="col5" datafld="Best Actor"></div></td>
            </tr>
          </table>
          <object id="oscars" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
            <param name="DataURL" value="Academy Awards.txt" />
            <param name="UseHeader" value="True" />
            <param name="FieldDelim" value="&#09;" />
          </object>
       </body>
    </html>
```

JavaScript: jsb26-09.js

```
function sortByYear(type)
{
   oscars.Sort = (type == "normal") ? "-Year" : "+Year";
   oscars.Reset();
}
function filterInCommon(form)
{
   var filterExpr1 = form.filter1.options[form.filter1.selectedIndex].value;
   var filterExpr2 = form.filter2.options[form.filter2.selectedIndex].value;
   var operator = (form.operator[0].checked) ? "=" : "<>";
   var filterExpr = filterExpr1 + operator + filterExpr2;
   oscars.Filter = filterExpr;
   oscars.Reset();
}
```

相关主题：recordNumber、table.dataPageSize 属性。

dir

值："ltr"|"rtl"， 读/写

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

dir 属性(基于每个面向文本的 HTML 元素的 dir 特性)控制元素文本是从左到右(默认)还是从右到左显示。大体上，只有覆盖 Unicode 标准定义的语言字符集的默认方向时，才需要该属性(和 HTML 特性)。

示例

在标准的 U.S.版本浏览器中更改此属性值，只会使右边界成为每行新文本的起点(换句话说，字符不是按相反顺序显示)。可在 The Evaluator 的表达式求值域中输入以下语句，来试用此属性：

```
document.getElementById("myP").dir = "rtl"
```

相关主题：lang 属性。

disabled

数值：Boolean，　　读写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

在 IE4 和 IE5 中，只有表单元素具有 disabled 属性，但在 IE5.5+中，所有 HTML 元素都具有这个属性。W3C DOM 浏览器仅将此属性应用于表单控件和 style 元素对象。禁用 HTML 元素(例如表单元素)通常使该元素的外观变暗，表示它没有激活。不可用的元素不接收任何事件，也不能接收焦点，不管焦点是手动还是通过脚本设置的。但用户仍然可选择和复制已禁用的主体文本元素。

注意：

如果禁用表单控件元素，该元素的数据就不与其他表单元素一并提交给服务器。如果需要"锁住"一个表单控件，而仍把它提交给服务器，应使用 form 元素的 onsubmit 事件处理程序，在表单提交前使表单控件可用。

示例

用 The Evaluator(参见第 4 章)试验表单元素(IE4+和 W3C)和普通 HTML 元素(WinIE5.5+)的 disabled 属性。对于 IE4+和 W3C 浏览器，在顶部文本框中输入以下语句，来禁用输出文本区域，看看会发生什么：

```
document.forms[0].output.disabled = true
```

文本区域禁止用户进行输入，但仍然可以通过脚本设置该域的 value 属性(这就是返回 true 的方式)。

如果使用 WinIE5.5+，在顶部文本框中输入以下语句，以禁用 myP 元素：

```
document.getElementById("myP").disabled = true
```

示例段落的文本将变成灰色。

相关主题：isDisabled 属性。

更多资源请访问稀酷客(www.ckook.com)

document

值：document 对象，　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera+，Chrome-

在 IE4+/Opera 显示的 HTML 元素对象中，document 属性是对包含对象的文档的引用。虽然不大可能需要使用这个属性，但如果复杂脚本和脚本库以通用方式处理对象，且不知道包含特定对象的文档的引用路径，那么使用 document 属性就很方便。有时可能需要使用文档的引用来查找文档中的相关对象。这个属性的 W3C 版本是 ownerDocument。

示例

以下简化的函数接收一个参数，该参数可以是文档层次结构中的任何对象。脚本找出包含文档的对象引用，以引用其他对象：

```
function getCompanionFormCount(obj)
{
   var ownerDoc = obj.document;
   return ownerDoc.forms.length;
}
```

由于 ownerDoc 变量包含了对 document 对象的有效引用，所以 return 语句可使用该引用返回文档对象层次结构中的一个典型属性。

相关主题：ownerDocument 属性。

filters[]

值：数组，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

滤镜是 IE 专用的样式表附件，它提供了更多的字体渲染效果(例如阴影)，并可使元素隐藏或可见。每个滤镜规范都是一个 filter 对象，filters 属性包括为当前元素定义的一组 filter 对象。滤镜可应用于下列元素集：bdo、body、button、fieldset、img、input、marquee、rt、ruby、table、td、textarea、th 以及定位的 div 和 span 元素。有关样式表滤镜的详情，请参阅第 38 章。

相关主题：filter 对象。

firstChild，lastChild

值：Node 对象引用，　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

基于 W3C DOM 的文档对象模型建立在一个称为节点图的结构上。HTML 定义的每个对象都是图中的一个节点。一个节点和文档中的其他节点有关系，用家谱的术语来说就是父节点、兄弟节点和子节点的关系。

子节点是被另一个元素包含的元素，容器是子节点的父节点。一个 HTML 元素可包含任意数量的子元素，同样，父节点可有 0 个或多个子节点。可以通过 childNodes 属性从对象中读取一列子节点(返回为一个数组)：

更多资源请访问稀酷客(www.ckook.com)

```
var nodeArray = document.getElementById("elementID").childNodes;
```

使用这个数组及其 length 属性可以得到第一个或最后一个子节点的引用，而 firstChild 和 lastChild 属性提供了对这些节点的快捷引用。要在所有节点之前或之后插入新的子节点，或者需要引用把元素添加到文档节点列表中的方法时，就可以使用这些属性。

示例

程序清单 26-10 包含的示例说明了如何使用 firstChild 和 lastChild 属性来访问子节点。这两个属性在此示例中用于为现有的 ol 元素添加和替换 li 元素。可在列表开头或结尾处输入要显示的文本。使用 firstChild 和 lastChild 属性简化了对列表两端的访问。该示例使用 replaceChild() 方法来替换子节点。对于 IE4+，也可以修改 firstChild 或 lastChild 属性返回的对象的 innerText 属性。将项添加到列表中时，此示例尤为有趣：浏览器自动对项重新编号，以便匹配列表的当前状态。

警告：

某些浏览器版本中存在幻影节点，详见本章前面对 childNodes 属性的讨论。这一问题可能影响 firstChild 和 lastChild 属性的使用。

注意：

尽管 innerHTML 属性很方便，但在 JavaScript 的严格 W3C 语法中不应使用它，因为它不是 W3C 标准的正式组成部分。不过，它通常是一个非常方便的强大属性，不能忽视，如本书中的许多代码所示。本书的一些示例也给出了对 innerHTML 的 W3C 节点操作替代方法。对于 innerHTML 的 W3C 替代方法的详细解释和示例，请参见第 29 章。

程序清单 26-10 使用 firstChild 和 lastChild 属性

HTML: jsb26-10.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>firstChild and lastChild Properties</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-10.js"></script>
   </head>
   <body>
      <h1>firstChild and lastChild Property Lab</h1>
      <hr />
      <form>
         <label>Enter some text to add to or replace in the OL element:</label>
         <br />
         <input type="text" name="input" size="50" />
         <br />
         <input type="button" value="Insert at Top"
               onclick="prepend(this.form)" />
         <input type="button" value="Append to Bottom"
```


更多资源请访问稀酷客(www.ckook.com)

```
                onclick="append(this.form)" />
            <br />
            <input type="button" value="Replace First Item"
                onclick="replaceFirst(this.form)" />
            <input type="button" value="Replace Last Item"
                onclick="replaceLast(this.form)" />
        </form>
        <ol id="myList">
            <li>Initial Item 1</li>
            <li>Initial Item 2</li>
            <li>Initial Item 3</li>
            <li>Initial Item 4</li>
        </ol>
    </body>
</html>
```

JavaScript: jsb26-10.js

```
// helper function for prepend() and append()
function makeNewLI(txt)
{
    var newItem = document.createElement("LI");
    newItem.innerHTML = txt;
    return newItem;
}
function prepend(form)
{
    var newItem = makeNewLI(form.input.value);
    var firstLI = document.getElementById("myList").firstChild;
    document.getElementById("myList").insertBefore(newItem, firstLI);
}
function append(form)
{
    var newItem = makeNewLI(form.input.value);
    var lastLI = document.getElementById("myList").lastChild;
    document.getElementById("myList").appendChild(newItem);
}
function replaceFirst(form)
{
    var newItem = makeNewLI(form.input.value);
    var firstLI = document.getElementById("myList").firstChild;
    document.getElementById("myList").replaceChild(newItem, firstLI);
}
function replaceLast(form)
{
    var newItem = makeNewLI(form.input.value);
    var lastLI = document.getElementById("myList").lastChild;
    document.getElementById("myList").replaceChild(newItem, lastLI);
}
```

相关主题： nextSibling、parentElement、parentNode、previousSibling 属性；appendChild()、

更多资源请访问稀酷客(www.ckook.com)

hasChildNodes()、removeChild()、removeNode()、replaceChild()、replaceNode()方法

height，width

值：整型或百分比字符串，　　　　读/写和只读

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

此处描述的 height 和"width"属性不同于元素样式的同名属性。这些属性反映了赋予元素(例如 img、applet 和 table 等)的 height 和 width 特性的值，它们可直接在对象中访问(例如 document.getElementById("myTable").width)，而不是通过 style 对象来访问(例如，document.getElementById ("myDIV").style.width)。只有 HTML 4.x 标准为元素提供了 height 和 width 特性，元素才有这些属性。

这些属性的值是整数像素值(数值或字符串)或百分数(仅字符串)。如果需要在现有的百分数上进行某些计算，可使用 parseInt()函数来提取数值，进行数学计算。如果元素的 height 和 width 特性设置为百分数，可使用许多现代浏览器中的 offsetHeight 和 offsetWidth 属性，来获得所显示元素的像素尺寸。

这些属性值对于大多数新版浏览器的 image 对象是可读/写的，因为可以在页面载入后，重新调整 image 对象的大小。这些属性对其他一些对象(例如 table 对象)也是可读/写的，但不见得包括支持这些属性的所有对象。

一般来说，这些属性的值不能小于显示元素所需的大小，对于表而言尤其如此。如果试图使 height 值小于显示表所需的像素尺寸(由它的样式设置定义)，则对该属性值的改变就不起作用(即使属性值仍为手工设置的较小值)。然而对其他对象而言，无论该属性设置为什么，浏览器都会相应缩放元素的内容(例如图像)。如果只想查看元素的一部分(换句话说，即剪裁元素)，可使用样式表来设置元素的剪切区域。

示例

以下示例演示了如何使用 width 属性将表格的宽度增加 10%：

```
var tableW = parseInt(document.getElementById("myTable").width);
document.getElementById("myTable").width = (tableW * 1.1) + "%";
```

由于 table 元素的 width 特性最初设置为百分数，因此脚本从百分数形式的宽度字符串中提取数字。在第二条语句中，给原有数值增加 10%，并在值后添加百分号，将其转换为百分比字符串。得到的字符串值赋给表格的 width 属性。

相关主题：clientHeight、clientWidth 属性；style.height、style.width 属性。

hideFocus

值：Boolean，　　　　读/写

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在 Windows IE 中，只要元素拥有焦点，按钮类型的表单控件和链接就在元素周围显示一个虚点框。如果在 IE5+中给其他类型的元素设置 tabindex 特性或 tabIndex 属性，这些元素也会在获得焦点时显示虚线框。将元素对象的 hideFocus 属性设置为 true(默认值为 false)，就仍可以让该元素接收焦点，但隐藏虚点框。

PDG

更多资源请访问稀酷客(www.ckook.com)

隐藏焦点不会使元素不可用。实际上，如果接收焦点的元素在视图之外，页面会自动滚动，使元素显示在视图中。响应键盘动作的表单控件(例如，按空格键来选中复选框控件，或者取消其选中状态)也会继续像平常一样工作。对一些设计者而言，焦点矩形影响了页面的设计，而 hideFocus 属性可以对外观进行更多的控制，同时继续执行页面的其他操作。HTML 标记中没有与此对应的特性，因此在页面载入后，可在页面上使用 onload 事件处理程序设置对象的 hideFocus 属性。

示例

用 The Evaluator(参见第 4 章)试验 WinIE5.5+中的 hideFocus 属性。在顶部文本框中输入以下语句，将一个 tabIndex 值赋给 myP 元素，该元素在默认情况下接收焦点，并显示虚点框：

```
document.getElementById("myP").tabIndex = 1
```

按几次 Tab 键，直至段落接收到焦点。现在禁用焦点矩形：

```
document.getElementById("myP").hideFocus = true
```

如果现在按几次 Tab 键，段落周围不会出现虚点框。要证明元素仍然具有焦点，将页面向下滚动到底部，以使段落不可见(可能需要调整窗口的大小)。单击页面底部某个可获得焦点的元素，然后慢慢地按 Tab 键，直到 Address 域工具栏拥有焦点为止。再按一次 Tab 键，页面滚动并显示出段落，但在该元素周围没有虚点框。

相关主题：tabIndex 属性；srcollIntoView()方法。

id

值：字符串，(参见正文)

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

id 属性返回在 HTML 代码中赋给元素 id 特性的标识符。脚本不能修改现有元素的 ID，也不能把 ID 赋给没有 ID 的元素。但如果脚本创建一个新元素对象，就可通过 id 属性给该对象指定标识符。

示例

极少需要在脚本中访问此属性，除非是开发一个编写工具，来遍历页面上的所有元素，提取由作者分配的 ID。通过 document.getElementById(elemID)方法了解到对象的 id 属性后，可以获得该对象的引用，但如果由于某种原因脚本不知道元素(如文档第二段)的 ID，可以采用以下方式提取该 ID：

```
var elemID = document.getElementsByTagName("p")[1].id;
```

相关主题：className 属性。

innerHTML，innerText

值：字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

Internet Explorer 显示元素内容的一种方式是使用 innerHTML 和 innerText 属性(NN6+/Moz 只提供了 innerHTML 属性)。这些内部属性定义的内容包括位于元素首尾标记中的文档数据，但不包括标记本身(见 outerText 和 outerHTML 属性)。页面载入后，设置这些内部属性是修改一部分页面内容的常用方法。

innerHTML 属性包括页面上显示的元素的文本内容，还包括与该内容相关的所有 HTML 标记(如果内容中没有标记，就按原样显示文本)。例如，考虑下面的 HTML 源代码段：

```
<p id="paragraph1">"How <em>are</em> you?" he asked.</p>
```

段落对象的 innerHTML 属性值(document.getElementById("paragraphl").innerHTML)是：

```
"How <em>are</em> you?" he asked.
```

浏览器将包含在字符串(赋给元素的 innerHTML 属性)中的所有 HTML 标记都解释为标记。这也意味着把 HTML 内容赋给元素的 innerHTML 属性，可引入全新的嵌套元素(在现代术语中称为子节点)，文档的对象模型会对新插入的内容进行调整。

与此形成对照的是，innerText 属性只知道元素容器的文本内容。在上例中，段落的 innerText 属性值(document.getElementById.("paragraphl").innerText)为：

```
"How are you?" he asked.
```

如果把一个字符串赋给元素的 innerText 属性，且这个字符串包含 HTML 标记，标记及其尖括号将出现在显示页上，不解释为有效标记。

W3C DOM Level 3 增加了一个 textContent 属性，作为 innerText 的标准对等属性。IE 不支持 textContent。

不要修改 innerHTML 属性，来调整 frameset、html、head 或 title 对象的 HTML。应该通过 innerHTML 或用不同的表相关方法创建或删除行、列和单元格，来修改表结构(见配书光盘中的第 41 章)。然而，设置单元格的 innerHTML 或 innerText 属性，可安全地修改单元格的内容。

当所插入的 HTML 包含<script>标记时，要确保在首标记中包含 defer 特性。这对包含函数定义的脚本来说更重要，因为用户可能认为脚本是自动延迟的，实际上并非如此。

W3C DOM 不支持 innerHTML 属性，但所有现代浏览器都支持它。用户可能认为，纯粹的 W3C DOM 节点操作方法比只将 HTML 代码赋给 innerHTML 更有条理，但在日常的脚本编程中，进行单个属性赋值常常更方便。只要有可能，本书中的示例都使用 W3C 方法来更改节点的 HTML 代码，但在个别情况下，innerHTML 属性是非常简洁的选择。

示例

程序清单 26-11 包含的示例说明了如何使用 innerHTML 和 innerText 属性来动态更改页面的内容。该程序清单生成的页面包含一个 hl 元素标签和一段文本，以说明 innerHTML 和 innerText 属性的不同用途。两个文本框包含了相同的文本和 HTML 标记组合，来替换段落标签内部的内容。

如果将第一个文本框的默认内容应用于 labell 对象的 innerHTML 属性，第一个词就显示为斜体样式。另外，圆括号中的文本通过<span>标记指定的 small 样式表规则进行显示。但如果将同一内容应用于 label 对象的 innerText 属性，标记将按原样显示。

更多资源请访问稀酷客(www.ckook.com)

将此示例作为练习，在两个文本框中输入其他一些内容，看看在两个文本框的某些文本中插入
标记，情况会如何。

程序清单 26-11　使用 innerHTML 和 innerText 属性

HTML: jsb26-11.html

```
<!DOCTYPE html>
<html>
  <head>
     <meta http-equiv="content-type" content="text/html;charset=utf-8">
     <title>innerHTML and innerText Properties</title>
     <style type="text/css">
        h1
        {
           font-size:18pt; font-weight:bold;
           font-family:"Comic Sans MS", Arial, sans-serif;
        }
        .small
        {
           font-size:12pt; font-weight:400; color:gray;
        }
     </style>
     <script type="text/javascript" src="../jsb-global.js"></script>
     <script type="text/javascript" src="jsb26-11.js"></script>
  </head>
  <body>
     <form>
        <p>
           <input type="text" name="HTMLInput"
                  value="&lt;I&gt;First&lt;/I&gt; Article &lt;span↵
                     class='small'&gt;(of ten)&lt;/span&gt;"
                  size="50" />
           <input type="button" value="Change Heading HTML"
                  onclick="setGroupLabelAsHTML(this.form)" />
        </p>
        <p>
           <input type="text" name="textInput"
                  value="&lt;I&gt;First&lt;/I&gt; Article &lt;span↵
                     class='small'&gt;(of ten)&lt;/span&gt;"
                  size="50" />
           <input type="button" value="Change Heading Text"
                  onclick="setGroupLabelAsText(this.form)" />
        </p>
     </form>
     <h1 id="label1">
        ARTICLE I
     </h1>
     <p>Congress shall make no law respecting an establishment of religion, or
        prohibiting the free exercise thereof; or abridging the freedom of
```


更多资源请访问稀酷客(www.ckook.com)

```
            speech, or of the press; or the right of the people peaceably to
            assemble, and to petition the government for a redress of grievances.
        </p>
    </body>
</html>
```

JavaScript: jsb26-11.js

```
function setGroupLabelAsText(form)
{
    var content = form.textInput.value;
        if (content)
        {
        document.getElementById("label1").innerText = content;
        }
}
function setGroupLabelAsHTML(form)
{
    var content = form.HTMLInput.value;
        if (content)
        {
            document.getElementById("label1").innerHTML = content;
        }
}
```

相关主题： outerHTML, outerText、textContent 属性；replaceNode()方法。

isContentEditable

值： Boolean，　　只读

兼容性： WinIE5.5+，MacIE-，NN-，Moz-，Safari 1.2+，Opera+，Chrome+

isContentEditable 属性返回 Boolean 值，表示某个元素对象是否设置为可编辑的(见本章前面对 contentEditable 属性的讨论)。这个属性是很有帮助的，因为父元素的 contentEditable 属性设为 true，而嵌套元素的 contentEditable 属性很可能设为其默认值 inherit。但因为其父元素是可编辑的，所以嵌套元素的 isContentEditable 属性返回 true。

示例

使用 The Evaluator(参见第 4 章)在 myP 和嵌套的 myEM 元素上试验 contentEditable 和 isContentEditable 属性(重新载入页面，从已知的版本开始)。在顶部的文本框中输入以下语句，来检查 myEM 元素的当前设置：

```
myEM.isContentEditable
```

该值为 false，因为在元素包含层次结构的上方，还没有元素设置为可编辑状态。接着启用外部 myP 元素的可编辑功能：

```
myP.contentEditable = true
```

此时，整个 myP 元素都是可以编辑的，因为其子元素默认设置为继承父元素的编辑状态。

更多资源请访问稀酷客(www.ckook.com)

为证明这一点，在顶部文本框中输入以下语句：

```
myEM.isContentEditable
```

尽管 myEM 元素显示为可以编辑，但其 contentEditable 属性并未改变：

```
myEM.ContentEditable
```

此属性值仍然是默认的 inherit。

相关主题：contentEditable 属性。

isDisabled

值：Boolean，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

isDisabled 属性返回 Boolean 值，表明某个元素对象是否设置为不可用(见本章前面对 disabled 属性的讨论)。这个属性是很有帮助的，如果父元素的 disabled 属性设置为 true，尽管嵌套元素的 disabled 属性可能设置为默认值 false，嵌套元素的 isDisabled 属性也返回 true，因为它的父元素是不可用的。换句话说，isDisabled 属性返回元素实际的禁用状态，而不管它的 disabled 属性到底如何设置。

示例

使用 The Evaluator(参见第 4 章)在 myP 和嵌套的 myEM 元素上试验 disabled 和 isDisabled 属性(重新载入页面，从已知的版本开始)。在顶部的文本框中输入以下语句，检查 myEM 元素的当前设置：

```
myEM.isDisabled
```

该值为 false，因为在元素包含层次的上方，还没有元素设置为禁用状态。接着禁用外部的 myP 元素：

```
myP.disabled = true
```

此时，整个 myP 元素(包括其子元素)都是禁用的。为证明这一点，在顶部的文本框中输入以下语句：

```
myEM.isDisabled
```

尽管 myEM 元素显示为禁用，但其 disabled 属性并没有改变：

```
myEM.disabled
```

此属性值仍是默认的 false。

相关主题：disabled 属性。

isMultiLine

值：Boolean，　　只读

更多资源请访问稀酷客(www.ckook.com)

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

isMultiLine 属性返回 Boolean 值，指定元素对象能否占据或显示多行文本。重要的是，这个值不表示元素是否实际占据多行，而是表示该元素是否有可能占据多行。例如，文本 input 元素不能换行，因此它的 isMultiLine 属性为 false。然而，button 元素的标记可以显示多行文本，因此它的 isMultiLine 属性为 true。

示例

用 The Evaluator(参见第 4 章)读取该页面上元素的 isMultiLine 属性。在顶部的文本框中输入以下语句：

```
document.body.isMultiLine
document.forms[0].input.isMultiLine
myP.isMultiLine
myEM.isMultiLine
```

除文本域表单控件外，所有元素都能够占据多行。

isTextEdit

值：Boolean，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

isTextEdit 属性显示一个对象能否用其内容创建 WinIE 的 TextRange 对象(参见配书光盘上的第 33 章)。只有 Windows IE4+中的少数对象可创建 TextRange 对象，如 body、button、图像、单选类型的 input 和 textarea。在 MacIE 中，这个属性总是返回 false。

示例

良好的编程实践要求，脚本在调用任何对象的 createTextRange()方法之前检查此属性。典型的实现语句如下：

```
if (document.getElementById("myObject").isTextEdit)
{
   var myRange = document.getElementById("myObject").createTextRange();
   // [more statements that act on myRange]
}
```

相关主题：createRange()方法；TextRange 对象(参阅配书光盘上的第 33 章)。

lang

值：ISO 语言代码字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

在覆盖默认的浏览器语言系统时，lang 属性控制用来显示元素文本内容的书面语言系统。这个属性的默认值是一个空字符串，除非在元素的标记中为相应的 lang 特性赋值。用脚本控件修改该属性值，在当前的浏览器实现方案中没有任何效果。

更多资源请访问稀酷客(www.ckook.com)

示例

lang 属性的值由包含有效 ISO 语言代码的字符串组成。此类代码至少有一个主要语言代码(例如，法语是"fr")，再加上一个可选的区域指定符(例如，瑞士法语是"fr-ch")。如下代码将“瑞士德语”赋给元素：

```
document.getElementById("specialSpan").lang = "de-ch";
```

language

值：字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-

IE4+的体系结构允许多个脚本引擎与浏览器一起工作。基本的 Windows 版本浏览器包括两个引擎：JScript(和 JavaScript 兼容)和 Visual Basic Scripting Edition (VBScript)。默认的脚本引擎是 JScript。但如果希望在嵌入到标记的事件处理程序特性的语句中使用 VBScript 或其他脚本语言，则可通过标记的 language 特性，指导浏览器把希望的脚本引擎应用到脚本语句上。Language 特性提供了对该属性的脚本访问。除非想修改事件处理程序的 HTML 代码，并用 VBScript(或浏览器中安装的其他非 JScript 兼容语言)语句替换它，否则不必修改这个属性(甚至不必读取这个属性)。

该属性的有效值是 JScript、javascript、vbscript 和 vbs。对于该值，第三方脚本引擎有自己的标识符。Internet Explorer 5 也支持 language="xml"。注意这个特性在 HTML4.01 中已废弃，所以根据所使用的 DOCTYPE，使用这个特性的代码可能是无效的。使用 type 具有同样的效果(参见第 4 章)。

相关主题：script 元素对象。

lastChild

(参见 firstChild)

length

值：整型、 只读和读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

length 属性返回对象集合或数组中项的数目，它最常见的应用是作为 for 循环中的边界条件。虽然数组和集合通常使用整数索引(总是以 0 开始)，但 length 值是组中数据项的实际数目。因此，为了遍历组中的所有项，条件表达式应使用小于(<)号，而非小于等于(<=)号，如下：

```
for (var i = 0; i < someArray.length; i++)
{
  // [statements for the loop to process]
}
```

如果对数组执行递减操作(换句话说，从数组中的最后一项开始向前搜索)，初始表达式必须将计数变量初始化为 length-1：

```
for (var i = someArray.length - 1; i >= 0; i--)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
    // [statements for the loop to process]
}
```

对大多数的数组和集合，length 属性是只读的，只受组中数据项数的控制。但在浏览器的新版本中，可给一些对象数组(areas、options 和 select 对象)赋值，为数据赋值创建占位符，详见 area、select 和 option 元素对象的讨论。也可用脚本修改普通 JavaScript 数组的 length 属性值，从数组的末尾裁减相应的项，或为以后的赋值保留空位。Array 对象的更多内容请参见第 18 章。

示例

可在 The Evaluator 的顶部文本框中输入以下语句序列，查看 length 属性的返回值(以及为某些对象设置该属性)。注意，某些语句只在部分浏览器版本中有效。

```
(All browsers) document.forms.length
(All browsers) document.forms[0].elements.length
(All browsers)   document.images.length
(NN4+)         document.layers.length
(All browsers)          document.all.length
(IE5+, W3C)    document.getElementById("myTable").childNodes.length
```

给出所有这些语句主要是为了完整性。除非有充分的理由支持旧浏览器，否则应该使用最后一种技术(IE5+, W3C)来访问 length 属性。

相关主题：area、select、option 和 Array 对象。

localName，namespaceURI，prefix

值：字符串，　　只读

兼容性：WinIE8+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

这三个属性(localName、namespaceURI 和 prefix)可应用于 XML 文档中把命名空间 URI 和一个 XML 标记关联起来的任何节点。虽然 NN6 为所有元素(和节点)对象提供了这三个属性，但这些属性不返回期望的值。然而，基于 Mozilla 的浏览器，包括 NN7+，解决了这个问题。为了更好地理解这三个属性的值，考虑下列 XML 内容：

```
<x xmlns:bk='http://bigbooks.org/schema'>
    <bk:title>To Kill a Mockingbird</bk:title>
</x>
```

标记为<bk:title>的元素和为块定义的命名空间 URI 关联起来，该元素的 namespaceURI 属性返回字符串 http://bigbooks.org/schema。标记名包括前缀(冒号之前)和局部名(冒号之后)。在上面的例子中，对于<bk:title>标记定义的元素，其 prefix 属性是 bk，而 localName 属性返回 title。任何节点的 localName 属性值和它的 nodeName 属性值相同，例如文本节点的#text。

XML 命名空间的更多内容请查看 http://www.w3.org/TR/REC-xml-names。

相关主题：scopeName，tagUrn 属性。

更多资源请访问稀酷客(www.ckook.com)

nextSibling，previousSibling

值：对象引用， 只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

兄弟节点是 HTML 文档中和另一个节点在相同嵌套层上的节点。例如，下面的 p 元素有两个子节点(em 和 span 元素)，这两个子节点就是兄弟元素。

```
<p>MegaCorp is <em>the</em> source of the
    <span class="hot">hottest</span> gizmos.</p>
```

兄弟元素的顺序仅由元素的源代码顺序决定。因此，在前面的例子中，em 节点没有 previousSibling 属性，span 元素也没有 nextSibling 属性(意味着这些属性返回 null)。这些属性提供了另一种遍历同一层上所有元素的方法。

示例

下面的函数将同一类名赋给元素的所有子节点：

```
function setAllChildClasses(parentElem, className)
{
    var childElem = parentElem.firstChild;
        while (childElem.nextSibling)
        {
            childElem.className = className;
            childElem = childElem.nextSibling;
        }
}
```

当然，此示例并不是完成这个任务的唯一方式。使用 for 循环来遍历父元素的 childNodes 集合也是有效的。

相关主题：firstChild、lastChild、childNodes 属性；hasChildNodes()、insertAdjacentElement()方法。

nodeName

值：字符串， 只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

对于 HTML 和 XML 元素，节点名和标记名相同。nodeName 属性的作用是与正式 W3C DOM 标准指定的节点体系结构保持一致。这个值就像 tagName 属性一样，是标记名的全大写字符串(即使 HTML 源代码使用小写标记)。

一些节点没有标记，例如元素的文本内容，此类节点的 nodeName 属性是一个特殊的值：#text。另一类节点是元素的特性，nodeName 是该特性的名字。Node 对象属性的更多内容请参见第 25 章。

示例

下面的函数展示了将新的类名赋予 IE5+文档中每个 p 元素的一种不大高效的方式：

更多资源请访问稀酷客(www.ckook.com)

```
function setAllPClasses(className)
{
   for (var i = 0; i < document.all.length; i++)
   {
      if (document.all[i].nodeName == "P")
      {
         document.all[i].className = className;
      }
   }
}
```

更高效的方式是使用 getElementsByTagName()方法获得所有 p 元素的集合，然后直接遍历它们。

相关主题： tagName 属性。

nodeType

值： 整型，　　　　只读

兼容性： WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C DOM 规范指定了一系列表示节点类型的常数值。每个节点都有一个值来标识其类型，但并非所有的浏览器都在所有节点类型上将 nodeType 属性看成对象。表 26-4 列出了 nodeType 值，所有这些值都是 W3C DOM Level 2 规范的组成部分。

表 26-4　nodeType 属性值

值	说　明
1	元素节点
2	特性节点
3	文本(#text)节点
4	CDATA 段节点
5	实体引用节点
6	实体节点
7	处理指令节点
8	注释节点
9	文档节点
10	文档类型节点
11	文档片断节点
12	符号节点

nodeType 值自动赋给一个节点，无论该节点是在文档 HTML 的源代码中，还是通过脚本动态生成的。如果脚本用某种方法创建新元素节点，例如，把嵌入 HTML 标记的字符串赋给 innerHTML 属性或明确调用 document.createElement()方法，新元素就假定 nodeType 为 1。

基于 Mozilla 浏览器和 Safari 为每个 nodeType 值提供了一套 Node 对象常量，在支持 W3C

更多资源请访问稀酷客(www.ckook.com)

DOM 规范方面又迈出了一步。表 26-5 列出了 DOM Level 2 规范定义的整个集合，用 nodeType 整数替代这些常量可提高脚本的可读性。例如，最好不要使用：

```
if (myElem.nodeType == 1)
{
   // [statements to process if true]
}
```

这样更简单：

```
if (myElem.nodeType == Node.ELEMENT_NODE)
{
   // [statements to process if true]
}
```

表 26-5　W3C DOM 的 nodeType 常量

引　　用	nodeType 值
Node.ELEMENT_NODE	1
Node.ATTRIBUTE_NODE	2
Node.TEXT_NODE	3
Node.CDATA_SECTION_NODE	4
Node.ENTITY_REFERENCE_NODE	5
Node.ENTITY_NODE	6
Node.PROCESSING_INSTRUCTION_NODE	7
Node.COMMENT_NODE	8
Node.DOCUMENT_NODE	9
Node.DOCUMENT_TYPE_NODE	10
Node.DOCUMENT_FRAGMENT_NODE	11
Node.NOTATION_NODE	12

示例

可在 The Evaluator 中查看 nodeType 属性值。ID 为 myP 的 p 元素是个合适的开端，p 元素本身的 nodeType 为 1：

```
document.getElementById("myP").nodeType
```

此元素有三个子节点：一个文本字符串(nodeName #text)、一个 em 元素(nodeName em)和元素内容的其余文本(nodeName #text)。两个文本部分的 nodeType 值是 3：

```
document.getElementById("myP").childNodes[0].nodeType
```

相关主题：nodeName 属性。

更多资源请访问稀酷客(www.ckook.com)

nodeValue

值：数值、字符串或空， 读/写

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

文本节点的 nodeValue 属性包括该节点的实际文本。这样的节点不能包含任何嵌套的元素，因此 nodeValue 属性提供了另一种方法来读取和修改 Internet Explorer 元素的 innerText 属性。但在 W3C DOM 中，只有引用元素的子文本节点，才能获得或设置其节点值。

在支持 W3C DOM 的浏览器实现的节点类型中，只有文本和特性类型有可读值，元素节点的 nodeValue 属性返回 null 值。特性节点的 nodeValue 属性由分配给该特性的值组成。根据 W3C DOM 标准，特性值应该显示为字符串，但 WinIE5 在该特性值全部为数字字符时返回 Number 类型的值。即使将数字的字符串版本赋给这类 nodeValue 属性，它也在内部转换为 Number 类型。其他浏览器总是将 nodeValue 值返回为字符串(并将数值转换为字符串)。

示例

可以使用 nodeValue 属性完成实际任务。例如，可使用 nodeValue 将 textarea 对象的宽度增加 10%。在执行数学运算和重新赋值之前，把 textarea 转换为整数：

```
function widenCols(textareaElem)
{
   var colWidth = parseInt(textareaElem.attributes["cols"].nodeValue, 10);
   textareaElem.attributes["cols"].nodeValue = (colWidth * 1.1);
}
```

另外，如果元素不包含额外的嵌套元素，还可以用该属性替换元素的文本：

```
function replaceText(elem, newText)
{
   if (elem.childNodes.length == 1 && elem.firstChild.nodeType == 3)
   {
      elem.firstChild.nodeValue = newText;
   }
}
```

函数添加了一个最终验证条件：元素只包含一个文本类型的子节点。在 IE 中可以使用 innerText 属性替代第二个例子中的赋值语句，得到相同的结果：

```
elem.innerText = newText;
```

也可在除 IE 外的浏览器中使用 textContent 属性，获得同样简洁的效果：

```
elem.textContent = newText;
```

相关主题：attributes、innerText、nodeType 属性。

offsetHeight，offsetWidth

值：整型， 只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

这些属性表面上报告元素的高度和宽度，由于 Microsoft 和 W3C 对 CSS 框模型的解释是有冲突的，因此这些属性有着曲折的发展历史。这两个属性都是由 Microsoft 为 IE4 创造的，尽管它们当前不是 W3C 标准的组成部分(它们包含在 CSSOM 视图模型的工作草案中)，但由于其对脚本开发人员非常有价值，其他现代浏览器都实现了这些属性，包括基于 Mozilla 的浏览器、基于 WebKit 的浏览器和基于 Presto 的浏览器。

假设要为内联(非定位)元素的宽度或高度指定样式表规则，则根据页面是否(通过 DOCTYPE)将浏览器置于标准兼容模式，offsetHeight 和 offsetWidth 属性的行为会有所不同。更确切地讲，当 IE6+(通过 DOCTYPE 属性，如第 25 章所述)设置为标准兼容模式时，这些属性会计算元素内容再加上填充符或边框的像素尺寸，不包括页边距，这也遵循 W3C 框模型的 Mozilla 和 WebKit 采用的默认方式。但在怪癖模式下，IE6+默认只返回元素内容的高度和宽度，不考虑填充符、边框或页边距。对 IE6 之前的 IE 版本，这是唯一的方式。

注意对于没有指定高度和宽度的常规块级元素，在所有文本都显示出来后，offsetHeight 是由内容的实际高度确定的。但 offsetWidth 总扩展其包含元素的最大宽度，因此，offsetWidth 属性不表示文本内容的显示宽度，它比父元素的总宽度小。例如，仅由几个词组成的 p 元素可能报告其 offsetWidth 有数百个像素，因为段落块的宽度等于包含 p 元素的父元素 body 的总宽。为了确定全宽块级元素中文本的实际宽度，可将文本放在一个内联元素中(如 span)，并检查 span 的 offsetWidth 属性。

示例

在 IE4+中，可用 offsetHeight 和 offsetWidth 属性代替程序清单 26-6 中的 clientHeight 和 clientWidth。因为这两个元素将其宽度硬连接到样式表中，这样，offsetWidth 属性遵循这一设置，而不是取其父元素(BODY)的默认宽度。

对于 IE5+和 W3C 浏览器，可用 The Evaluator 查看页面上各种对象的 offsetHeight 和 offsetWidth 属性值。在顶部文本框中输入以下语句：

```
document.getElementById("myP").offsetWidth
document.getElementById("myEM").offsetWidth
document.getElementById("myP").offsetHeight
document.getElementById("myTable").offsetWidth
```

相关主题：clientHeight，clientWidth 属性。

offsetLeft，offsetTop

值：整型，　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

当边框、页边距和填充符与元素相关，还要考虑 DOCTYPE 选项时，offsetLeft 和 offsetTop 属性与 offsetHeight 和 offsetWidth 属性一样，会由于版本冲突带来一些问题。然而，在元素没有明确定位的情况下，offsetLeft 和 offsetTop 属性可以在父元素的定位上下文内提供元素的像素坐标。

更多资源请访问稀酷客(www.ckook.com)

注意：

MacIE 中已定位元素的 offsetLeft 和 offsetTop 属性不会返回与 style.left 和 style.top 属性相同的值。

offsetParent 属性返回的元素可作为这些属性的坐标上下文。这意味着要确定元素的精确位置，必须添加一些代码，从 offsetParent 层次开始遍历，直到属性返回 null 为止。

与 offsetHeight 和 offsetWidth 属性一样，offsetLeft 和 offsetTop 属性不是 W3C DOM 规范的一部分，但大多数浏览器都实现了这些属性，因为它们很便于完成一些支持脚本的动态 HTML 任务。脚本通过这两个属性，可读出块级或内置元素的像素坐标。该坐标是相对于 body 元素的，但将来可能会改变。详见本章后面关于 offsetParent 属性的讨论。

示例

下面的 IE 脚本语句使用 4 个 offset 维度属性来度量和定位一个 div 元素，使它完全覆盖 p 元素中的 span 元素。这可用于一个填空测验，该测验在页面的其他位置提供了文本输入域。当用户给出正确答案时，div 元素 blocker 就隐藏起来，以显示正确答案。

```
document.all.blocker.style.pixelLeft = document.all.span2.offsetLeft
document.all.blocker.style.pixelTop = document.all.span2.offsetTop
document.all.blockImg.height = document.all.span2.offsetHeight
document.all.blockImg.width = document.all.span2.offsetWidth
```

由于 span 元素的 offsetParent 属性是 body 元素，因此已定位的 div 元素可使用相同的定位上下文(这是默认上下文)来设置 pixelLeft 和 pixelTop 样式属性(注意，定位属性属于元素的 style 对象)。offsetHeight 和 offsetWidth 属性可以读取 span 元素的尺寸(本示例没有考虑边框、页边距或填充符)，并将它们作为 div 元素 blocker 包含的图像的尺寸。

在某些实现方案中，此示例有点危险。如果 span2 的文本换行了，新的 offsetHeight 值应有足够的空间来容纳这两行。但 blockImg 和 blockerdiv 元素都是显示为简单矩形的块级元素。换句话说，blocker 元素不会变成两个单独的块，来覆盖分布在两行上的 span2 块。

相关主题：clientLeft，clientTop，offsetParent 属性。

offsetParent

值：对象引用，　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

offsetParent 属性返回对象的引用，用作当前元素的定位上下文。OffsetLeft 和 offsetTop 属性的值相对于 offsetParent 对象的左上角进行测量。

返回的对象通常是(但并不总是)最近的外部块级容器。对于大多数文档元素而言，offsetParent 对象是 document.body 对象(在一些浏览器中，一些元素有不同的 offsetParent 元素)。

例如，表单元格在不同的浏览器中有不同的 offsetParent 元素，如表 26-6 所示：

更多资源请访问稀酷客(www.ckook.com)

表 26-6　不同浏览器中不同的 offsetParent 元素

浏览器	td offsetParent
WinIE4	tr
WinIE5+/NN7+/基于 Moz/基于 WebKit/基于 Presto	table
MacIE	table
NN6	body

幸好，在大多数现代浏览器中，定位元素的 offsetParent 属性都是可以预测的。例如，一级定位元素的 offsetParent 元素是 body，嵌套定位元素的 offsetParent(例如，另一个元素中一个绝对定位的 div)是下一层外部容器(换句话说，内部元素的定位上下文)。

示例

可以使用 offsetParent 属性来帮助定位页面上的嵌套元素。程序清单 26-12 演示了脚本如何向上访问 Windows IE 中的 offsetParent 对象层次结构，计算出页面上嵌套元素的位置。程序清单 26-12 的目标是将图像定位到第二个表单元格的左上角，整个表格在页面上居中显示。

onload 事件处理程序调用了 setImagePosition()函数。该函数首先设置一个布尔标志，来决定计算操作是基于客户端属性集还是基于偏移属性集。WinIE4 和 MacIE5 使用客户端属性，而 WinIE5+使用偏移属性，while 循环平衡了这种差异。此循环从单元格的 offsetParent 开始，向外遍历 offsetParent 层次结构，直到 document.body 对象。在 IE5+和其他浏览器中，while 循环只执行 3 次(分别应用于 td、table 和 body 元素)，因为在单元格和 body 之间只有 table 元素；在 IE4 中，循环执行 4 次，以处理层次结构中的 tr 和 table 元素。循环在 offsetParent 是 null 时停止。最后，向左测量和向上测量的累计值应用于 div 对象样式的定位属性，把图像显示出来。

程序清单 26-12　使用 offsetParent 属性

HTML: jsb26-12.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>offsetParent Property</title>
      <style type="text/css">
         #myImg
         {
            position:absolute; height:12px; width:12px;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-12.js"></script>
   </head>
   <body onload="setImagePosition()">
      <h1>The offsetParent Property</h1>
      <hr />
      <p>After the document loads, the script positions a small image in the
```


更多资源请访问稀酷客(www.ckook.com)

```
            upper left corner of the second table cell.
        </p>
        <table border="1" align="center">
            <tr>
                <td>This is the first cell</td>
                <td id="myCell">This is the second cell.</td>
            </tr>
        </table>
        <img id="myImg" alt="image" src="end.gif" height="12" width="12"
            style="visibility:hidden;" />
    </body>
</html>
```

JavaScript: jsb26-12.js

```
function setImagePosition()
{
    var x = 0;
    var y = 0;
    var offsetPointer = document.getElementById("myCell"); // cElement;
        while (offsetPointer)
        {
            x += offsetPointer.offsetLeft;
            y += offsetPointer.offsetTop;
            offsetPointer = offsetPointer.offsetParent;
        }
    // correct for MacIE body margin factors
    if (navigator.userAgent.indexOf("Mac") != -1 &&
        typeof document.body.leftMargin != "undefined")
    {
        x += document.body.leftMargin;
        y += document.body.topMargin;
    }
    document.getElementById("myImg").style.left = x + "px";
    document.getElementById("myImg").style.top = y + "px";
    document.getElementById("myImg").style.visibility = "visible";
}
```

相关主题：offsetLeft，offsetTop，offsetHeight 和 offsetWidth 属性。

outerHTML，outerText

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari 1.3+，Opera+，Chrome+

Internet Explorer、基于 WebKit 的浏览器和基于 Presto 的浏览器对整个元素进行脚本编程的一种方式是使用 outerHTML 和 outerText 属性。这两个属性的主要区别在于：outerHTML 包括元素的首尾标记，而 outerText 只包括元素中的显示文本(包括来自嵌套元素的文本)。

outerHTML 属性不仅包含元素在页面上可见的文本内容，而且包括与内容相关的每个 HTML 标记。例如，考虑下列 HTML 源代码段：

更多资源请访问稀酷客(www.ckook.com)

```
<p id="paragraph1">"How <em>are</em> you?" he asked.</p>
```

p 对象的 outerHTML 属性值(document.all.paragraph1.outerHTML)和源代码中的完全相同。

浏览器将解释元素 outerHTML 属性字符串中的 HTML 标记。所以可以用此属性删除(将属性设置为空字符串)或替换整个标记，文档的对象模型随之调整，以适应对 HTML 进行的调整。

相反，outerText 属性只知道元素容器的文本内容。在前面的例子中，段落的 outerText 属性值(document.all.paragraph1.innerText)为：

```
"How are you?" he asked.
```

这看起来很熟悉，因为在多数情况下，现有对象的 innerText 和 outerText 属性返回相同的字符串。

示例

程序清单 26-13 演示了如何使用 outerHTML 和 outerText 属性，来访问和动态修改网页的内容，所生成的页面(仅适用于 WinIE4+/基于 WebKit 或 Presto 的浏览器)包含一个 h1 元素标签和一段文本，以说明演示 outerHTML 和 outerText 在不同用途。两个文本框包含的文本和 HTML 标记可替代创建段落标签的元素。

如果将第一个文本框的默认内容应用于 labell 对象的 outerHTML 属性，h1 元素就被一个 span 元素替代，该 span 的 class 特性具备在文档前面定义的另一个样式表规则。注意，新 span 元素的 ID 与原 h1 元素的 ID 相同，这允许第二个按钮的脚本寻址对象，但第二段脚本用纯文本(包括标记)替换了元素。元素没有了，任何试图更改 labell 对象的 outerHTML 或 outerText 属性的操作都将引发错误，因为文档中不再有 labell 对象。

使用此示例在两个文本框中通过其他内容进行试验。

程序清单 26-13 使用 outerHTML 和 outerText 属性

HTML: jsb26-13.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>outerHTML and outerText Properties</title>
      <style type="text/css">
         h1
         {
            font-size:18pt; font-weight:bold;
            font-family:"Comic Sans MS", Arial, sans-serif;
         }
         .heading
         {
            font-size:20pt; font-weight:bold;
            font-family:"Arial Black", Arial, sans-serif;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <script type="text/javascript" src="jsb26-13.js"></script>
    </head>
    <body>
        <form>
            <p id="myP1">
                <input type="text" name="HTMLInput"
                       value="&lt;span id='label1' class='heading'&gt;i
                            Article the First&lt;/span&gt;"
                       size="55" />
                <input type="button" value="Change Heading HTML"
                       onclick="setGroupLabelAsHTML(this.form)" />
            </p>
            <p id="myP2">
                 <input type="text" name="textInput"
                        value="&lt;span id='label1' class='heading'&gt;i
                             Article the First&lt;/span&gt;"
                        size="55" />
                 <input type="button" value="Change Heading Text"
                        onclick="setGroupLabelAsText(this.form)" />
            </p>
        </form>
        <h1 id="label1">ARTICLE I</h1>
        <p>Congress shall make no law respecting an establishment of religion, or
            prohibiting the free exercise thereof; or abridging the freedom of
            speech, or of the press; or the right of the people peaceably to
            assemble, and to petition the government for a redress of grievances.
        </p>
    </body>
</html>
```

JavaScript: jsb26-13.js

```
function setGroupLabelAsText(form)
{
    var content = form.textInput.value;
        if (content)
        {
            document.getElementById("label1").outerText = content;
        }
}
function setGroupLabelAsHTML(form)
{
    var content = form.HTMLInput.value;
        if (content)
        {
          document.getElementById("label1").outerHTML = content;
        }
}
```

相关主题： innerHTML、innerText 属性；replaceNode()方法。

更多资源请访问稀酷客(www.ckook.com)

ownerDocument

值：Document 对象引用，　　只读

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C DOM 中的任何元素或节点都有 ownerDocument 属性，其值为最终包含这个元素或节点的文档的引用。如果脚本遇到一个元素或节点的引用(也许它作为参数传递给函数)，该对象的 ownerDocument 属性就提供一种方式，来建立相同文档中其他对象的引用，或者访问 document 对象的属性和方法。该属性的 IE 版本是 document。

示例

使用 The Evaluator(参见第 4 章)来研究 ownerDocument 属性。在顶部文本框中输入如下语句：

```
document.body.childNodes[5].ownerDocument
```

结果是对 document 对象的引用，可以用它来查看文档的属性，如下所示。应该在顶部文本框中输入该语句：

```
document.body.childNodes[5].ownerDocument.URL
```

这将返回拥有子节点的文档的 document.URL 属性。

相关主题：document 对象。

parentElement

值：元素对象引用或 null，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari 1.2+，Opera+，Chrome+

parentElement 属性从当前元素返回次外层 HTML 元素的引用。元素的这个父子关系通常(但不总是)和节点的父子关系相同(参见本章后面的 parentNode 属性)，差别是 parentElement 属性只处理作为 document 对象的 HTML 元素，但节点未必是 HTML 元素(例如，节点可以是一个特性或文本块)。

parentElement 和 offsetParent 属性也有区别，后者返回一个元素，它可能是与给定元素相隔几代的元素，但它是定位上下文最直接的父元素。例如，td 元素的 parentElement 属性可能是 tr 容器元素，但 td 元素的 offsetParent 属性却是 table 元素(除非使用 WinIE4)。

通过 parentElement 属性，脚本可从一个元素沿元素层次结构向外移动。父元素链的顶部是 html 元素，它的 parentElement 属性返回 null。

示例

可在 The Evaluator 中试验 parentElement 属性。假定文档包含一个名为 myP 的 p 元素，在顶部的表达式求值文本框中输入表 26-7 左列中的各语句，按 Enter 键查看结果。

更多资源请访问稀酷客(www.ckook.com)

表 26-7　输入的语句

表达式	IE	WebKit/Presto
document.getElementById("myP").tagName	p	P
document.getElementById("myP").parentElement	[object]	Object HTMLHtmlElement
document.getElementById("myP").parentElement.tagName	body	body
document.getElementById("myP").parentElement.parentElement	[object]	Object HTMLBodyElement
document.getElementById("myP").parentElement.parentElement. tagName	html	html
document.getElementById("myP").parentElement.parentElement. parentElement	null	null

相关主题：offsetParent，parentNode 属性。

parentNode

值：Node 对象引用或 null，　　只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

parentNode 属性返回次外层节点的引用，该节点是文档中的一个对象。对于标准元素对象，parentNode 属性和 IE/WebKit/Presto 浏览器的 parentElement 一样，因为这两个对象都有直接的父子节点关系和父子元素关系。

然而，其他类型的内容可以是节点，这包括元素内的文本段。文本段的 parentNode 属性是次外层节点或包围这个片段的元素。IE/Opera 中的文本节点对象没有 parentElement 属性。

示例

使用 The Evaluator 查看元素和非元素节点的 parentNode 属性值。首先输入以下两条语句，观察每条语句的结果：

```
document.getElementById("myP").parentNode.tagName
document.getElementById("myP").parentElement.tagName (IE/WebKit/Presto browsers)
```

现在，在 myP 段落元素中查看第一个文本片断节点的属性：

```
document.getElementById("myP").childNodes[0].nodeValue
document.getElementById("myP").childNodes[0].parentNode.tagName
document.getElementById("myP").childNodes[0].parentElement (WebKit only)
```

注意在 IE 和 Opera 中，文本节点没有 parentElement 属性。

相关主题：childNodes、nodeName、nodeType、nodeValue 和 parentElement 属性。

parentTextEdit

值：元素对象引用或 null，　　只读

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 对象模型中只有少数对象能创建文本范围(见配书光盘中第 33 章的 TextRange 对象)。使用 parentTextEdit 属性可以找到能够生成文本范围的次外层对象容器。如果一个元素在层次结构中，则该属性返回该元素的对象引用，否则(例如 document.body.parentTextEdit)该值为 null。MacIE 总是返回 null 值，因为该浏览器不支持 TextRange 对象。

示例

程序清单 26-14 包含的示例演示了如何使用 parentTextEdit 属性来创建文本范围，其生成的页面包含了一段拉丁文本和三个单选按钮，用于选择段落块大小：一个字符、一个词或一个句子。如果单击大段落中的任何位置，onclick 事件处理程序将调用 selectChunk()函数。该函数首先检查选择了哪个单选按钮，以确定在用户单击位置周围突出显示(选择)多少段落内容。

脚本使用 parentTextEdit 属性测试被单击的元素是否有能够创建文本范围的有效父元素之后，再次调用此属性来帮助创建文本范围。之后，TextRange 对象方法将范围缩小为一个插入点，将该插入点移到离单击时鼠标指针位置最近的地方，扩充选择区，以包含所需的内容块，并选择该块文本。

注意对 TextRange 对象的 expand()方法异常的一个变通处理：如果指定一个句子，IE 不会自动将 p 元素的起点作为句子的起点，而是将一个伪装的(白色文本)句点添加到前一个元素的结尾，以强制 TextRange 对象只扩充到目标 p 元素中第一个句子的开头。

程序清单 26-14　使用 parentTextEdit 属性

HTML: jsb26-14.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>parentTextEdit Property</title>
      <style type="text/css">
         p
         {
            cursor:hand
         }
         #placeHolder
         {
            color:white;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-14.js"></script>
   </head>
   <body bgcolor="white">
      <form>
         <p>Choose how much of the paragraph is to be selected when you click
            anywhere in it:
            <br />
```


更多资源请访问稀酷客(www.ckook.com)

```
                <input type="radio" name="chunk" value="character"
                        checked="checked" />Character
                <input type="radio" name="chunk"
                        value="word" />Word
                <input type="radio" name="chunk"
                        value="sentence" />Sentence
                <span id="placeHolder">.</span>
            </p>
        </form>
        <p onclick="selectChunk()">Lorem ipsum dolor sit amet, consectetaur
          elit, sed do eiusmod tempor incididunt ut labore et dolore magnaaliqua.
          Ut enim adminim veniam, quis nostrud exercitation ullamco laboris nisi
          ut aliquip ex ea commodo consequat. Duis aute irure dolor in
          reprehenderit involuptate velit esse cillum dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </p>
    </body>
</html>
```

JavaScript: jsb26-14.js

```
function selectChunk()
{
    var chunk, range;
    for (var i = 0; i < document.forms[0].chunk.length; i++)
    {
        if (document.forms[0].chunk[i].checked)
        {
            chunk = document.forms[0].chunk[i].value;
            break;
        }
    }
    var x = window.event.clientX;
    var y = window.event.clientY;
    if (window.event.srcElement.parentTextEdit)
    {
        range = window.event.srcElement.parentTextEdit.createTextRange();
        range.collapse();
        range.moveToPoint(x, y);
        range.expand(chunk);
        range.select();
    }
}
```

相关主题：isTextEdit 属性；TextRange 对象(参见配书光盘上的第 33 章)。

prefix

(参见 localName)

更多资源请访问稀酷客(www.ckook.com)

previousSibling

(参见 nextSibling)

readyState

值：字符串(对于 OBJECT 对象，其值是整型)，　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

在其他语句操作对象或其数据之前，脚本可在 IE 中检验元素是否载入了所有附属数据(例如，外部图像文件或其他媒体文件)。readyState 属性可显示元素的载入状态。

表 26-8 列出了 readyState 的值及其含义。

表 26-8　readyState 属性值

HTML 值	OBJECT 值	说　明
Complete	4	元素和数据已经完全载入
Interactive	3	数据可能未完全载入，但用户可与元素交互
Loaded	2	数据载入，但对象可能正在启动
Loading	1	数据正在载入
Uninitialized	0	对象尚未开始载入数据

大多数 HTML 元素的这个属性总是返回 complete。其他大部分状态由 img、embed 和 object 等元素使用，它们载入外部数据，并启动其他处理过程(例如 ActiveX 控件)。

注意，readyState 属性不会显示对象是否存在于文档中(例如 uninitialized)。如果对象不存在，就不可能有 readyState 属性，对于未定义的对象，其结果是一个脚本错误。如果想在每个元素及其数据完全载入后运行脚本，则应使用 body 元素的 onLoad 事件处理程序或对象的 onreadystatechange 事件处理程序来触发函数(并检查 readyState 属性是否是 complete)。

示例

要看到标准 HTML 的 readyState 属性是 complete 之外的值，可在紧随<img>标记的脚本中查看该属性：

```
...
<img id="myImg" src="someImage.gif">
<script type="text/javaScript">
   alert(document.getElementById("myImg").readyState);
</script>
...
```

将此段代码放入可以通过慢速网络访问的文档。如果图像不在浏览器的缓存中，readyState 属性的值可能是 uninitialized 或 loading。前者意味着 img 对象存在，但尚未开始从服务器上接收图像数据。如果重新载入页面，图像很可能立即从缓存中载入，readyState 的属性值就是 complete。

相关主题：onreadystatechange 事件处理程序。

更多资源请访问稀酷客(www.ckook.com)

recordNumber

值：整型或 null，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

实际上，每个对象都有 recordNumber 属性，但它只应用于 Internet Explorer 数据绑定中的某些元素，以表示重复的数据。例如，如果在表格中显示来自外部数据存储的 30 条记录，表中的 tr 元素在 HTML 中就只出现一次，然而，浏览器会重复表行(和它的组成单元格)，来容纳所有 30 行数据。若单击了其中一行，则可使用 tr 对象的 recordNumber 属性来查看单击了哪个记录。这个工具通常应用在允许更新记录的数据绑定中。例如，可编写一个表，单击一行不可编辑的数据，就会使该记录的数据显示在页面上其他地方的可编辑文本框中。如果一个对象不与数据源绑定，或它是与数据源绑定的数据的非重复对象，recordNumber 属性就是 null。

示例

程序清单 26-15 显示了如何使用 recordNumber 属性导航到数据序列中的特定记录。数据源是一个用 Tab 分隔的小文件，包含 20 条奥斯卡奖数据记录。这样，显示字段子集的表格绑定到数据源对象上，嵌入到页面顶部附近段落中的三个 span 对象也绑定到数据源对象上。当用户单击一行数据时，被单击记录中的三个字段就放入绑定的 span 对象。

此页面的脚本部分只有一条语句。当用户触发重复的 tr 对象的 onclick 事件处理程序时，该函数把对 tr 对象的引用作为参数。数据存储对象包含 recordset 对象中数据的一份内部副本，此 recordset 对象的一个属性是 AbsolutePosition，它是数据对象指向的当前记录的整数值(它一次只能指向一行，默认是第一行)。脚本语句将 recordset 对象的 AbsolutePosition 属性设置为用户单击的行的 recordNumber 属性。由于三个 span 元素绑定到同一个数据源，因此它们会立即更新，以反映对数据对象的当前记录内部指针的改变。还要注意，第三个 span 对象绑定到一个未在表格中显示的数据源字段。由于数据源对象包含整个数据源内容，因此用户可以访问记录的任何字段。但实际上，如果最佳男演员所演电影的名称信息没有显示在表中，而网站访问者看到了这个信息，就会令人不安(一些访问者甚至可能开始发出抱怨：“出错了”)。

程序清单 26-15　使用数据绑定的 recordNumber 属性

HTML：jsb26-15.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Data Binding (recordNumber)</title>
      <style type="text/css">
         #instructions
         {
            fon-weight:bold;
         }
         .filmTitle
         {
            font-style:italic;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
        #columnHeaders
        {
            background-color:yellow; text-align:center;
        }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-15.js"></script>
  </head>
  <body>
   <p><span id="instructions">Academy Awards 1978-1997</span> (Click on a table
        row to extract data from one record into the following paragraph.)
   </p>
   <p>The award for Best Actor of <span datasrc="#oscars" datafld="Year"></span>
       went to <span datasrc="#oscars" datafld="Best Actor"></span>
       for his outstanding achievement in the film
     <span class="filmTitle" datasrc="#oscars" datafld="Best Actor Film"></span>.
   </p>
   <table border="1" datasrc="#oscars" align="center">
      <thead id="columnHeaders">
          <tr>
              <td>Year</td>
              <td>Film</td>
              <td>Director</td>
              <td>Actress</td>
              <td>Actor</td>
          </tr>
      </thead>
      <tr id="repeatableRow" onclick="setRecNum(this)">
          <td><div id="col1" datafld="Year"></div></td>
          <td><div class="filmTitle" id="col2" datafld="Best Picture"></div></td>
          <td><div id="col3" datafld="Best Director"></div></td>
          <td><div id="col4" datafld="Best Actress"></div></td>
          <td><div id="col5" datafld="Best Actor"></div></td>
      </tr>
   </table>
   <object id="oscars" classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
      <param name="DataURL" value="Academy Awards.txt" />
      <param name="UseHeader" value="True" />
      <param name="FieldDelim" value="&#09;" />
   </object>
  </body>
</html>
```

JavaScript: jsb26-15.js

```
// set recordset pointer to the record clicked on in the table.
function setRecNum(row)
{
   document.oscars.recordset.AbsolutePosition = row.recordNumber;
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：dataFld、dataSrc 属性；table、tr 对象(配书光盘中的第 41 章)。

runtimeStyle

值：style 对象，　　只读

兼容性：WinIE5+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

利用 runtimeStyle 属性，可确定浏览器的样式表特性的默认设置。这个属性返回一个 style 对象，它包含所有的样式特性和页面载入时的默认设置。该属性不反映通过文档中的样式表或脚本赋给元素的值。这个属性返回的默认值和 currentStyle 属性返回的值不同，currentStyle 属性包括与值相关的数据，这些值并非明确由样式表赋予，而受浏览器显示引擎的默认行为的影响；而 runtimeStyle 属性将未赋值的样式值显示为空白或 0。

示例

要更改样式属性设置，可以通过元素的 style 对象访问属性。使用 The Evaluator(参见第 4 章)比较一个元素的 runtimeStyle 和 style 对象的属性。假设未修改的 The Evaluator 版本包含了一个 ID 为 myEM 的 em 元素。在顶部文本框中依次输入以下语句：

```
document.getElementById("myEM").style.color
```

和

```
document.getElementById("myEM").runtimeStyle.color
```

最初两个值均为空。现在在顶部文本框中为 style 属性指定颜色：

```
document.getElementById("myEM").style.color = "red"
```

如果在顶部文本框中输入前面的两条语句，style 对象就会反映这个变化，而 runtimeStyle 对象的(空)值仍保持不变。

相关主题：currentStyle 属性；style 对象(参阅第 38 章)。

scopeName，tagUrn

值：字符串，　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

scopeName 属性主要与嵌入文档中的 XML 代码相关。当包括 XML 时，可指定一个或多个 XML 命名空间，来定义一个自定义标记名的拥有者，防止文档中来自不同源的相同自定义标记相互冲突。

XML 命名空间赋给<html>标记的一个属性，此标记包含了整个文档：

```
<html xmlns:fred='http://www.someURL.com'>
```

之后，命名空间值放在所有链接到此命名空间的自定义标记之前：

```
<fred:FIRST_Name id="fredFirstName"/>
```

要确定元素的命名空间拥有者，可读取该元素的 scopeName 属性。对于上例而言，scopeName

更多资源请访问稀酷客(www.ckook.com)

返回 fred；对于普通 HTML 元素，scopeName 的返回值总是 HTML。tagURN 属性与 scopeName 类似，它存储了命名空间的 URI。

scopeName 属性仅在 IE5+的 Win32 和 UNIX 版本中可用。在 W3C DOM 中，与 scopeName 和 tagURN 对应的属性是 prefix 和 namespaceURI。

示例

如果示例文档包含 XML 和命名空间说明，可用 document.write()或 alert()方法来查看 scopeName 属性值。语法为：

```
document.getElementById("elementID").scopeName
```

相关主题：tagUrn 属性。

scrollHeight，scrollWidth

值：整型，　　只读

兼容性：WinIE4+，MacIE4+，NN7+，Moz1.0.1+，Safari+，Opera+，Chrome+

scrollHeight 和 scrollWidth 属性包括对象的像素尺寸，而不管该对象是否在页面上可见。因此，如果浏览器窗口显示垂直滚动条，而文档主体一直扩展到窗口中可见部分的底部，scrollHeight 就会考虑整个主体的高度，就像向下滚动查看整个元素一样。对于大多数没有滚动条的元素而言，scrollHeight 和 scrollWidth 属性与 clientHeight 和 clientWidth 属性的值相同。

示例

使用 The Evaluator(参见第 4 章)试验 textarea 对象的这两个属性，该对象显示求值结果和属性列表。首先在底部的单行文本框中输入以下语句，列出 body 对象的属性：

```
document.body
```

这会显示 body 对象的长属性列表。现在，在顶部的单行文本框中输入以下属性表达式，看看 textarea 在包含数十个属性列表行时的 scrollHeight 属性：

```
document.getElementById("output").scrollHeight
```

结果(可能是几百大小)现在显示在 textarea 中，这意味着可以垂直滚动 output 元素的内容，可滚动高度就是该像素值。再次单击 Evaluate 按钮，结果(13 或 14)是只包含上次结果的 textarea 的 scrollHeight 属性值。该内容的可滚动高度仅为 13 或 14 个像素——textarea 中的字体高度。textarea 的 scrollWidth 属性由分配给元素 cols 特性的宽度来确定(由浏览器计算，来确定文本区域在页面上显示为多宽)。

相关主题：clientHeight、clientWidth 属性；window.scroll()方法。

scrollLeft，scrollTop

值：整型，　　只读

兼容性：WinIE4+，MacIE4+，NN7+，Moz1.0.1+，Safari+，Opera+，Chrome+

如果元素是可滚动的(换句话说，它有滚动条)，则可通过 scrollLeft 和 scrollTop 属性确定元

更多资源请访问稀酷客(www.ckook.com)

素在水平和垂直方向滚动了多远，这些属性值的单位是像素。对于不可滚动的元素，这些属性的值总是 0，即使它们可以放在可滚动的元素中也是如此。例如，如果垂直滚动浏览器窗口(或多框架环境下的框架)，body 对象的 scrollTop 属性是对象顶部(现在看不见)和元素的第一个可见像素行之间的像素距离，但文档中表的 scrollTop 值始终是 0。

版本 7 (Mozilla)之前的 Netscape 浏览器从窗口的角度来看待 body 元素的滚动。如果要确定当前页面在这些浏览器中的滚动距离，可使用 window.scrollX 和 window.scrollY。

在 IE 中，跟踪鼠标事件的脚本需要考虑 body 的 scrollLeft 和 scrollTop 属性，以便补偿页面的滚动。参见第 32 章中的 Event 对象。

示例

用 The Evaluator(参见第 4 章)来试验 textarea 对象的这两个属性，textarea 对象显示了求值结果和属性列表。首先，在底部的单行文本框中输入以下语句，列出 body 对象的属性：

```
document.body
```

这显示了 body 对象的长属性列表。使用 textarea 的滚动条向下翻页数次，现在在顶部的单行文本框中输入以下属性表达式，看一下在滚动后 textarea 的 scrollTop 属性：

```
document.getElementById("myTextArea").scrollTop
```

结果(某个数字)现在显示在 textarea 中。这意味着垂直滚动了 output 元素的内容。再次单击 Evaluate 按钮，结果(0)是只包含上次结果的 textarea 的 scrollHeight 属性值。在 textarea 中没有足够多的内容可以滚动，因此内容根本没有滚动。这样，scrollTop 属性为 0。textarea 元素的 scrollLeft 属性始终为 0，因为 textarea 元素设置为对溢出元素宽度的文本换行。在这种情况中，水平滚动条不显示，scrollLeft 属性永远不变。

相关主题：clientLeft 属性，clientTop 属性；window.scroll()方法。

sourceIndex

值：整型，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

sourceIndex 属性返回对象在整个文档中的数字下标(从 0 开始)，它是文档中所有元素的分组。

示例

尽管此属性的操作很明了，但 document.all 属性显示的元素顺序可能不是这样。所以，可在 IE4+中使用 The Evaluator(参见第 4 章)试验 sourceIndex 属性的返回值，看一下 document.all 集合的下标值是否与源代码对应。

首先，重新载入 The Evaluator，在顶部的文本框中输入以下语句，设置一个预先初始化的全局变量：

```
a=0
```

计算此表达式时，在 Results 框中会显示 0。接下来在顶部文本框中输入以下语句：

```
document.all[a].tagName + " [" + a++ + "]"
```


更多资源请访问稀酷客(www.ckook.com)

此语句中有许多加号，因此一定要输入正确。在连续对此语句求值时(重复单击 Evaluate 按钮)，全局变量(a)会递增，以按照源代码的顺序遍历所有元素。每个 HTML 标记的 sourceIndex 值显示在 Results 框的方括号中。通常首先显示以下序列：

```
html [0]
head [1]
title [2]
```

继续单击 Evaluate，直至遍历完所有的元素，此时会显示一条错误消息，因为 a 的值超过了 document.all 数组中的元素数量。将结果与 The Evaluator 的 HTML 源代码视图进行比较。

相关主题： item()方法。

style

值： style 对象引用， 读/写

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

style 属性是设置元素样式表的入口。该属性的值为 style 对象，style 对象的属性允许读写元素的样式表设置。尽管脚本不经常将 style 对象看作一个整体，但在 DHTML 页面上，通常用脚本获取或设置 style 对象的多个属性，来提高元素的动画效果、可见性，获得所有的外观参数。注意，通过此对象返回的样式属性都是通过元素的 style 特性或脚本明确设置的。

在不同版本的 IE 和 NN 中，style 对象的属性范围有相当大的区别。style 对象详见第 38 章。

示例

style 属性的大多数操作都与 style 对象的属性有关，因此可使用 The Evaluator 来研究在各种 DHTML 兼容浏览器上可用的 style 对象属性列表。首先，在底部的单行文本框中输入以下语句，来查看 document.body 对象的 style 属性：

```
document.body.style
```

现在在 The Evaluator 原始版本中查看表格元素的 style 属性，在底部文本框中输入以下语句：

```
document.getElementById("myTable").style
```

在这两种情况下，默认分配给 style 对象的属性值都相当有限。

相关主题： currentStyle 属性，runtimeStyle 属性；style 对象(参阅第 38 章)。

tabIndex

值： 整型， 读/写

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

tabIndex 属性控制当前对象接收焦点的 Tab 顺序，该属性显然只应用于可接收焦点的元素中。IE5+允许接收焦点的元素比其他浏览器多；但在兼容该属性的所有浏览器中，都包括了用户想控制其焦点(即表单输入元素)的基本元素。

通常，浏览器默认将表单元素看成可获得焦点的元素。非表单元素通常不接收焦点，除非设置了它们的 tabIndex 属性(或 tabindex 标记特性)。如果将一个表单元素的 tabIndex 属性设为 1，

更多资源请访问稀酷客(www.ckook.com)

该元素将是 Tab 顺序中的第一个元素。连续按下 Tab 键时，其他元素将按照它们在源代码中出现的顺序接收焦点。如果将两个元素的 tabIndex 属性都设为 1，这两个元素的 Tab 顺序将覆盖它们的源代码顺序，然后焦点再从页面顶部开始按照源代码顺序移动到其他元素。

在 Internet Explorer 和 Mozl.8+中，将元素的 tabIndex 属性设置为-1，可将元素从 Tab 顺序中完全删除。用户仍可以单击这些元素，来修改表单元素设置，但按 Tab 键时将忽略这些元素。

示例

程序清单 26-16 包含的示例脚本演示了如何通过 tabIndex 属性来控制表单的 Tab 顺序。此示例不仅演示了实时修改表单的 Tab 顺序的方式，而且说明了如何在 IE 中强制将表单元素从 Tab 顺序中彻底删除。在此页面中，顶部的表单 lab 包含 4 个元素，由底部表单中的按钮调用的脚本控制着 Tab 顺序。注意，所有底部表单元素的 tabIndex 特性都设置为-1，这意味着这些控制按钮不在 Tab 顺序中。

载入页面时，lab 表单控件元素使用默认的 tab 顺序(默认设置为 0)。如果按下 Tab 键，结果首先取决于所使用的浏览器。在 IE 中，首先选中地址域，然后焦点移动到窗口上(或框架，如果此页面是在框架集中)，最后，焦点移动到 lab 表单。继续按 Tab 键，看看浏览器如何将焦点赋予每个元素类型。在基于 WebKit 和 Presto 的浏览器中，以及 Mozilla 的最新版本中，焦点先移动到 lab 表单。但在 NN6 中，只有在内容上单击，才能使 Tab 键在表单控件上起作用。

示例脚本通过 for 循环反转了 Tab 顺序，该循环初始化了两个变量，这些变量在循环中反向工作，给最后一个元素指定了最低的 tabIndex 值。skip2()函数将第二个文本框的 tabIndex 属性设置为-1，将其完全从 Tab 顺序中删除(以前只有 IE 支持它，但目前所有现代浏览器都支持它)。但要注意，用户仍可以单击输入域，并输入文本(参考本章前面的 disabled 属性，看看如何防止编辑输入域)。NN6+没有提供强制浏览器跳过表单控件的 tabIndex 属性设置；要跳过表单控件，应该禁用该控件。

程序清单 26-16　控制 tabIndex 属性

HTML：jsb26-16.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>tabIndex Property</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-16.js"></script>
    </head>
    <body>
        <h1>tabIndex Property Lab</h1>
        <hr />
        <form name="lab">
            Text box no. 1: <input type="text" name="text1" />
            <br />
            Text box no. 2: <input type="text" name="text2" />
            <br />
            <input type="button" value="A Button" />
```


更多资源请访问稀酷客(www.ckook.com)

```
            <br />
            <input type="checkbox" />And a checkbox
        </form>
        <hr />
        <form name="control">
            <input type="button" value="Invert Tabbing Order" tabindex="-1"
                    onclick="invert()" />
            <br />
            <input type="button" value="Skip Text box no. 2"
                    tabindex="-1" onclick="skip2()" />
            <br />
            <input type="button" value="Reset to Normal Order" tabindex="-1"
                    onclick="resetTab()" />
        </form>
    </body>
</html>
```

JavaScript: jsb26-16.js

```
function invert()
{
     var form = document.lab;
     for (var i = 0, j = form.elements.length; i < form.elements.length; i++, j--)
     {
        form.elements[i].tabIndex = j;
     }
}
function skip2()
{
     document.lab.text2.tabIndex = -1;
}
function resetTab()
{
    var form = document.lab;
    for (var i = 0; i < form.elements.length; i++)
    {
        form.elements[i].tabIndex = 0;
    }
}
```

最后的 resetTab()函数将所有 lab 表单元素的 tabIndex 属性值都设置为 0，恢复了默认顺序。

相关主题：blur()、focus()方法。

tagName

值：字符串，　　　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

tagName 属性返回对象的 HTML 或 XML 标记名字符串。即使源代码以小写或混合大小写的形式编写 tagName 属性，所有 tagName 值也都以全大写字母的形式返回。这种一致性更便于比较字符串，例如，可创建一个通用函数，它包含一个 switch 语句，为一些标记执行动作。这

更多资源请访问稀酷客(www.ckook.com)

个函数的框架如下：

```
function processObj(objRef)
{
   switch (objRef.tagName)
   {
      case "TR":
         [statements to deal with table row object]
         break;
      case "TD":
         [statements to deal with table cell object]
         break;
      case "COLGROUP":
         [statements to deal with column group object]
         break;
      default:
         [statements to deal with all other object types]
   }
}
```

示例

在本章前面讨论的 sourceIndex 属性示例中，也使用了 tagName 属性。在那个示例中，tagName 属性按源代码顺序从一个对象序列中读取。

相关主题：nodeName 属性；getElementsByTagName()方法。

tagUrn

参见 scopeName。

textContent

值：字符串，　　　　只读

兼容性：WinIE-，MacIE-，NN-，Moz1.7+，Safari+，Opera+，Chrome+

此属性存储节点的文本串，包括元素内的任何组合文本节点。这意味着即使节点嵌套了其他元素(如 em)，节点的内容也作为单个文本串反映在 textContent 属性中。如果设置 textContent 属性，用一串文本替换节点内容，所有以前的节点内容都会替换，包括嵌套的元素。可将 textContent 看成 W3C DOM 中与 IE 的 innerText 相当的属性。

相关主题：innerText 属性。

title

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C 标准指出，应将 title 属性和 title 特性当作备选属性，用于残障人士的设备使用 title 属性和 title 特性，所以最好总是给它们赋值。大多数浏览器将此值解释为赋给工具提示的文本(当光标放到元素上时，工具提示会立即显示出来)。这个属性的优点是脚本可修改元素的工具提示文本，来响应页面上的其他用户交互操作。工具提示可提供对页面上图标或链接的简略帮

更多资源请访问稀酷客(www.ckook.com)

助，还可从链接的目标上传递关键内容的摘要，向用户显示重要的信息，而不必导航到其他页面上。

与设置状态栏一样，最好不要使用工具提示向用户传递关键的信息，因为不是所有的用户都有耐心地把指针停在某个元素上，等待工具提示的出现；有些用户可能更注意工具提示而不是状态栏消息(即使后者是即刻出现的)。

示例

程序清单 26-17 说明了如何使用 title 属性来建立页面的工具提示。将一个简单的段落元素的 title 属性设置为“First Time!”，这是页面载入后，用户将鼠标指针停留在段落上所显示的工具提示。但该元素的 onmouseover 事件处理程序在脚本中增加了一个全局变量计数器，每次鼠标移动到段落上时，段落对象的 title 属性都会改变，count 值是分配给 title 属性的字符串的一部分。注意，title 属性和变量之间没有活动连接，新值明确设置了 title 属性。

程序清单 26-17　控制 title 属性

HTML: jsb26-17.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>title Property</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-17.js"></script>
   </head>
   <body>
      <h1>title Property Lab</h1>
      <hr />
      <p id="myP" title="First Time!" onmouseover="incrementCount(this)">Roll
         the mouse over this paragraph a few times.
         <br />
         Then pause atop it to view the tooltip.
      </p>
   </body>
</html>
JavaScript: jsb26-17.js
// global counting variable
var count = 0;

function setToolTip(elem)
{
   elem.title = "You have previously rolled atop this paragraph " + count + "
        time(s).";
}

function incrementCount(elem)
{
   count++;
```


更多资源请访问稀酷客(www.ckook.com)

```
    setToolTip(elem);
}
```

相关主题： window. status 属性。

uniqueID

值： 字符串，　　　　只读

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

利用 uniqueID 属性，可让 WinIE5+浏览器为页面上动态生成的元素创建一个标识符(id 属性)。这个属性应该小心使用，因为 uniqueID 属性在给定时刻创建的 ID 和下一时刻在页面上创建的 ID 不同。因此，当脚本需要让一个未知元素拥有 id 属性，而算法并没有指定任意特定的标识符时，应使用 uniqueID 属性。

为了保证对象存在于内存中时，元素只有一个 ID，可以给该对象(而不是某个其他对象)的 uniqueID 属性赋值。一旦得到了对象的 uniqueID 属性，不管访问该属性多少次，其属性值都不变。一般情况下，可将 uniqueID 属性返回的值赋给对象的 id 属性，以便执行其他处理。例如，getElementById()方法就需要对象的 id 属性值作为参数。

示例

程序清单 26-18 给出了获取和应用浏览器生成的对象标识符的推荐语法。在文本框中输入一些文本后，单击按钮，addRow()函数就在表格中添加一行。左列显示了通过表格行对象的 uniqueID 属性生成的标识符。IE5+生成 ms__idn 格式的标识符，对于当前浏览器会话而言，n 是一个从 0 开始的整数。由于 addRow()函数为行和每行中的单元格都分配了 uniqueID 值，因此每行的整数都比前一行大 3。无法保证以后的浏览器版本也遵循这一格式，因此在脚本中不要依赖此格式或顺序。

程序清单 26-18　使用 uniqueID 属性

HTML: jsb26-18.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Inserting an WinIE5+ Table Row</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-18.js"></script>
   </head>
   <body>
      <table id="myTable" border="1">
         <tr>
            <th>Row ID</th>
            <th>Data</th>
         </tr>
         <tr id="firstDataRow">
            <td>firstDataRow</td>
            <td>Fred</td>
```


更多资源请访问稀酷客(www.ckook.com)

```
            </tr>
            <tr id="secondDataRow">
                <td>secondDataRow</td>
                <td>Jane</td>
            </tr>
        </table>
        <hr />
        <form>
            Enter text to be added to the table:
            <br />
            <input type="text" name="input" size="25" />
            <br />
            <input type='button' value='Insert Row'
                    onclick='addRow(this.form.input.value) ' />
        </form>
    </body>
</html>
```

JavaScript: jsb26-18.js

```
function addRow(item1)
{
   if (item1)
   {
      // assign long reference to shorter var name
      var theTable = document.getElementById("myTable");
      // append new row to the end of the table
      var newRow = theTable.insertRow(theTable.rows.length);
      // give the row its own ID
      newRow.id = newRow.uniqueID;

      // declare cell variable and data variable
      var newCell;
      var newText;

      // an inserted row has no cells, so insert the cells
      newCell = newRow.insertCell(0);
      // give this cell its own id
      newCell.id = newCell.uniqueID;
      // display the row's id as the cell text
      newText = document.createTextNode(newRow.id);
      newCell.appendChild(newText);
      newCell.bgColor = "yellow"
      // re-use cell var for second cell insertion
      newCell = newRow.insertCell(1);
      newCell.id = newCell.uniqueID;
      newText = document.createTextNode(item1);
      newCell.appendChild(newText);
   }
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：id 属性；getElementById()方法。

unselectable

值：字符串常量(“on”或“off”)，　　读/写

兼容性：WinIE5.5+，MacIE-，NN-、Moz-，Safari-，Opera+，Chrome-

此属性控制元素的可选择性，即元素内容可否供用户选择。可使用此属性来防止敏感数据被选择和复制。

4. 方法

addBehavior("URL")

返回值：整数 ID。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

addBehavior()方法导入一个外部 Internet Explorer 操作，并将它与当前对象相关联，以扩展对象的属性和方法(有关 IE 操作的详细信息，请参见配书光盘中的第 51 章)。addBehavior()方法唯一的参数是指向操作组件代码的 URL 指针，该组件可放在一个外部文件(扩展名.htc)中，此时，参数可以是相对或绝对的 URL；IE 还包含内置(默认)的操作库，它的 URL 采用下列格式：

```
#default#behaviorName
```

这里，behaviorName 是一个默认操作。如果操作通过 object 标记导入到文档中，addBehavior()方法的参数就是下列格式的元素 ID：

```
#objectID
```

添加操作时，外部代码的载入是异步进行的。这意味着即使方法立刻返回一个值，操作也可能还没有准备好。只有完全载入操作，它才会响应事件，或者允许访问它的属性和方法。从外部文件载入的操作会遵守域安全规则。

示例

程序清单 26-19a 列举了操作文件的一个示例，来演示程序清单 26-19b 中的 addBehavior()方法。操作组件和载入它的 HTML 页面必须来自相同的服务器和域，还必须通过相同的协议(例如，http://、https://和 file://是相互排斥的不匹配协议)载入。

程序清单 26-19a　makeHot.htc 操作组件

```
<PUBLIC:ATTACH EVENT="onmousedown" ONEVENT="makeHot()" />
<PUBLIC:ATTACH EVENT="onmouseup" ONEVENT="makeNormal()" />
<PUBLIC:PROPERTY NAME="hotColor" />
<PUBLIC:METHOD NAME="setHotColor" />
<script type="text/jscript">
   var oldColor
   var hotColor = "red"
```


更多资源请访问稀酷客(www.ckook.com)

```
    function setHotColor(color)
    {
       hotColor = color
    }

    function makeHot()
    {
       if (event.srcElement == element)
       {
          oldColor = style.color
          runtimeStyle.color = hotColor
       }
    }
    function makeNormal()
    {
       if (event.srcElement == element)
       {
          runtimeStyle.color = oldColor
       }
    }
</script>
```

该组件附属在一个简单的段落对象上，如程序清单 26-19b 所示。载入页面时，没有关联该操作，因此单击段落文本没有任何效果。

通过调用 turnOn()函数启动操作时，addBehavior()方法将 makeHot.htc 组件的代码连接到 myP 对象上。此时，myP 对象多了一个属性、一个方法和两个事件处理程序，它们由组件代码编写为公共元素。如果想将该操作应用于文档中的多个段落，就必须对每个段落对象调用 addBehavior()方法。

在开始载入操作文件后，将会调用 setInitialColor()函数，将段落的新颜色属性设置为用户在 select 列表中所做的选择，但只有在组件完全载入后才进行这个设置。因此，在调用组件的函数之前，setInitialColor()函数会检查 myP 的 readyState 属性，来确认组件是否完全载入。如果 IE 仍在载入组件，就在 500ms 后再次调用该函数。

只要载入了操作，就可以更改颜色，将段落变为热区。函数首先检查对象是否有新的颜色属性，来确保组件已载入。如果组件已载入，就调用组件的方法(来说明如何展示和调用组件的方法)。也可以简单地设置属性值。

程序清单 26-19b　使用 addBehavior()和 removeBehavior()

HTML: jsb26-19.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>addBehavior() and removeBehavior() Methods</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-19.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
    </head>
    <body>
        <h1>addBehavior() and removeBehavior() Method Lab</h1>
        <hr />
        <p id="myP">This is a sample paragraph. After turning on the behavior, it
            will turn your selected color when you mouse down anywhere in this
            paragraph.
        </p>
        <form>
            <input type="button" value="Switch On Behavior" onclick="turnOn()" />
            Choose a 'hot' color:
            <select name="colorChoice" onchange="setColor(this, this.value)">
                <option value="red">red</option>
                <option value="blue">blue</option>
                <option value="cyan">cyan</option>
            </select>
            <br />
            <input type="button" value="Switch Off Behavior"
                    onclick="turnOff()" />
            <p>
                <input type="button" value="Count the URNs"
                    onclick="showBehaviorCount()" />
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb26-19.js

```
var myPBehaviorID;
function turnOn()
{
   myPBehaviorID = document.getElementById("myP").addBehavior("makeHot.htc");
   setInitialColor();
}

function setInitialColor()
{
   if (document.getElementById("myP").readyState == "complete")
   {
      var select = document.forms[0].colorChoice;
      var color = select.options[select.selectedIndex].value;
      document.getElementById("myP").setHotColor(color);
   }
   else
   {
      setTimeout("setInitialColor()", 500);
   }
}

function turnOff()
```


更多资源请访问稀酷客(www.ckook.com)

```
    {
       document.getElementById("myP").removeBehavior(myPBehaviorID);
    }

    function setColor(select, color)
    {
       if (document.getElementById("myP").hotColor)
       {
          document.getElementById("myP").setHotColor(color);
       }
       else
       {
          alert("This feature is not available. Turn on the Behavior first.");
          select.selectedIndex = 0;
       }
    }

    function showBehaviorCount()
    {
       var num = document.getElementById("myP").behaviorUrns.length;
       var msg = "The myP element has " + num + " behavior(s). ";
       alert(msg);
    }
```

要停用操作，可调用 removeBehavior()方法。注意，removeBehavior()方法与 myP 对象相关联，其参数为前面添加的操作的 ID。如果将多个操作与一个对象相关联，则删除一个操作不会影响其他操作，因为每个操作都有唯一的 ID。

相关主题：readyState 属性；removeBehavior()方法；IE 操作(参见配书光盘中的第 51 章)。

addEventListener("eventType", listenerFunc, useCapture), removeEventListener("eventType", listenerFunc,useCapture)

返回值：无。

兼容性：WinIE-， MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C DOM 的事件机制支持事件冒泡和滴流(参阅第 32 章)。新的机制支持一个长期存在的概念：通过 HTML 特性(如旧的 onclick 事件处理程序)把事件绑定到元素上，但最好通过注册元素的事件监听器来绑定事件。在支持 W3C 事件模型的浏览器中，绑定事件的其他方法(例如事件处理属性)会在内部转换为注册事件。

为了告诉 DOM 元素应该监听某个事件，可在元素对象上使用 addEventListener ()方法。该方法需要三个参数，第一个是 DOM 元素应该监听的事件类型的字符串文本。事件类型字符串不包括事件处理程序常用的 on 前缀，只包含事件名，并经常是全部小写的(除了 DOM 使用的一些系统范围的特殊事件)。表 26-9 显示了 W3C DOM 规范支持的所有事件(包括一些尚未在浏览器中实现的新 DOM 事件)。

更多资源请访问稀酷客(www.ckook.com)

表 26-9　W3C DOM 事件监听器的类型

abort	error
blur	focus
change	load
click	mousedown
DOMActivate	mousemove
DOMAttrModified	mouseout
DOMCharacterDataModified	mouseover
DOMFocusIn	mouseup
DOMFocusOut	reset
DOMNodeInserted	resize
DOMNodeInsertedIntoDocument	scroll
DOMNodeRemoved	select
DOMNodeRemovedFromDocument	submit
DOMSubtreeModified	unload

注意，DOM Level 2 指定的事件类型比 IE 4+定义的事件更有限。而 W3C 只临时列出了键盘事件，直到 DOM Level 3 才实现键盘事件。幸好，大多数 W3C 兼容浏览器都实现了键盘事件，这些事件将来可能会作为 W3C DOM Level 3 的一部分。

addEventListener()方法的第二个参数是要调用的 JavaScript 函数的引用，其形式与把函数赋给对象的事件属性相同(例如 objReference.onclick=someFunction)，且不应该是加引号的字符串。这也意味着不能在函数调用中指定参数，因此，如果函数需要引用表单或表单控件元素，就必须自己构建引用(使用 event 对象的属性，来说明哪个对象是事件的目标)。

默认情况下，W3C DOM 事件模型中的事件从事件的目标对象(例如被单击的按钮)开始，沿着元素容器层次结构向上冒泡。然而，如果将 addEventListener()方法的第三个参数指定为 true，则只要当前对象是事件的目标，就会给这个事件类型启用事件捕获功能。这意味着任何将当前对象作为目标的事件类型都可向上冒泡，除非它也有与对象相关的事件监听器，而且第三个参数设为 true。

使用 addEventListener()方法的要求是，与之相关的对象已经存在。因此，可在初始化函数中使用这个方法，该初始化函数由页面的 onload 事件处理程序触发(document 对象可立即给载入事件使用 addEventListener()，因为在载入过程的初期，document 对象就存在)。

脚本也可删除前面添加的事件监听器，removeEventListener()方法的参数和 addEventListener()一样，这意味着关闭一个事件监听器，不会干扰其他监听器。实际上，因为可为事件和监听函数添加两个监听器(一个设置为启动捕获功能，另一个没有这样设置，这实际上是很少见的)，所以 removeEventListener()的三个参数可以指定要从对象中移除哪个监听器。

不管目标如何，W3C DOM 事件模型都没有提供某个事件类型的“全局”捕获机制，这与 NN4 的事件捕获机制不同。至于 Internet Explorer，addEventListener()方法和 IE5+的 attachEvent()方法非常相似。IE5+中的事件捕获机制是通过 setCapture()方法启用的。W3C 和 IE 事件模型都

更多资源请访问稀酷客(www.ckook.com)

使用自己的语法来绑定对象与事件处理函数，因此实际函数只有使用只为事件绑定设置的浏览器版本，才能服务于这两个模型。关于使用这两个事件模型的事件处理程序，请参阅第 32 章。

示例

程序清单 26-20 提供了一个简洁的工作平台来研究和试验基本的 W3C DOM 事件模型。载入页面时，浏览器没有注册事件监听器(当然，控制按钮除外)，但可以用冒泡和/或捕获模式，为包含 span(span 包含着文本行)的 body 元素或 p 元素添加一个 click 事件的监听器。如果添加了事件监听器并单击文本，就会显示处理事件的元素和表示事件阶段为冒泡(3)还是捕获(1)的信息。若使用了所有事件监听器，要注意事件的处理顺序。一次删除一个监听器，看看对事件处理的影响。

程序清单 26-20　W3C 事件练习

HTML: jsb26-20.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>W3C Event Model Lab</title>
      <style type="text/css">
         td
         {
            text-align:center
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-20.js"></script>
   </head>
   <body id="myBODY">
      <h1>W3C Event Model Lab</h1>
      <hr />
      <p id="myP"><span id="mySPAN">This paragraph (a SPAN element nested inside
         a P element) can be set to listen for "click" events.</span>
      </p>
      <hr />
      <form name="controls" id="controls">
         <p>Examine click event characteristics: 
           <input type="button" value="Clear" onclick="clearTextArea()" />
           <br />
           <textarea name="output" cols="80" rows="6" wrap="virtual"></textarea>
         </p>
         <table cellpadding="5" border="1">
            <caption style="font-weight:bold">Control Panel</caption>
            <tr style="background-color:#ffff99">
               <td rowspan="2">"Bubble"-type click listener:</td>
               <td>
                  <input type="button" value="Add to BODY"
                         onclick="addBubbleListener('myBODY')" />
```


更多资源请访问稀酷客(www.ckook.com)

```
                </td>
                <td>
                    <input type="button" value="Remove from BODY"
                           onclick="removeBubbleListener('myBODY')" />
                </td>
            </tr>
            <tr style="background-color:#ffff99">
                <td>
                    <input type="button" value="Add to P"
                           onclick="addBubbleListener('myP')" />
                </td>
                <td>
                    <input type="button" value="Remove from P"
                           onclick="removeBubbleListener('myP')" />
                </td>
            </tr>
            <tr style="background-color:#ff9999">
                <td rowspan="2">"Capture"-type click listener:</td>
                <td>
                    <input type="button" value="Add to BODY"
                           onclick="addCaptureListener('myBODY')" />
                </td>
                <td>
                    <input type="button" value="Remove from BODY"
                           onclick="removeCaptureListener('myBODY')" />
                </td>
            </tr>
            <tr style="background-color:#ff9999">
                <td>
                    <input type="button" value="Add to P"
                           onclick="addCaptureListener('myP')" />
                </td>
                <td>
                    <input type="button" value="Remove from P"
                           onclick="removeCaptureListener('myP')" />
                </td>
            </tr>
        </table>
    </form>
  </body>
</html>
```

JavaScript: jsb26-20.js

```
// add event listeners
function addBubbleListener(elemID)
{
   document.getElementById(elemID).addEventListener("click",
     reportEvent, false);
}
function addCaptureListener(elemID)
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   document.getElementById(elemID).addEventListener("click", reportEvent, true);
}
// remove event listeners
function removeBubbleListener(elemID)
{
   document.getElementById(elemID).removeEventListener("click",
     reportEvent, false);
}
function removeCaptureListener(elemID)
{
   document.getElementById(elemID).removeEventListener("click",
     reportEvent, true);
}

// display details about any event heard
function reportEvent(evt)
{
   var elem = (evt.target.nodeType == 3) ? evt.target.parentNode : evt.target;
   if (elem.id == "mySPAN")
   {
     var msg = "Event processed at " + evt.currentTarget.tagName +
               " element (event phase = " + evt.eventPhase + ").\n";
     document.controls.output.value += msg;
   }
}
// clear the details textarea
function clearTextArea()
{
     document.controls.output.value = "";
}
```

相关主题：attachEvent()、detachEvent()、dispatchEvent()、fireEvent()和 removeEventListener()方法。

appendChild(elementObject)

返回值：Node 对象引用。

兼容性：WinIE5+、 MacIE5+、 NN6+、Moz+、 Safari+、Opera+、Chrome+

appendChild()方法插入一个元素或文本节点(由它之前的其他代码定义)，作为当前元素的最后一个新子节点或元素。除了把新子元素添加到子节点序列的尾部之外，appendChild()方法也可以构建元素对象及其内容，之后在现有文档中添加、替换或插入元素。document.createElement()方法创建元素的引用，该元素的标记名用作方法的参数。

appendChild()方法返回新添加的 Node 对象的引用。这个引用和传递给方法的对象参数不同，因为返回的对象是文档的一部分，而不是内存中的独立对象。

示例

程序清单 26-21 包含的示例演示了如何用 appendChild()、removeChild()和 replaceChild()

更多资源请访问稀酷客(www.ckook.com)

方法来修改文档中的子元素。由于许多 W3C DOM 浏览器将源代码中的回车看作文本节点(从而成为其父元素的子节点)，因此，在程序清单 26-21 中，受影响元素的 HTML 在元素之间没有回车。

append()函数创建了一个新的 li 元素，然后使用 appendChild()方法将文本框中的文本添加为该项的显示文本。嵌套表达式 document.createTextNode(newDate)计算为合法的节点，该节点添加到新的 li 项中。所有这些操作都在新 li 项添加到文档之前执行。函数的最后一条语句在 ul 元素的位置上调用 appendChild()，从而将 li 元素添加为 ul 元素的子节点。

在 rplace()函数中调用 replaceChild()方法时使用了部分相同的代码。主要区别在于 replaceChild()方法需要第二个参数：要替换的子元素引用。此示例替换了 ul 列表的最后一个子节点，因此函数利用了元素的 lastChild 属性来获得对最后一个嵌套子元素的引用，该引用是 replaceChild()的第二个参数。

程序清单 26-21　各种子节点方法

HTML: jsb26-21.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>appendChild(), removeChild(), and replaceChild() Methods</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-21.js"></script>
   </head>
   <body>
      <h1>Child Methods</h1>
      <hr />
      Here is a list of items:
      <ul id="myUL"><li>First Item</li><li>Second Item</li></ul>
      <form>
         Enter some text to add/replace in the list:
         <input type="text" id="newListData" size="30" />
         <br />
         <input type="button" value="Append to List"
                onclick="append()" />
         <input type="button" value="Replace Final Item"
                onclick="rplace()" />
         <input type="button" value="Truncate List to 2 Items"
                onclick="truncateList()" />
      </form>
   </body>
</html>
```

JavaScript: jsb26-21.js

```
function append()
{
   var newData = document.getElementById("newListData").value;
```


更多资源请访问稀酷客(www.ckook.com)

```
        if (newData)
        {
            var newItem = document.createElement("LI");
            newItem.appendChild(document.createTextNode(newData));
            document.getElementById("myUL").appendChild(newItem);
        }
    }

    // Opera throws an error when use keyword replace, so....
    function rplace()
    {
        var newData = document.getElementById("newListData").value;
        if (newData)
        {
            var newItem = document.createElement("LI");
            var lastChild = document.getElementById("myUL").lastChild;
            newItem.appendChild(document.createTextNode(newData));
            document.getElementById("myUL").replaceChild(newItem, lastChild);
        }
    }

    function truncateList()
    {
        var oneChild;
        var mainObj = document.getElementById("myUL");
        while (mainObj.childNodes.length > 2)
        {
            oneChild = mainObj.lastChild;
            mainObj.removeChild(oneChild);
        }
    }
```

示例的最后部分使用 removeChild()方法，来剥离 ul 元素的所有子元素，仅保留两项。同样，truncatelist()函数也使用 lastChild 属性，不断删除最后一个子节点，直至只剩两个节点。

相关主题：removeChild()方法、replaceChild()方法；节点和子节点(参阅第 25 章)。

applyElement(*elementObject*[, *type*])

返回值：无。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

applyElement()方法可以插入新元素，作为当前对象的父元素或子元素。这个方法的重要功能是用新对象包括当前对象(如果新元素要成为父元素)，或包括当前对象的内容(如果新元素要成为子元素)。把新元素作为子元素时，所有以前创建的子元素就嵌套到更深的一层，成为新元素的直接子元素。该方法的结果会影响当前元素的包含层次，因此在使用 applyElement()方法时必须小心。

该方法需要一个参数：被应用对象的引用。这个对象可从 document.createElement()等结构中创建，也可以从子元素中创建，或从返回对象的节点方法中得到。第二个参数是可选参数，

更多资源请访问稀酷客(www.ckook.com)

它必须是表 26-10 中的值之一：

表 26-10 值

参 数 值	说 明
outside	新元素要成为当前对象的父元素
inside	新元素要成为当前对象的直接子元素

如果忽略第二个参数，就使用其默认值(outside)。程序清单 26-22 显示了 applyElement()方法如何使用默认值，以及如何不使用默认值。

示例

为理解 applyElement()方法使用不同参数设置的影响，程序清单 26-22 将一个新元素(em 元素)应用到段落中的 span 元素。用户可随时查看整个 p 元素的 HTML，以便了解 em 元素的应用位置及其对段落的元素包含层次结构的影响。

载入页面后，先查看段落的 HTML，再执行其他操作。注意 span 元素及其嵌套的 font 元素，这两个元素都包含一个单词。如果在 span 元素中应用 em 元素(单击中间的按钮)，span 元素的第一个(并且是唯一的)子元素就变为 em 元素，code 元素则是新 em 元素的子元素。

程序清单 26-22 使用 applyElement()方法

HTML：jsb26-22.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>applyElement() Method</title>
      <style type="text/css">
         #mySpan
         {
            font-size:x-large;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-22.js"></script>
   </head>
   <body>
      <h1>applyElement() Method</h1>
      <hr />
      <p id="myP">A simple paragraph with a <span id="mySpan"><code>
      special</code></span> word in it. How special? Click an apply button,
         then click the show button.
      </p>
      <form>
         <input type="button" value="Apply &lt;EM&gt; Outside"
               onclick="applyOutside()" />
         <input type="button" value="Apply &lt;EM&gt; Inside"
```


更多资源请访问稀酷客(www.ckook.com)

```
                onclick="applyInside()" />
            <input type="button" value="Show &lt;P&gt; HTML..."
                onclick="showHTML()" />
            <br />
            <input type="button" value="Restore Paragraph"
                onclick="location.reload()" />
        </form>
    </body>
</html>
```

JavaScript: jsb26-22.js

```
function applyOutside()
{
   var newItem = document.createElement("EM");
   newItem.id = newItem.uniqueID;
   // no 2nd argument--use default: outside
   document.getElementById("mySpan").applyElement(newItem);
}

function applyInside()
{
   var newItem = document.createElement("EM");
   newItem.id = newItem.uniqueID;
   document.getElementById("mySpan").applyElement(newItem, "inside");
}

function showHTML()
{
   alert(document.getElementById("myP").outerHTML);
}
```

在本示例中，在 span 元素内外应用 em 元素的显示结果相同。但可从 HTML 结果中看到，每个元素对元素层次结构的影响完全不同。

相关主题： insertBefore()，appendChild()和 insertAdjacent.Element()方法。

attachEvent("*eventName*", *functionRef*)，*detachEvent*("*eventName*", *functionRef*)

返回值： 布尔值。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera+，Chrome-

attachEvent()方法最初是给 IE 操作绑定事件的方式(参见配书光盘上的第 51 章)，现在是 W3C 中 addEventListener()事件绑定方法的 IE 替代方法。为了说明该方法的用法，应先考虑下面的例子，它是绑定事件处理程序的典型属性赋值方式：

```
myObject.onmousedown = setHilite;
```

使用 attachEvent()的版本如下：

```
myObject.attachEvent("onmousedown", setHilite);
```


更多资源请访问稀酷客(www.ckook.com)

该方法需要两个参数，第一个参数是事件名的字符串版本(不区分大小写)；第二个是触发该对象的事件时要调用的函数的引用。函数引用是一个没有引号、区分大小写的函数标识符，不带括号(所以不能在这个函数调用中传递参数)。

在事件属性绑定方法中使用 attachEvent()有一个不很明显的优势：使用 attachEvent()时，如果事件绑定成功，方法返回布尔值 true。如果函数引用失败，IE 将触发一个脚本错误，因此不要依赖于返回的 false 值，来捕获这类错误。另外，也不能保证对象能识别事件名。

如果使用 attachEvent()绑定事件处理程序与对象事件，则可用 detachEvent()方法中断该绑定，参数和 attachEvent()一样。detachEvent()方法不能中断某些事件的绑定，这些事件的关联通常是通过标记特性或事件属性设置的。

W3C DOM 事件模型也提供了与这些 IE 方法相同的功能：addEventListener()和 removeEvent Listener()。

示例

用 The Evaluator(参见第 4 章)创建一个匿名函数，在页面上第一个段落的 onmousedown 事件触发时调用该函数。首先在顶部文本框中将匿名函数赋给全局变量 a(已在 The Evaluator 中初始化)：

```
a = new Function("alert('Function created at " + (new Date()) + "')")
```

引号和圆括号很容易混淆，因此输入此表达式时要小心。成功输入表达式后，Results 框会显示函数的文本。现在，在顶部文本框中输入以下语句，将此函数分配给 myP 元素的 onmousedown 事件：

```
document.getElementById("myP").attachEvent("onmousedown", a)
```

如果成功，Results 框就显示 true。如果在 Evaluator 面板后面的第一个段落上按下鼠标，一个警告框将显示匿名函数的创建日期和时间(当计算 new Date()表达式时)。

现在，在顶部文本框中输入以下语句，从对象上断开事件关联：

```
document.getElementById("myP").detachEvent("onmousedown", a)
```

相关主题：addEventListener()、detachEvent()、dispatchEvent()、fireEvent()和 removeEvent Listener()方法；事件绑定(第 32 章)。

blur()，focus()

返回值：无。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

blur()方法从元素中移去焦点，而 focus()方法把焦点赋给元素。blur()和 focus()方法自从最早的脚本浏览器开始就存在，但不是所有可接收焦点的对象从一开始就可以使用这些方法，在 IE4 和 NN6 之前的浏览器中，这些方法主要用于 window 对象和表单控件元素。

1. 窗口

对于 window 对象，blur()方法(NN3+、IE4+，Opera)把引用的窗口放到所有其他打开窗口的后

更多资源请访问稀酷客(www.ckook.com)

面。如果其他浏览器附属窗口(例如电子邮件或新闻阅读窗口)是打开的，接收 blur ()方法的窗口也放在这些窗口之后。

在 Safari 和 Firefox 中，blur()方法把引用的窗口放到打开它的窗口后面。在 Chrome 中，window.blur()方法无效，就像在程序清单 26-23a 中单击 Put Me in Back 按钮一样。在这方面，window.focus()方法在 Chrome 中也无效，就像在程序清单 26-23a 中单击 Bring Main to Front 按钮一样。window.focus()方法在 IE8 中也无效。

警告：

window.blur()方法生效时，并不调整当前窗口在不同浏览器中的堆栈顺序，但窗口中的脚本可以调用另一个窗口的 focus()方法，把该窗口显示在前面(假定在两个窗口之间有一个脚本关联，如 window.opener 属性)。

在 Web 站点环境中创建另一个窗口时，必须注意窗口层的管理。因为浏览器窗口很容易由最轻微的鼠标单击激活，所以小窗口有时会突然隐藏在大窗口后。大多数没有经验的用户不会去查看 Windows 任务栏或浏览器菜单栏(如果浏览器有菜单栏)，来检查小窗口是否仍然是打开的，并将它激活。如果那个子窗口对站点设计很重要，则应该在每个窗口中显示一个按钮或其他控件，让用户在窗口间安全地切换，window.focus()方法会把被引用窗口放在所有窗口的前面。

除了在页面上提供独立的按钮，把隐藏窗口放在前面之外，还可以构建窗口打开函数，这样如果窗口已打开，该函数会自动将那个窗口放在最前面(如程序清单 26-23a 所示)，这就减轻了访问者管理窗口的负担。

使用这种方法的关键是确保窗口的引用是正确的。因此，如果子窗口需要使主窗口获得焦点，最好使用 window.opener 属性来引用主窗口。

2. 表单控制元素

blur()和 focus()方法基本上应用于面向文本的表单控件：text input、select 和 textarea 元素。

就像相机镜头在焦距之外会变得模糊一样，文本对象在失去焦点时也会变得模糊(在文本域外单击或按 Tab 键)。在脚本的控制下，blur()将取消在域内的任何选择，文本插入指针也会退出此域。在域外单击鼠标时，指针不会按照 Tab 顺序指向下一个域，同样，按下 Tab 键，使域失去焦点时，也会有这种情形。

文本对象获得焦点，意味着文本插入指针会在文本对象域上闪烁，而使一个域获得焦点就像打开它进行手工编辑一样。

使文本域或 textarea 获得焦点，不允许在域内的任何指定位置放置光标，光标经常出现在文本的开头。为使输入域移除现有的文本，需要依次使用 focus()和 select()方法。

在使用 focus()和 select()来预选文本域的内容从而进行编辑时，要注意：Internet Explorer 的许多版本由于内部计时问题，不能得到期望的结果。而通过 setTimeout()方法来启动获得焦点和选择操作，就可以解决这个问题(且仍和其他浏览器兼容)。

一个常见的设计要求是在文本域或 textarea 的末尾定位插入指针，使用户可马上开始在现有内容后添加文本。这可以在 IE4+中使用 TextRange 对象来实现。下面的脚本段把文本插入指针移到 ID 为 myTextarea 的 textarea 元素的末尾处：

更多资源请访问稀酷客(www.ckook.com)

```
var range = document.getElementById("myTextarea").createTextRange();
range.move("textedit");
range.select();
```

在组合 blur()或 focus()方法以及 onblur 和 onfocus 事件处理程序时应该非常小心，在事件处理程序显示警告框时尤其如此。这些事件和方法的许多组合可能会导致一个无限循环，在该无限循环中不可能完全消除警告对话框；而对于没有为文本框提供 disabled 属性的旧浏览器，这是一个有用的组合。下面的文本框事件处理程序可阻止用户在文本框中输入文本：

```
onfocus = "this.blur()";
```

一些操作系统和浏览器允许给元素提供焦点，如按钮(包括单选按钮和复选框)和超文本链接(包括 a 和 area 元素)。通常情况下，一旦这类元素拥有了焦点，在键盘上按下空格键，就可以完成与鼠标单击相同的动作，这对难以使用鼠标的人十分有用。

在 Win32 系统中，按钮焦点的一个副作用是，用户单击按钮后，按钮周围会一直显示一个虚点框，直到另一个对象得到焦点。在按钮中包括下列事件处理程序，可为实现了 onmouseup 事件处理程序的浏览器和对象消除这个虚点框：

```
onmouseup = "this.blur()";
```

IE5.5+认识到虚点框的副作用，所以允许脚本将元素的 hideFocus 属性设置为 true，使该元素在拥有焦点时隐藏虚点框。然而，对用户来说这是权宜之计，因为哪个对象具有焦点，并没有可视化的反馈信息。

3. 其他元素

对支持 focus()方法的其他类型的元素，可通过 scrollIntoView()方法使元素可见。在 IE 的 Windows 版本中，用户使 Link(a)和 area 元素获得焦点时，它们周围会显示虚点框。为了取消此效果，可使用与表单控件相同的事件处理程序(或 IE5.5+的 hideFocus 属性)：

```
onmouseup = "this.blur()";
```

示例

程序清单 26-23a 包含的示例使用 focus ()和 blur ()方法来改变窗口的焦点。此示例创建了一个双窗口环境，在每个窗口中都可将另一个窗口显示在前端。主窗口使用 window.open()返回的对象来获得对新窗口的引用。在子窗口中(其内容完全由 JavaScript 实时创建)，调用 self.opener 来引用主窗口，而 self.blur()对子窗口本身进行操作。使一个窗口模糊并将焦点转到另一窗口，会产生将窗口放至窗口列表之后的效果。

程序清单 26-23a　window.focus()和 window.blur()方法

HTML：jsb26-23.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <title>Window Focus() and Blur()</title>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-23.js"></script>
    </head>
    <body>
        <h1>Window focus() and blur() Methods</h1>
        <hr />
        <form>
            <input type="button" name="newOne" value="Show New Window"
                   onclick="makeNewWindow()" />
        </form>
    </body>
</html>
```

JavaScript: jsb26-23.js

```
// declare global variable name
var newWindow = null;

function makeNewWindow()
{
    // check if window already exists
    if (!newWindow || newWindow.closed)
    {
        // store new window object in global variable
        newWindow = window.open("","","width=250,height=250");
        // pause briefly to let IE3 window finish opening
        setTimeout("fillWindow()",100);
    }
    else
    {
        // window already exists, so bring it forward
        newWindow.focus();
    }
}

// assemble new content and write to subwindow
function fillWindow()
{
    var newContent = "<html><head><title>Another Sub Window<\/title><\/head>";
    newContent += "<body bgColor='salmon'>";
    newContent += "<h1>A Salmon-Colored Subwindow.<\/h1>";
    newContent += "<form><input type='button' value='Bring Main to Front'
        onclick='self.opener.focus()'>";
    newContent += "<input type='button' value='Put Me in Back'
        onclick='self.blur()'>";
    newContent += "<\/form><\/body><\/html>";
    // write HTML to new window document
    newWindow.document.write(newContent);
    newWindow.document.close();
}
```


更多资源请访问稀酷客(www.ckook.com)

在程序清单 26-23a 中，makeNewWindow()函数的关键成功因素是第一个条件表达式。由于在页面载入时 newWind 初始化为 null 值，因此这是其在首次调用函数时的值。但在首次打开子窗口后，给 newWind 分配了一个值(子窗口对象)，则即使用户关闭了窗口，该值仍保持不变。因此，该值不会自动变回 null。为了确定用户是否关闭了窗口，条件表达式还查看窗口是否关闭。如果窗口关闭了，就生成一个新的子窗口，该新窗口的引用值重新分配给 newWind 变量。另一方面，如果窗口引用存在并且窗口未关闭，focus()方法将该子窗口显示到前端。

在程序清单 26-23a 中可以看出，子窗口的内容完全由 JavaScript 实时创建。IE 和 Firefox 中一个很好的小工具是可以查看生成的源代码。源代码较难理解，因为所有代码都放在同一行上。程序清单 26-23b 格式化了这些生成的源代码，它们是完全由程序清单 26-23a 创建的代码。

程序清单 26-23b　由程序清单 26-23a 实时创建的源代码

```
<html>
   <head>
      <title>Another Sub Window</title>
   </head>
   <body bgcolor="salmon">
      <h1>A Salmon-Colored Subwindow.</h1>
      <form>
         <input value="Bring Main to Front" onclick="self.opener.focus()"
               type="button">
         <input value="Put Me in Back" onclick="self.blur()" type="button">
      </form>
   </body>
</html>
```

在第 36 章讨论文本对象的 select()方法时，使用了文本对象的 focus ()方法。

相关主题：window.open()方法，document.formObject.textObject.select()方法。

clearAttributes()

返回值：无。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

clearAttributes()方法从元素中删除除 name 和 id 值之外的所有特性，因此，样式和事件处理程序被删除了，由 HTML 源代码或脚本指定的自定义特性也删除了。clearAttributes()方法不会改变元素 attributes 集合的长度，因为该集合总是包含元素的所有可能的特性(见本章前面的 attributes 元素属性)。

如果希望为元素构造一套全新的特性且喜欢从头开始，这个方法是很方便的。然而，除非脚本马上给元素设置新特性，否则元素的外观将恢复为完全没有修饰过的形式，直到用户为它指定新特性为止。这意味着定位元素都变回它们的源代码顺序，直到指定新的定位样式为止。如果仅想改变元素的一个或多个特性值，使用 setAttribute()方法或调整相应的属性会更快。

为在 NN6+/Moz 中实现 IE5+中 clearAttributes()的结果，必须遍历元素的所有特性，并通过 removeAttribute()方法来移除除 id 和 name 外的所有特性。

更多资源请访问稀酷客(www.ckook.com)

示例

在应用 clearAttributes()之前和之后使用 The Evaluator(参见第 4 章)查看元素的特性。首先，在顶部文本框中输入以下语句，显示页面上表格元素的 HTML：

```
myTable.outerHTML
```

注意与<table>标记相关的特性。查看显示的表格，看看各种特性如 border 和 width 如何影响表格的显示。现在，在顶部文本框中输入以下语句，从这个元素中删除所有可删除的特性：

```
myTable.clearAttributes()
```

首先看一下表格。边框没有了，表格的宽度是显示内容所需的宽度，且没有单元格填充。最后，再次在表格的 outerHTML 中查看 clearAttributes()方法的结果：

```
myTable.outerHTML
```

源代码文件没有变化，但浏览器内存中的对象模型反映了用户所做的更改。

相关主题：attributes 属性；getAttribute()、setAttribute()、removeAttribute()、mergeAttributes()和 setAttributeNode()方法。

click()

返回值：无。

兼容性：WinIE4+，MacIE4+，NN2+，Moz+，Safari+，Opera+，Chrome+

click()方法允许脚本执行与单击元素几乎相同的动作。在 NN4 和 IE4 之前，在按钮上调用 click()方法，不会激活对象的 onclick 事件处理程序。如果期望按钮的 onclick 事件处理程序在脚本运行该“单击”操作时也起作用，这个方法就很重要。早期的浏览器版本必须直接调用事件处理程序语句。另外，因为是脚本“单击”一个按钮，所以并非所有平台上的所有按钮都会改变它们的外观。例如，当远程单击时，Mac 上的 NN4 不会改变复选框的状态。

如果想编写单击按钮的动作，则可直接调用事件处理函数。如果元素是单选按钮或复选框，它将立即处理状态的改变(例如，设置复选框的 checked 属性)，而不是期望浏览器设置状态。

示例

使用 The Evaluator(参见第 4 章)试验 click()方法。页面底部包括各类按钮，可在顶部文本框中输入以下语句，来单击复选框：

```
document.myForm2.myCheckbox.click()
```

如果使用最新的浏览器版本，则每次执行语句时，复选框就会在选中和未选中状态之间切换。

相关主题：onclick 事件处理程序。

cloneNode(*deepBoolean*)

返回值：Node 对象引用。

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

cloneNode()方法可以创建当前 Node 对象的准确副本。该副本没有父节点，与其他元素也没有任

更多资源请访问稀酷客(www.ckook.com)

何关系(当然，原始节点保持不变)。这个副本不会成为文档对象模型(节点树)的一部分，除非明确地将其插入或添加到页面上的某处。该副本包括所有元素特性，包括 id 特性。因为 cloneNode()方法的返回值是一个真正的 Node 对象，所以可在它处于非文档对象状态时，使用 Node 对象的方法操作它。

cloneNode()方法的 Boolean 参数指定节点的副本是包括所有的子节点(true)，还是仅包括节点本身(false)。例如，在复制段落元素时，其副本可以只包括源元素(标记对的替代物，包括首标记的特性)，而不包括其内容；也可以包括子节点，此时段落元素内的所有内容都是副本的一部分。在 IE5 中该参数是可选的(默认为 false)，但在其他 W3C 兼容浏览器中它是必需的。

示例

使用 The Evaluator(参见第 4 章)复制、重命名和添加源代码中的元素。首先复制名为 myP 的段落元素及其全部内容，在顶部文本框中输入以下语句：

```
a = document.getElementById("myP").cloneNode(true)
```

变量 a 现在保存着原始节点的副本，因此可输入以下语句，更改其 id 特性：

```
a.setAttribute("id", "Dolly")
```

如果想查看节点副本的属性，可在底部文本框中输入 a。所显示的属性列表取决于正在使用的浏览器；无论使用什么浏览器，都应该能找到 id 属性，其值为 Dolly。

最后，在顶部文本框中输入以下语句，将此新命名的节点添加到 body 元素的末尾：

```
document.body.appendChild(a)
```

向下滚动到页面底部，查看复制的内容。由于两个节点有不同的 id 特性，因此访问它们的脚本不会混淆它们。

相关主题：Node 对象(参阅第 25 章)；appendChild()、removeChild()、removeNode()、replace Child()和 replaceNode()方法。

compareDocumentPosition(nodeRef)

返回值：整型。

兼容性：WinIE-，MacIE-，NN6+，Moz1.4+，Safari+，Opera+，Chrome+

此方法确定一个节点相对于节点树上另一个节点的位置。更确切地说，就是比较参数提供的 nodeRef 对象(B 节点)与在其上调用方法的对象(A 节点)。该方法返回一个整数值，它可以包含表 26-11 中列出的一个或多个比较掩码。

表 26-11 比较返回标志

整数值	常量	说明
0		B 节点和 A 节点是同一个节点
1	DOCUMENT-POSITION-DISCONNECTED	两个节点间没有联系
2	DOCUMENT_POSITION_PRECEDING	B 节点在 A 节点之前
4	DOCUMENT-POSITION-FOLLOWING	B 节点在 A 节点之后
8	DOCUMENT__POSITION_CONTAINS	B 节点包含 A 节点(因此在 A 节点之前)

更多资源请访问稀酷客(www.ckook.com)

(续表)

整数值	常　　量	说　　明
16	DOCUMENT-POSITION-CONTAINED-BY	B 节点被 A 节点包含(因此在 A 节点之后)
32	DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC	比较结果由浏览器决定

compareDocumentPosition()方法返回的整数实际上是一个位掩码，所以表 26-11 中的值都是 2 的幂。于是，方法可以将它们加在一起，来返回多个比较值。例如，返回值 20 表明 B 节点被 A 节点包含(16)，同时 B 节点在 A 节点之后((4)。

相关主题：contains()方法。

componentFromPoint(x, y)

返回值：字符串。

兼容性：WinIE5+、 MacIE-、 NN-、 Moz-、 Safari-、 Opera-、 Chrome-

componentFromPoint()方法有助于完成一些与事件相关的任务，可在某些冲突检测中使用它(换言之，确定事件发生在特定的元素之内还是之外)。如果该元素有滚动条，该方法可提供事件的附加信息，例如用户激活了滚动条的哪个部分。

该方法的一个要点是可在任何用作参考点的元素上调用它。例如，要确定 mouseup 事件是否发生在 ID 为 myTable 的元素上，应调用如下方法：

```
var result = document.getElementById("myTable").componentFromPoint(↩
   event.clientX, event.clientY);
```

该方法的参数是 x 和 y 坐标，这些坐标不见得来自一个事件，但最可能的情况是将这个方法和某类事件链接起来，最好是鼠标事件(onclick 除外)。

该方法的返回值是一个字符串，它提供关于当前元素坐标点位置的信息。如果该坐标点在元素矩形的内部，返回值就是一个空字符串；相反，如果这个点完全在元素之外，返回值就是字符串 outside。对于滚动条，返回值的列表非常长(如表 26-12 所示)。

表 26-12　componentFromPoint()的返回值

返回的字符串	坐标点处的元素组件	返回的字符串	坐标点处的元素组件
(空)	在元素内容区域内	scrollbarDown	滚动条下箭头
Outside	在元素内容区域外	scrollbarHThumb	水平滚动条上的滚动块
handleBottom	底部的调整句柄	scrollbarLeft	滚动条左箭头
handleBottomLeft	左下方的调整句柄	scrollbarPageDown	滚动条向下翻页区域
handleBottomRight	右下方的调整句柄	scro1lbarPageLeft	滚动条向左翻页区域
handleLeft	左侧的调整句柄	scrollbarPageRight	滚动条向右翻页区域
handleRight	右侧的调整句柄	scro1lbarPageUp	滚动条向上翻页区域
handleTop	顶部的调整句柄	scrollbarRight	滚动条右箭头
handleTopLeft	左上方的调整句柄	scrol1barUp	滚动条上箭头
handleTopRight	右上方的调整句柄	scrol1barVThumb	垂直滚动条上的滚动块

更多资源请访问稀酷客(www.ckook.com)

对于大多数冲突或事件检测，不必使用这一方法。event 对象的 srcElement 属性返回一个对接收事件的对象的引用。

示例

程序清单 26-24 演示了如何使用 componentFromPoint()方法来准确地确定鼠标事件的发生位置。如程序清单所示，该方法与一个 textarea 对象相关联，该对象的大小做了专门调整，以同时显示垂直和水平滚动条。在单击 textarea 的不同区域和页面上的其他位置时，状态栏通过 componentFromPoint()方法显示事件的位置信息。

脚本使用 event.srcElement 属性和 componentFromPoint()方法来帮助区分如何将每种方法用于不同类型的事件处理。srcElement 属性最初用作筛选器，来决定状态栏是否进一步显示 textarea 元素的事件处理细节。

body 元素中的 onmousedown 事件处理程序会处理所有事件。IE 事件沿着对象层次结构向上冒泡(在此页面上没有取消任何事件)，因此所有 mousedown 事件最终都将到达 body 元素，然后，whereInWorld()函数比较来自任何元素的 mousedown 事件与文本区域的位置。

程序清单 26-24　使用 componentFromPoint()方法

HTML: jsb26-24.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>componentFromPoint() Method</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-24.js"></script>
   </head>
   <body onmousedown="whereInWorld()">
      <h1>componentFromPoint() Method</h1>
      <hr />
      <p>Tracking the mouseDown event relative to the textarea object. View
         results in status bar.
      </p>
      <form>
         <textarea id="myTextarea" wrap="off" cols="12" rows="4">
            This is Line 1
            This is Line 2
            This is Line 3
            This is Line 4
            This is Line 5
            This is Line 6
         </textarea>
      </form>
   </body>
</html>
```

JavaScript: jsb26-24.js

更多资源请访问稀酷客(www.ckook.com)

```
function whereInWorld(elem)
{
   var x = event.clientX;
   var y = event.clientY;
   var component = document.getElementById("myTextarea").componentFromPoint(x,y);
   if (window.event.srcElement == document.getElementById("myTextarea"))
   {
       if (component == "")
       {
           status = "mouseDown event occurred inside the element";
       }
       else
       {
           status = "mouseDown occurred on the element\'s " + component;
       }
   }
   else
   {
       status = "mouseDown occurred " + component + " of the element";
   }
}
```

相关主题： event 对象。

contains(*elementObjectReference*)

返回值： Boolean。

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari+-，Opera+，Chrome+

contains()方法报告当前对象在 HTML 包含层次结构中是否包含另一个已知的对象，注意这不是重叠元素的位置冲突检测，而是确定一个元素是否嵌套在另一个元素中。

contains()方法可深入到当前对象的层次结构中，找到需要的对象。实际上，contains()方法检查元素 all 数组中的所有元素。因此，该方法可用作 for 循环的快捷替换方式，来检查容器的每个嵌套元素，以检验是否存在特定的元素。

contains()方法的参数是一个对象引用。如果只有元素的 ID 字符串可用，则可使用 document.getElementById()方法来创建嵌套元素的有效引用。

注意：
元素总是包含自身。

示例

在 The Evaluator(参阅第 4 章)的顶部文本框中输入以下语句，看看 contains()方法如何响应以下各条语句中的对象组合：

```
document.body.contains(document.getElementById("myP"))
document.getElementById("myP").contains(document.getElementById("myEM"))
document.getElementById("myEM").contains(document.getElementById("myEM"))
document.getElementById("myEM").contains(document.getElementById("myP"))
```


更多资源请访问稀酷客(www.ckook.com)

尽可以测试此页面中的其他对象组合。

相关主题： item()方法，document.getElementById()方法。

createControlRange("*param*")

返回值： 整数 ID。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

createControlRange()方法用于为所选文本创建控制范围。尽管多个元素都实现了此方法，但它主要供 selection 对象使用，因此，应只在 selection 对象上使用它。

相关主题： selection 对象。

detachEvent()

参见 attachEvent()。

dispatchEvent(*eventObject*)

返回值： Boolean。

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

dispatchEvent()方法允许脚本对支持某事件的对象触发该事件，这是 W3C 事件模型的通用机制，早期浏览器有时用 click()和 focus()等对象方法来模仿该机制。

生成这类事件的过程类似于脚本生成一个新节点，并将该节点插入 DOM 中某个位置的过程。不过对于事件，对象是通过 document.createEvent()方法生成的 Event 对象。按这种方式生成的事件只是事件的描述信息，事件的细节使用事件对象的属性来提供，如其坐标或鼠标按钮，然后调用该目标对象的 dispatchEvent()方法，并把新创建的 Event 对象作为唯一的参数，将事件分派到目标对象。

dispatchEvent()方法返回的 Boolean 值不太好解释。浏览器通过对该对象和事件类型有效的事件传播通道(冒泡或捕获)，来跟踪被分派的事件。如果由该分派事件触发的事件监听器函数调用了 preventDefault()方法，dispatchEvent()方法就返回 false，以便表示事件没有激活对象的固有动作；否则，dispatchEvent()方法返回 true。注意，此返回值没有指定传播类型，也没有说明由于分派此事件会触发多少个事件监听器。

警告：

尽管 NN6 实现了 dispatchEvent()方法，但没有提供从头开始生成新事件的方法。如果试图通过 dispatchEvent()方法将一个现有事件重定向到另一个对象，浏览器很容易崩溃。换句话说，对于使用 dispatchEvent()方法的脚本，选择基于 Mozilla 的浏览器更合适。

示例

程序清单 26-25 演示了如何使用 W3C DOM 的 dispatchEvent()方法以编程方式触发事件。注意 doDispatch()函数中创建和初始化新鼠标事件的语法，基于 Mozilla 的浏览器对该语法的支持最可靠。此示例的操作与本章稍后的程序清单 26-26 相同，该程序清单演示了 IE5.5+中的对等方法 fireEvent()。

更多资源请访问稀酷客(www.ckook.com)

程序清单 26-25　使用 dispatchEvent()方法

HTML: jsb26-25.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Using the dispatchEvent() Method</title>
    <style type="text/css">
      #mySPAN
      {
        font-style:italic;
      }
      #ctrlPanel
      {
        font-weight:bold;
      }
    </style>
    <script type="text/javascript" src="../jsb-global.js"></script>
    <script type="text/javascript" src="jsb26-25.js"></script>
  </head>
  <body id="myBODY" onload="init()">
    <h1>dispatchEvent() Method</h1>
    <hr />
    <p id="myP">This is a paragraph <span id="mySPAN">(with a nested SPAN)</span>
             that receives click events.
    </p>
    <hr />
    <p>
      <span id="ctrlPanel">Control Panel</span>
    </p>
    <form name="controls">
      <p>
        <input type="checkbox" name="bubbleOn"
               onclick="event.stopPropagation()" />Cancel event bubbling.
      </p>
      <p>
        <input type="button" value="Fire Click Event on BODY"
               onclick="doDispatch('myBODY', event)" />
      </p>
      <p>
        <input type="button" value="Fire Click Event on myP"
               onclick="doDispatch('myP', event)" />
      </p>
      <p>
        <input type="button" value="Fire Click Event on mySPAN"
               onclick="doDispatch('mySPAN', event)" />
      </p>
    </form>
  </body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

JavaScript: jsb26-25.js

```
// assemble a couple event object properties
function getEventProps(evt)
{
   var msg = "";
   var elem = evt.target;
   msg += "event.target.nodeName: " + elem.nodeName + "\n";
   msg += "event.target.parentNode: " + elem.parentNode.id + "\n";
   msg += "event button: " + evt.button;
   return msg;
}

// onClick event handlers for body, myP, and mySPAN
function bodyClick(evt)
{
   var msg = "Click event processed in BODY\n\n";
   msg += getEventProps(evt);
   alert(msg);
   checkCancelBubble(evt);
}
function pClick(evt)
{
  var msg = "Click event processed in P\n\n";
  msg += getEventProps(evt);
  alert(msg);
  checkCancelBubble(evt);
}
function spanClick(evt)
{
  var msg = "Click event processed in SPAN\n\n";
  msg += getEventProps(evt);
  alert(msg);
  checkCancelBubble(evt);
}

// cancel event bubbling if checkbox is checked
function checkCancelBubble(evt)
{
  if (document.controls.bubbleOn.checked)
  {
  evt.stopPropagation();
  }
}

// assign onClick event handlers to three elements
function init()
{
  document.body.onclick = bodyClick;
```


更多资源请访问稀酷客(www.ckook.com)

```
  document.getElementById("myP").onclick = pClick;
  document.getElementById("mySPAN").onclick = spanClick;
}

// invoke dispatchEvent() on object whose ID is passed as parameter
function doDispatch(objID, evt)
{
  // create empty mouse event
  var newEvt = document.createEvent("MouseEvents");
  // initialize as click with button ID 3
  newEvt.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, false,
       false, false, false, 3, null);
  // send event to element passed as param
  document.getElementById(objID).dispatchEvent(newEvt);
  // don't let button clicks bubble
  evt.stopPropagation();
}
```

相关主题： fireEvent()方法。

doScroll("*scrollAction*")

返回值： 无。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

doScroll()方法通过触发元素的滚动条，来控制元素的滚动。doScroll()不将滚动条移动到特定位置，而是模仿单击滚动条的效果，其区别很细微。结果是触发一个 onscroll 事件，这是模拟滚动希望得到的结果。

doScroll()的字符串参数可以是以下值之一，表示要发生哪种滚动：scrollbarUp、scrollbarDown、scrollbarLeft、scrollbarRight、scrollbarPageUp、scrollbarPageDown、scrollbarPageLeft、scrollbar-PageRight、scrollbarHThumb 或 scrollbarVThumb。

dragDrop()

返回值： Boolean。

兼容性： WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

dragDrop()方法可以触发 ondragstart 事件，来启动一个鼠标拖放序列，返回值是一个 Boolean 值，表示用户是否释放了鼠标按钮(true)。

fireEvent("*eventType*"[, *eventObjectRef*])

返回值： Boolean。

兼容性： WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

一些对象包含仿效物理事件的方法，例如，click()和 focus()方法，而 WinIE5.5+允许脚本将有效事件引导到对象上来，以便实现这个机制，fireEvent()方法是实现这个机制的媒介。

该方法的一个必选参数是事件类型，并格式化为一个字符串。IE 事件类型的编码方式与事件处理程序的属性名相同(例如 onclick、onmouseover 等)。

更多资源请访问稀酷客(www.ckook.com)

第二个可选的参数是现有 event 对象的引用，这个对象可以是某个用户动作或系统动作触发的事件(即 fireEvent()方法在事件处理程序调用的一个函数中)，还可以是 IE5.5+中 document. createEventObject()方法创建的一个对象。在这两种情况下，提供现有 event 对象的目的是为了设置 event 对象的属性，这个对象由 fireEvent()方法创建。事件类型是方法的第一个参数，但如果要设置其他属性(例如，坐标或键盘键代码)，这些属性就从现有对象中获取。下面的例子创建了一个新的 mousedown 事件，把一些值填充到它的属性中，然后在页面的一个元素上触发该事件：

```
var newEvent = document.createEventObject();
newEvent.clientX = 100;
newEvent.clientY = 30;
newEvent.cancelBubble = false;
newEvent.button = 1;
document.getElementById("myElement").fireEvent("onmousedown", newEvent);
```

fireEvent()方法创建的事件与 IE 中常规的 window.event 对象一样，它们都有浏览器预设的几个重要的 event 对象属性。重要的是，cancelBubble 设置为 false，returnValue 设置为 true——与用户或系统引发的常规事件一样。这意味着如果想阻止事件冒泡，或阻止事件源元素的默认动作，事件处理函数就必须像 IE 中的常规事件处理程序那样设置这些 event 对象属性。

fireEvent()方法返回由事件的 returnValue 属性返回的 Boolean 值，如果在事件处理过程中 returnValue 属性设置为 false，则 fireEvent()方法返回 false；在常规的处理中，该方法返回 true。

W3C DOM (Level 2)事件模型包括 dispatchEvent()方法，以响应脚本创建的事件(以及使用 Event 对象方法创建的 event 对象)，它大致等效于 fireEvent()方法。

示例

程序清单 26-26 包含的脚本代码显示了如何使用 fireEvent()方法以编程方式触发事件。示例页面中的三个按钮可以将一个单击事件定向到三个元素中，这些元素都有为其定义的事件处理程序。采用这种方式触发的事件是人为的，通过 createEventObject()方法来生成。为了便于演示，这些脚本事件的 button 属性设置为 3，此属性值赋给 event 对象，最终定向到元素上。

启动事件冒泡功能时，通过 fireEvent()发送的事件的行为就像是在元素上物理单击一样。同样，如果禁用事件冒泡功能，第一个处理事件的处理程序就取消冒泡，对该事件不做进一步的处理。注意，事件冒泡功能是在处理事件的处理程序中取消的。为防止对复选框和操作按钮的单击触发 body 元素的 onclick 事件处理程序，需要立即关闭按钮的事件冒泡功能。

程序清单 26-26　使用 fireEvent()方法

```
HTML: jsb26-26.html
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Using the fireEvent() Method</title>
      <style type="text/css">
         #mySPAN
```


更多资源请访问稀酷客(www.ckook.com)

```
            {
               font-style:italic;
            }
            #ctrlPanel
            {
               font-weight:bold;
            }
         </style>
         <script type="text/javascript" src="../jsb-global.js"></script>
         <script type="text/javascript" src="jsb26-26.js"></script>
      </head>
      <body id="myBODY" onload="init()">
       <h1>fireEvent() Method</h1>
       <hr />
       <p id="myP">This is a paragraph <span id="mySPAN">(with a nested SPAN)</span>
                   that receives click events.
       </p>
       <hr />
       <p>
          <span id="ctrlPanel">Control Panel</span>
       </p>
       <form name="controls">
          <p>
             <input type="checkbox" name="bubbleOn"
                    onclick="event.cancelBubble=true" />Cancel event bubbling.
          </p>
          <p>
             <input type="button" value="Fire Click Event on BODY"
                    onclick="doFire('myBODY')" />
          </p>
          <p>
             <input type="button" value="Fire Click Event on myP"
                    onclick="doFire('myP')" />
          </p>
          <p>
             <input type="button" value="Fire Click Event on mySPAN"
                    onclick="doFire('mySPAN')" />
          </p>
       </form>
      </body>
   </html>
```

JavaScript: jsb26-26.js

```
// assemble a couple event object properties
function getEventProps()
{
   var msg = "";
   var elem = event.srcElement;
   msg += "event.srcElement.tagName: " + elem.tagName + "\n";
   msg += "event.srcElement.id: " + elem.id + "\n";
```


更多资源请访问稀酷客(www.ckook.com)

```
    msg += "event button: " + event.button;
    return msg;
}

// onClick event handlers for body, myP, and mySPAN
function bodyClick()
{
    var msg = "Click event processed in BODY\n\n";
    msg += getEventProps();
    alert(msg);
    checkCancelBubble();
}
function pClick()
{
   var msg = "Click event processed in P\n\n";
   msg += getEventProps();
   alert(msg);
   checkCancelBubble();
}
function spanClick()
{
   var msg = "Click event processed in SPAN\n\n";
   msg += getEventProps();
   alert(msg);
   checkCancelBubble();
}

// cancel event bubbling if checkbox is checked
function checkCancelBubble()
{
   event.cancelBubble = document.controls.bubbleOn.checked;
}

// assign onClick event handlers to three elements
function init()
{
   document.getElementById("myBODY").onclick = bodyClick;
   document.getElementById("myP").onclick = pClick;
   document.getElementById("mySPAN").onclick = spanClick;
}

// invoke fireEvent() on object whose ID is passed as parameter
function doFire(objID)
{
   var newEvt = document.createEventObject();
   newEvt.button = 3;
   document.all(objID).fireEvent("onclick", newEvt);
   // don't let button clicks bubble
   event.cancelBubble = true;
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题： dispatchEvent()方法。

focus()

(参见 blur())。

getAdjacentText("*position*")

返回值： 字符串。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

getAdjacentText()方法允许提取元素对象的纯文本组件(换句话说，不提取任何 HTML 标记信息)。此方法的唯一参数是 4 个不区分大小写的字符串常量中的一个，这些常量指定提取文本的起始位置(相对于当前对象)，其值如表 26-13 所示。

表 26-13　值

参 数 值	说　明
beforeBegin	紧靠元素标记之前的文本，直到前一个标记
afterBegin	在元素标记内开始的文本，直到下一个标记(无论它是一个嵌套元素还是元素的结束标记)
beforeEnd	紧靠元素结束标记之前的文本，直到前一个标记(无论它是一个嵌套元素还是元素的开始标记)
afterEnd	紧跟元素结束标记的文本，直到下一个标记

如果当前对象没有嵌套元素，afterBegin 和 beforeEnd 版本的返回值就和对象 innerText 属性相同；如果当前对象直接包含在另一个元素中(例如，tr 元素中的 td 元素)，在元素的开头之前或元素的结尾之后没有文本，所以这些值返回空字符串。

该方法返回的字符串和 W3C DOM 中的文本段节点值大致相当，但 IE5+把这些数值看成字符串类型，而不是文本节点类型。这 4 个版本的 W3C DOM 等效语句如下：

```
document.getElementById("objName").previousSibling.nodeValue
document.getElementById("objName").firstChild.nodeValue
document.getElementById("objName").lastChild.nodeValue
document.getElementById("objName").nextSibling.nodeValue
```

示例

使用 The Evaluator(参见第 4 章)查看该文档中的 myP 和嵌套元素 myEM 的 4 种相邻文本。在顶部文本框中输入以下各条语句，并查看结果：

```
document.getElementById("myP").getAdjacentText("beforeBegin")
document.getElementById("myP").getAdjacentText("afterBegin")
document.getElementById("myP").getAdjacentText("beforeEnd")
document.getElementById("myP").getAdjacentText("afterEnd")
```

第一条和最后一条语句返回空字符串，因为 myP 元素没有包含它的文本片断；afterBegin 语句返回 myP 元素的文本片断，直到(但不包括)嵌套在其中的 EM 元素；beforeEnd 字符串返回

更多资源请访问稀酷客(www.ckook.com)

从嵌套的 EM 元素结尾开始直到 myP 结尾的所有文本。

下面分析嵌套 myEM 元素的情况：

```
document.getElementById("myEM").getAdjacentText("beforeBegin")
document.getElementById("myEM").getAdjacentText("afterBegin")
document.getElementById("myEM").getAdjacentText("beforeEnd")
document.getElementById("myEM").getAdjacentText("afterEnd")
```

由于此元素没有嵌套元素，因此 afterBegin 和 beforeEnd 字符串是相同的——与元素的 innerText 属性值相同。

相关主题：childNodes，data，firstChild，lastChild，nextSibling，nodeValue 和 previousSibling 属性。

getAttribute("*attributeName*"[, *caseSensitivity*])

返回值：见文本。

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

getAttribute()方法返回当前对象的特定特性值，可以用作获取对象属性的替代方法，特别是脚本将特性名显示为字符串时(而不是对象及其属性的引用)。因此，下面的语句会得到相同的数据：

```
var mult = document.getElementById("mySelect").multiple;
var mult = document.getElementById("mySelect").getAttribute("multiple");
```

getAttribute()的返回值类型是字符串(包括非引用数值的特性值)或 Booleans(例如 select 元素对象的 multiple 属性)。

注意：

W3C DOM Level 2 标准推荐用 getAttribute()和 setAttribute()来读写元素对象的属性值，而不是通过相应的属性来读写。虽然可对 XML 元素使用这些方法，但 DOM 标准会发出冲突信号，因为它为 HTML 元素对象定义了所有属性。当然，浏览器将来可能支持通过属性进行访问，因此不一定要改变访问属性值的方式。

所有支持 getAttribute()方法的浏览器都需要一个参数，它是特性名的字符串。默认情况下，这个参数不区分大小写。但注意它会影响在文档中赋给 HTML 或 XML 元素的自定义特性。当自定义特性变成对象属性时，特性名自动转换为小写。因此，特性名必须避免重用，即使在源代码赋值中也不要使用仅大小写字母不同的特性名。

IE 包括该方法的可选扩展，允许第二个参数指定第一个参数的大小写形式。第二个参数默认为 false，表示第一个参数不区分大小写；true 则表示第一个参数区分大小写。只有使用 setAttribute()把一个参数添加到现有对象中，且该方法的 IE 版本必定区分大小写，它才起作用。setAttribute ()默认考虑特性名的大小写，setAttribute()对 IE attributes 属性的影响请参见本章后面讨论的 setAttribute()方法。

更多资源请访问稀酷客(www.ckook.com)

示例

使用 The Evaluator(参见第 4 章)对页面中的元素试验 getAttribute()方法。可以在顶部文本框中输入以下示例语句，来查看特性值：

```
document.getElementById("myTable").getAttribute("cellpadding")
document.getElementById("myTable").getAttribute("border")
```

相关主题：attributes 属性；document.createAttribute()方法，setAttribute()方法。

getAttributeNode("attributeName")

返回值：特性 Node 对象。

兼容性：WinIE6+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

在 W3C DOM 中，attribute 对象继承了 Node 对象的所有属性(参阅第 25 章)。attribute 对象代表特性的“名-值”对，它在对象的标记中明确定义。当使用 XML 而不是 HTML 时，把特性作为节点对象更加重要，但在文档的 W3C DOM 面向对象视图环境中，理解特性节点更有益。重点在于，attribute 节点不是文档层次结构的节点，因此，attribute 节点不是定义特性的元素的子节点。

寻址对象的 attributes 属性内容时，特性的“节点性”很重要。W3C 将 attributes 属性建立在 DOM 的正式结构上，返回一个称为(内部的)命名节点图的对象。与数组一样，命名节点图也有 length 属性(便于 for 循环遍历节点图)，还有几个方法可以在节点图内插入、删除、读写特性“名-值”对。

attribute 对象继承了 Node 对象的所有属性。表 26-14 列出了 attribute 对象的属性。

表 26-14 W3C DOM 兼容浏览器的 attribute 对象属性

Attributes	nodeType
childNodes	nodeValue
firstChild	ownerDocument
lastChild	parentNode
name	previousSibling
nextSibling	specified
nodeName	value

下面该解释 W3C DOM 的 getAttributeNode()方法了，它返回一个 W3C DOM attribute 对象，该方法的唯一参数是不区分大小写的特性名字符串。使用表 26-14 所示的属性可以得到或设置特性值。当然，HTML 特性常显示为 HTML 元素的属性，因此更容易直接读写对象的属性。

示例

使用 The Evaluator(参见第 4 章)来研究 getAttributeNode()方法。textarea 元素 Results 提供了多个特性以供检查。由于此方法返回一个对象，所以在底部的文本框中输入以下语句，就可以查看方法返回的 attribute 节点对象的属性：

更多资源请访问稀酷客(www.ckook.com)

```
document.getElementById("myTextArea").getAttributeNode("cols")
document.getElementById("myTextArea").getAttributeNode("rows")
document.getElementById("myTextArea").getAttributeNode("wrap")
document.getElementById("myTextArea").getAttributeNode("style")
```

除最后一条语句外，其他所有语句都显示了每个 attribute 节点对象的属性列表，但最后一条语句没有返回值，因为没有为元素指定 style 特性。

也可以使用 The Evaluator 显示 getAttributeNode()方法返回的 attribute 对象属性。为此，在顶部文本框中输入以下语句：

```
document.getElementById("myTextArea").getAttributeNode("cols").value
```

相关主题：attributes 属性；getAttribute()、removeAttributeNode()和 setAttributeNode()方法。

getAttributeNodeNS("namespaceURI", "localName")

返回值：attribute 节点对象。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+-，Opera+，Chrome-

此方法返回一个 W3C DOM attribute 对象。它的第一个参数是 URI 字符串，该字符串与分配给文档中的标签的 URI 相匹配，第二个参数是要获得的特性的本地名称部分。

相关主题：attributes、namespaceURI、localName 属性；getAttributeNode()方法，setAttributeNodeNS()方法。

getAttributeNS("namespaceURI", "localName")

返回值：(参见正文)。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+-，Opera+，Chrome+

特性名由文档中的 XML 命名空间来定义时，此方法返回当前对象的该特性值。该方法的第一个参数是 URI 字符串，该字符串匹配赋予命名空间标签的 URI(在文档前面定义的标记中)，第二个参数是要获得其值的特性的本地名称部分。

getAttributeNS()的返回值类型是字符串(包括赋予属性的未加引号的数值)或 Boolean(例如，select 元素对象的 multiple 属性)。在 W3C DOM、Netscape、Safari 和 Opera 中，返回值总是字符串。

相关主题：attributes、namespaceURI、localName 属性；getAttribute()、getAttributeNodeNS()、setAttributeNodeNS()方法。

getBoundingClientRect()，getClientRects()

返回值：getBoundingClientRect()返回 TextRectangle 对象；getClientRects()返回 TextRectangle 对象数组。

兼容性：WinIE5+，MacIE-，NN-，Moz+-，Safari+-，Opera+，Chrome+

从 IE5+开始，大多数现代浏览器都给每个包含内容的元素指定一个矩形，来描述页面上元素占据的空间，该矩形叫做边框矩形，表示为 TextRectangle 对象(内容甚至可以是一个图像或其他类型的对象)。TextRectangle 对象有 4 个属性(top、left、bottom 和 right)，它们定义了矩形

更多资源请访问稀酷客(www.ckook.com)

的像素坐标。在不支持 TextRectangle 对象的旧浏览器中，使用 offsetTop、offsetLeft、offsetHeight 和 offsetWidth 可以获得这些信息。getBoundingClientRect()方法返回一个 TextRectangle 对象，以描述当前对象的边框矩形。下例可以访问对象边框矩形的一个尺寸：

```
var parTop = document.getElementById("myP").getBoundingClientRect().top;
```

对于包含文本(例如段落)的元素，用于元素中每行文本的 TextRectangle 的大小影响着边框矩形的大小。例如，如果段落包含两行，第二行只延伸到第一行的中间，则第二行的 TextRectangle 对象和第二行的实际文本一样宽。但因为第一行接近右页边，边框矩形的宽度就由第一行的 TextRectangle 对象控制。因此，元素的边框矩形和它最宽的行一样宽，和段落中所有 TextRectangle 对象的总高度一样高。

另一个方法 getClientRects()，可以得到元素每行的 TextRectangle 对象的集合。该方法返回当前对象在调用该方法时包含的所有 TextRectangle 对象的数组。每个 TextRectangle 对象都有自己的 top、left、bottom 和 right 坐标属性。接着，就可以遍历这个数组中的所有对象，计算每一行的像素宽度。如果希望确定整个集合的总高度和/或最大宽度，就可以使用 getBoundingClient Rect()方法作为一种快捷方式。

getClientRects()返回的集合的长度在不同的浏览器中有所不同。对于基于 WebKit 的浏览器和基于 Mozilla 的浏览器的早期版本，在 getClientRects()返回的集合中，仅给通过参数传送来的每个块元素包含一个 TextRectangle 对象。在 IE 和基于 Mozilla 的浏览器的最新版本中，只有给块元素指定宽度，getClientRects()才返回这样的集合。在基于 Presto 的浏览器中，getClientRects()总是返回只包含一个 TextRectangle 对象的集合，而无论该方法涉及多少个元素(不管它们是否是块元素)。而参数不是由块元素指定的(例如，在程序清单 26-27a 中，传送给 getClientRects()的参数是一个 span 元素)。如果参数是一个块元素(如 div)，则只有在 IE 浏览器中，getClientRects()才给每一行返回一个 TextRectangle 对象集合。显然，这会令脚本人员非常沮丧。在各种浏览器中查看程序清单 26-27a 和 26-27b 时要注意这一点。

示例

程序清单 26-27a 只使用了 getClientRects()方法(稍后的程序清单 26-27b 使用了两个方法)。传送给 getClientRects()方法的元素是一个 span，它包含两个段落和一个有序列表。它包含的区域显示为蓝色。还有一个按钮生成一个警告框，显示该方法返回的集合的长度。

程序清单 26-27a　使用 getClientRects()方法

HTML: jsb26-27a.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>getClientRects() Method</title>
      <style type="text/css">
         #main
         {
            color:blue;
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
        </style>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-27a.js"></script>
    </head>
    <body>
        <h1>getClientRects() Method</h1>
        <hr />
        <p>The area we're interested in is colored blue.</p>
        <form>
            <button onclick="howMany();">How many TextRectangle objects
                                        are in the collection?</button>
        </form>
        <span id="main">
            <p>Lorem ipsum dolor sit amet, consectetaur adipisicing
                elit, sed do eiusmod tempor incididunt ut labore et dolore magna
                aliqua. Ut enim adminim veniam, quis nostrud exercitation
                ullamco:
            </p>
            <ul>
                <li>laboris</li>
                <li>nisi</li>
                <li>aliquip ex ea commodo</li>
            </ul>
            <p>Duis aute irure dolor in reprehenderit involuptate velit esse cillum
                dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
                non proident, sunt in culpa qui officia deseruntmollit anim id est
                laborum. Et harumd und lookum like Greek to me, dereud facilis est
                er expedit distinct.
              </p>
        </span>
    </body>
</html>
```

JavaScript: jsb26-27a.js

```
function howMany()
{
    var clientRects = document.getElementById("main").getClientRects();
    alert("The length of the rectangle returned by getClientRects() is ="
          + clientRects.length + ".");
}
```

程序清单 26-27b 使用了 getBoundingClientRect()和 getClientRects()方法，以说明它们的区别。该程序清单可在所有主流浏览器中工作，但因为如前所述，getClientRects()方法返回的集合在不同的浏览器中是不同的，所以只能在 IE 中看到有意义的结果，且应只在 IE 中使用它们。但注意，尽管这两个方法不必一起使用，但不管使用什么浏览器，仍可以仅使用 getBoundingClientRect()方法完成许多工作。

与程序清单 26-27a 一样，一组元素在名为 main 的 span 元素中组合在一起；该组由两个段

更多资源请访问稀酷客(www.ckook.com)

落和一个无序列表构成，它们的文本都显示为蓝色。

两个控件允许将底部突出显示的矩形的位置设置为用户选择的任何行。复选框允许把突出显示的矩形设置为仅与行一样宽，还是与整个 span 元素的边框矩形一样宽。

所有代码都在 hilite()函数中，select 和复选框元素都调用了此函数。在函数的前面，为 main 元素调用了 getClientRects()方法，以获得整个元素的所有 TextRectangles 的快照。当脚本需要为在 select 元素中选择的某行获取矩形的坐标时，就可以使用此数组。

只要用户从 select 列表中选择一个数字，且该值小于 clientRects 中的 TextRectangle 对象总数，函数就开始计算底部突出显示的黄色矩形的尺寸和位置。选中 Full Width 复选框后，就从 getBoundingClientRect()方法中获取左、右坐标，因为本例需要整个 span 元素的矩形空间，否则就要从 clientRects 数组选择的矩形中获得 left 和 right 属性。

接下来要将位置和尺寸值赋给 hiliter 对象的 sty1e 属性。顶部和底部总是固定到所选的行，因此会遍历 clientRects 数组，以获得所选项的 top 和 bottom 属性。前面计算的 left 值赋给 hiliter 对象的 pixelLeft 属性，同时从 right 坐标中减去 left 来计算宽度。注意，top 和 left 坐标还考虑了整个文档体的垂直或水平滚动。如果将窗口调小，自动换行会使原来的行计数失效。但是，在 onresize 事件处理程序中调用 hilite()，会将当前所选行号应用到调整窗口大小后归入该行的内容。

由于 hiliter 元素的 z-index 样式属性设置为-1，因此该元素总是显示在页面主要内容的下面。如果所选的行号超过了 main 元素中的当前行数，将会隐藏 hiliter 元素。

程序清单 26-27b　使用 getBoundingClientRect()

HTML: jsb26-27b.html

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>getClientRects() and getBoundClientRect() Methods</title>
      <style type="text/css">
         #main
         {
            color:blue;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-27b.js"></script>
   </head>
   <body id="myBody" onresize="hilite()">
      <h1>getClientRects() and getBoundClientRect() Methods</h1>
      <hr />
      <form>
         Choose a line to highlight:
         <select id="choice" onchange="hilite()">
            <option value="1">1</option>
            <option value="2">2</option>
            <option value="3">3</option>
            <option value="4">4</option>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <option value="5">5</option>
            <option value="6">6</option>
            <option value="7">7</option>
            <option value="8">8</option>
            <option value="9">9</option>
            <option value="10">10</option>
            <option value="11">11</option>
            <option value="12">12</option>
            <option value="13">13</option>
            <option value="14">14</option>
            <option value="15">15</option>
         </select>
         <br />
         <input name="fullWidth" type="checkbox" onclick="hilite()" />
         Full Width (bounding rectangle)
      </form>
      <span id="main">
         <p>Lorem ipsum dolor sit amet, consectetaur adipisicing
            elit, sed do eiusmod tempor incididunt ut labore et dolore magna
            aliqua. Ut enim adminim veniam, quis nostrud exercitation
            ullamco:
         </p>
         <ul>
            <li>laboris</li>
            <li>nisi</li>
            <li>aliquip ex ea commodo</li>
         </ul>
         <p>Duis aute irure dolor in reprehenderit involuptate velit esse cillum
            dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat
            non proident, sunt in culpa qui officia deseruntmollit anim id est
            laborum Et harumd und lookum like Greek to me, dereud facilis est
            er expedit distinct.
         </p>
      </span>
      <div id="hiliter"
      style="position:absolute; background-color:yellow;i
             z-index:-1; visibility:hidden">
      </div>
   </body>
</html>
```

JavaScript: jsb26-27b.js

```
function hilite()
{
   var hTop, hLeft, hRight, hBottom, hWidth;
   var selectList = document.getElementById("choice");
   var n = parseInt(selectList.options[selectList.selectedIndex].value) - 1;
   var clientRects = document.getElementById("main").getClientRects();
   var mainElem = document.getElementById("main");
```


更多资源请访问稀酷客(www.ckook.com)

```
        if (n >= 0 && n < clientRects.length)
        {
           if (document.forms[0].fullWidth.checked)
           {
              hLeft = mainElem.getBoundingClientRect().left;
              hRight = mainElem.getBoundingClientRect().right;
           }
           else
           {
              hLeft = clientRects[n].left;
              hRight = clientRects[n].right;
           }
           document.getElementById("hiliter").style.pixelTop =
                clientRects[n].top + document.getElementById("myBody").scrollTop;
           document.getElementById("hiliter").style.pixelBottom
                = clientRects[n].bottom;
           document.getElementById("hiliter").style.pixelLeft = hLeft
                + document.getElementById("myBody").scrollLeft;
           document.getElementById("hiliter").style.pixelWidth = hRight - hLeft;
           document.getElementById("hiliter").style.visibility = "visible";
        }
        else if (n > 0)
        {
        alert("The content does not have that many lines.");
        document.getElementById("hiliter").style.visibility = "hidden";
    }
}
```

相关主题：TextRectangle 对象(配书光盘中的第 33 章)。

getElementsByTagName("*tagName*")

返回值：元素对象数组。

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

getElementsByTagName()方法返回一个包含当前对象中所有元素的数组，该对象的标记匹配方法的唯一参数的标记名，标记名参数必须是字符串，且不区分大小写。数组中返回的元素组只包括当前对象包含的元素，因此，如果文档中有两个 table 对象，在其中一个 table 上调用 getElementsByTagName("td")方法时，返回的表单元格元素列表就是当前 table 对象的单元，当前元素并不包含在返回的数组中。

对于MacIE5、WinIE6+和所有其他支持该方法的浏览器，此方法可以使用通配符("*")来匹配后代元素，而不考虑标记名。所得到的元素数组几乎与 IE4+通过 document.all 集合返回的数组相同。

示例

使用 The Evaluator(参见第 4 章)来试验 getElementsByTagName()方法。在顶部文本框中一次输入一条以下语句，并分析结果：

更多资源请访问稀酷客(www.ckook.com)

```
document.body.getElementsByTagName("div")
document.body.getElementsByTagName("div").length
document.getElementById("myTable").getElementsByTagName("td").length
```

由于 getElementsByTagName()方法返回一个对象数组，所以可以将这类返回值作为一个有效的元素引用：

```
document.getElementsByTagName("form")[0].getElementsByTagName("input").length
```

相关主题： getElementByTagNameNS()、getElementById()、tags()方法。

getElementsByTagNameNS("*namespaceURI*", "*localName*")

返回值： 元素对象数组。

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

此方法返回的数组包含在两个参数中指定的当前对象(在 XML 文档中)的所有元素。方法的第一个参数是与分配给文档中标签的 URI 相匹配的 URI 字符串，第二个参数是要获取其值的属性的本地名称部分。

getAttributeNS()的返回值类型是字符串(包括赋予特性的未加引号的数值)或 Boolean(例如，select 元素对象的 multiple 属性)。

相关主题： attributes、namespaceURI、localName 属性；getElementsByTagNameNS()、getElementById()、tags()方法。

getExpression("*attributeName*")

返回值： 字符串。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

getExpression()方法返回表达式的文本，表达式通过 setExpression()方法赋予元素的特性。返回值不是表达式的值，而是表达式本身。如果想确定表达式的当前值(假定用到的变量在脚本的作用域内)，可使用 eval()函数来调用 getExpression()，将字符串转换为一个 JavaScript 表达式，并返回求得的结果。

该方法需要把特性名的字符串版本作为参数。

示例

查看程序清单 26-32 中的 setExpression()方法，此程序清单演示了 getExpression()的返回值。

相关主题： document.recalc()、removeExpression()、setExpression()方法。

getFeature("*feature*", "*version*")

返回值： 对象。

兼容性： WinIE-，MacIE-，NN-，Moz1.7.2+，Safari-，Opera+，Chrome-

根据 W3C DOM 规范的要求，getFeature()方法接受一个脚本功能和版本，返回一个为该功能实现了 API 的对象。此方法的功能参数示例是 Core 和 Events，它们对应于 DOM 模块。

直到 Mozilla 1.8.1 (Firefox 2.0)，getFeature()方法才返回一个对象，但该对象没有向脚本提供 API 功能。

更多资源请访问稀酷客(www.ckook.com)

相关主题：implementation.hasFeature()方法。

getUserData("*key*")

返回值：对象。

兼容性：WinIE-，MacIE-，NN6-，Moz1.7.2+，Safari+-，Opera-，Chrome+

getUserData()方法允许访问已与节点关联的自定义用户数据。指定的节点可以有多个用户数据对象，在这种情况下每个用户数据对象通过一个文本关键字来识别。此关键字要传递给 getUserData()，获得用户数据对象。直到 Mozilla 1.8.1 (Firefox 2.0)，此方法都只实现了一部分，因此仍然不可用。

hasAttribute("*attributeName*")

返回值：Boolean

兼容性：WinIE+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如果当前对象的一个特性名称与方法唯一的参数相匹配，hasAttribute()方法就返回 true；否则返回 false。

相关主题：hasAttributeNS()，hasAttributes()方法。

hasAttributeNS("*namespaceURI*", "*localName*")

返回值：Boolean。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如果当前对象有两个参数表示的特性，hasAttributeNS()方法就返回 true；否则返回 false。方法的第一个参数是与分配给文档中标签的 URI 相匹配的 URI 字符串，第二个参数是要获取其值的特性的本地名称部分。

相关主题：attributes、namespaceURI、localName 属性；hasAttribute()、hasAttributes()方法。

hasAttributes()

返回值：Boolean。

兼容性：WinIE+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如果当前对象有在标记内明确指定的特性，hasAttributes()方法返回 true；否则返回 false。

相关主题：hasAttribute()、hasAttributeNS()方法。

hasChildNodes()

返回值：Boolean。

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

如果当前对象有嵌套的子节点，hasChildNodes()方法返回 true；否则返回 false。子节点和子元素未必相同，因此当前对象有至少一个子节点时，下列两个表达式都返回 true：

```
document.getElementById("myObject").hasChildNodes()
document.getElementById("myObject").childNodes.length > 0
```


更多资源请访问稀酷客(www.ckook.com)

不能用下列语句(它使用 children 属性)与第二个表达式互换使用：

```
document.getElementById("myObject").children.length > 0
```

一般在条件表达式中使用 hasChildNodes()方法来确保在执行操作前，存在子节点：

```
if (document.getElementById("myObject").hasChildNodes()
{
    // [statements that apply to child nodes]
}
```

示例

使用 The Evaluator(参见第 4 章)试验 hasChildNodes()方法。如果在顶部文本框中输入以下语句：

```
document.getElementById("myP").hasChildNodes()
```

返回值为 true。可以通过读取 childNodes 数组的 length 属性，来确定有多少个节点：

```
document.getElementById("myP").childNodes.length
```

此表达式显示一共有三个节点：两个文本节点和它们之间的 em 元素。检查第一个文本节点是否有子节点：

```
document.getElementById("myP").childNodes[0].hasChildNodes()
```

结果为 false，因为文本片断没有任何嵌套节点。检查 myP 元素的第二个子节点 em 元素：

```
document.getElementById("myP").childNodes[1].hasChildNodes()
```

结果为 true，因为 em 元素有一个嵌套的文本片断节点。毫无疑问，语句：

```
document.getElementById("myP").childNodes[1].childNodes.length
```

得到的节点数为 1。也可以直接在引用中访问 em 元素：

```
document.getElementById("myEM").hasChildNodes()
document.getElementById("myEM").childNodes.length
```

如果要查看 em 元素中的文本片断节点的属性，可在底部文本框中输入以下语句：

```
document.getElementById("myEM").childNodes[0]
```

文本片断的 data 和 nodeValue 属性都返回文本 all。

相关主题：childNodes 属性；appendChild()、removeChild()、replaceChild()方法。

insertAdjacentElement("*location*", *elementObject*)

返回值：对象。

兼容性：WinIE5+，　MacIE-，　NN-，　Moz-，Safari+，Opera+，　Chrome+

innertAdjacentElement()方法在相对于当前对象的特定位置插入一个元素对象(来自不同的源)。该方法需要两个参数，第一个必选参数是用于插入元素的 4 个位置，它们并不区分大小写，

更多资源请访问稀酷客(www.ckook.com)

如表 26-15 所示：

表 26-15　位置及说明

位　　置	说　　明
beforeBegin	当前元素的始标记之前
afterBegin	起始标记之后，但在所有其他嵌套内容之前
beforeEnd	结束标记之前，但在所有其他嵌套内容之后
afterEnd	结束标记之后

这些位置是相对于当前对象的，当前对象的元素类型(块级或内嵌元素)对所插入元素的显示有很大影响。例如，使用 document.createElement()创建 b 元素，并将一些内部文本赋给它。接着使用 insertAdjacentElement()在 p 元素的文本前插入这个 b 元素。因为 p 元素是一个块级元素，所以 beforeBegin 将 b 元素放在 p 元素的首标记前。这意味着 p 元素开头的文本行以粗体显示，因为<p>标记在它的容器的左边界开始一个新块(除非受样式表的控制)。结果，HTML 语句如下：

```
<b>The new element.</b><p>The original paragraph element.</p>
```

为了使新的 b 元素成为 p 元素的一部分(在现有 p 元素内容之前)，应使用 afterBegin 参数。其 HTML 语句如下：

```
<p><b>The new element.</b>The original paragraph element.</p>
```

为了演示 4 个位置类型，下面是 beforeEnd 的结果：

```
<p>The original paragraph element. <b>The new element.</b></p>
```

这是 afterEnd 的结果：

```
<p>The original paragraph element.</p><b>The new element.</b>
```

要插入的对象是元素对象的引用，对象引用可以来自其值为元素对象的任何表达式，也可能来自 document.createElement()方法的结果。document.createElement()创建的对象刚开始没有内容，所有特性值都设为默认值。此外，对象通过引用传递到 insertAdjacentElement()中，意味着只有对象的一个实例。如果在两个地方用两个语句插入此对象，对象将从第一个位置移到第二个位置；如果需要复制一个现有对象，以使原始对象不移动，或不受此方法的干扰，则应使用 cloneNode()方法，并指定 true 参数来获得节点的所有嵌套内容。

示例

在 WinIE5+中用 The Evaluator(参见第 4 章)试验 insertAdjacentElement()方法，以便在 myP 元素上插入一个新的 hl 元素。

所有动作都要求在顶部文本框中输入一组语句。首先将新元素存储在全局变量 a 中：

```
a = document.createElement("h1")
```

给新对象指定一些文本：

更多资源请访问稀酷客(www.ckook.com)

```
a.innerText = "New Header"
```

在 myP 对象的开始位置之前插入此元素：

```
myP.insertAdjacentElement("beforeBegin", a)
```

注意，没有为新元素分配 id 属性值。但由于元素是通过引用插入的，因此可以修改存储在 a 变量中的对象，来修改插入的对象：

```
a.style.color = "red"
```

插入的元素也是文档层次结构的一部分，因此可以通过层次结构引用(如 myP.previousSibling)来访问它。

新插入元素的父元素是 body。这样，可在顶部文本框中输入以下语句，来查看所显示页面的当前 HTML：

```
document.body.innerHTML
```

如果向下滚动过第一个表单，可以找到已添加的<hl>元素以及 style 特性。

相关主题：document.createElement()方法，applyElement()方法。

insertAdjacentHTML("*location*", "*HTMLtext*")，insertAdjacentText("*location*", "*text*")

返回值：无

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

这两个方法在相对于当前元素的一个位置插入 HTML 或文本。它们在页面载入后使用，而不用于在页面载入时插入内容(这种情况下，可使用 document.write()在页面上期望的地方编写需要的内容)。

第一个必选参数是 4 个插入位置之一，它们不区分大小写，如表 26-16 所示。

表 26-16　4 个插入位置

位　置	说　明
beforeBegin	当前元素的始标记之前
afterBegin	起始标记之后，但在所有其他嵌套内容之前
beforeEnd	结束标记之前，但在所有其他嵌套内容之后
afterEnd	结束标记之后

这些位置参数的作用和本章前面讨论的 insertAdjacentElement()函数相同。

使用 insertAdjacentHTML()还是 insertAdjacentText()取决于内容的种类，以及浏览器的处理方式。如果内容包含 HTML 标记，而且希望浏览器解释和显示它，就好像它是页面源代码的一部分，就应使用 insertAdjacentHTML()方法，这样所有标记就成了文档对象模型中的对象。但如果只想显示一些文本(包括“原始”格式的 HTML 标记)，则应使用 insertAdjacentText()，这时显示引擎不会解释包含在第二个字符串参数中的标记，而是把这些标记显示为页面上的字符。这个区别和 innerHTML 与 innerText 之间的区别相同。

更多资源请访问稀酷客(www.ckook.com)

insertAdjacentHTML()和 insertAdjacentElement()之间的区别是插入内容的种类，前者允许将 HTML 累积为一个字符串；而后者需要创建一个元素对象。另外，本节的两个方法都适用于 IE4+(包括 Mac 版本)，而 insertAdjacentElement()需要 WinIE5+的新对象模型。

如果传递给 insertAdjacentHTML()的第二个参数的 HTML 包含<script>标记，则必须在首标记中设置 defer 特性，这样在插入脚本语句时，浏览器就不会执行它们。

示例

使用 The Evaluator(参见第 4 章)来试验这两个方法。这里的示例演示了用两种方法在 myP 元素的开始位置添加一些 HTML 的效果。首先为全局变量 a 赋予一个 HTML 代码串：

```
a = "<b id='myB'>Important News!</b>"
```

由于此 HTML 与 myP 段落的开始位置在同一行上，因此为插入方法使用了 ofterBegin 参数：

```
myP.insertAdjacentHTML("afterBegin", a)
```

注意，在插入 HTML 的感叹号后没有空格。为了证明插入的 HTML 的确是文档对象模型的一部分，在 ID 为 myB 的 b 元素后插入一个空格：

```
myB.insertAdjacentText("afterEnd", " ")
```

每当执行前面的语句时(不断单击 Evaluate 按钮，或当鼠标指针在顶部文本框中时，按下 Enter 键)，都会添加一个空格。

再看看要用 insertAdjacentText()插入的字符串包含 HTML 标记的情况。重新载入 The Evaluator，在顶部文本框中输入以下两条语句，依次执行每条语句：

```
a = "<b id='myB'>Important News!</b>"
myP.insertAdjacentText("afterBegin", a)
```

HTML 并未进行解释，而是显示为纯文本。在执行最后一个插入方法后，没有插入 myB 对象。

相关主题：innerText、innerHTML、outerText、outerHTML 属性；insertAdjacentElement()、replaceAdjacentText()方法。

insertBefore(*newChildNodeObject*, *referenceChildNode*)

返回值：Node 对象。

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

insertBefore()方法是将新的子节点插入现有元素的 W3C DOM 语法。其两个参数是节点引用，它们必须是有效的 Node 对象，包括 document.createElement()创建的对象。

这个方法有时似乎违反了常理，如果该方法包括第二个参数(当前元素中现有子节点的引用——在 IE 中是可选的)，新子节点就插入到现有节点之前。但是如果忽略第二个参数(或其值为 null)，新子节点就插入为当前元素的最后一个子节点，在这种情况下，该方法和 appendChild()方法相同。只有指定第二个参数，才能显现出这个方法的作用；从父元素的角度来看，可将一

更多资源请访问稀酷客(www.ckook.com)

个新子节点放到它的现有子节点中的任何位置。如果插入的节点已存在于文档树中，就删除以前存在的节点。

insertBefore()方法只在父元素上起作用，而 Internet Explorer 提供了附加的方法，例如 insertAdjacentElement()，它在子元素上执行操作。

示例

程序清单 26-28 演示了 insertBefore()方法如何根据第二个参数，将子元素(li)插入父元素(ol)中的不同位置。文本框允许用户选择要插入到 ol 元素内不同位置的文本和/或 HTML。如果未指定位置，insertBefore()的第二个参数就是 null——意味着新的子节点添加到现有子节点的末尾。从选择列表中选择要插入新项目的位置，select 列表选项的值是 ol 元素的前三个子节点的下标。

程序清单 26-28　使用 insertBefore()方法

HTML: jsb26-28.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>insertBefore() Method</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-28.js"></script>
   </head>
   <body>
      <h1>insertBefore() Method</h1>
      <hr />
      <form onsubmit="return false">
         <p>Enter text or HTML for a new list item:
            <input type="text"
                   name="newText" size="40" value="" />
         </p>
         <p>Before which existing item?
            <select name="itemIndex">
               <option value="null">None specified</option>
               <option value="0">1</option>
               <option value="1">2</option>
               <option value="2">3</option>
            </select>
         </p>
         <input type="button" value="Insert Item"
                onclick="doInsert(this.form)" />
      </form>
      <ol id="myUL">
         <li>Originally the First Item</li>
         <li>Originally the Second Item</li>
         <li>Originally the Third Item</li>
      </ol>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb26-28.js

```
function doInsert(form)
{
   if (form.newText)
   {
       var newChild = document.createElement("LI");
       newChild.innerHTML = form.newText.value;
       var choice = form.itemIndex.options[form.itemIndex.selectedIndex].value;
       var insertPoint = (isNaN(choice)) ?
          null : document.getElementById("myUL").childNodes[choice];
       document.getElementById("myUL").insertBefore(newChild, insertPoint);
   }
}
```

相关主题：appendChild()、replaceChild()、removeChild()、insertAdjacentElement()方法。

isDefaultNamespace("*namespaceURI*")

返回值：Boolean。
兼容性：WinIE-，MacIE-，NN6-，Moz1.7.2+，Safari+，Opera+，Chrome+
此方法检查指定的命名空间是否匹配当前节点的默认命名空间。

isEqualNode(*nodeRef*)，isSameNode(*nodeRef*)

返回值：整数 ID
兼容性：WinIE-，MacIE-，NN-，Moz1.7.2+，Safari+，Opera+，Chrome+

在谈到节点时，一个节点与另一个节点相同明显不同于一个节点与另一个节点相同。节点相等有非常特殊的含义：如果两个节点的 attributes、childNodes、 localname、namespaceURI、nodeName、nodeType、nodeValue 和 prefix 属性值都相同，这两个节点就是相等的。这些属性基本上反映了节点的内容，但它们没有反映节点在文档中的相对位置，这意味着节点可以相等，但处于节点树的不同位置。如果两个节点是同一个节点，这两个节点就是相同的。isEqualNode()方法检查节点是否相等，而 isSameNode()方法检查两个节点是否为同一节点。这两个方法都需要把一个节点引用作为其唯一的参数。当然，Opera 不支持 isEqualNode()方法。

isSupported("*feature*", "*version*")

返回值：Boolean。
兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如果当前节点支持指定 W3C DOM 模块和版本的必需部分，isSupported()方法就返回 true；否则返回 false。该方法的第一个参数接受以下 DOM 模块的字符串名称，这些字符串区分大小写：Core、XML、HTML、Views、StyleSheets、CSS、CSS2、Events、UIEvents、MouseEvents、MutationEvents、HTMLEvents、Range 和 Traversal；第二个参数接受主要和次要 DOM 模块版本的字符串表示，如 DOM Level 2 表示为 2.0。

更多资源请访问稀酷客(www.ckook.com)

示例

使用 The Evaluator(参见第 4 章)试验 isSupported()方法。如果有多个版本的 NN6+和 Mozilla，请尝试一下以下(和其他)语句，看看对不同模块的支持是如何发展的：

```
document.body.isSupported("CSS", "2.0")
document.body.isSupported("CSS2", "2.0")
document.body.isSupported("Traversal", "2.0")
```

如果可以访问 Safari、Opera 或 Chrome，在其中尝试运行相同的方法，看看其与基于 Mozilla 的浏览器相比在支持的模块上的差异。

item(*index* | "*index*" [, *subIndex*])

返回值：对象。

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

item()方法处理的大多数对象本身是其他对象的集合，在 W3C DOM 术语中，这类对象称为命名节点列表(对 Node 对象和 attribute 对象)或 HTML 集合(对表单元素等对象)。可使用一个数值参数调用 item()方法，该参数是集合内期望对象的下标值。如果知道项的下标值，则可以改用 JavaScript 数组语法。下列两条语句返回相同的对象引用：

```
document.getElementById("myTable").childNodes.item(2)
document.getElementById("myTable").childNodes[2]
```

此方法还支持在集合内使用对象的 ID 字符串，而对于 attributes、rules 和 TextRectangle 而言，该方法需要整数值。此外，如果集合中有多个 ID 相同的对象(除非有必要，否则最好避免这种情况)，第二个数值参数允许选择所需的命名组(在子组内使用基于 0 的下标值)。很明显，这不能应用到集合中，例如属性和规则，它们没有相关的 ID。

该方法返回参数指定的对象的引用。

示例

使用 The Evaluator(参见第 4 章)试验 item()方法。在顶部文本框中输入以下语句，查看每条语句的结果。

W3C 和 IE5：

```
document.getElementById("myP").childNodes.length
document.getElementById("myP").childNodes.item(0).data
document.getElementById("myP").childNodes.item(1).nodeName
```

W3C、IE4 和 IE5：

```
document.forms[1].elements.item(0).type
```

当脚本处理对象名称的字符串版本时，第一个示例很有帮助。如果脚本已经知道了对象引用，第二种方法更加简洁高效。

相关主题：返回其他对象集合(数组)的所有对象元素属性。

更多资源请访问稀酷客(www.ckook.com)

lookupNamespaceURI("*prefix*")，lookupPrefix("*namespaceURI*")

返回值： 命名空间或前缀字符串(参见描述)。

兼容性： WinIE-，MacIE-，NN-，Moz1.7.2+，Safari+，Opera+，Chrome+

这两个方法使用一个信息来查找另一个信息。lookupNamespaceURI()方法把前缀作为其唯一的参数，如果前缀匹配一个以前定义的命名空间，则方法返回节点的 URI 字符串。lookupPrefix()方法则反向操作，它接受一个命名空间 URI 字符串，如果命名空间参数匹配一个以前定义的命名空间，则方法返回节点的前缀字符串。

mergeAttributes("*sourceObject*")

返回值： 无。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

mergeAttributes()方法是在新建元素中复制特性的一种快捷方式，不必一次添加一个特性。如果一个对象的特性可作为其他元素的原型，则这些特性(除 id 特性外)可立刻应用到新创建的元素中。此方法的默认动作不是复制元素的 id 或 name 特性。不过 IE5.5+引入了一个额外的布尔参数 preserveIDs，将该参数设置为 false(默认值为 true)，就可以复制这两个特性。

示例

程序清单 26-29 演示了 mergeAttributes()的用法，它在复制同一个表单输入域的同时为每个新表单输入域分配一个唯一的 ID。为便于用户看到结果，本例在每个输入域中显示了域的 HTML。

doMerge()函数首先生成两个新元素：p 元素和 input 元素。由于这些新建元素没有相关联的属性，所以通过 uniqueID 属性把唯一的 ID 分配给 input 元素。源代码中域(field1)的特性并入新的 input 元素。这样，除了 name 和 id 之外，所有特性都复制到新元素中。input 元素插入到 p 元素中，p 元素添加到文档表单元素的末尾。最后，新元素的 outerHTML 显示在其域中。注意，除了 name 和 id 特性外，所有其他特性都复制了，包括样式表特性和事件处理程序。为了证明事件处理程序在新元素中有效，可为任一元素添加一个空格，然后按 Tab 键，以触发 onchange 事件处理程序，将内容改为全大写形式。

程序清单 26-29 使用 mergeAttributes()方法

```
HTML: jsb2-29.html
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>mergeAttributes() Method</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-29.js"></script>
   </head>
   <body
   onload="document.expandable.field1.value = ↩
           document.expandable.field1.outerHTML">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <h1>mergeAttributes() Method</h1>
        <hr />
        <form name="expandable" onsubmit="return false">
          <p>
            <input type="button" value="Append Field 'Clone'"
                   onclick="doMerge(this.form)" />
          </p>
          <p>
            <input type="text" name="field1" id="FIELD1" size="120" value=""
                   style="font-size:9pt" onchange="upperMe(this)" />
          </p>
        </form>
    </body>
</html>
```

JavaScript: jsb26-29.js

```
function doMerge(form)
{
   var newPElem = document.createElement("p");
   var newInputElem = document.createElement("input");
   newInputElem.id = newInputElem.uniqueID;
   newInputElem.mergeAttributes(form.field1);
   newPElem.appendChild(newInputElem);
   form.appendChild(newPElem);
   newInputElem.value = newInputElem.outerHTML;
}

// called by onChange event handler of fields
function upperMe(field)
{
  field.value = field.value.toUpperCase();
}
```

相关主题：clearAttributes()、cloneNode()、removeAttributes()方法。

normalize()

返回值：无。

兼容性：WinIE6+，MacIE5+，NN7+，Moz+，Safari1.2+，Opera+，Chrome+

在添加、插入、删除和替换元素子节点的过程中，两个文本节点很可能彼此相邻。虽然这不影响内容的显示，但如果某些 XML 应用程序严重依赖文档节点层次结构来正确解释内容，文本节点就不可能彼此相邻。节点层次结构的“正确”格式是使一个文本节点被其他节点类型包围。normalize()方法可以去除当前节点对象的子节点，并把相邻的文本节点组合到一个文本节点中，这显然会影响元素的子节点数目，还会删除嵌套节点层次。

示例

使用 The Evaluator(参见第 4 章)试验 normalize()方法。以下语句添加了一个文本节点，使之紧邻 myP 元素中的一个文本节点，随后调用 normalize()方法，删除了相邻文本节点之

更多资源请访问稀酷客(www.ckook.com)

间的界限。

首先确认 myP 元素的子节点数量：

```
document.getElementById("myP").childNodes.length
```

元素中最初有三个节点。接下来创建一个文本节点，并将其添加为 myP 元素的最后一个子节点：

```
a = document.createTextNode("This means you!")
document.getElementById("myP").appendChild(a)
```

新文本现在显示在页面上，子节点的数量增加到了 4 个：

```
document.getElementById("myP").childNodes.length
```

myP 的最后一个子节点是刚才创建的文本节点：

```
document.getElementById("myP").lastChild.nodeValue
```

在 myP 上调用 normalize()方法，把所有相邻文本节点合并为一个节点：

```
document.getElementById("myP").normalize()
```

myP 元素现在又有三个子节点了，最后一个子节点是以前两个不同但相邻的文本节点的组合：

```
document.getElementById("myP").childNodes.length
document.getElementById("myP").lastChild.nodeValue
```

相关主题：document.createTextNode()、appendChild()、insertBefore()、removeChild()、replaceChild()方法。

releaseCapture()，setCapture(*containerBoolean*)

返回值：无。

兼容性：WinIE5+，MacIE-，NN-、Moz-，Safari-，Opera-，Chrome-

通过 WinIE 的专用方法 setcapture()，可在页面上用一个对象捕获所有的鼠标事件(onmousedown、onmouseup、onmousemove、onmouseout、onmouseover、onclick 和 ondblclick)。这种鼠标事件捕获功能的一个主要用途是，当页面上显示某些内容时，我们希望它们成为访问者注意的中心，例如下拉菜单、上下文菜单或模拟的模式窗口区域。此时，应禁用所有鼠标事件，但应用于菜单或当前可见伪窗口的鼠标事件除外。当这些内容消失时，就启用鼠标事件，这样其他元素可以(例如页面其他地方的按钮和链接)响应鼠标事件。

事件捕获并不阻止事件的冒泡，相反，事件重定向到设置为捕获所有鼠标事件的对象上。事件从这个对象开始冒泡，除非明确地取消它(见第 32 章)。例如，文档中的一个<body>标记包含 onclick 事件处理程序，它总是控制整个文档。如果要为文档某处的 div 启用事件捕获功能，单击事件将首先走到 div 处。那个 div 可能有一个 onclick 事件处理程序，若单击事件发生在它的子元素中，它就处理单击事件；如果 div 的事件处理程序没有取消这个单击事件的冒泡，body 元素的 onclick 事件处理最后会接收和处理单击事件，而 div 最先捕获单击事件。

确定应该由哪个对象捕获事件是一个十分重要的设计问题。启用事件捕获功能后，所有的

更多资源请访问稀酷客(www.ckook.com)

鼠标事件(不管它们发生在何处)都会传递到捕获事件的对象上。因此，如果应用程序的界面设计为包括可单击和可拖放的元素，则可设置一个元素(甚至是 document 对象)来执行事件捕获。而对弹出式区域，更具条理、更方便的编码方法是把捕获机制指定给弹出内容的主要容器(通常是一个定位 div 元素)。

setCapture( )方法有一个可选的 Boolean 参数，用于确定发生在捕获对象的子元素中的鼠标事件是否在事件捕获机制的控制之下。默认值(true)表示当前元素对象的所有鼠标事件都由当前对象捕获，而不是原始目标，这是在弹出式菜单和上下文菜单中一种最可能使用 setCapture()的方式。但是如果将该参数设为 false，则发生在捕获容器的子元素中的鼠标事件将直接接收它们的事件。从那里开始，事件从目标向上冒泡(见第 32 章)。

设置为捕获鼠标事件的区域(例如文本输入域和 select 列表)包含表单元素时，可能会出现一些奇怪的现象。因为只有鼠标事件得到焦点才能与这些元素交互，而事件捕获机制会约束对这些元素的访问。要解决这个问题，可以检查鼠标单击事件的 srcElement 属性，确定是否单击了这些元素，并编写脚本使元素得到焦点(或指导用户按 Tab 键，直到元素获得焦点为止)。

一旦对象设置为捕获事件，则其他代码必须确定哪个事件执行操作；并决定事件是否应该从捕获元素向上冒泡。只有在高于元素包含层次结构的元素中包括鼠标处理，才需要考虑冒泡。在启用事件捕获功能时，可能并不触发事件处理程序。在这种情况下，需要在捕获对象上取消事件冒泡。

如果应用程序设计为需要隐藏弹出式区域，并让事件处理程序正常返回(例如在用户在弹出式菜单做出选择后)，则使用 releaseCapture()方法并隐藏容器。因为事件捕获机制一次只可用于一个元素，所以可从容器或 document 对象上调用 releaseCapture()方法，来释放事件捕获机制。

当用户执行下列动作时，会自动释放事件捕获机制。

- 其他窗口得到焦点。
- 显示系统模式对话框窗口(例如警告窗口)。
- 滚动页面。
- 打开浏览器上下文菜单(通过右键单击)。
- 使用 Tab 键使浏览器窗口中的 Address 域得到焦点。

因此，可设置 document 对象的 onlosecapture 事件处理程序，来隐藏与该事件捕获相关的容器。

还要注意，即使可以捕获鼠标事件，来阻止鼠标访问页面的其他部分，也不会捕获键盘事件。因此，使用事件捕获机制来模仿模式窗口并不安全：用户可以用 Tab 键移到页面上的任何表单元素或链接，再按下空格键或 Enter 键来触发该元素。

W3C DOM 中的事件捕获和 WinIE 事件捕获不同。在 W3C DOM 中，可让浏览器用任何类型的事件捕获机制替代常规的事件冒泡行为。例如，可将事件监听器与 body 元素关联起来，在事件到达目标元素前，查看所有单击事件，这些事件以 body 元素包含的元素为目标。参看第 25 章和第 32 章，来了解 W3C DOM 事件模型的更多信息，以及如何把它集成到跨浏览器的应用程序中。

示例

程序清单 26-30 演示了 setCapture()和 releaseCapture()在一个临时的 WinIE5+上下文菜单中

更多资源请访问稀酷客(www.ckook.com)

的用法。上下文菜单的作用是为页面上的有序项列表显示一个编号样式列表。只要用户在 ol 元素上打开上下文菜单，自定义的上下文菜单就会显示出来。在此过程中打开事件捕获功能，以防止页面上其他位置的鼠标操作中断上下文菜单的选择。在显示上下文菜单时，禁止单击哪怕是设置为列表标题的链接。在上下文菜单外任何位置单击都将隐藏菜单。单击菜单中的选项将更改 ol 对象的 listStyleType 属性，并隐藏菜单。只要上下文菜单隐藏起来，就关闭事件捕获功能，以使单击页面的操作(如链接)能正常进行。

为实现这一设计，将 onclick、onmouseover 和 onmouseout 事件处理程序赋予包含上下文菜单的 div 元素。为了显示上下文菜单，ol 元素有一个 oncontextmenu 事件处理程序，它会调用 showContextMenu()函数。在此函数中，将事件捕获功能赋予包含上下文菜单的 div 对象。在 div 设置为可见之前，它也定位到单击位置。为防止系统的常规上下文菜单也显示出来，将 event 对象的 returnValue 属性设置为 false。

既然页面上的所有鼠标事件都要经过 div 对象 contextMenu，下面看看用户动作触发的各种不同事件。用户滚动鼠标时，会触发大量的 mouseover 和 mouseout 事件，这些事件由分配给 div 的事件处理程序管理。但要注意，只有当事件的 srcElement 属性是 div 中的某个菜单项时，highlight()和 unhighlight()这两个事件处理程序才执行操作。由于页面没有为包含层次结构上方的元素定义其他 onmouseover 或 onmouseout 事件处理程序，因此不必取消这些事件的冒泡。

用户单击鼠标按钮时，会发生不同情况，这取决于是否启用了事件捕获功能。若没有启用事件捕获功能，click 事件从其发生位置冒泡到 body 元素中的 onclick 事件处理程序(此时会显示一个警告对话框，说明事件何时到达 body)。但若启用了事件捕获功能(上下文菜单正在显示)，只要单击某个上下文菜单项，handleClick()事件处理程序就会接管，应用所需的选项。对于由此函数处理的所有 click 事件，都会隐藏上下文菜单，取消 click 事件的向上冒泡(不显示警告对话框)。无论用户是在上下文菜单中进行了选择，还是在页面其他位置单击，都会执行上述操作。在后一种情况下，只需让上下文菜单像真正的上下文菜单一样消失。为了更加保险，当用户执行上述取消捕获的动作时，onlosecapture 事件处理程序都会隐藏上下文菜单。

程序清单 26-30　使用 setCapture()和 releaseCapture()

HTML: jsb26-30.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <style type="text/css">
         #contextMenu
         {
            position:absolute; background-color:#cfcfcf;
            border-style:solid; border-width:1px;
            border-color:#EFEFEF #505050 #505050 #EFEFEF;
            padding:3px 10px; font-size:8pt; font-family:Arial, Helvetica;
            line-height:150%; visibility:hidden;
         }
         .menuItem
         {
```


更多资源请访问稀酷客(www.ckook.com)

```
            color:black;
         }
         .menuItemOn
         {
            color:white
         }
         ol
         {
            list-style-position:inside; font-weight:bold; cursor:hand;
         }
         li
         {
            font-weight:normal;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-30.js"></script>
   </head>
   <body onclick="alert('You reached the document object. ')">
      <ol id="shapesList" oncontextmenu="showContextMenu()">
        <li style="list-style: none"><a
         href="javascript:alert('A%20sample%20link. ')">Three-DimensionalShapes</a>
        </li>
        <li value="1">Circular Cylinder</li>
        <li>Cube</li>
        <li>Rectangular Prism</li>
        <li>Regular Right Pyramid</li>
        <li>Right Circular Cone</li>
        <li>Sphere</li>
      </ol>
      <div id="contextMenu" onlosecapture="hideMenu()" onclick="handleClick()"
           onmouseover="highlight()" onmouseout="unhighlight()">
         <span id="menuItem1" class="menuItem"
               listtype="upper-alpha">A,B,C,...</span>
         <br />
         <span id="menuItem2" class="menuItem"
               listtype="lower-alpha">a,b,c,...</span>
         <br />
         <span id="menuItem3" class="menuItem"
               listtype="upper-roman">I,II,III,...</span>
         <br />
         <span id="menuItem4" class="menuItem"
               listtype="lower-roman">i,ii,iii,...</span>
         <br />
         <span id="menuItem5" class="menuItem"
               listtype="decimal">1,2,3,...</span>
      </div>
   </body>
</html>
```

JavaScript: jsb26-30.js

更多资源请访问稀酷客(www.ckook.com)

```
function showContextMenu()
{
   contextMenu.setCapture();
   contextMenu.style.pixelTop = event.clientY + document.body.scrollTop;
   contextMenu.style.pixelLeft = event.clientX + document.body.scrollLeft;
   contextMenu.style.visibility = "visible";
   event.returnValue = false;
}

function revert()
{
   document.releaseCapture();
   hideMenu();
}

function hideMenu()
{
   contextMenu.style.visibility = "hidden";
}
function handleClick()
{
   var elem = window.event.srcElement;
   if (elem.id.indexOf("menuItem") == 0)
   {
      document.getElementById("shapesList").style.listStyleType = elem.listtype;
   }
   revert();
   event.cancelBubble = true;
}

function highlight()
{
   var elem = event.srcElement;
   if (elem.className == "menuItem")
   {
       elem.className = "menuItemOn";
   }
}

function unhighlight()
{
   var elem = event.srcElement
   if (elem.className == "menuItemOn")
   {
       elem.className = "menuItem";
   }
}
```

相关主题： addEventListener()、dispatchEvent()、fireEvent()、removeEventListener()方法；onlosecapture 事件；Event 对象(参阅第 32 章)。

更多资源请访问稀酷客(www.ckook.com)

removeAttribute("*attributeName*"[, *caseSensitivity*])

返回值：Boolean (IE)；无(NN/DOM)。

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

如果用 setAttribute()方法创建特性，则可使用 removeAttribute()方法从元素对象中删除该特性。该方法的参数是特性名。IE 允许设置和删除特性，特性名是区分大小写的。IE 中 removeAttribute()的第二个 Boolean 参数默认为 false，因此，如果 setAttribute()中区分大小写的参数为 true，则应将 removeAttribute()中的参数设为 true，以确保在创建和删除特性时保持平衡。

removeAttribute()方法的 W3C DOM(NN/Moz-/基于 WebKit)版本有一个参数(不区分大小写的特性名)，而且没有返回值。在 IE 中，如果删除成功，返回的值为 true；如果不成功(或者用其他方法设置了特性)，则返回 false。

示例

使用 The Evaluator(参见第 4 章)对页面中的元素试验 removeAttribute()方法。查看本章稍后的 setAttribute()方法示例，并在顶部文本框中输入相应的 removeAttribute()语句。在语句中交替使用 getAttribute()，来验证每个特性是否存在。

相关主题：attributes 属性；document.createAttribute()、getAttribute()、setAttribute()方法。

removeAttributeNode(*attributeNode*)，setAttributeNode(*attributeNode*)

返回值：Attribute 对象。

兼容性：WinIE6+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如本章前面的 getAttributeNode()方法所述，W3C DOM 把“名-值”特性对作为 attribute 对象，attribute 对象是命名节点图(属于元素的 attribute 对象集合)内的独特节点。理解命名节点图和 attribute 对象在 XML 环境中很有帮助，在该环境中，attribute 对象不仅包含有用的数据，而且不会在 DOM 中显示为能通过脚本访问的属性。不是访问对象的属性，而是使用实际的特性。

如果想在正式的 W3C 方法中插入一个特性，可使用 document.createAttribute()生成新的 attribute 对象。然后，脚本语句把值赋给 nodeName 和 nodeValue 属性，给 attribute 对象提供一个传统的“名-值”对。然后通过 setAttributeNode()方法，把新的 attribute 对象插入对象的特性列表中。该方法唯一的参数是 attribute 对象，返回值是新插入 attribute 对象的引用。

为使用该语法从元素中删除 attribute 节点，可使用 removeAttributeNode()方法，此方法的唯一参数是 attribute 对象。如果脚本只知道特性名，则可使用 getAttributeNode()来得到 attribute 对象的有效引用。removeAttributeNode()方法返回被删除 attribute 对象的引用，此对象仍保留在浏览器的内存中，但不是文档层次结构的一部分。在变量中保留这个已删除的 attribute 对象，可以修改它，并赋给文档中的另一个对象。

实际上，很少需要将 attribute 作为节点。当脚本只有元素的特性名(字符串)时，其他方法可以完成上述任务，例如 getAttribute()、removeAttribute()和 setAttribute()。

示例

使用 The Evaluator(参见第 4 章)对页面中的 p 元素试验 setAttributeNode()和 removeAttribute-

更多资源请访问稀酷客(www.ckook.com)

Node()方法。任务是创建一个 style 特性，并添加到 p 元素中。首先创建一个新特性，并临时存储在全局变量 a 中：

```
a = document.createAttribute("style")
```

为 attribute 对象赋值：

```
a.nodeValue = "color:red"
```

将新特性插入到 p 元素中：

```
document.getElementById("myP").setAttributeNode(a)
```

段落会改变颜色，以响应新增加的特性。

在 NN6 中不允许方法返回对新插入特性节点的引用，但可以人为获得这一引用：

```
b = document.getElementById("myP").getAttributeNode("style")
```

最后，使用对新增特性的引用，将其从元素中删除：

```
document.getElementById("myP").removeAttributeNode(b)
```

删除该特性后，段落恢复为其最初的颜色。参见本章稍后的 setAttribute()方法示例，了解如何用 setAttribute()执行此类操作。

相关主题： attributes 属性，document.createAttribute()、getAttribute()、getAttributeNode()、setAttribute()方法。

removeAttributeNS("*namespaceURI*", "*localName*")

返回值： 无。

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

此方法删除在两个参数中指定的属性。方法的第一个参数是一个与分配给文档中标签的 URI 相匹配的 URI 字符串，第二个参数是要删除其值的特性的本地名称部分。

相关主题： attributes、namespaceURI、localName 属性；removeAttribute()、getAttributeNS()、setAttributeNS()方法。

removeBehavior(*ID*)

返回值： Boolean。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

removeBehavior()方法使对象与操作解除关联，它假定操作是用 addBehavior()方法添加到对象中的。addBehavior()方法的返回值是特定操作的唯一标识符，该标识符是 removeBehavior()方法的必选参数。因此，可将两个操作添加一个对象中，在需要时看仅删除其中之一。如果成功删除，removeBehavior()方法返回 true；否则返回 false。

示例

有关 addBehavior()和 removeBehavior()的用法，请参见本章前面的程序清单 26-19a 和

更多资源请访问稀酷客(www.ckook.com)

26-19b。

相关主题：addBehavior()方法。

removeChild(*nodeObject*)

返回值：Node 对象引用。

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

removeChild()方法从当前元素中删除子元素，此后和子元素相关的内容在页面上不再可见，该子对象也不再是文档对象层次结构的一部分。

这似乎具有破坏性，但删除的对象信息未必丢失，removeChild()方法返回已删除节点的引用。把这个值赋给一个变量，就可以保存对象信息，用于将来的插入操作。还可把这个变量用作 appendChild()、replaceChild()、swapNode()和 insertBefore()等方法的参数。

removeChild()是在父元素上调用的。如果只想删除一个元素，则可直接使用 removeNode()方法(仅限于 WinIE5+)。IE 的 removeNode()方法还允许删除节点本身，这不可能通过 removeChild()来实现。

示例

在本章前面的程序清单 26-21 中，可以看到 removeChild()的例子。

相关主题：appendChild()，replaceChild()，removeNode()方法。

removeEventListener()

(详见 addEventListener())

removeExpression("*propertyName*")

返回值：Boolean。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

如果通过 setExpression()方法把一个表达式赋给对象的属性(包括对象的 style 对象)，可使用 removeExpression()方法在脚本的控制下删除它。该方法唯一的参数是属性名的字符串版本，属性名是区分大小写的。

如果删除成功，该方法就返回 true；否则返回 false。注意，删除一个表达式不会改变当前赋给属性的值，换句话说，可使用 setExpression()来设置属性值，然后删除表达式，则文档重新计算表达式时，属性值不会变化。如果只想达到这个目标，最好通过脚本直接设置属性。

示例

可在 The Evaluator(参阅第 4 章)中试验这三个表达式方法。以下语句给页面上 myP 元素的样式表属性添加一个表达式，然后将其删除。

首先，在 The Evaluator 底部的单行文本框中输入数字 24(但不要按 Enter 键或单击 List Properties 按钮)，这个值用来在表达式中确定 myP 对象的 fontSize 属性；接下来在顶部文本框中输入以下语句，将一个表达式赋给 myP 对象的 style 对象：

更多资源请访问稀酷客(www.ckook.com)

```
myP.style.setExpression("fontSize","document.forms[0].↵
inspector.value","JScript")
```

现在可在底部文本框中输入不同的字体大小，并立即将该值应用于 fontSize 属性(文本框中的键盘事件自动触发，重新计算该属性)。默认单位是 px，也可在文本框中给值使用其他单位(如 pt)，看一下不同的度量单位如何影响同一数值。

在执行下一步操作之前，输入不是 16(默认的 fontSize 值)的值。最后，在顶部文本框中输入以下语句，使表达式与属性断开连接：

```
myP.style.removeExpression("fontSize")
```

注意，尽管不再能在底部文本框中调整字体的大小，但最新分配给它的值仍然对元素有效。为证明这一点，在顶部文本框中输入以下语句，来查看 fontSize 的当前值：

```
myP.style.fontSize
```

相关主题：document.recalc()、getExpression()、setExpression()方法。

removeNode(*removeChildrenFlag*)

返回值：Node 对象引用。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在 WinIE5+中可使用 removeNode()方法从元素层次结构中删除当前节点。此方法的唯一参数是 Boolean 值，它指定方法是删除节点自身(不删除其子节点)，还是删除节点及其所有的子节点(值为 true 时)。该方法返回删除的 Node 对象的引用，DOM 不再能访问删除的对象。但返回值包含对象删除前所拥有的所有属性(包括 outerHTML 等属性，以及明确设置的样式表规则)。因此，可使用该返回值作为一个参数，在文档的其他地方插入节点。

W3C DOM 没有实现 removeNode()方法，与 removeNode()最类似的跨浏览器方法是 removeChild()。removeChild()方法的作用域比 removeNode()方法的对象所在的对象层次高一层。

示例

查看程序清单 26-21 中的 appendChild()方法，以便理解 removeChild()和 removeNode()之间的区别。在 IE5+中，可以用

```
mainObj.removeChild(oneChild);
```

替换

```
oneChild.removeNode(true);
```

它们之间的区别很微妙，但理解这一区别十分重要。参见本章稍后程序清单 26-31 中的另一个 removeNode()方法示例。

相关主题：Node 对象；appendChild()、cloneChild()、removeChild()、replaceChild()、replaceNode()方法。

更多资源请访问稀酷客(www.ckook.com)

replaceAdjacentText("location", "text")

返回值：字符串。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

replaceAdjacentText()方法允许用一个文本块替换另一个文本块，替换的文本块在相对于当前对象的特定位置上。注意，该方法只对普通文本起作用，对 HTML 标记不起作用。此方法的返回值是替换的文本字符串。

该方法需要两个参数，第一个必选参数是 4 个插入位置之一，位置的值不区分大小写，如表 26-17 所示。

表 26-17 位置及说明

位 置	说 明
beforeBegin	当前元素的起始标记之前
afterBegin	起始标记之后，但在所有其他嵌套内容之前
beforeEnd	结束标记之前，但在所有其他嵌套内容之后
afterEnd	结束标记之后

使用 beforeBegin 和 afterEnd 参数时，该方法最好用在内嵌(而不是块)元素内。例如，如果在两个连续段落元素的第二个上使用 replaceAdjacentText()和 beforeBegin，替换文本将插入第一段的末尾。另外，还可考虑在文本段节点上使用 replaceAdjacentText()方法，该方法用新文本替换文本段代码(使用 4 个参数中的任何一个位置参数)。替换简单元素的文本时使用 afterBegin 或 beforeEnd 参数，与把该文本赋给对象的 innerText 属性是一样的。

示例

使用 The Evaluator(参见第 4 章)来试验 replaceAdjacentText()方法。在顶部文本框中输入以下各条语句，并在 myP 元素(及其嵌套的 myEM 元素)中观察结果：

```
document.getElementById("myEM").replaceAdjacentText("afterBegin", "twenty")
```

注意，myEM 元素的新文本获得了元素的行为。同时，方法返回已被替换的文本(all)，并显示在 Results 框中：

```
document.getElementById("myEM").replaceAdjacentText("beforeBegin", "We need")
```

文本片断的所有字符(包括空格)都被替换了。因此，如这里所示，如果替换的片断有空格，可能需要提供一个尾部空格：

```
document.getElementById("myP").replaceAdjacentText("beforeEnd", "good people.")
```

这是替换 myEM 元素后的文本片断的另一种方法，该文本片段也是相对于外部 myP 元素的。如果现在要替换 myP 块级元素结尾之后的文本：

```
document.getElementById("myP").replaceAdjacentText("afterEnd", "Hooray!")
```

文本片段插入到 myP 元素标记对的后面。该片段在 DOM 中只看成一个未加标签的文本节点。

更多资源请访问稀酷客(www.ckook.com)

相关主题： innerText、outerText 属性；getAdjacentText()、insertAdjacentHTML()、insertAdjacentText()方法。

replaceChild(*newNodeObject*, *oldNodeObject*)

返回值： Node 对象引用。

兼容性： WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

replaceChild()方法允许用新节点对象替换现有的子节点对象。replaceChild()方法的参数是节点对象引用，顺序必须是要替换的对象跟在新对象的后面。旧的对象必须是用于调用该方法的父节点的一个直接子节点，新对象也必须是文档包含性层次内的一个合法子元素。

该方法返回新对象所替换的子对象的引用，该引用可作为任何面向节点的插入或替换方法的参数。

记住，replaceChild()是在父元素上调用的。如果只想改变一个元素，在 WinIE5+中使用 swapNode()方法或 replaceNode()方法会更直接。

示例

在本章前面的程序清单 26-21(用于 appendChild 属性)中，可看到 replaceChild()的例子。

相关主题： appendChild()、removeChild()、replaceNode()、swapNode()方法。

replaceNode("*newNodeObject*")

返回值： Node 对象引用。

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

replaceNode()方法与 replaceChild()方法相关，但该方法在要替换的实际节点上调用，而不是其父节点。该方法的唯一参数是节点对象的有效引用，该引用可通过 document.createElement()方法创建，或从现有节点中复制。此方法的返回值是被替换对象的引用。将此结果保存在变量中，就可以保留一份被替换节点的副本，供将来使用。

如果所替换的节点包含其他节点，replaceNode()方法就从文档中删除原始节点包含的所有节点。因此，如果想改变容器节点，同时保留原始子节点，则脚本必须捕获这些子节点，并将它们放在新节点中，如下例所示。

示例

程序清单 26-31 演示了三个与节点相关的方法：removeNode()、replaceNode()和 swapNode()，这些方法仅可用于 WinIE5+。

程序清单 26-31 显示的页面最初是一个由 4 项组成的 ul 类型列表，4 个按钮控制着此列表元素的节点结构的各个方面。第一个按钮调用 replace()函数，该函数将 ul 类型变为 ol。为此，函数必须将原来 ul 元素的所有子节点临时存储起来，以便以后将它们添加回新的 ol 元素。同时，旧的 ul 节点存储在一个全局变量中(oldNode)，以便在另一个函数中还原。

为了用 ol 替换 ul 节点，replace()函数创建了一个新的空 ol 元素，并将 ID 值 myOL 赋给它；接下来，子节点(li 元素)作为数组保存到变量 innards 中，然后使用 insertBefore()方法将子节点插入空的 ol 元素。注意，把 innards 数组中的每个子元素插入 ol 元素时，这些子元素也从 innards 数组中删除。所以插入子节点的 while 循环不断将 innards 数组的第一项插入到新元素中。最后，

更多资源请访问稀酷客(www.ckook.com)

replaceNode()方法将新节点放到旧节点的位置上，旧节点(只是 ul 元素)保存在 oldNode 中。

restore()函数的操作与 replace()函数相反，它同样需要处理嵌套的子节点。

第三个按钮调用 swap()函数，该函数的脚本交换第一个节点和最后一个节点。与本节讨论的其他方法一样，swapNode()方法也在节点上执行操作。因此，该方法依附于某个被交换的节点，另一个节点则指定为参数。由于 ol 元素的性质，数字顺序保持不变，但 li 节点的文本交换了。

为了演示 removeNode()方法，第四个函数删除列表的最后一个子节点。每个对 removeNode()的调用都使用 true 参数，以确保也删除嵌套在每个 li 节点中的文本节点。将参数设置为 false(默认值)，试验一下此方法。注意在删除 li 节点时父子关系如何变化。

程序清单 26-31　使用与节点相关的方法

HTML: jsb26-31.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>removeNode(), replaceNode(), and swapNode() Methods</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-31.js"></script>
   </head>
   <body>
     <h1>Node Methods</h1>
     <hr />
     Here is a list of items:
     <ul id="myUL">
       <li>First Item</li>
       <li>Second Item</li>
       <li>Third Item</li>
       <li>Fourth Item</li>
     </ul>
     <form>
       <input type="button" value="Change to OL List"
              onclick="replace()" />  
       <input type="button"
              value="Restore LI List" onclick="restore()" />  
       <input type="button" value="Swap First/Last" onclick="swap()" />  
       <input type="button" value="Remove Last" onclick="remove()" />
     </form>
   </body>
</html>
```

JavaScript: jsb26-31.js

```
// store original node between changes
var oldNode;

// replace UL node with OL
```


更多资源请访问稀酷客(www.ckook.com)

```
function replace()
{
     if (document.getElementById("myUL"))
     {
          var newNode = document.createElement("OL");
          newNode.id = "myOL";
          var innards = document.getElementById("myUL").children;
          while (innards.length > 0)
          {
               newNode.insertBefore(innards[0]);
          }
          oldNode = document.getElementById("myUL").replaceNode(newNode);
     }
  }

  // restore OL to UL
  function restore()
  {
  if (document.getElementById("myOL") && oldNode)
  {
    var innards = document.getElementById("myOL").children;
    while (innards.length > 0)
    {
        oldNode.insertBefore(innards[0]);
    }
    document.getElementById("myOL").replaceNode(oldNode);
  }
}

// swap first and last nodes
function swap()
{
    if (document.getElementById("myUL"))
    {
        document.getElementById("myUL").firstChild.swapNode(
            document.getElementById("myUL").lastChild);
    }
    if (document.getElementById("myOL"))
    {
        document.getElementById("myOL").firstChild.swapNode(
            document.getElementById("myOL").lastChild);
    }
}

// remove last node
function remove()
{
    if (document.getElementById("myUL"))
    {
        document.getElementById("myUL").lastChild.removeNode(true);
```


更多资源请访问稀酷客(www.ckook.com)

```
    }
    if (document.getElementById("myOL"))
    {
        document.getElementById("myOL").lastChild.removeNode(true);
    }
}
```

可使用 W3C DOM 以跨浏览器方式实现程序清单 26-31 的功能。要替代 removeNode()和 replaceNode()方法，可在 ul 和 ol 对象的父元素 document.body 上调用 removeChild()和 replaceChild()方法，并将 document.all 引用改为 document.getElement ById()。

相关主题：removeChild()、removeNode()、replaceChild()、swapNode()方法。

scrollIntoView(*topAlignFlag*)

返回值：无。

兼容性：WinIE4+，MacIE4+，NN7+，Moz+，Safari 2.02，Opera+，Chrome+

scrollIntoView()方法可根据需要垂直或水平滚动页面，使当前对象在包含它的窗口或框架中可见。此方法需要一个 Boolean 参数，来控制元素在可视空间内的位置。参数为 true(默认)将显示元素，它的顶部与窗口或框架的顶部平齐(假定它下面的文档足够长，允许执行这种滚动)；而 false 值使元素的底部与可视区域的底部平齐。大多数情况下使用前者，使页面开始于可视区域的顶部。但如果跳到新视图时，不希望用户看到某个元素下面的内容，则使用 false 参数。

对于表单元素，必须使用一般形式的表单元素引用(document.formName.elementName.scrollIntoView())，除非还为元素指定了 ID 特性(document.getElementById("elementID").scrollIntoView ())。

示例

使用 The Evaluator(参阅第 4 章)试验 scrollIntoView()方法。调整浏览器窗口的高度，使其只显示顶部文本框和 Results 文本区域。在顶部文本框中输入以下语句，看看 myP 元素显示的位置：

```
myP.scrollIntoView()
myP.scrollIntoView(false)
```

增加浏览器窗口的高度，直至可以看到页面下方的表格部分。如果在顶部文本框中输入：

```
myTable.scrollIntoView(false)
```

页面会滚动，将表格底部显示在窗口底部。但如果使用默认参数(true 或空)：

```
myTable.scrollIntoView()
```

页面只滚动到元素顶部尽可能与窗口顶部对齐的位置。页面滚动不能超出其正常的最大滚动范围(如果元素是一个定位元素，可使用动态定位功能将其放到任何位置，包括在页面之外)。同样，如果缩小窗口，试图将表格顶部滚动到窗口顶部，注意 table 元素包含了一个 caption 元素，因此标题将与窗口顶部平齐。

相关主题：window.scroll()、window.scrollBy()、window.scrollTo()方法。

更多资源请访问稀酷客(www.ckook.com)

setActive()

返回值：无

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

setActive()方法允许脚本指定一个元素对象作为当前活动的元素。然而，与 focus()方法不同，setActive()方法不会滚动窗口，使活动元素显示出来。调用 setActive()将激活为元素定义的 onFocus 事件处理程序，不需要浏览器赋予元素焦点。

示例

使用 The Evaluator(参阅第 4 章)比较 setActive()和 focus()方法。将页面滚动到顶部，调整窗口，不显示页面底部附近的示例复选框，在顶部文本框中输入以下语句：

```
document.getElementById("myCheckbox").setActive()
```

向下滚动，确保复选框有操作焦点(按空格键查看)。现在滚动回顶部，并输入以下语句：

```
document.getElementById("myCheckbox").focus()
```

这次复选框会获得焦点，页面自动滚动，将复选框对象显示出来。

相关主题：focus()方法。

setAttribute("*attributeName*", value[, *caseSensitivity*])

返回值：无。

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

setAttribute()方法把新值赋给当前对象的一个现有特性，或在当前对象的特性中插入全新的特性“名-值”对，该方法是另一种直接设置对象属性的语法。

注意：

W3C DOM Level 2 标准推荐用 getAttribute()和 setAttribute()方法读写元素对象的特性值，而不是通过相应的属性读写那些值。这些方法适用于 XML 元素，但 DOM 标准会发出冲突信号，因为它为 HTML 元素对象定义了所有属性。当然，浏览器将来可能支持通过属性进行访问，因此不一定要改变读写属性值的方式。

setAttribute()的前两个参数是必需的，第一个参数是特性名，该方法默认区分特性名的大小写，因此，如果以默认模式使用 setAttribute()来调整现有特性的值，第一个参数必须根据当前文档的对象模型来匹配特性的大小写。注意，所有作为内嵌源代码的特性名都会自动转换为小写字母。

第二个参数是赋给特性的值，为了获得跨浏览器的兼容性，该值应该是一个字符串或 Boolean 数据类型。

IE 提供了可选的第三个参数，来控制特性名的大小写，默认值(true)对对象有不同的影响，这取决于是使用 setAttribute()给新特性赋值，还是重新给现有的特性赋值。对于前一种情况，第三个参数为 true，意味着第一个参数要区分大小写，这个参数是赋给对象的属性名；对于后一种情况，第三个参数为 true 意味着，除非第一个参数与当前对象相关特性的大小写匹配，否

更多资源请访问稀酷客(www.ckook.com)

则不给特性重新赋值，但会创建一个大小写不同的新特性。

试图管理新建特性的大小写是很危险的，特别是在重用仅大小写不同的名字时，强烈建议对 setAttribute()和 getAttribute()使用默认值：区分大小写。

参见本章前面 getAttributeNode()和 removeAttributeNode()的讨论，看看 W3C DOM 如何将特性处理为 Node 对象。

示例

使用 The Evaluator(参见第 4 章)对页面中的元素试验 setAttribute()方法。设置特性对页面布局有直接的影响(就像设置对象属性一样)。在顶部文本框中输入以下示例语句，并查看特性值：

```
document.getElementById("myTable").setAttribute("width", "80%")
document.getElementById("myTable").setAttribute("border", "5")
```

相关主题：attributes 属性；document.createAttribute()、getAttribute()、getAttributeNode()、removeAttribute()、removeAttributeNode()、setAttributeNode()方法。

setAttributeNode()

详见 removeAttributeNode()。

setAttributeNodeNS("*attributeNode*")

返回值：Attribute 对象。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+。

此方法插入或替换当前元素中的特性。方法的唯一参数是一个 Attribute 对象，返回值是对新插入的 Attribute 对象的引用。调用该方法时，浏览器在节点间查找匹配的本地名称和命名空间 URI，如果有匹配，节点将替换匹配的节点；否则插入该节点。

相关主题：attributes、namespaceURI、localName 属性；removeAttributeNS()、getAttributeNS()和 setAttributeNS()方法。

setAttributeNS("namespaceURI", "qualifiedName", "value")

返回值：无。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

此方法插入或替换当前元素中由三个参数指定的特性。方法的第一个参数是一个与分配给文档中标签的 URI 相匹配的 URI 字符串，第二个参数是要获取其值的特性的本地名称部分。如果在当前元素中找到了匹配这些参数的项，就将第三个参数中的值赋给现有特性；否则将该值插入为新特性。

相关主题：attributes、namespaceURI、localName 属性；removeAttributeNS()、getAttributeNS()和 setAttributeNodeNS()方法。

setCapture(*containerBoolean*)

(参见 releaseCapture())。

更多资源请访问稀酷客(www.ckook.com)

setExpression("*propertyName*", "*expression*",["*language*"])

返回值：无。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

setExpression()方法可以把可执行表达式的结果赋给元素对象的属性。该方法可把值赋给 HTML 元素对象以及其中的样式对象。

setExpression()方法是把表达式赋给特性的脚本方法，但使用 expression()语法可把表达式直接赋予元素 HTML 标记中的样式表定义，如下例：

```
<p style="width:expression(document.body.style.width * 0.75)">
```

setExpression()方法需要三个参数，第一个参数是赋予表达式的属性名(字符串形式)，属性名区分大小写；第二个参数是表达式的字符串形式，它为属性提供一个值。表达式可引用同一文档中的全局变量或其他对象的属性(但属性不能是数组)，还可包含数学操作符。

要特别注意表达式结果的数据类型，该结果值必须是对属性有效的数据类型，例如，body 背景图像的 URL 必须是一个字符串。但对于数值而言，可互换使用数值和字符串类型，因为该值会转换为属性适用的类型。最好将结果为数值的表达式放在引号内。不是所有情况都需要这样做，但如果有使用引号的习惯，字符串或需要它们的复杂表达式就很少出现问题。

表达式的语言不要局限于 JavaScript，也可在可选的第三个参数中指定表达式的脚本语言，语言可接受的参数值为：

```
JScript
JavaScript
VBScript
```

JScript 和 JavaScript 在所有方面都是一样的，这两种语言都符合 ECMA-262 标准。JavaScript 是 language 参数的默认值。

对动态属性使用 setExpression()的一个原因是让属性总是响应页面上的当前状况。例如，如果属性设置为依赖 body 当前的宽度，则当用户重置窗口的大小时，需要重新计算属性。浏览器会自动响应许多事件，并更新所有的动态属性。本质上，浏览器会重新计算表达式，并把新值应用到属性中。尤其是键盘事件会触发这类自动重新计算操作。但如果脚本自行执行动作(换句话说，动作不是由事件触发的)，则需要强制重新计算表达式。document.recalc()方法会处理这些重新计算操作，但在这些情况下，必须调用它，来强制重新计算动态属性。

示例

程序清单 26-32 显示了在基于 DHTML 的时钟中工作的 setExpression()、recalc()和 getExpression()方法。图 26-1 显示了此时钟。随着时间的推移，表示小时、分钟和秒的横条会调整其宽度，以反映当前时间。同时，每个横条右侧的 span 元素的 innerHTML 属性会显示横条的当前值。

为了获得此例中的动态计算值，需要不断地创建新的 Date 对象，从客户端计算机时钟中获取当前时间。小时、分钟和秒值都从 now 变量存储的 Date 对象中获取。示例还涉及其他一些计算，用于将这些时间值转换为横条宽度的像素值。横条分为 24 个(对于小时)和 60 个(对于分钟和秒)小块，因此这两种类型的刻度不同。在此示例中，对于有 60 个增量的横条，递增量设置为 5 个像素(存储在 shortWidth 中)，对于 24 个增量的横条，递增量是 shortWidth 的 2.5 倍。

更多资源请访问稀酷客(www.ckook.com)

文档载入时，没有给彩色横条的三个 span 元素提供宽度，所以它们使用默认宽度 0。但在文档载入后，onload 事件处理程序调用 init()函数，该函数设置每个横条宽度的初始值和三个标签 span 的文本(innerHTML)。设置了这些初始值后，init()函数调用 updateClock()函数。

在 updateClock()函数中，为当前时间创建了一个新的 Date 对象。然后调用 document.recalc()方法，让浏览器重新计算在 init()函数中设置的表达式，再将新值赋给属性。为使时钟不断计时，setTimeout()方法设置为每秒调用一次 updateClock()函数。

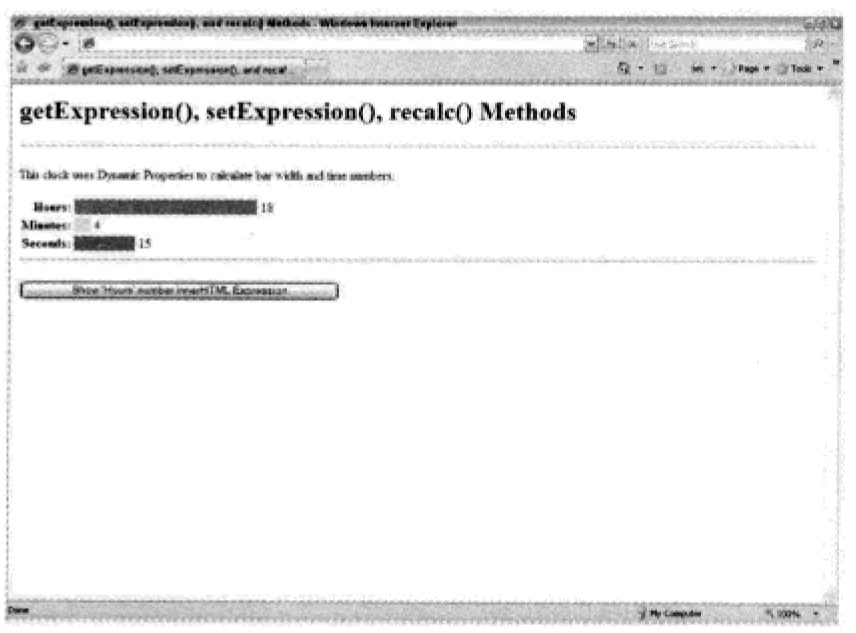


图 26-1　用动态表达式创建的条形图时钟

要看到 getExpression()方法的结果，可单击页面上的按钮，它将显示某个使用 setExpression()分配的一个特性的返回值。

程序清单 26-32　动态属性

HTML: jsb26-32.html

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>getExpression(), setExpression(), and recalc() Methods</title>
      <style type="text/css">
         th
         {
            text-align:right;
         }
         span
         {
            vertical-align:bottom;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-32.js"></script>
   </head>
   <body onload="init()">
      <h1>getExpression(), setExpression(), recalc() Methods</h1>
      <hr />
```


更多资源请访问稀酷客(www.ckook.com)

```
        <p>This clock uses Dynamic Properties to calculate bar width and time
           numbers:
        </p>
        <table border="0">
           <tr>
              <th>Hours:</th>
              <td>
                 <span id="hoursBlock" style="background-color:red"></span>
                  <span id="hoursLabel"></span>
              </td>
           </tr>
           <tr>
              <th>Minutes:</th>
              <td>
                 <span id="minutesBlock" style="background-color:yellow"></span>
                  <span id="minutesLabel"></span>
              </td>
           </tr>
           <tr>
              <th>Seconds:</th>
              <td>
                 <span id="secondsBlock" style="background-color:green"></span>
                  <span id="secondsLabel"></span>
              </td>
           </tr>
        </table>
        <hr />
        <form>
           <input type="button" value="Show 'Hours' number innerHTML Expression"
                  onclick="showExpr()" />
        </form>
    </body>
</html>
```

JavaScript: jsb26-32.js

```
var now = new Date();
var shortWidth = 5;
var multiple = 2.5;

function init()
{
   with (document.all)
   {
     hoursBlock.style.setExpression("width",
        "now.getHours() * shortWidth * multiple","jscript");
     hoursLabel.setExpression("innerHTML",
        "now.getHours()","jscript");
     minutesBlock.style.setExpression("width",
        "now.getMinutes() * shortWidth","jscript");
     minutesLabel.setExpression("innerHTML",
```


更多资源请访问稀酷客(www.ckook.com)

```
            "now.getMinutes()","jscript");
        secondsBlock.style.setExpression("width",
            "now.getSeconds() * shortWidth","jscript");
        secondsLabel.setExpression("innerHTML",
            "now.getSeconds()","jscript");
    }
    updateClock();
}
function updateClock()
{
    now = new Date();
    document.recalc();
    setTimeout("updateClock()",1000);
}
function showExpr()
{
    alert("Expression for the \'Hours\' innerHTML property is:\r\n" +
        document.getElementById("hoursLabel").getExpression("innerHTML") +
        ".");
}
```

相关主题：document.recalc()、removeExpression()、setExpression()方法。

setUserData("*key*", *dataObj*, *dataHandler*)

返回值：对象。

兼容性：WinIE-，MacIE-，NN-，Moz1.7.2+，Safari+，Opera-，Chrome+

setUserData()方法允许向节点添加用户数据，此用户数据表示为对象，通过一个字符串键与节点相关联。setUserData()方法把用户数据对象的关键字作为参数，允许在单个节点上设置多个数据(对象)。该方法的最后一个参数是事件处理函数的引用，只要复制、导入、删除、重命名或采纳数据对象，就调用该函数。

尽管在 Mozl.7.2 中增加了一些对 setUserData()方法的支持，但直到 Mozl.8.1 才允许使用此方法。

swapNode(*otherNodeObject*)

返回值：Node 对象引用。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

swapNode()方法可在元素层次结构内互换两个节点的位置，在交换过程中，两个节点的内容保持不变。此方法的唯一参数必须是有效的节点对象，该对象可以用 document.create Element()创建，或从现有节点中复制。返回值是调用 SwapNode()方法的对象引用。

示例

参见程序清单 26-31 (replaceNode()方法)，来了解 swapNode()方法的用法。

相关主题：removeChild()、removeNode()、replaceChild()、replaceNode()方法。

更多资源请访问稀酷客(www.ckook.com)

tags("*tagName*")

返回值：元素对象数组。

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

tags()方法不属于对象集合的任何元素，而是整个对象集合的方法(例如 all、forms 和 elements)。该方法是当前集合中元素的一种筛选器，例如，为了得到包括文档内所有 p 元素的数组，可使用如下表达式：

```
document.all.tags("P")
```

必须给该方法传递一个参数字符串，它包含要从集合中提取的标记名，标记名不区分大小写。

方法的返回值是当前集合中其标记匹配参数的对象的引用数组。如果没有匹配，返回数组的长度就是 0。如果需要跨浏览器的兼容性，则使用本章前面的 getElementsByTagName()方法，并传递一个通配符“*”。

示例

使用 The Evaluator(参见第 4 章)来试验 tags()方法。在顶部文本框中一次输入一条以下语句，并分析结果：

```
document.all.tags("div")
document.all.tags("div").length
myTable.all.tags("td").length
```

由于 tags()方法返回一个对象数组，因此可将这类返回值用作有效的元素引用：

```
document.all.tags("form")[1].elements.tags("input").length
```

支持标记的浏览器不完全支持 HTML 元素。例如，基于 WebKit 的浏览器不支持把 tags 作为 table 的一部分。

相关主题：getElementsByTagName()方法。

toString("*param*")

返回值：字符串。

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

toString()方法返回元素对象的字符串表示，但是，它对于不同的浏览器可能有不同的含义。在不同的浏览器上它不会得到完全一致的结果，尤其是 IE 只返回一般的字符串"[object]"。

urns("*behaviorURN*")

返回值：元素对象的数组。

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，opera-，Chrome-

urns()方法不属于对象集合中的任何元素，而是整个对象集合的方法。必须给它传递一个字符串参数，它包含赋给该集合中一个或多个元素的操作资源(最常见的是.htc)的 URN (Uniform Resource Name，统一资源名称)。此方法的参数不包括文件的扩展名，如果指定的参数没有匹配的操作 URN，urns()方法就返回一个长度为 0 的数组。该方法与 behaviorUrns 属性相关，该

更多资源请访问稀酷客(www.ckook.com)

属性包含一组赋给单个元素对象的操作 URN。

示例

为防止 urns()方法在将来重新连接到其他元素上，可在程序清单 26-19b 中添加一个按钮和一个函数，以确定 makeHot.htc 操作是否附属于 myP 元素。这个函数如下所示：

```
function behaviorAttached()
{
    if (document.all.urns("makeHot"))
    {
        alert("There is at least one element set to \'makeHot\'.");
    }
}
```

相关主题：behaviorUrns 属性。

5. 事件处理程序

onactivate，ondeactivate

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

onactivate 和 ondeactivate 事件处理程序分别与 onfocus 和 onblur 事件处理程序、IE5.5+中的 onfocusin 和 onfocusout 事件类似。从 IE5.5+开始，就可以分别管理元素激活和元素焦点。onactivate 和 ondeactivate 事件响应元素的激活，而 onfocusin 和 onfocusout 负责处理元素的焦点。在许多情况下，元素激活和获得焦点是同时进行的，但并非总是如此。

如果激活了元素，则在激活之前触发该元素的 onactivate 事件；与此相反，在元素失去焦点之前，按顺序触发 onbeforedeactivate、ondeactivate 和 onblur 事件。只有本来就能自动激活的元素(例如链接和表单输入控件)，或设置了 tabindex 特性的元素，才能成为活动元素(从而触发这些事件)。

WinIE5.5+保留了原始的 onfocus 和 onblur 事件处理程序，但因为它们的行为与 onactivate 好 ondeactivate 事件十分相近，所以最好不要在编写脚本时混用新旧事件处理程序。如果脚本仅用于 IE5.5+(目前这不太可能)，则可以始终使用新术语。如果在 IE 中要跟踪元素的焦点，则考虑使用 onfocus 和 onfocusout。

示例

可修改本章稍后的程序清单 26-34，用 onactivate 代替 onfocus，用 ondeactivate 代替 onblur。

使用 The Evaluator(参见第 4 章)试验 onbeforedeactivate 事件处理程序。首先设置 myP 元素，使其能够接受焦点：

```
myP.tabIndex = 1
```

不断按下 Tab 键，使 myP 段落最终接收到焦点——由其周围的虚点框表示。为了防止元素失去焦点，为 onbeforedeactivate 事件处理程序分配一个匿名函数，如下所示：

```
myP.onbeforedeactivate = new Function("event.returnValue=false")
```

现在随意按下 Tab 键，或随意单击其他可获得焦点的元素，myP 元素将不会失去焦点，直

更多资源请访问稀酷客(www.ckook.com)

到重新载入页面为止(这将清除事件处理程序)。除非要激怒、疏远站点访问者，否则请不要在页面上这样做。

相关主题：onblur、onfocus、onfocusin、onfocusout 事件处理程序。

onafterupdate，onbeforeupdate

兼容性：WinIE4+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

只要更新绑定数据对象中的数据，就在该对象上触发 onafterupdate 和 onbeforeupdate 事件处理程序。onbeforeupdate 事件在更新之前触发，而 onafterupdate 在数据成功更新之后触发。

onbeforecopy

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari1.3+，Opera-，Chrome+

只要用户通过 Edit 菜单(包括 Windows 的 Ctrl+C 组合键和 Mac 的 Command+C 组合键)或右击上下文菜单，来启动复制操作，onbeforecopy 事件处理程序都会在执行实际的复制动作前触发。在所有浏览器中，如果用户通过组合键执行复制操作，复制操作就会完成。如果用户通过 Edit 菜单或上下文菜单访问 Copy 命令，不同的浏览器会有不同的反映。在基于 WebKit 的浏览器中，会显示菜单，用户选择复制选项，就会触发 onbeforecopy 事件。在 IE 中，onbeforecopy 事件在任一个菜单显示前调用。实际上，即使期望事件只调用一次，事件也可能会调用两次。如果只调用了 onbeforecopy 事件，并不能保证用户会完成复制操作(例如，在用户选择前，上下文菜单可能已关闭)。

与粘贴相关的事件不同，onbeforecopy 事件处理程序不使用表单输入元素，而可以处理其他 HTML 元素。

示例

在程序清单 26-33 中，Latin 段落元素的 onbeforecopy 事件处理程序会调用一个函数，该函数仅显示一个警告框。如果用户只在自己的浏览器中使用该程序清单，就可以使用 onbeforecopy 事件处理程序，在执行实际的复制操作之前预处理信息。

程序清单 26-33　onbeforecopy 事件处理程序

HTML: jsb26-33.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onbeforecopy event handler</title>
      <style type="text/css">
         .keyword
         {
            color: blue; font-weight:bold;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-33.js"></script>
```


更多资源请访问稀酷客(www.ckook.com)

```
    </head>
    <body>
        <h1>onbeforecopy event handler</h1>
        <hr />
        <p>Select one or more characters in the Latin paragraph.
            then execute a copy command:
            <ul>
                <li>edit menu</li>
                <li>context menu</li>
                <li>ctrl-c (on Windows)</li>
                <li>command-c (on Mac)</li>
            </ul>
            Browsers that support <span class="keyword">onbeforecopy</span>,
            do not behave consistently. You may not be able to use all
            of the copy commands normally available to you. Test all the
            different ways to copy text and see what happens.
        </p>
        <p>
            Browsers that do not support <span class="keyword">onbeforecopy</span>
            will still allow you to copy -- you just won't see an alert.
        </p>
        <p id="myp" onbeforecopy="beforeCopy()">lorem ipsum dolor sit amet,
            consectetaur adipisicing elit, sed do eiusmod tempor incididunt ut
            labore et dolore magna aliqua. ut enim adminim veniam, quis nostrud
            exercitation ullamco laboris nisi ut aliquip ex commodo consequat.</p>
        <form>
            <p>paste results here:<br />
                <textarea name="output" cols="60" rows="5"></textarea>
            </p>
        </form>
    </body>
</html>
```

JavaScript: jsb26-33.js

```
function beforeCopy()
{
    alert("onbeforecopy is supported by your browser");
}
```

相关主题：onbeforecut、oncopy 事件处理程序。

onbeforecut

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari 1.3+，Opera-，Chrome+

只要用户通过 Edit 菜单(包括 Ctrl+X 组合键)或上下文菜单启动一个内容剪切操作，onbeforecut 事件处理程序都会在执行实际的剪切动作前触发。如果把 onbeforecut 事件处理程序添加到一个 HTML 元素中，上下文菜单常常会使 Cut 菜单项失效；但如果把一个 JavaScript 调用赋给这个事件处理程序，就会使 Cut 菜单项可用。如果用户通过 Edit 菜单或上下文菜单访问 Cut 命令，不同的浏览器会有不同的反映。在基于 WebKit 的浏览器中，会显示菜单，用户选择

更多资源请访问稀酷客(www.ckook.com)

Cut 选项，就会触发 onbeforecut 事件。在 IE 中，onbeforecut 事件在任一个菜单显示前调用。实际上，即使期望事件只调用一次，事件也可能会调用两次。如果只调用了 onbeforecut 事件，就不能保证用户会完成剪切操作(例如，在用户选择前，上下文菜单可能已关闭)。

示例

可使用 onbeforecut 事件处理程序在实际剪切操作之前预处理信息。可编辑程序清单 26-33 的副本来进行试验，将 onbeforecopy 事件处理程序更改为 onbeforecut。

相关主题：onbeforecopy，oncut 事件处理程序。

onbeforedeactivate

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-

详见 onactivate 事件处理程序。

onbeforeeditfocus

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

只要在 Microsoft 的 DHTML Editing ActiveX 控件等环境下编辑页面上的元素，或在 IE5.5+可编辑页面内容的环境下编辑页面上的内容，就会激活 onbeforeeditfocus 事件处理程序。这里只讨论后一种情况，因为它完全在客户端 JavaScript 的范围内。onbeforeeditfocus 事件只在元素接收焦点前调用，此时屏幕上可能没有表示编辑功能已经启用的反馈，但可以自己编写反馈。每次用户单击元素时，都将触发该事件，即使元素在其他地方刚刚接收了焦点也是如此。

示例

使用 The Evaluator(参见第 4 章)研究 WinIE5.5+中的 onbeforeeditfocus。以下语句给 myP 元素的 onbeforeeditfocus 事件处理程序分配一个匿名函数。触发事件处理程序时，该函数将元素的文本颜色变为红色：

```
document.getElementById("myP").onbeforeeditfocus = new
Function("document.getElementById('myP').style.color='red'")
```

现在启用 myP 元素的内容编辑功能：

```
document.getElementById("myP").contentEditable = true
```

如果现在单击页面上的 myP 元素，以编辑其内容，则在开始编辑之前文本变为红色。在为此类用户界面编写的页面中，应该包括某个关闭编辑功能并将颜色变为正常的控件。

相关主题：document.designMode、contentEditable、isContentEditable 属性。

onbeforepaste

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari 1.3+，Opera-，Chrome+

与 onbeforecopy 和 onbeforecut 一样，onbeforepaste 事件在不同的浏览器上也是不同的，在基于 WebKit 的浏览器中，会显示菜单，用户选择 paste 选项，就会触发 onbeforepaste 事件。在 IE 中，选择当前对象(或选择了其中一个元素)后，就会调用 onbeforepaste 事件，然后显示上下

更多资源请访问稀酷客(www.ckook.com)

文菜单或菜单栏的 Edit 菜单。使用脚本控制复杂对象的复制和粘贴过程时，该事件非常重要。这类复杂对象可能有相关的多种数据，但脚本只会捕获一个数据类型。也可将一些与复制项(例如，元素的 id 属性)相关的数据放到剪贴板中。使用 onbeforepaste 事件处理程序将 event.returnValue 属性的值设置为 false，可保证所粘贴的项在上下文菜单或 Edit 菜单中可用(假定剪贴板包含一些内容)。接着，onpaste 调用的处理程序应把剪贴板上指定的数据子集应用于当前选中的项。

示例

在本章后面有关 onpaste 事件处理程序的程序清单 26-44 中，可查看 onbeforepaste 和 onpaste 事件处理程序的用法。

相关主题： oncopy、oncut、onpaste 事件处理程序。

onbeforeupdate

参见 onafterupdate。

onblur

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

一个元素要接收焦点，而使另一个具有焦点的元素失去焦点时，就会触发 onblur 事件。例如，用户使用 Tab 键从文本输入元素移动到表单中的下一个元素时，文本输入元素就触发 onblur 事件，第一个元素的 onblur 事件在下一元素的 onfocus 事件之前调用。

onblur 事件的可用性随着脚本浏览器的不断推出而提高。在早期的版本中，onblur 和 onfocus 仅用于面向文本的输入元素(包括 select 元素)，这些事件可在所有的脚本浏览器版本中安全地使用。window 对象从 NN3 和 IE4 开始就接收 onblur 事件处理程序；IE4 还将该事件处理程序扩展到更多的表单元素中，主要在 Windows 操作系统上，因为该操作系统的用户界面用虚点框表示按钮和链接等项接收到焦点(以通过键盘的空格键启动它们)。IE5+中的 onblur 事件处理程序可用于每个 HTML 元素。然而对于其中大多数元素来说，除非给元素标记的 tabindex 特性赋值，否则就没有 onblur 和 onfocus。例如，如果在<p>标记内指定 tabindex="1"，就可以单击段落，使段落获得焦点，或按下 Tab 键，直到该项接收到焦点。

如果准备在窗口或面向文本的输入元素上使用 onblur 事件处理程序，可能会有一些无法预料、不想要的脚本结果。例如，在 IE 中，如果要使页面上的任何元素获得焦点，则当前具有焦点的 window 对象就会失去焦点(并触发 onblur 事件)。同样，onblur、onfocus 和 alert()对话框之间的交互也可能使文本输入元素产生错误，所以一般推荐使用 onchange 事件处理程序来引发表单验证程序。如果要对同一个元素使用 onblur 和 onchange 事件处理程序，onchange 事件就在 onblur 之前触发。使用这个事件处理程序进行数据验证的详细信息，可参见配书光盘中的第 46 章。

IE5.5+添加了 ondeactivate 事件处理程序，它在 onblur 事件处理程序之前触发。如果 onbeforedeactivate 事件处理函数将 event.returnValue 设置为 false，就会禁止触发 onblur 和 ondeactivate 事件。

示例

页面编写者经常使用 onblur 事件处理程序对用户严加控制，如防止用户退出文本框，除非用户在框中输入了内容。这不是 Web 的友好实践方式，最好不要这样做，而应采用更聪明的方式，确保在提交表单前，用户已在域中输入了内容(参见配书光盘上的第 46 章)。程序清单 26-34

更多资源请访问稀酷客(www.ckook.com)

简单地演示了 tabindex 特性对 onblur 和 onfocus 事件的影响。注意，在按下 Tab 键时，只有第二段触发了事件，尽管所有三个段落都分配了事件处理程序。

程序清单 26-34　onblur 和 onfocus 事件处理程序

HTML: jsb26-34.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onblur and onfocus Event Handlers</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-34.js"></script>
   </head>
   <body>
      <h1 id="H1" tabindex="2">onblur and onfocus Event Handlers</h1>
      <h2>Start tabbing and see what happens</h2>
      <hr />
      <p id="P1" onblur="showBlur()" onfocus="showFocus()">Lorem ipsum dolor
         sit amet, consectetaur adipisicing elit, sed do eiusmod tempor
         incididunt ut labore et dolore magna aliqua. Ut enim adminim veniam,
         quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
         commodo consequat.
      </p>
      <p id="P2" tabindex="1" onblur="showBlur()" onfocus="showFocus()">Bis
         nostrud exercitation ullam mmodo consequet. Duis aute involuptate
         velit esse cillum dolore eu fugiat nulla pariatur. At vver eos et
         accusam dignissum qui blandit est praesent luptatum delenit
         aigueexcepteur sint occae.
      </p>
      <p id="P3" onblur="showBlur()" onfocus="showFocus()">Unte af phen
         neigepheings atoot Prexs eis phat eit sakem eit vory gast te Plok
         peish ba useing phen roxas. Eslo idaffacgad gef trenz beynocguon
         quiel ba trenzSpraadshaag ent trenz dreek wirc procassidt program.
      </p>
   </body>
</html>
```

JavaScript: jsb26-34.js

```
function showBlur()
{
   var id = event.srcElement.id;
   alert("Element \"" + id + "\" has blurred.");
}
function showFocus()
{
   var id = event.srcElement.id;
   alert("Element \"" + id + "\" has received focus.");
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：blur()、focus()方法；ondeactivate、onbeforedeactivate、onfocus、onactivate 事件处理程序。

oncellchange

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

oncellchange 事件处理程序是 IE 数据绑定的组成部分，当数据提供器(通常是一个绑定控件)中的数据改变时，就会触发该事件。在响应此事件时，可分析 dataFld 属性，确定记录集中的哪个字段发生了变化。

onclick

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

当指针指在元素上时，如果用户按下基本鼠标键并释放(必须在同一个元素矩形内向下和向上按键)，就触发 onclick 事件。在操作系统如 Windows 中，也可用非鼠标单击的方式触发该事件。例如，可通过键盘使一个可单击对象获得焦点，然后按下空格键或 Enter 键，来执行与单击该元素相同的动作。在 IE 中，如果元素对象支持 click()方法，onclick 事件就和该方法一起调用(注意，这不能用于 Navigator 或其他浏览器)。

onclick 事件与其他鼠标事件密切相关。这些相关的事件是 onmousedown、onmouseup 和 ondblclick。当用户使用鼠标向下单击时，会触发 onmousedown 事件，接下来触发 onmouseup 事件(当释放鼠标按钮时)。只有先触发同一对象的 onmousedown 和 onmouseup 事件，才触发 onclick 事件。关于 onmousedown 和 onmouseup 事件的讨论，请参见本章后面使用它们的例子。

与 ondblclick 事件的交互十分简单：首先调用 onclick 事件(首次单击后)，接着触发 ondblclick 事件(第二次单击后)。本章后面讨论 ondblclick 事件处理程序时，会介绍这两个事件处理程序的更多交互内容。

与用户单击时执行固有动作的对象(即链接和区域)一起使用时，onclick 事件处理程序可执行所有动作，包括导航到元素 href 特性指向的目的地。例如，为与所有脚本浏览器兼容，如果用<a>链接标记包围图像的标记，图像就是可单击的，这样该标记的 onclick 事件就会替代早期<img>标记所没有的 onclick 事件处理程序。如果指定一个没有特殊保护的 onclick 事件处理程序，该事件处理程序会执行，也会执行元素的固有动作，因此需要阻止固有动作的执行。为此，该事件处理程序必须返回 false。这有两种方式：第一是把 return false 语句添加到指定该事件处理程序的脚本语句中：

```
<a href="#" onclick="yourFunction(); return false"><img...></a>
```

另外，还可以让事件处理程序调用的函数提供 return false 语句的 false 部分，如下所示：

```
function yourFunction()
{
   // [statements that do something here]
   return false;
}
...
<a href="#" onclick="return yourFunction()"><img...></a>
```


更多资源请访问稀酷客(www.ckook.com)

这两种方法都是可行的，第三种方法根本不使用 onclick 事件处理程序，只把 javascript:伪 URL 赋给 href 特性(详见第 30 章的 Link 对象)。

IE4+中的事件模型还提供了一种方法，来阻止用户单击对象时触发对象的固有动作。如果 onclick 事件处理函数将 event 对象的 returnValue 属性设置为 false，就会取消固有动作。为此，只需在事件处理程序调用的函数中包含下列语句：

```
event.returnValue = false;
```

W3C DOM 的事件模型可用另一种方法来取消默认动作：在事件的处理函数中，调用 eventObj.cancelDefault()方法。

脚本编程的初学者常犯的一个错误是，将提交类型的输入按钮用作执行脚本动作的按钮，而不是提交表单的按钮。典型的情况是给 submit 类型的 input 元素赋予 onclick 事件处理程序，来执行一些本地动作。提交输入按钮有一个固有操作，就像链接和区域一样。如前所述，尽管可以禁止该固有操作，但应使用一个 button 类型的 input 元素。

如果在实现 onclick 事件处理程序时遇到了困难(例如，试图确定使用哪个鼠标按钮进行单击)，则可能是操作系统或浏览器的默认操作妨碍了脚本。通常可以通过 onmousedown 事件处理程序来得到需要的数据，但要注意使用鼠标右键单击对象时，不会调用 onmouseup 事件。尽可能使用 onclick 事件处理程序来捕获用户的单击，因为该事件的操作最像用户在日常计算工作中所习惯的那样，但遇到问题时最好使用 onmousedown。

示例

onclick 事件处理程序是最容易掌握和使用的事件处理程序之一。程序清单 26-35 演示了该处理程序与 ondblclick 事件处理程序的交互，还说明了在与 click 事件一起使用时，如何防止触发链接的固有动作。在单击和/或双击链接时，span 元素中的文本显示与每个事件相关的消息。注意，如果是双击，会先触发 click 事件，接着第一条消息立即被第二条消息替代。为便于演示，这里给出了取消链接的固有动作的两种向后兼容方式。在实践中，应确定一种方式并坚持使用。

程序清单 26-35　使用 onclick 和 ondblclick 事件处理程序

HTML: jsb26-35.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onclick and ondblclick Event Handlers</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-35.js"></script>
   </head>
   <body>
      <h1>onclick and ondblclick Event Handlers</h1>
      <hr />
      <a href="#" onclick="showClick();return false"
         ondblclick="return showDblClick()">A sample link.</a>
```


更多资源请访问稀酷客(www.ckook.com)

```
      (Click type: <span id="clickType"></span>)
   </body>
</html>
```

JavaScript: jsb26-35.js

```
var timeout;
function clearOutput()
{
   document.getElementById("clickType").innerHTML = "";
}
function showClick()
{
   document.getElementById("clickType").innerHTML = "single";
   clearTimeout(timeout);
   timeout = setTimeout("clearOutput()", 3000);
}
function showDblClick()
{
   document.getElementById("clickType").innerHTML = "double";
   clearTimeout(timeout);
   timeout = setTimeout("clearOutput()", 3000);
   return false;
}
```

相关主题：click()方法；oncontextmenu、ondblclick、onmousedown、onmouseup 事件处理程序。

oncontextmenu

兼容性：WinIE5+，MacIE-，NN7+，Moz+，Safari+，Opera-，Chrome+

用鼠标右键单击对象时，会触发 oncontextmenu 事件。用右键触发的、仅与单击相关的事件是 onmousedown 和 oncontextmenu。

为了禁止 oncontextmenu 事件显示固有的应用程序菜单，可以使用浏览器中的任意事件取消方法，如前面的 onclick 事件处理程序所述，两种事件取消方法分别是让事件处理程序返回 false；把 false 赋给 event.returnValue 属性。我们常常要阻止上下文菜单的显示，但这样用户就不能下载图像的副本，或查看框架的源代码。然而，如果用户关闭了 WinIE5+中的 Active Scripting，该事件处理程序就不能阻止上下文菜单的显示。

该事件的另一个可能操作是显示用其他 DHTML 工具构造的自定义上下文菜单。在这种情况下，也必须禁用内在的上下文菜单，这样两个菜单就不会同时显示。

示例

有关使用自定义上下文菜单和 oncontextmenu 事件处理程序的例子，请参见程序清单 26-30。

相关主题：releaseCapture()、setCapture()方法。

oncontrolselect

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

当页面处于编辑模式，用户在可编辑元素上进行选择时，就会触发 oncontrolselect 事件。

更多资源请访问稀酷客(www.ckook.com)

注意，触发这个事件的是被选中的元素本身，而非元素中的内容。

相关主题：onresizeend、onresizestart 事件处理程序。

oncopy，oncut

兼容性：WinIE5+，MacIE4+，NN-，Moz+，Safari1.3+，Opera-，Chrome+

用户或脚本在当前对象上启动复制或剪切编辑动作后，就会触发 oncopy 和 oncut 事件。每个事件都在相关的 before 事件之后触发，并在 Edit 菜单或上下文菜单显示前触发，如果用键盘快捷键启动，则这些事件在复制或剪切动作之前触发。

这些事件处理程序通常为不允许复制或剪切的元素提供编辑功能。这种情况下，需要将 onbeforecopy 或 onbeforecut 事件处理程序的 event.returnValue 设置为 false，使上下文菜单或 Edit 菜单中的 Copy 或 Cut 菜单项可用。然后 oncopy 或 oncut 事件处理程序必须通过 IE 中 clipboardData 对象的 setdata()方法，把一个值手动填充到剪贴板中。在 oncopy 或 oncut 事件处理程序中使用 setdata()方法时，也必须在处理函数中将 event.returnValue 属性设置为 false，以避免默认的复制或剪切动作擦去剪贴板的内容。

编程人员控制着剪贴板中存储的数据，所以剪贴板可以不只包含数据的一个直接副本。例如，剪贴板可以存放 image 对象的 src 属性值，以便用户把它粘贴到页面上的其他地方。

在 oncut 事件处理程序中，脚本还负责从页面上剪切元素或选择内容。要删除元素的所有内容，可将元素的 innerHTML 或 innerText 属性设置为空字符串。可使用 selection.createRange()方法创建 TextRange 对象，其内容可通过 TextRange 对象的方法来操作。

示例

程序清单 26-36 显示了 onbeforecut 和 oncut 事件处理程序(以及 onbeforepaste 和 onpaste)的用法。注意，handleCut()函数不仅将所选的词填充到 IE 的 clipboardData 对象中，还从表格单元格元素中删除了所选的文本。如果用 onbeforecopy 和 oncopy 替换 onbeforecut 和 oncut 事件处理程序(并更改 handleCut()，以免删除事件源元素的内部文本)，就通过复制和粘贴而不是剪切和粘贴来完成操作。本章稍后的程序清单 26-44 将说明这一点。

程序清单 26-36 脚本控制下的剪切和粘贴

HTML: jsb26-36.html

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onbeforecut and oncut Event Handlers Using a range object</title>
      <style type="text/css">
         td
         { text-align:center;
         }
         th
         { text-decoration:underline;
         }
         .blanks
         { text-decoration:underline;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
    </style>
    <script type="text/javascript" src="../jsb-global.js"></script>
    <script type="text/javascript" src="jsb26-36.js"></script>
  </head>
  <body>
    <h1>onbeforecut and oncut Event Handlers</h1>
    <hr />
    <p>Your goal is to cut and paste one noun and one adjective from the
       following table into the blanks of the sentence. Select a word from
       the table and use the Edit or context menu to cut it from the table.
       Select one or more spaces of the blanks in the sentence and choose
       Paste to replace the blank with the clipboard contents.
    </p>
    <table cellpadding="5" onbeforecut="selectWhole()" oncut="handleCut()">
       <tr>
          <th>Nouns</th>
          <th>Adjectives</th>
       </tr>
       <tr>
          <td>truck</td>
          <td>round</td>
       </tr>
       <tr>
         <td>doll</td>
         <td>red</td>
       </tr>
       <tr>
         <td>ball</td>
         <td>pretty</td>
       </tr>
    </table>
    <p id="myP" onbeforepaste="handleBeforePaste()" onpaste="handlePaste()">
       Pat said, "Oh my, the <span id="blank1"
       class="blanks">     </span> is so <span
       id="blank2" class="blanks">     </span> !"
    </p>
    <button onclick="location.reload()">Reset</button>
  </body>
</html>
```

JavaScript: jsb26-36.js

```
function selectWhole()
{
   var obj = window.event.srcElement;
   var range = document.body.createTextRange();
   range.moveToElementText(obj);
   range.select();
   event.returnValue = false;
}
```


更多资源请访问稀酷客(www.ckook.com)

```
function handleCut()
{
   var rng = document.selection.createRange();
   clipboardData.setData("Text",rng.text);
   var elem = event.srcElement;
   elem.innerText = "";
   event.returnValue = false;
}

function handlePaste()
{
   var elem = window.event.srcElement;
       if (elem.className == "blanks")
       {
           elem.innerHTML = clipboardData.getData("Text");
       }
   event.returnValue = false;
}

function handleBeforePaste()
{
   var elem = window.event.srcElement;
      if (elem.className == "blanks")
      {
         event.returnValue = false;
      }
}
```

相关主题：onbeforecop、onbeforecut、onbeforepaste 和 onpaste 事件处理程序。

ondataavailable，ondatasetchanged，ondatasetcomplete

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

这三个事件是 IE 数据绑定的一部分，触发它们可帮助反映所传输数据的状态。当数据从数据源中传送出来时，会触发 ondataavailable 事件，而 ondatasetcomplete 事件表明记录集已从数据源中下载完毕。当以某种方式改变数据源的记录集时，会触发 ondatasetchanged 事件。

ondblclick

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

双击动作在第二次单击后会调用 ondblclick 事件，单击之间的时间间隔取决于客户机的鼠标控制面板设置，onclick 事件也会触发，但只在第一次单击后触发。

一般情况下，在鼠标单击时完成一个任务，而双击时完成另一个任务并不是很好的设计。在现代浏览器中使用该事件序列是不现实的(甚至用户在第二次单击时，onclick 事件也总是会触发)。但让鼠标按下的动作触发一些辅助动作是很常见的。在大多数基于图标的文件系统中可看到这个应用：如果单击一个文件图标，在鼠标按下时它是高亮显示的，以选择该文件；还可双击该文件来启动它。在上述情况下，一个事件的动作不会阻止另一个事

更多资源请访问稀酷客(www.ckook.com)

件，也不会使用户感到迷惑。

示例

有关 ondblclick 事件的用法，请参见程序清单 26-35(用于 onclick 事件处理程序)。

相关主题：onclick、onmousedown、onmouseup 事件处理程序。

ondrag，ondragend，ondragstart

兼容性：WinIE5+，MacIE-，NN-，Moz+，Safari 1.3+，Opera-，Chrome+

在 ondragstart 事件之后会触发 ondrag 事件，用户在屏幕上拖动一个选项或对象时，该事件会重复触发。onmousemove 事件仅当光标在屏幕上移动时触发，而 ondrag 事件即使在光标静止时也会触发。在大多数浏览器中，用户可将对象拖到其他浏览器窗口或其他应用程序中，当拖动操作扩展到浏览器窗口之外时，就会触发该事件。

因为不管被拖动的对象是什么，该事件都会触发，所以可在游戏或培训环境中使用它，在这种环境中，用户完成拖动操作(例如匹配对象的相似对)的时间是固定的。如果浏览器支持可下载的光标，ondrag 事件可遍历一系列光标版本，来形成一个动画光标。

如果脚本需要对动作进行微观管理(通常没必要对基本的拖放操作进行微观管理)，理解用户拖动过程中拖动相关事件的触发次序会有帮助。考虑图 26-2 中的拖放操作。

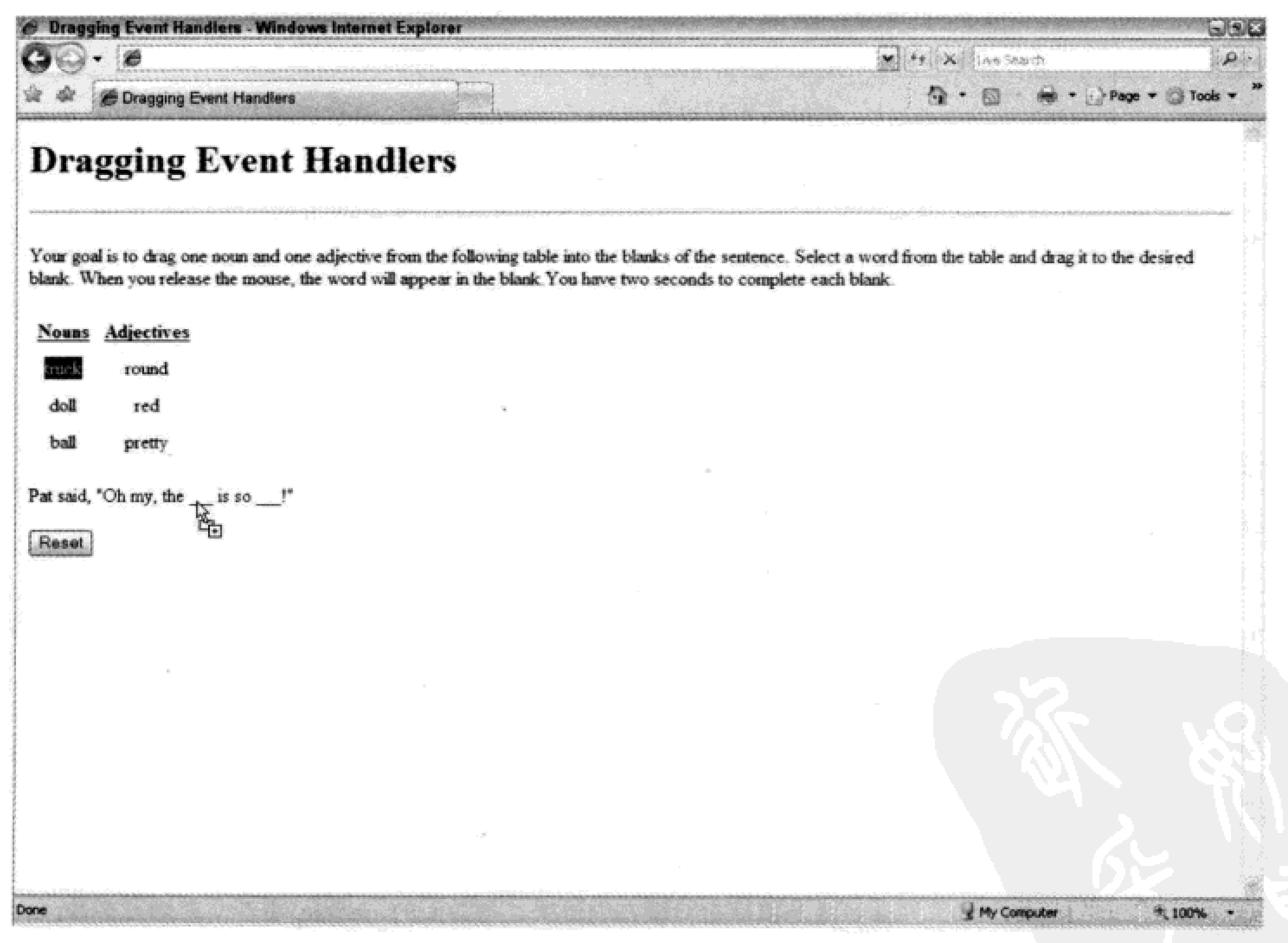


图 26-2 典型的拖放操作

将包含可拖动内容的表单元格像电子表格单元那样命名会很有帮助：truck 是 Al 单元，round 是 Bl 单元，doll 是 A2 单元等。在拖动过程中，许多对象是各种拖动相关事件的目标。表 26-18 列出了事件顺序和事件目标。

更多资源请访问稀酷客(www.ckook.com)

表 26-18 事件以及事件在典型拖放操作期间的目标

事　件	目　标	说　明
ondragstart	单元格 A1	拖放操作中触发的第一个事件
ondrag	单元格 A1	在拖放过程中不断在此目标上触发此事件，但其他事件分散到其他目标上
ondragenter	单元格 A1	在源元素中发生第一次移动时，即使鼠标指针尚未从单元格 A1 移开，也会触发 ondragenter 事件
ondragover	单元格 A1	在鼠标指针此时停留的元素上不断触发。如果用户在拖动过程中只是按下鼠标按钮，但没有移动鼠标指针，ondrag 和 ondragover 事件不断交替触发
(repetition)	单元格 A1	当鼠标指针停留在单元格 A1 上时，ondrag 和 ondragover 事件交替触发
ondragenter	table	当鼠标指针触及 table 元素(由边框和/或单元格填充表示)时，table 元素会接收 ondragenter 事件
ondragleave	单元格 A1	在另一个元素上触发 ondragenter 事件后，触发 ondragleave 事件
ondrag	单元格 A1	仍然不断触发
ondragover	table	此事件的源元素转为表格，因为这是鼠标指针此时所在的位置。如果鼠标指针没有从这个地方移走，ondrag(单元格 A1)和 ondragover(表格)事件不断交替触发
ondragenter	单元格 B1	拖动操作正在从 table 边框空间前进到单元格 B1
ondragleave	table	当鼠标指针退出 table 元素的空间时，table 元素接收到 ondragleave 事件
ondrag	单元格 A1	ondrag 事件不断在单元格 A1 对象上触发
ondragover	单元格 B1	现在鼠标指针在单元格 B1 上，因此触发该对象的 ondragover 事件。此事件与前面的 ondrag 事件交替触发多次(取决于计算机和用户拖动操作的速度)
		当鼠标指针从单元格 B1 经过 table 边框到单元格 B2，再经过 table、单元格 B3，最后到达 table 的最外缘时，这个事件会触发多次
ondragenter	body	在 table 之外的空 body 元素上自由拖动
ondragleave	table	用户刚刚离开了 table
ondrag	单元格 A1	仍然活动着，并接收此事件
ondragover	body	这是鼠标指针现在的位置。与前面的 ondrag 事件交替触发多次(取决于计算机和用户拖动操作的速度)
ondragenter	blankl	鼠标指针到达 ID 为 blank1 的 span 元素，这是空下划线所在的位置
ondragleave	body	刚刚离开 body 到达空白位置
ondrag	单元格 A1	仍然在不停地触发
ondragover	blankl	这是鼠标指针现在的位置。与前面的 ondrag 事件交替触发多次(取决于计算机和用户拖动动作的速度)
ondrop	blankl	span 元素获得最近一次拖动的通知
ondragend	单元格 A1	最初的源元素获得最后通知，拖动操作完成。即使拖动不成功，也触发此事件，因为拖动不是在拖动目标上结束

更多资源请访问稀酷客(www.ckook.com)

实际上，表 26-18 中的一些事件是不会触发的，大部分事件都和需要捕获多少事件处理程序来执行脚本有关。另一个主要因素是用户执行拖放操作的物理速度(与 CPU 处理速度互相影响)。最可能跳过的事件是 ondragenter 和 ondragleave，如果用户在 ondragover 事件触发前滑过一个对象，可能还会跳过一些 ondragover 事件。

尽管拖动相关事件的可靠性是不确定的，但总会触发几个重要事件。ondragstart、ondrop(如果位于拖放目标上)和 ondragend 事件，还有一些 ondrag 事件肯定会在屏幕上拖动元素时触发。除 ondrop 外的所有事件都在源元素上触发，而 ondrop 在事件目标上触发。

示例

程序清单 26-37 给出了几个与拖动相关的事件处理程序。其页面类似于程序清单 26-36 中的示例，但页面的脚本完全不同。在此示例中，用户要从 Nouns 和 Adjectives 列中选择一些词，并将其拖动到句子的空白处。为加强演示效果，程序清单 26-37 显示了如何将数组数据从拖动源传递到拖动目标。另外，用户完成每次拖动操作的时间是固定不变的(2s)。

ondragstart 和 ondrag 事件处理程序放在＜body＞标记中，因为这些事件会从用户试图拖动的元素上向上冒泡。这些事件处理程序调用的脚本会筛选事件，使表格中的热点元素只触发所需的操作。对事件处理程序采用这种处理，就不必为每个表单元格重复事件处理程序。

ondragstart 事件处理程序调用 setupDrag()函数。此函数取消 ondragstart 事件，除非目标元素(要拖动的元素)是表格中的某个 td 元素。为了使此应用程序知道哪个词拖动到哪个空白位置，它不仅传递了词的文本，还传递了单词的一些额外信息，这允许另一个事件处理程序确保名词拖动到第一个空白位置，而形容词拖动到第二个空白位置。为帮助完成这一操作，把类名分配给 td 元素，以便区分来自 Nouns 列和来自 Adjectives 列的词。setupDrag()函数生成一个由事件源元素的 innerText 和元素类名组成的数组，但 event.dataTransfer 对象不能存储数组类型的数据，因此 Array.join()方法将数组转换为用分号分隔各项的字符串，然后，此字符串填充到 event.dataTransfer 对象中。此对象会在拖放操作过程中显示鼠标指针，这样当鼠标指针位于拖放目标上时，变成复制样式。最后，setupDrag()是在拖动操作中执行的第一个函数，因此把计时器设置为当前时钟时间，对拖动操作计时。

ondrag 事件处理程序(在 body 中)捕获 ondrag 事件，ondrag 事件由作为动作源元素的表单元格元素触发。每次触发该事件时(在拖动操作中会触发大量该事件)，就调用 timeIt()函数，来比较当前时间和拖动开始时设置的参考时间(全局变量 timer)。如果时间超过 2 秒(2000 微秒)，就显示一个警告对话框，通知用户。要关闭警告对话框，用户必须释放鼠标按钮，以终止拖动操作。

要将空白 span 元素变为拖放目标，其 ondragenter、ondragover 和 ondrop 事件处理程序必须将 event.returnValue 设置为 false，event.dataTransfer.dropEffect 属性也应设置为所需的效果(这里是 copy)。为简单起见，这些事件处理程序放在包含两个 span 元素的 p 元素中。但注意，只有目标元素是其 ID 以空白开头的某个 span 元素，cancelDefault()函数才会工作。

当用户释放鼠标按钮时，ondrop 事件处理程序就调用 handleDrop()函数。此函数从 event.dataTransfer 中获取字符串数据，并用 String.split()方法将其还原为数组数据类型。进行一些测试，确保词性(名称或形容词)与适当的空白位置相关联。这样，源元素的文本设置为拖放目标的 innerText 属性，否则将显示一条错误消息，来帮助用户弄清问题所在。

更多资源请访问稀酷客(www.ckook.com)

程序清单 26-37　使用与拖动相关的事件处理程序

HTML: jsb26-37.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Dragging Event Handlers</title>
    <style type="text/css">
      td
      {
        text-align:center;
      }
      th
      {
        text-decoration:underline;
      }
      .blanks
      {
        text-decoration:underline;
      }
    </style>
    <script type="text/javascript" src="../jsb-global.js"></script>
    <script type="text/javascript" src="jsb26-37.js"></script>
  </head>
  <body ondragstart="setupDrag()" ondrag="timeIt()">
    <h1>Dragging Event Handlers</h1>
    <hr />
    <p>Your goal is to drag one noun and one adjective from the following
      table into the blanks of the sentence. Select a word from the table
      and drag it to the desired blank. When you release the mouse, the word
      will appear in the blank. You have two seconds to complete each blank.
    </p>
    <table cellpadding="5">
      <tr>
        <th>Nouns</th>
        <th>Adjectives</th>
      </tr>
      <tr>
        <td class="noun">truck</td>
        <td class="adjective">round</td>
      </tr>
      <tr>
        <td class="noun">doll</td>
        <td class="adjective">red</td>
      </tr>
      <tr>
        <td class="noun">ball</td>
        <td class="adjective">pretty</td>
      </tr>
```


更多资源请访问稀酷客(www.ckook.com)

```
    </table>
    <p id="myP" ondragenter="cancelDefault()" ondragover="cancelDefault()"
       ondrop="handleDrop()">Pat said, "Oh my, the <span id="blank1"
       class="blanks">     </span> is so <span
       id="blank2" class="blanks">     </span> !"
    </p>
    <button onclick="location.reload()">Reset</button>
  </body>
</html>
```

JavaScript: jsb26-37.js

```
var timer;
function setupDrag()
{
  if (event.srcElement.tagName != "TD")
  {
     // don't allow dragging for any other elements
     event.returnValue = false;
  }
  else
  {
     // setup array of data to be passed to drop target
     var passedData = [event.srcElement.innerText, event.srcElement.className];
     // store it as a string
     event.dataTransfer.setData("Text", passedData.join(":"));
     event.dataTransfer.effectAllowed = "copy";
     timer = new Date();
  }
}

function timeIt()
{
  if (event.srcElement.tagName == "TD" && timer)
  {
     if ((new Date()) - timer > 2000)
     {
        alert("Sorry, time is up. Try again.");
        timer = 0;
     }
  }
}

function handleDrop()
{
  var elem = event.srcElement;
  var passedData = event.dataTransfer.getData("Text");
  var errMsg = "";
      if (passedData)
      {
         // reconvert passed string to an array
```


更多资源请访问稀酷客(www.ckook.com)

```
            passedData = passedData.split(":");
            if (elem.id == "blank1")
            {
                if (passedData[1] == "noun")
                {
                    event.dataTransfer.dropEffect = "copy";
                    event.srcElement.innerText = passedData[0];
                }
                else
                {
                    errMsg = "You can't put an adjective into the noun placeholder.";
                }
            }
            else if (elem.id == "blank2")
        {
            if (passedData[1] == "adjective")
            {
                event.dataTransfer.dropEffect = "copy";
                event.srcElement.innerText = passedData[0];
            }
            else
            {
                errMsg = "You can't put a noun into the adjective placeholder.";
            }
        }
        if (errMsg)
        {
            alert(errMsg);
        }
    }
}
function cancelDefault()
{
    if (event.srcElement.id.indexOf("blank") == 0)
    {
        event.dataTransfer.dropEffect = "copy";
        event.returnValue = false;
    }
}
```

程序清单 26-37 未显示的一个事件处理程序是 ondragend。使用此事件可显示每个成功的拖动操作所用的时间。由于事件在拖动源元素上触发，因此可在<body>标记中实现它，并筛选事件，其方式类似于 ondragstart 或 ondrag 事件处理程序筛选 td 元素的事件。

相关主题：event.dataTransfer 对象；ondragenter、ondragleave、ondragover、ondrop 事件处理程序。

ondragenter，ondragleave，ondragover

兼容性：WinIE5+，MacIE-，NN-，Moz+，Safari 1.3+，Opera-，Chrome+

更多资源请访问稀酷客(www.ckook.com)

这些事件在拖动过程中触发。光标进入页面上元素的矩形空间时，就触发该元素上的 ondragenter 事件。紧接着触发另一个元素上的 ondragleave 事件，光标从这个元素来到当前元素。这似乎是按物理动作的顺序发生，但事件总以这个顺序触发。根据客户计算机 CPU 的速度和用户拖动动作的速度，这些事件的一个或多个也许不会触发，特别是如果物理动作太快，计算机就可能无法触发这些事件。

如果拖动时光标停在一个元素上，会不断地触发 ondragover 事件。从页面上的一个位置拖动到另一个位置的过程中，ondragover 事件的目标随着光标下的元素而改变。如果没有触发其他与拖动相关的事件(在拖动过程中仍然按着鼠标按钮，但光标没有移动)，ondrag 和 ondragover 事件会不断地交替触发。

应让拖放目标元素的 ondragover 事件处理程序将 event.returnValue 属性设置为 false。请参见本章前面讨论的 ondrag 事件处理程序，来进一步了解拖动相关事件的触发顺序。

示例

程序清单 26-38 给出了 ondragenter 和 ondragleave 事件处理程序的用法。简单的页面 (通过状态栏)显示光标进入页面上某个元素的时间。当拖动鼠标指针离开该元素时，ondragleave 事件处理程序就隐藏状态栏消息。由于没有为此页面定义拖放目标，因此拖动项目时，鼠标指针显示为不能放下的形状。

程序清单 26-38　使用 ondragenter 和 ondragleave 事件处理程序

HTML: jsb26-38.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>ondragenter and ondragleave Event Handlers</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-38.js"></script>
   </head>
   <body>
      <h1 ondragenter="showEnter()" ondragleave="clearMsg()">
         ondragenter and ondragleave Event Handlers
      </h1>
      <hr />
      <p>Select any character(s) from this paragraph, and slowly drag it around
         the page. When the dragging action enters the large header above, the
         status bar displays when the onDragEnter event handler fires. When you
         leave the header, the message is cleared via the onDragLeave event
         handler.
      </p>
   </body>
</html>
```

JavaScript: jsb26-38.js

```
function showEnter()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   status = "Entered at: " + new Date();
   event.returnValue = false;
}
function clearMsg()
{
   status = "";
   event.returnValue = false;
}
```

相关主题：ondrag、ondragend、ondragstart、ondrop 事件处理程序。

ondragstart

(参见 ondrag)

ondrop

兼容性：WinIE5+，MacIE-，NN-，Moz+，Safari 1.3+，Opera-，Chrome+

一旦用户在拖放操作结束时放开鼠标按钮，就在拖放目标元素上触发 ondrop 事件。对于 IE，Microsoft 推荐在目标元素上使用 ondragenter、ondragover 和 ondrop 事件处理程序来表示拖放目标。在每个事件处理程序中，应将 dataTransfer.dropEffect 设置为在拖放操作中显示的效果(对每种类型使用不同的光标表示)。这些设置应该匹配通常在 ondragstart 事件处理程序中设置的 dataTransfer.effectAllowed 属性。这三个与拖放相关的处理程序都应将 event.returnValue 属性设置为 false，以覆盖事件的默认操作。参见本章前面讨论的 ondrag 事件处理程序，来进一步了解拖动相关事件的触发顺序。

示例

参见 ondrag 事件处理程序的程序清单 26-37，了解如何在典型的拖放情形中应用 ondrop 事件处理程序。

相关主题：event.dataTransfer 对象；ondrag、ondragend、ondragenter、ondragleave、ondragover、ondragstart 事件处理程序。

onerrorupdate

兼容性：WinIE4+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

onerrorupdate 事件处理程序是 IE 数据绑定的一部分，在数据源对象中更新数据时，如果发生错误，则触发此事件。

onfilterchange

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

只要对象的可见滤镜切换到新状态或转换完成时(转换可能随着时间而扩展)，就会触发 onfilterchange 事件。只有 IE 中包含滤镜和转换的对象(主要是块元素和表单控件)才能接收这个事件。

onfilterchange 事件的一个常见用法是触发一系列转换操作中的下一个转换，这可能包括无

更多资源请访问稀酷客(www.ckook.com)

限循环的转换，在该循环中，接收事件的对象在两个转换状态间切换。如果不想陷入该循环，就把不同的内容集放到它们自己的定位元素中，并在一个元素中使用 onfilterchange 事件处理程序，来触发另一个元素上的转换。

示例

程序清单 26-39 演示了 onfilterchange 事件处理程序如何在一个转换完成后，触发第二个转换。onload 事件处理程序触发第一个转换。尽管 onfilterchange 事件处理程序能处理 IE4 中大多数与 IE5 相同的对象，但滤镜对象转换属性没有以方便的形式表示出来。程序清单 26-39 中的语法使用 IE5.5+中更现代的 ActiveX 滤镜控件(参见第 38 章)。

程序清单 26-39　使用 onFilterChange 事件处理程序

HTML: jsb26-39.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onfilterchange Event Handler</title>
      <style type="text/css">
         #image1
         {
            position:absolute; top:150px; left:150px;
            filter:progID:DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE',
                motion='in')
         }
         #image2
         {
            position:absolute; top:150px; left:150px;
            filter:progID:DXImageTransform.Microsoft.Iris(irisstyle='CIRCLE',
                motion='out')
         }
         .anImage
         {
            height:90px; width:120px;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-39.js"></script>
   </head>
   <body onload="init()">
      <h1>onfilterchange Event Handler</h1>
      <hr />
      <p>The completion of the first transition ("circle-in") triggers the
         second ("circle-out").
         <button onclick="location.reload()">Play It Again</button>
      </p>
      <div id="image1" style="visibility:visible;"
         onfilterchange="finish()">
```


更多资源请访问稀酷客(www.ckook.com)

```
            <img class="anImage" alt="image" src="desk1.gif" />
        </div>
        <div id="image2" style="visibility:hidden;" >
            <img class="anImage" alt="image" src="desk3.gif" />
        </div>
    </body>
</html>
```

JavaScript: jsb26-39.js

```
function init()
{
   image1.filters[0].apply();
   image2.filters[0].apply();
   start();
}

function start()
{
   image1.style.visibility = "hidden";
   image1.filters[0].play();
}
function finish()
{
   // verify that first transition is done (optional)
   if (image1.filters[0].status == 0)
   {
      image2.style.visibility = "visible";
      image2.filters[0].play();
   }
}
```

相关主题： filter 对象。

onfocus

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在一个对象失去焦点之后，另一个元素接收焦点，然后触发 onfocus 事件。通常在当前对象接收 onfocus 事件前，失去焦点的元素接收 onblur 事件。例如，用户通过键盘在表单中导航，使用 Tab 键移到文本输入元素上时，就会触发 onfocus 事件。单击一个元素也能使该元素得到焦点，就像使浏览器成为客户机桌面上最前端的应用程序一样。

onfocus 事件的可用性随着脚本浏览器的不断更新换代而提高。在早期版本中，onblur 和 onfocus 主要用于面向文本的输入元素(例如 select 元素)。window 对象从 NN3 和 IE4 开始就接收 onfocus 事件处理程序；IE4 也将该事件处理程序应用于更多的表单元素，主要在 Windows 操作系统上，因为该操作系统的用户界面用虚点框表示按钮和链接等项接收到焦点(以允许用户通过键盘的空格键启动它们)。IE5+中的 onfocus 事件处理程序几乎可用于每个 HTML 元素。然而对大多数元素来说，除非给元素标记的 tabindex 特性赋值，否则就不能使用 onblur 和 onfocus。例如，如果在<p>标记内指定 tabindex="1"，可以单击段落，使段落获得焦点(Windows 中使用虚点

更多资源请访问稀酷客(www.ckook.com)

框高亮显示)，或按下 Tab 键，直到该项接收到焦点。

WinIE5.5 还添加了 onfocusin 事件处理程序，它在 onfocus 事件处理程序之前触发。可使用两者中的任何一个，但没必要对同一对象使用这两个事件处理程序，除非希望临时锁定一项，使其不能接收焦点。IE5.5+为了防止对象接收焦点，允许在对象的 onfocusin 事件处理程序中包括 event.returnValue=false 语句。在其他浏览器中，将 onfocus="this.blur()"赋给元素(例如表单控件)作为事件处理程序，也能防止对象接收焦点。然而，这不是阻止用户改变控件设置的可靠方法。不过，目前几乎没有禁用此控件的可靠替代方法。

示例

onfocus 和 onblur 事件处理程序的示例请参见本章前面的程序清单 26-34。

相关主题：onactivate、onblur、ondeactivate、onfocusin、onfocusout 事件处理程序。

onfocusin，onfocusout

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

触发 onfocusin 和 onfocusout 事件，就表示元素准备接收焦点或刚刚失去焦点。这些事件与 onactivate 和 ondeactivate 密切相关，只是在 IE5.5+中，激活和获得焦点可以彼此区分。例如，如果使用 setActive()将元素设置为活动元素，该元素就是活动的，但它没有获得输入焦点。但如果调用 focus()来设置元素的焦点，就会激活元素，并获得输入焦点。

相关主题：onactivate，onblur，ondeactivate，onfocus 事件处理程序。

onhelp

兼容性：WinIE4+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

在 Windows PC 上，只要文档的一个元素具有焦点，且用户按下 F1 键，就会触发 onhelp 事件处理程序。但在 MacIE5 中，该事件只在窗口上触发(换句话说，事件处理程序在<body>标记中指定)，而且必须通过 Mac 键盘上专门的 Help 键触发。在浏览器上选择 Help 菜单选项不会激活这个事件。为防止显示浏览器的 Help 窗口，事件处理程序必须返回 false(IE4+)，或将 event.returnValue 属性设置为 false(IE5+)。因为在 Windows 版本中，事件处理程序可与文档的各个元素相关，所以可创建一个上下文相关的帮助系统。然而，如果焦点在浏览器窗口的 Address 域中，则不能阻止这个事件，此时浏览器的 Help 窗口会显示出来。

示例

程序清单 26-40 是一个上下文相关帮助系统的基本示例，该系统显示的帮助消息适用于不同文本框的不同文本输入。当用户给某个文本框提供焦点时，就会显示一个小图例，提醒用户可通过按 F1 键获得帮助。MacIE5 只提供了一般性帮助。

程序清单 26-40　创建上下文相关帮助

HTML：jsb26-40.html

```
<!DOCTYPE html>
<html>
   <head>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>onhelp Event Handler</title>
        <style type="text/css">
           #legend
           {
              font-size:10px;
           }
        </style>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-40.js"></script>
     </head>
     <body onload="init()" onhelp="return showGenericHelp()">
        <h1>onhelp Event Handler</h1>
        <hr />
        <p id="legend" style="visibility:hidden;"> </p>
        <form>
           Name:
           <input type="text" name="name" size="30"
                  onfocus="showLegend()" onblur="hideLegend()"
                  onhelp="return showNameHelp()" />
           <br />
           Year of Birth:
           <input type="text" name="YOB" size="30"
                    onfocus="showLegend()" onblur="hideLegend()"
                    onhelp="return showYOBHelp()" />
        </form>
     </body>
  </html>
```

JavaScript: jsb26-40.js

```
function showNameHelp()
{
   alert("Enter your first and last names.");
   event.cancelBubble = true;
   return false;
}
function showYOBHelp()
{
   alert("Enter the four-digit year of your birth. For example: 1972");
   event.cancelBubble = true;
   return false;
}
function showGenericHelp()
{
   alert("All fields are required.");
   event.cancelBubble = true;
   return false;
}
function showLegend()
{
```


更多资源请访问稀酷客(www.ckook.com)

```
        document.getElementById("legend").style.visibility = "visible";
    }
    function hideLegend()
    {
        document.getElementById("legend").style.visibility = "hidden";
    }
    function init()
    {
        var msg = "";
            if (navigator.userAgent.indexOf("Mac") != -1)
            {
                msg = "Press \'help\' key for help.";
            }
            else if (navigator.userAgent.indexOf("Win") != -1)
            {
                msg = "Press F1 for help.";
            }
        document.getElementById("legend").style.visibility = "hidden";
        document.getElementById("legend").innerHTML = msg;
    }
```

相关主题：window.showHelp()、window.showModalDialog()方法。

onkeydown，onkeypress，onkeyup

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

按下和释放键盘键时，三个事件会以快速连续的方式触发。首次接触键时，触发 onkeydown 事件；紧跟着触发 onkeypress 事件；释放键时，触发 onkeyup 事件。如果按住一个字符键，使其开始自动重复，则在该字符每次重复时，都会触发 onkeydown 和 onkeypress 事件。

事件顺序在一些键盘事件的处理中非常关键。例如，在用户输入固定数目的字符后(例如，日期、月份和两位数的年)，文本域的焦点应能自动前进到下一个域。触发 onkeyup 事件时，与按键动作相关的字符就添加到域中，此时可以准确地确定域中文本的长度，如下所示：

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Simple entry form</title>
      <script type="text/javascript">
         function jumpNext(fromFld, toFld)
         {
            if (fromFld.value.length == 2)
            {
               document.dateForm.elements[toFld].focus();
               document.dateForm.elements[toFld].select();
            }
         }
      </script>
   </head>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <body>
        <form name="dateForm">
            Month: <input name="month" type="text" size="3" value=""
                          onkeyup="jumpNext(this, 'day')" maxlength="2" />
            Day: <input name ="day" type="text" size="3" value=""
                          onkeyup ="jumpNext(this, 'year')" maxlength="2" />
            Year: <input name="year" type="text" size="3" value=""
                          onkeyup ="jumpNext(this, 'month')" maxlength="2" />
        </form>
    </body>
</html>
```

在支持键盘事件的所有浏览器版本中，并不是典型 PC 键盘上的所有键都触发这三个事件。对于前述兼容性表中的所有浏览器，支持这些事件的唯一键是 ASCII 值表示的字母数字键，包括空格键和 Enter(Mac 上的 Return)，但不包括所有功能键、箭头键和其他导航键。另外，一些修改键也会生成一些事件，例如 Shift、Ctrl (PC)、Alt (PC)、Command(Mac)和 Option (Mac)，具体取决于浏览器和版本。然而，其他键盘事件调用的函数总是可检查这些修改键的按下状态。

警告：
在基于 Mozilla 的浏览器中，onkeydown 事件处理程序只能用于 Mozilla1.4+(和 Netscape 7.1+)。

脚本键盘事件经常用来检查按下了哪个键，以便在该键上执行一些处理或验证操作。如果要编写跨浏览器的程序，事情就变得非常复杂。在一些情况下，即使仅为 Internet Explorer 编写程序就十分棘手，因为非数字字母键只生成 onkeydown 和 onkeyup 事件。

实际上，为完全理解键盘事件，需要区分键编码和字符编码。每个 PC 键盘键都具有与其相关的键编码，这个键编码总是相同的，不管是否同时按下其他键。然而，只有数字字母键(字母、数字和空格键等)会创建字符编码，这个编码表示该键创建的输入字符。如果按下修改键，这个编码就可能会改变。例如，如果按下 A 键，就会生成小写 a 字母(字符代码 97)；如果按住 Shift 键的同时按下 A 键，就会生成大写 A 字母(字符代码 65)。但无论如何，这个键的键代码(在西语键盘上是 65)总是不变。

现在看看如何在脚本中使用这些编码。在所有情况下，编码信息都传递为浏览器 event 对象的一个或两个属性。IE 的 event 对象只有一个这样的属性：keyCode，它包含 onkeydown 和 onkeyup 事件的键代码，不包含 onkeypress 事件的字符编码；另一方面，NN6+/Moz 的 event 对象包含两个属性 charCode 和 keyCode。这些 event 对象属性的细节和例子参见第 32 章。

对于非数字字母键(如功能键、箭头和页面导航键等)的事件，至少应考虑使用 onkeydown 或 onkeyup 事件处理程序。而要处理文本框中的字符，请使用 onkeypress 事件处理程序。可在程序清单 26-41 和第 32 章的例子中试验这些事件和代码。

6. 常用的键盘事件任务

WinIE4+允许修改用户在编辑文本框时输入的字符。Onkeypress 事件处理程序可修改 event.keyCode 属性，并允许继续触发该事件(换句话说，不返回 false 或将 event.returnValue 属性设置为 false)。下面的 IE 函数由 onkeypress 事件处理程序调用，它确保即使在文本域中输入

更多资源请访问稀酷客(www.ckook.com)

小写的文本，显示出来的文本也是全大写的，：

```
function assureUpper()
{
  if (event.keyCode >= 97 && event.keyCode <= 122)
  {
     event.keyCode = event.keyCode - 32;
  }
}
```

这样做会迷惑用户或使用户感到不快，因此要仔细考虑。

为了阻止按下的键盘键成为文本域中的一个输入字符，可使用 onkeypress 事件处理程序阻止事件的默认动作。例如，下列 HTML 页面显示了如何检查文本域的输入是否只包含数字：

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Keyboard Capture</title>
      <script type="text/javascript">
      function checkIt(evt)
      {
         var charCode = (evt.charCode) ?
               evt.charCode : ((evt.which) ? evt.which : evt.keyCode);
            if (charCode > 31 && (charCode < 48 || charCode > 57))
            {
                alert("Please make sure entries are numbers only.");
                return false;
            }
         return true;
      }
   </script>
</head>
<body>
   <form>
      Enter any positive integer:
      <input type="text" name="numeric" onkeypress="return checkIt(event)">
   </form>
</body>
</html>
```

只要用户输入非数字字符，就会收到一个警告消息，该字符也不会添加到文本框的文本中。

用户在文本框中按下 Enter(Mac 上的 Return)键时，键盘事件也允许通过脚本提交表单。Enter/Return 键的 ASCII 值为 13，因此，可检查在文本框中按下的每个键，只要其值为 13，就提交表单，如以下的函数所示：

```
function checkForEnter(evt)
{
   evt = (evt) ? evt : event;
   var charCode = (evt.charCode) ?
```


更多资源请访问稀酷客(www.ckook.com)

```
            evt.charCode : ((evt.which) ? evt.which : evt.keyCode);

        if (charCode == 13)
            {
            document.forms[0].submit();
            return false;
            }
        return true;
    }
```

将 checkForEnter()函数赋给每个域的 onkeypress 事件处理程序，就会给一般的 HTML 表单添加一些额外功能。

还可在 HTML 页面上截取 Ctrl+键(只能是字母)组合，这在 Internet Explorer 中非常有效，但只有浏览器本身不使用该组合键，这才能成功。换句话说，不能重定向浏览器用于自身控件的 Ctrl+键组合。Ctrl+ A～Z 字母键组合的 onkeypress 键码值是 1～26(但用于浏览器的 Ctrl+键组合不会触发键盘事件)。

示例

程序清单 26-41 是一个有用的练习，可用于更好地理解键盘事件代码和修改键在 IE5+和 W3C 浏览器中的工作方式。在使用该程序清单时，观察页面比观察程序清单的实际代码更重要。对于按下的每个键或键组合，页面都会显示 onkeydown、onkeypress 和 onkeyup 事件的 keyCode 值。如果在按键的同时按下一个或多个修改键，修改键名称就会在这三个事件中突出显示。注意，在基于 NN6+/Moz/WebKit 的浏览器中运行时，keyCode 值不是字符代码。在旧浏览器中，可能需要单击页面，document 对象才能识别键盘事件。

为观察键盘事件发生时的情况，最佳方式是按住一个键，查看 onkeydown 和 onkeypress 事件的键码，然后释放按键，查看 onkeyup 事件的代码。注意，如果按下 A 键的同时没有按下任何修改键，onkeydown 事件的键码就是 65(A)，但 IE 中的 onkeypress 键码(以及基于 NN6+/Moz/WebKit 的浏览器中的 charCode 属性)是 97(a)。如果按下 A 键的同时按下了 Shift 键，这三个事件都会生成键码 65(A)(Shift 修改键标签也会突出显示)。释放 Shift 键，onkeyup 事件就会显示 Shift 键的键码。

在另一个试验中，按下 4 个箭头键中的一个，则不会把键码传递给 onkeypress 事件，因为这些键不触发此事件，但会触发 onkeydown 和 onkeyup 事件。

程序清单 26-41　键盘事件处理程序练习

HTML: jsb26-41.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Keyboard Event Handler Lab</title>
      <style type="text/css">
         td
         {
            text-align:center;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-41.js"></script>
   </head>
   <body onload="init()">
      <h1>Keyboard Event Handler Lab</h1>
      <hr />
      <form>
         <table border="2" cellpadding="2">
            <tr>
               <th></th>
               <th>onKeyDown</th>
               <th>onKeyPress</th>
               <th>onKeyUp</th>
            </tr>
            <tr>
               <th>Key Codes</th>
               <td id="downKeyCode">0</td>
               <td id="pressKeyCode">0</td>
               <td id="upKeyCode">0</td>
            </tr>
            <tr>
               <th>Char Codes (IE5/Mac; Moz; WebKit)</th>
               <td id="downCharCode">0</td>
               <td id="pressCharCode">0</td>
               <td id="upCharCode">0</td>
            </tr>
            <tr>
               <th rowspan="3">Modifier Keys</th>
               <td><span id="shiftDown">Shift</span></td>
               <td><span id="shift">Shift</span></td>
               <td><span id="shiftUp">Shift</span></td>
            </tr>
            <tr>
               <td><span id="ctrlDown">Ctrl</span></td>
               <td><span id="ctrl">Ctrl</span></td>
               <td><span id="ctrlUp">Ctrl</span></td>
            </tr>
            <tr>
               <td><span id="altDown">Alt</span></td>
               <td><span id="alt">Alt</span></td>
               <td><span id="altUp">Alt</span></td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

JavaScript: jsb26-41.js

```
function init()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
    document.onkeydown = showKeyDown;
    document.onkeyup = showKeyUp;
    document.onkeypress = showKeyPress;
}

function showKeyDown(evt)
{
    evt = (evt) ? evt : window.event;
    document.getElementById("pressKeyCode").innerHTML = 0;
    document.getElementById("upKeyCode").innerHTML = 0;
    document.getElementById("pressCharCode").innerHTML = 0;
    document.getElementById("upCharCode").innerHTML = 0;
    restoreModifiers("");
    restoreModifiers("Down");
    restoreModifiers("Up");
    document.getElementById("downKeyCode").innerHTML = evt.keyCode;
        if (evt.charCode)
        {
            document.getElementById("downCharCode").innerHTML = evt.charCode;
        }
    showModifiers("Down", evt);
}

function showKeyUp(evt)
{
    evt = (evt) ? evt : window.event;
    document.getElementById("upKeyCode").innerHTML = evt.keyCode;
        if (evt.charCode)
        {
            document.getElementById("upCharCode").innerHTML = evt.charCode;
        }
    showModifiers("Up", evt);
    return false;
}

function showKeyPress(evt)
{
   evt = (evt) ? evt : window.event;
   document.getElementById("pressKeyCode").innerHTML = evt.keyCode;
      if (evt.charCode)
      {
          document.getElementById("pressCharCode").innerHTML = evt.charCode;
      }
   showModifiers("", evt);
   return false;
}

function showModifiers(ext, evt)
{
```


更多资源请访问稀酷客(www.ckook.com)

```
        restoreModifiers(ext);
        if (evt.shiftKey)
        {
            document.getElementById("shift" + ext).style.backgroundColor = "#ff0000";
        }
        if (evt.ctrlKey)
        {
            document.getElementById("ctrl" + ext).style.backgroundColor = "#00ff00";
        }
        if (evt.altKey)
        {
            document.getElementById("alt" + ext).style.backgroundColor = "#0000ff";
        }
    }

    function restoreModifiers(ext)
    {
        document.getElementById("shift" + ext).style.backgroundColor = "#ffffff";
        document.getElementById("ctrl" + ext).style.backgroundColor = "#ffffff";
        document.getElementById("alt" + ext).style.backgroundColor = "#ffffff";
    }
```

花上一些时间完成此练习，尝试所有的键和键组合，以理解事件和键码的工作方式。

相关主题：String.fromCharCode()方法。

onlayoutcomplete

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在当前布局矩形(LayoutRect 对象)上完成打印或打印预览布局操作时，将触发 onlayout complete 事件处理程序。该事件主要用于在打印过程中将内容从一个页面溢出到另一个页面。为响应 onlayoutcomplete 事件，可检查 contentOverflow 属性，以确定页面内容是否溢出当前布局矩形。

onlosecapture

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

如果对象不再拥有事件捕获功能，但它启用了该捕获功能，就会触发 onlosecapture 事件处理程序。用户执行下列动作时，会自动解除事件捕获功能：

- 其他窗口拥有焦点。
- 显示系统模式对话框(例如警告窗口)。
- 滚动页面。
- 打开一个浏览器上下文菜单(右击)。
- 使用 Tab 使浏览器窗口中的 Address 域拥有焦点。

与 onlosecapture 事件处理程序相关的函数应该执行环境清理工作，因为对象不再捕获鼠标事件。

更多资源请访问稀酷客(www.ckook.com)

示例

本章前面的程序清单 26-30 说明了如何在事件捕获场景中使用 onlosecapture 来显示上下文菜单。当用户执行使菜单失去鼠标捕获功能的动作时，onlosecapture 事件处理程序将隐藏上下文菜单。

相关主题：releaseCapture()、setCapture()方法。

onmousedown，onmouseup

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

用户按下任何鼠标按钮，都会触发 onmousedown 事件处理程序；用户释放鼠标按钮时，会触发 onmouseup 事件处理程序(假定接收该事件的对象也接收 onmousedown 事件)。用户在对象上执行典型的鼠标单击操作时，鼠标事件的触发顺序如下：onmousedown、onmouseup、onclick。但如果用户在对象上按下鼠标，然后把光标从对象上移开，则只触发 onmousedown 事件。

这些事件允许制作者和设计者在用作动作按钮或图标按钮的图像上添加更多的应用程序操作。大多数按钮的工作方式是：按下鼠标按钮时，按钮的外观会改变；释放鼠标按钮(或把光标拖出按钮之外)时，按钮恢复其原始样式。这些事件能模仿这种操作。

鼠标按钮动作创建的每个 event 对象都有一个属性，来表示用户按下了哪个鼠标按钮。NN4 的事件模型将这个属性称为 which，而 IE4+和基于 NN6+/Moz/WebKit 的浏览器称它为 button 属性(但是，对于按钮，该属性有不同的值)。在 onmousedown 或 onmouseup 事件(而不是 onclick 事件)上测试鼠标按钮最可靠，因为 onclick 事件对象不总是包含按钮信息。

示例

为响应鼠标滚动到图像上、在其上按下鼠标、释放鼠标按钮、从图像上移开鼠标，而更改按钮图像，程序清单 26-42 提供了一对小导航按钮(左箭头按钮和右箭头按钮)。在页面载入时，这些图像预先载入到浏览器缓存中，以便在用户首次调用图像的新版本时能即时响应。

程序清单 26-42 使用 onmousedown 和 onmouseup 事件处理程序

HTML：jsb26-42.html

```
<!DOCTYPE html>
<html>
  <head>
     <meta http-equiv="content-type" content="text/html;charset=utf-8">
     <title>onmousedown and onmouseup Event Handlers</title>
     <style type="text/css">
        #imgHolder
        {
           text-align:center;
        }
        .images
        {
           height:16px; width:16px; border:0px;
        }
     </style>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-42.js"></script>
    </head>
    <body>
        <h1>onmousedown and onmouseup Event Handlers</h1>
        <hr />
        <p>Roll atop and click on the buttons to see how the link event handlers
            swap the subtlety different images:
        </p>
        <div id="imgHolder">
            <img alt="image" class="images"
                 name="Left" src="LeftNorm.gif"
                 onmouseover="return setImage('Left', 'Up')"
                 onmousedown="return setImage('Left', 'Down')"
                 onmouseup="return setImage('Left', 'Up')"
                 onmouseout="return setImage('Left', 'Norm')" />
              
            <img alt="image" class="images"
                 name="Right" src="RightNorm.gif"
                 onmouseover="return setImage('Right', 'Up')"
                 onmousedown="return setImage('Right', 'Down')"
                 onmouseup="return setImage('Right', 'Up')"
                 onmouseout="return setImage('Right', 'Norm')" />
        </div>
    </body>
</html>
```

JavaScript: jsb26-42.js

```
var RightNormImg = new Image(16,16);
var RightUpImg = new Image(16,16);
var RightDownImg = new Image(16,16);
var LeftNormImg = new Image(16,16);
var LeftUpImg = new Image(16,16);
var LeftDownImg = new Image(16,16);

RightNormImg.src = "RightNorm.gif";
RightUpImg.src = "RightUp.gif";
RightDownImg.src = "RightDown.gif";
LeftNormImg.src = "LeftNorm.gif";
LeftUpImg.src = "LeftUp.gif";
LeftDownImg.src = "LeftDown.gif";
function setImage(imgName, type)
{
   var imgFile = eval(imgName + type + "Img.src");
   document.images[imgName].src = imgFile;
   return false;
}
```

相关主题：onclick 事件处理程序。

更多资源请访问稀酷客(www.ckook.com)

onmouseenter，onmouseleave

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

WinIE5.5 引入了 onmouseenter 和 onmouseleave 事件处理程序。这两个事件处理程序的操作与 onmouseover 和 onmouseout 事件处理程序一样。Microsoft 只提供了新术语。新、旧事件都可在 IE5.5+ 中触发。对于把光标移动到对象上和从对象上移出光标的动作，旧事件在新事件之前触发。如果只给 IE5.5+编写脚本，则应该使用新术语；否则使用旧版本。

示例

用 IE5.5 语法修改程序清单 26-43，用 onmouseenter 替代 onmouseover，用 onmouseleave 替代 onmouseout，效果是相同的。

相关主题：onmouseover、onmouseout 事件处理程序。

onmousemove

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

只要光标在当前对象上，则移动鼠标，即使只移动一个像素，也会触发 onmousemove 事件处理程序。虽然该事件通常用在元素拖动操作中，也不必按下鼠标按钮来触发该事件(在不能使用 ondrag 事件处理程序的 NN/FF3.4-/Opera 中尤其如此)。

即使该事件的间隔粒度是像素级，也不应该使用事件触发次数作为测量手段。因为事件依赖于光标移动速度和客户计算机的性能，所以它不可能在每个像素位置上触发。

在 IE4+和 W3C DOM 兼容的浏览器中，可将 onmousemove 事件处理程序赋给任何元素(虽然只能拖动定位元素)。用户在所设计的页面上拖动多个项时，通常将 onmousemove 事件处理程序赋给 document 对象，让所有这些事件向上冒泡到 document 对象处进行处理。

示例

有关使用鼠标事件控制元素在页面上拖动的例子，参见配书光盘中的第 43 章和第 59 章。

相关主题：ondrag、onmousedown 和 onmouseup 事件处理程序。

onmouseout，onmouseover

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

只要光标滚动到屏幕对象的矩形空间中，就会触发 onmouseover 事件。把光标移出对象的矩形时，会触发 onmouseout 事件处理程序。这些事件常在窗口的状态栏中显示对象的解释文本，并影响图像的变换(所谓的鼠标滚动)。使用 onmouseover 事件处理程序可将状态改为高亮状态；而使用 onmouseout 事件处理程序可恢复图像，或使状态栏返回它的常规状态。

虽然这两个事件从一开始时就在脚本浏览器的对象模型中，但它们不可用于早期浏览器的大多数对象。IE4+和 W3C DOM 兼容浏览器为占据屏幕空间的每个元素提供了这些事件。IE5.5+ 包括两个额外的事件处理程序：onmouseenter 和 onmouseleave，它们与 onmouseover 和 onmouseout 事件相同，但使用了不同术语。旧事件处理程序在新版本之前触发。

更多资源请访问稀酷客(www.ckook.com)

注意：

如果事件与靠近框架或窗口边缘的元素有关，并且用户将鼠标指针快速移出了当前框架，通常不会触发 onmouseout 事件处理程序。

示例

程序清单 26-43 使用带 4 个链接的文本来演示如何使用 onmouseover 和 onmouseout 事件处理程序。注意，对于每个链接，事件处理程序都运行一个设置窗口状态消息的通用函数。如果该函数的返回值为 true，事件处理程序就会生成设置状态栏所需的 return true 语句。在一个状态消息中，提供了一个放在圆括号中的 URL，让用户评价此类状态消息对访问者有多大帮助。

程序清单 26-43 使用 onmouseover 和 onmouseout 事件处理程序

HTML: jsb26-43.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onmouseover and onmouseout Event Handlers</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-43.js"></script>
   </head>
   <body>
      <h1>onmouseover and onmouseout Event Handlers</h1>
      <hr />
      <h1>Pledge of Allegiance</h1>
      <hr />
      I pledge
      <a href="javascript:emulate()"
               onmouseover="return setStatus('View dictionary definition')"
               onmouseout="return setStatus('')">allegiance</a>
      to the
      <a href="javascript:emulate()"
         onmouseover="return setStatus('Learn about the U.S. flag↵
              (http://lcweb.loc.gov) ')"
         onmouseout="return setStatus('')">flag</a>
      of the
      <a href="javascript:emulate()"
         onmouseover="return setStatus('View info about the U.S. government')"
         onmouseout="return setStatus('')">United States of America</a>,
      and to the Republic for which it stands, one nation
      <a href="javascript:emulate()"
         onmouseover="return setStatus('Read about the history of this phrase↵
             in the Pledge')"
         onmouseout="return setStatus('')">under God</a>,
      indivisible, with liberty and justice for all.
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

JavaScript: jsb26-43.js

```
function setStatus(msg)
{
   window.status = msg;
   return true;
}

// destination of all link HREFs
function emulate()
{
   alert("Not going there in this demo.");
}
```

相关主题：onmouseenter、onmouseleave、onmousemove 事件处理程序。

onmousewheel

兼容性：WinIE6+，MacIE-，NN-，Moz+，Safari+，Opera+，Chrome+

用户滚动鼠标的滚轮时，就会触发 onmousewheel 事件处理程序。在响应 onmousewheel 事件时，可检查 wheelDelta 属性，来确定鼠标滚轮滚动了多远。wheelDelta 属性以 120 的倍数来表示鼠标滚轮滚动的距离，正值表示从用户一方朝外滚动，负值表示向用户方向滚动。W3C 的对应事件是 DOMMouseScroll。

相关主题：onmousemove 事件处理程序。

onmove

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

切勿混淆 onmove 与 onmousemove，onmove 事件与鼠标无关。只要移动可定位元素，就触发 onmove 事件。例如，如果 div 元素创建为一个绝对定位的可移动元素，就可以通过响应 onmove 事件来跟踪其移动。在此事件处理程序中，可使用 offsetLeft 和 offsetTop 属性来确定元素的准确位置。

相关主题：onmoveend、onmovestart 事件处理程序。

onmoveend，onmovestart

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

可定位元素在页面上移动时，将会触发 onmovestart 和 onmoveend 事件处理程序。更确切地说，元素开始移动时，会触发 onmovestart 事件；而元素停止移动时，会触发 onmoveend 事件。在 onmovestart 和 onmoveend 事件触发之间，可能会触发多个 omove 事件，以表示元素在移动。

相关主题：onmove 事件处理程序。

onpaste

兼容性：WinIE5+，MacIE-，NN-，Moz+，Safari 1.3+，Opera-，Chrome+

更多资源请访问稀酷客(www.ckook.com)

用户或脚本启动当前对象的粘贴编辑动作后，会立即触发 onpaste 事件。该事件在 onbeforepaste 事件之前触发，onbeforepaste 事件在显示编辑菜单或上下文菜单前触发(或用键盘快捷键启动的粘贴动作之前)。

可以使用该事件处理程序给通常不允许粘贴的元素提供编辑功能。在这种情况下，需要将 onbeforepaste 事件处理程序的 event.returnValue 设置为 false，以允许使用上下文菜单或 Edit 菜单中的 Paste 菜单项。然后 onpaste 事件处理程序必须(通过 clipboardData 对象的 getData()方法)手动从剪贴板中获得数据，并将数据插入当前对象。

编程人员控制着剪贴板中存储的数据，所以剪贴板可以不只包含数据的一个直接副本。例如，剪贴板可存放 image 对象的 src 属性值，以便用户把它粘贴到页面上的其他地方。

示例

程序清单 26-44 演示了如何使用 onbeforepaste 和 onpaste event 事件处理程序(与 onbeforecopy 和 oncopy 结合使用)，让脚本在用户的复制-粘贴操作中控制数据传输过程。该示例要把一个表格包含的词(一列名词，一列形容词)复制并粘贴到段落的空白位置。把 onbeforecopy 和 oncopy 事件处理程序赋予 table 元素，因为来自 td 元素的事件会冒泡到 table 容器，这样要处理的 HTML 代码较少。

在段落中，两个 span 元素包含带下划线的空白。要将文本粘贴到这些空白位置，用户必须先选择至少一个字符(参考程序清单 26-37，它提供了此应用程序的拖放版本)。段落中的 onbeforepaste 事件处理程序(它捕获从两个 span 向上冒泡的事件)将 event.returnValue 属性设置为 false，从而允许“粘贴”选项显示在上下文和“编辑”菜单中(在 HTML body 内容中一般不显示该选项)。

在粘贴时，目标 span 的 innerHTML 属性设置为存储在剪贴板中的文本数据。这里，event.returnValue 属性也设置为 false，以防止正常的系统粘贴操作被脚本控制的粘贴操作干扰。

程序清单 26-44　使用 onbeforepaste 和 onpaste 事件处理程序

HTML: jsb26-44.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onbeforepaste and onpaste Event Handlers</title>
      <style type="text/css">
         td
         {
            text-align:center;
         }
         th
         {
            text-decoration:underline;
         }
         .blanks
         {
            text-decoration:underline;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
        </style>
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript" src="jsb26-44.js"></script>
    </head>
    <body>
        <h1>onbeforepaste and onpaste Event Handlers</h1>
        <h6>Be very precise when selecting characters</h6>
        <hr />
        <p>Your goal is to copy and paste one noun and one adjective from the
            following table into the blanks of the sentence. Select a word from
            the table and use a menu to copy it to the clipboard. Select one or
            more spaces of the blanks in the sentence and choose Paste from a
            menu to replace the blank with the clipboard contents.
        </p>
        <table cellpadding="5" onbeforecopy="selectWhole()"
               oncopy="handleCopy()">
            <tr>
                <th>Nouns</th>
                <th>Adjectives</th>
            </tr>
            <tr>
                <td>truck</td>
                <td>round</td>
            </tr>
            <tr>
                <td>doll</td>
                <td>red</td>
            </tr>
            <tr>
                <td>ball</td>
                <td>pretty</td>
            </tr>
        </table>
        <p id="myP" onbeforepaste="handleBeforePaste()" onpaste="handlePaste()">
            Pat said, "Oh my, the <span id="blank1" class="blanks">
                 </span>
            is so <span id="blank2" class="blanks">
                 </span> !"
        </p>
        <button onclick="location.reload()">Reset</button>
    </body>
</html>
```

JavaScript: jsb26-44.js

```
function selectWhole()
{
    var obj = window.event.srcElement;
    var range = document.body.createTextRange();
    range.moveToElementText(obj);
    range.select();
```


更多资源请访问稀酷客(www.ckook.com)

```
      event.returnValue = false;
   }
   function handleCopy()
   {
      var rng = document.selection.createRange();
      clipboardData.setData("Text",rng.text);
      event.returnValue = false;
   }

   function handlePaste()
   {
      var elem = window.event.srcElement;
          if (elem.className == "blanks")
          {
             elem.innerHTML = clipboardData.getData("Text");
          }
     event.returnValue = false;
   }
   function handleBeforePaste()
   {
      var elem = window.event.srcElement;
          if (elem.className == "blanks")
          {
             event.returnValue = false;
          }
   }
```

相关主题：oncopy、oncut、onbeforepaste 事件处理程序。

onpropertychange

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

只要脚本修改对象的属性，就在 WinIE5+中触发 onpropertychange 事件，这包括修改对象的样式属性。另外，使用 setAttribute()方法改变属性也可触发该事件。

脚本可检查属性的值是否改变，因为 event.propertyName 属性包含被改变属性的名称(字符串)。在改变对象的 style 对象时，event.propertyName 值以 style.开头，例如 style.backgroundcolor。

改变对象的属性后，可使用该事件处理程序本地化该对象专属的后期处理。一般不是将后期处理语句放在执行改变操作的函数内，而可以使该函数通用化(以便修改多个对象的属性)。

示例

程序清单 26-45 显示了如何采用编程方式响应对象的属性变化。此程序清单生成的页面包含 4 个单选按钮，来改变段落的 innerHTML 和 style.color 属性。段落的 onpropertychange 事件处理程序调用了 showChange()函数，该函数提取事件的相关信息，并在窗口的状态栏中显示数据。注意，在修改样式表属性时，属性名包括 style.。

程序清单 26-45　使用 onPropertyChange 属性

HTML: jsb26-45.html

更多资源请访问稀酷客(www.ckook.com)

```
<!DOCTYPE html>
<html>
   <head>
     <meta http-equiv="content-type" content="text/html;charset=utf-8">
     <title>onpropertychange Event Handler</title>
     <script type="text/javascript" src="../jsb-global.js"></script>
     <script type="text/javascript" src="jsb26-45.js"></script>
   </head>
   <body onload="init()";>
     <h1>onpropertychange Event Handler</h1>
     <hr />
     <p id="myP" onpropertychange="showChange()">This is a sample paragraph.</p>
     <form>
        Text:
        <input type="radio" name="btn1" checked="checked"
               onclick="normalText()" />Normal
        <input type="radio" name="btn1"
               onclick="shortText()" />Short
        <br />
        Color:
        <input type="radio" name="btn2" checked="checked"
               onclick="normalColor()" />Black
        <input type="radio" name="btn2"
               onclick="hotColor()" />Red
     </form>
   </body>
</html>
```

JavaScript: jsb26-45.js

```
function init()
{
   elem = document.getElementById("myP");
}
function normalText()
{
   if (elem.textContent)
   {
      elem.textContent = "This is a sample paragraph.";
   }
   else
   {
      elem.innerText = "This is a sample paragraph.";
   }
}
function shortText()
{
   if (elem.textContent)
   {
      elem.textContent = "Short stuff.";
   }
```


更多资源请访问稀酷客(www.ckook.com)

```
        else
        {
            elem.innerText = "Short stuff.";
        }
    }
    function normalColor()
    {
        elem.style.color = "black";
    }
    function hotColor()
    {
        elem.style.color = "red";
    }
    function showChange()
    {
        var objID = event.srcElement.id;
        var propName = event.propertyName;
        var newValue = eval(objID + "." + propName);
        var msg = "The " + propName + " property of the " + objID;
        msg += " object has changed to \"" + newValue + "\".";
        alert(msg);
    }
```

相关主题：style 属性；setAttribute()方法。

onreadystatechange

兼容性：WinIE4+，MacIE-，NN7+，Moz1.0.1+，Safari 1.2+，Opera+，Chrome+

只要改变对象的准备状态，就会触发 onreadystatechange 事件处理程序，本章前面讨论 readyState 属性时，列出了这些状态的细节(注意 IE4 的局限性)。状态的改变不能保证对象已准备好，允许脚本语句访问它的属性，所以常常需要在 onreadystatechange 事件处理程序调用的脚本中，检查对象的 readyState 属性。

该事件由能载入数据的对象触发：applet、document、frame、frameset、iframe、img、link、object、script 和 XML 对象；它不能由其他类型的对象触发，除非该对象关联了 Microsoft DHTML 操作。onreadystatechange 事件不冒泡，也无法取消它。

相关主题：readyState 属性。

onresize

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

只要对象重置了大小，以响应各种用户动作或脚本动作，就会触发 onresize 事件处理程序。多数元素都包括这个事件处理程序，这里假定对象给表示大小的样式特性赋了值(例如高度、宽度或位置)。

onresize 事件不冒泡。重置浏览器窗口或框架的大小，不会触发窗口的 onload 事件处理程序。

示例

如果要捕获用户调整浏览器窗口(或框架)大小的操作，可为 onresize 事件处理程序分配一个

更多资源请访问稀酷客(www.ckook.com)

函数，脚本如下：

```
window.onresize = handleResize;
```

也可以使用 body 元素的一个 HTML 特性来指定该函数：

```
<body onresize="handleResize()">
```

相关主题：window.resize()方法。

onresizeend，onresizestart

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

onresizeend 与 onresizestart 事件处理程序仅用于 Windows 编辑模式下可调整大小的对象。

相关主题：oncontrolselect 事件处理程序。

onrowenter，onrowexit

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在 IE 数据绑定中，记录集数据的当前行改变时，将会触发 onrowenter 和 onrowexit 事件。更确切地说，当前行的数据已改变，新数据在数据源对象上可用时，就触发 onrowenter 事件。当前行正在改变时，会触发 onrowexit 事件，这意味着用户正在选择另一行数据；在该行改变之前触发 onrowexit。

相关主题：onrowsdelete，onrowsinserted 事件处理程序。

onrowsdelete，onrowsinserted

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

一行或多行数据要从 IE 数据绑定的记录集中删除时，将触发 onrowsdelete 事件。相反，一行或多行数据已插入记录集时，触发 onrowsinserted 事件。

相关主题：onrowenter、onrowexit 事件处理程序。

onscroll

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera-，Chrome+

只要重新定位元素滚动条中的滚动块，就触发 onscroll 事件处理程序。更简单地说，用户用鼠标单击并拖动滚动块时，触发 onscroll 事件。不过，这不是触发该事件的唯一操作。用户单击滚动箭头，单击滚动条，或按下 Home、End、Space、Page Up 或 Page Down 中的任何键时，也会触发 onscroll 事件。调用 doScroll()方法也触发此事件，就像用户按住向上箭头或向下箭头键一样。

相关主题：doScroll()方法。

onselectstart

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari 1.3+，Opera-，Chrome+

用户开始在页面上选择内容时，会触发 onselectstart 事件处理程序。所选内容可以是内联文本、图像或可编辑文本框中的文本。如果用户选择了多个对象，就会在所选的第一个对象上触发该事件。

更多资源请访问稀酷客(www.ckook.com)

示例

使用程序清单 26-46 生成的页面，看一下用户选择页面上的多个元素时，onselectstart 事件处理程序的工作方式。用户开始在页面的任何位置上执行选择操作时，状态栏就会显示接收该事件的对象的 ID。注意，只有实际做出了选择，才会触发该事件。当鼠标指针下没有其他元素时，body 元素就会触发事件。

程序清单 26-46　使用 onselectstart 事件处理程序

HTML: jsb26-46.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onselectstart Event Handler</title>
      <style type="text/css">
         h1, th
         {
           padding:5px;
         }
         td
         {
           text-align:center;
         }
      </style>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb26-46.js"></script>
   </head>
   <body id="myBody" onselectstart="showObj()">
      <h1 id="myH1">
         onselectstart Event Handler
      </h1>
      <hr id="myHR" />
      <p id="myP">This is a sample paragraph.</p>
      <table border="1">
         <tr id="row1">
            <th id="header1">Column A</th>
            <th id="header2">Column B</th>
            <th id="header3">Column C</th>
         </tr>
         <tr id="row2">
            <td id="cellA2">text</td>
            <td id="cellB2">text</td>
            <td id="cellC2">text</td>
         </tr>
         <tr id="row3">
            <td id="cellA3">text</td>
            <td id="cellB3">text</td>
            <td id="cellC3">text</td>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb26-46.js

```
function showObj()
{
   var objID = event.srcElement.id;
   alert("Selection started with object: " + objID);
}
```

相关主题： 各种对象的 onselect 事件处理程序。

更多资源请访问稀酷客(www.ckook.com)

第27章

window 对象和 frame 对象

在第 25 章的基本文档对象模型图(见图 25-1)中，window 对象是所有文档相关对象最外层的全局容器，这些文档对象可以用 JavaScript 编写。所有 HTML 和 JavaScript 活动都在一个窗口中进行，该窗口可以是标准的 Windows、Mac 或者 XWindows 应用程序窗口，具有滚动条、工具栏和其他窗框控件，也可以生成只包含某些典型窗框控件的窗口。框架也是一个窗口，但框架除了滚动条以外没有其他窗框控件。JavaScript 总是从 window 对象开始引用对象。现代浏览器将框架集处理为一个特殊类型的 window 对象，因此本章也会介绍框架集。

本章包含哪些内容？
编写在多个框架之间进行通信的脚本
创建和管理新窗口
控制浏览器窗口的大小、位置和外观
有关 window、frame、frameset 和 iframe 对象的详细信息

在与浏览器脚本编程相关的所有对象中，window 对象以及与 window 相关的对象是与对象关系最为密切的术语，这需要占用较长的篇幅来讨论。

27.1 window 对象术语

第一次学习文档对象模型时，常常觉得 window 对象令自己感到困惑。window 对象有大量的同义词：top、self、parent 和 frame。更糟的是，这些术语也是 window 对象的属性。在有些情况下，窗口是它自己的父窗口，但如果定义包含两个框架的框架集，3 个 window 对象中就只有一个父对象，这个问题令人头疼不已。

如果不在 Web 应用程序中使用框架，这些头疼的问题就不会浮出水面。但是当框架是设计规划的一部分时，就应该了解框架如何影响对象模型。

更多资源请访问稀酷客(www.ckook.com)

27.2　框架

如何应用框架是 Web 设计人员争论已久的问题，一些人支持它，一些人反对它。经过慎重考虑，我们相信，有时使用框架是有道理的。从历史观点上说，包含层次结构或依赖文件系统的大文档就需要使用框架。例如，文档多屏显示时，常需要包含一个随时可见的静态导航控件集。在另一个框架里放置这些控件(它们可能是链接或者图像地图)，就可以即时访问这些控件，而不管主文档的滚动条件是什么。另外，还可以使用 Ajax 框架提供这些导航控件，来替代 HTML 框架。注意在框架中创建书签和打印文档虽然是可能的，但非常困难。框架对于残障人士而言是一个问题。大多数 Web 设计人员都通过链接为这类情形提供了替代的普通 HTML 文档。

27.2.1　创建框架

无论是否使用 JavaScript，在文档中定义框架的工作都是相同的，最简单的框架集文档包含用于建立框架集的标记，如下所示：

```
<html>
    <head>
      <title>My Frameset</title>
    </head>
<frameset>
    <frame name="Frame1" src="document1.html">
    <frame name="Frame2" src="document2.html">
</frameset>
</html>
```

前面的 HTML 文档(用户永远看不到这个文档)为整个浏览器窗口定义了框架集，每个框架必须有文档的一个 URL 引用(通过 src 特性指定)，以使文档载入框架。用 name 特性为每个框架指定一个名称，可以极大地简化框架内容的脚本编程。

27.2.2　框架对象模型

在任何给定时刻，浏览器内存中的对象模型由当前加载文档中的 HTML 标记决定，理解这一点是成功创建框架脚本的关键所在。本书中所有封装的对象模型图都不能准确反映文档或文档集中的对象模型，如图 27-1 所示。

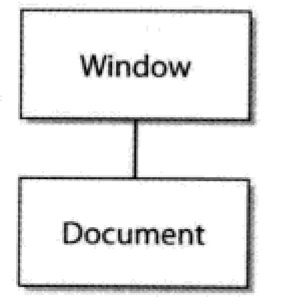


图 27-1　最简单的 window-document 关系

对于单个无框架文档，对象模型最初都只有一个 window 对象，该对象包含一个文档，如图 27-1 所示。在这个简单结构中，window 对象是任何已加载对象引用的起点。因为窗口总是存在(只

更多资源请访问稀酷客(www.ckook.com)

有存在窗口，才能载入文档)，所以文档中的任何对象引用都可以忽略当前窗口的引用。

在包含两个框架的简单框架集模型中(见图 27-2)，浏览器把初始框架集文档的容器当作父窗口，文档存在的唯一可见证据是框架集文档的标题显示在浏览器窗口的标题栏里。

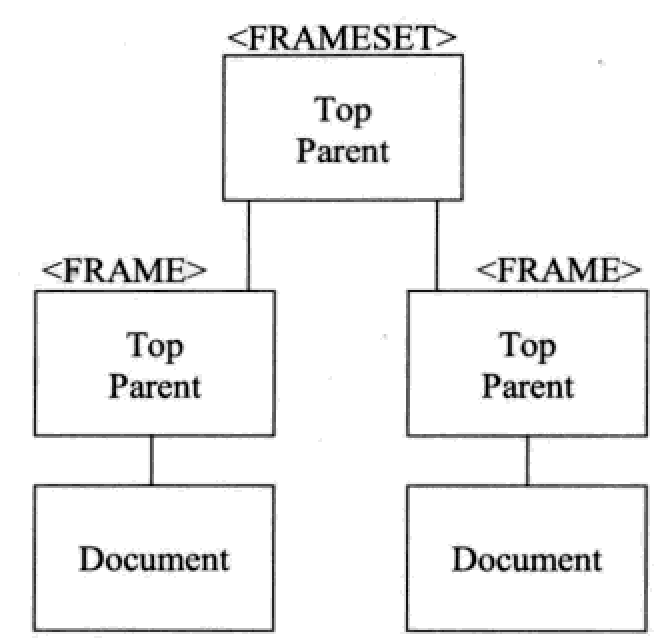


图 27-2　父框架和框架都是对象模型的一部分

<frameset>标记集中的每个<frame>标记都会创建另一个加载文档的 window 对象。因此，每个框架都有一个与其关联的 document 对象。对于给定的文档而言，它只有一个窗口容器，如图 27-1 所示的模型。尽管用户看不见 parent 对象，但它仍然存在于内存的对象模型中。parent 对象的存在常常使变量数据更便于由多个子框架共享，或者在一个子框架中加载不同的文档时保留变量的数据。

在更复杂的结构中，如图 27-3 所示，子框架本身也可以加载框架集文档。在这种情况下，parent 和 top 对象之间的区别就非常重要，top 窗口是图 27-3 中所有框架共有的唯一窗口。如后面所述，当一个框架需要与其他框架(及其文档)通信时，必须通过所有框架都拥有的 window 对象来引用远程对象。

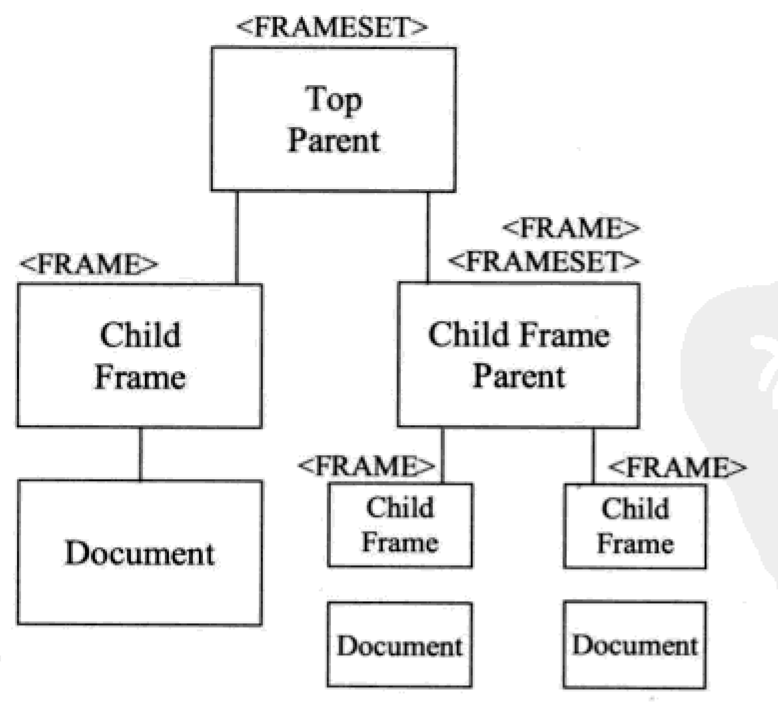


图 27-3　window 对象的 3 代

更多资源请访问稀酷客(www.ckook.com)

27.2.3 引用框架

对象引用的作用是帮助 JavaScript 在内存的当前对象模型中定位所期望的对象。引用为浏览器提供了一条路径，使其可以跟踪(例如)某个文档中某个文本域的值。因此，在构建引用时，要考虑脚本在对象模型的什么地方，以及引用如何帮助浏览器确定在何处找到远程对象。在如图 27-2 所示的两代中，两代之间存在三种引用关系。

- 父对子
- 子对父
- 子对子

假设要访问临近框架的对象、函数或变量，可以使用相关的引用结构：frameName. objFuncVarName、parent.objFuncVarName 和 parent.frameName.objFuncVarName。

规则是：只要引用需要指向另一个框架，该引用就必须从两个框架共有的 window 对象开始。下面用图 27-3 中的复杂模型来演示这个规则，如果左子框架的文档需要引用右下部的文档，引用结构是：

```
top.frameName.frameName.document. ...
```

从顶端 window 对象往下，经过两个框架，即可到达最终的文档，JavaScript 必须采用这条路径，所以引用也必须采用这条路径。

27.2.4 top 和 parent

在前面的对象地图和引用例子中，为什么不在所有的转换框架引用中把 top 用作首对象？从对象模型角度看，可以这么做，因为在两代环境中，parent 框架也是 top 窗口。但不能确保框架集文档在别人的浏览器中总是 top 窗口对象。例如，如果某 Web 站点将其他 Web 站点载入自己的一个框架中，则 top 窗口对象就可能在其他 Web 站点上。如果总是在引用中用 top 表示父窗口，则该引用无效，还可能产生脚本错误。因此，最好在表示当前文档的父对象时，在引用中使用 parent。

27.2.5 防止在其他 Web 站点的框架中显示自己的页面

可以运用 top 和 parent 引用的知识，来防止在其他 Web 站点的框架集中显示自己的页面。顶层文档必须验证它是否载入其 top 或 parent 窗口中。当文档在自己的 top 窗口中时，当前窗口的 top 属性引用就是对当前窗口的引用(在语法上，这里使用窗口的同义词 self 似乎最合适)。如果两个值不相等，可为文档编写脚本，将其重载为顶层文档。文档是顶层文档这一点很关键，所以将程序清单 27-1 中的脚本放在文档的 head 部分。

程序清单 27-1 防止在其他 Web 站点的框架中显示自己的页面

```
<script type="text/javascript">
   if (top != self)
   {
      top.location = location;
   }
```


更多资源请访问稀酷客(www.ckook.com)

```
</script>
```

文档会暂时显示在另一个站点的框架集中，接着清空它，让顶级文档控制浏览器窗口。

27.2.6　确认页面载入框架集

在框架集上设计 Web 应用程序时，应确保页面总是载入完整的框架集。如果访问者仅将一个框架添加到书签列表中，则下次访问时，只有添加了书签的页面才会显示在浏览器中，而不会显示框架集，框架集可能包含有价值的站点导航功能。

脚本可以比较 top 窗口和 self 窗口的 URL，来确保页面总是载入其框架集。如果 URL 相同，则意味着页面需要载入框架集。程序清单 27-2 显示了这种技术的最简单版本，它载入了一个固定的框架集。本书还提供了一个更完整的实现过程，它能将参数传递给框架集，在其中一个框架中打开特定页面，具体内容请参见第 28 章中的 location.search 属性。

程序清单 27-2　强制加载框架集

```
<script type="text/javascript">
   if (top.location.href == window.location.href)
   {
      top.location.href = " myFrameset.html";
   }
</script>
```

27.2.7　从有框架转换为无框架

某些站点默认将自己载入框架集，并为用户提供了去掉框架的选项。现代浏览器允许实时修改框架集的 cols 或 rows 属性，来模仿为当前视图添加或删除框架的效果(参见本章稍后的 frameset 元素对象)；另一方面，旧浏览器不允许在框架集载入后动态修改其构成，但可将框架集的内容页面载入到主窗口中。要解决旧浏览器的这个问题，可在站点中包括一个按钮或链接，其动作是将该文档载入 top 窗口对象：

```
top.location.href = "mainBody.html";
```

要切换回有框架版本，只需要加载框架集文档。

27.2.8　继承性和封装性

在面向对象的编程环境中，富有经验的脚本开发人员可能希望框架能继承在父对象中定义的属性、方法、函数和变量，但脚本浏览器不是这样。当使用对父对象的完全引用调用子对象时，仍然可以访问那些父对象。例如，如果想在所有子框架共享的框架集父文档中定义一个延迟函数，框架中的脚本就可以用下列引用指向该函数：

```
parent.myFunc()
```

可以为这些函数传递参数，得到返回值。

更多资源请访问稀酷客(www.ckook.com)

27.2.9　框架的同步

一些脚本开发人员计划将实时脚本包含在框架集文档中，这是十分危险的。这些脚本依赖的文档位于这个框架集文档创建的框架中。如果框架尚未创建完毕，其文档还未加载，实时脚本就可能崩溃。

此问题的预防措施之一是在所有框架集的 onload 事件处理程序中触发所有这些脚本。在理论上，只有所有文档都成功载入框架集定义的子框架中后，才执行这个处理程序。同时要注意在载入框架集框架的文档中的 onload 事件处理程序。如果某个脚本依赖另一个框架(其兄弟框架)中的文档，最终必将失败。速度较慢的网络、服务器或者调制解调器都会阻止其他文档以理想的顺序载入框架。

解决这些问题的办法是在父文档中创建一个 Boolean 变量，作为成功载入子框架的标志。当文档载入框架时，其 onload 事件处理程序可以将标志设置为 true，表示文档已经加载。只要脚本依赖于正在加载的页面，都要用 if 结构来检测该标志的值。

最好构造一些代码，让父框架的 onload 事件处理程序触发所有要在加载后执行的脚本。即使如此，还是要彻底检查页面，以防任何人改变窗口尺寸或者单击 Reload 而发生错误。

27.2.10　空白框架

通常最好在框架集中创建没有任何文档的框架；直到用户与其他框架中的各种控件或者其他用户界面元素交互时，才添加文档。多数现代浏览器都在其内部 URL 中包括一个空文档(about:blank)，然而，无法确保这个 URL 在所有浏览器上可用。如果需要空白框架，最好让框架集文档直接通过框架的 src 特性将一个通用的 HTML 文档直接写入框架，如程序清单 27-3 所示。载入一个空 HTML 文档无需任何其他处理。

程序清单 27-3　创建空白框架

```
<html>
   <head>
      <script type="text/javascript">
      <! —
         function blank()
         {
            return "<html></html>";
         }
      // — >
      </script>
   </head>
   <frameset>
      <frame name="Frame1" src="someURL.html">
      <frame name="Frame2" src="javascript:parent.blank()">
   </frameset>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

27.2.11　查看框架源代码

研究其他脚本人员的程序是学习 JavaScript(或者任意编程语言)的主要方法。在大多数脚本浏览器中，很容易查看任何框架的源代码，包括内容完全或者部分使用 JavaScript 产生的框架。单击所需的框架以激活它(在某些浏览器版本中，框架也许会显示一个边框，不过没有也不要紧)，然后从 View 菜单(或者右击子菜单)中选择 Frame Source 或等效元素，即可以查看框架的源代码。还可以打印或保存选中的框架。

27.2.12　框架和 frame 元素对象

对象模型把每个 HTML 元素运用于脚本编程中。随着对象模型的扩展，出现了一个术语冲突。本章前面对框架的讨论都是指原始对象模型，而框架是另一种窗口，其引用方式稍有区别，即使在最新的浏览器中也是如此。

但在对象模型也包含 HTML 元素时，frame 元素对象的含义就与原始模型的框架对象有所不同。frame 元素对象的属性受<frame>标记特性的控制，因此可以访问某些在原始框架对象中没有的属性，例如框架的边框和滚动条。

框架和 frame 元素对象的引用也不同，本章前面介绍了许多旧风格框架的引用，但必须通过 frame 元素的 id 特性或者 frameset 容器元素的子节点关系(不能用 parentNode 属性甩开当前文档，而访问封装文档的 frame 元素)来访问 frame 元素对象。最好给<frame>标记指定一个 id 特性，用框架集层次的 parent(或 top)中的 document 对象来访问 frame 元素对象。因此，要从框架的脚本中访问 frame 元素对象的 frameBorder 属性，语法是：

```
parent.document.all.frame1ID.frameBorder
```

对于 IE5+/Moz/W3C 是：

```
parent.document.getElementById("frame1ID").frameBorder
```

当引用访问 frame 元素对象时，可以通过元素对象的 contentWindow 或 contentDocument 属性，到达该框架中的 document 对象(参见本章后面的 frame 元素对象)。

27.3　window 对象属性

属　　性	方　　法	事件处理程序
appCore	addEventListener()†	onabort††
clientInformation	alert()	onafterprint
clipboardData	attachEvent()†	onbeforeprint
closed	back()	onbeforeunload
components[]	blur()†	onblur†
content	clearInterval()	onchange††
controllers[]	clearTimeout()	onclick††

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　　性	方　　法	事件处理程序
crypto	close()	onclose††
defaultStatus	confirm()	onerror
dialogArguments	createPopup()	onfocus†
dialogHeight	detachEvent()†	onhelp
dialogLeft	dispatchEvent()†	onkeydown††
dialogTop	dump()	onkeypress††
dialogWidth	execScript()	onkeyup††
directories	find()	onload
document	fireEvent()†	onmousedown††
event Event	focus()†	onmousemove††
external	forward()	onmouseout††
frameElement	geckoActiveXObject()	onmouseover††
frames[]	getComputedStyle()	onmouseup††
fullScreen	getSelection()	onmove
history	home()	onreset††
innerHeight	moveBy()	onresize
innerWidth	moveTo()	onscroll
length	navigate()	onselect††
location	open()	onsubmit††
locationbar	openDialog()	onunload
menubar	print()	
name	prompt()	
navigator	removeEventListener()†	
netscape	resizeBy()	
offscreenBuffering	resizeTo()	
opener	scroll()	
outerHeight	scrollBy()	
outerWidth	scrollByLines()	
pageXOffset	scrollByPages()	
pageYOffset	scrollTo()	
parent	setActive()†	
personalbar	setInterval()	
pkcs11	setTimeout()	
prompter	showHelp()	
returnValue	showModalDialog()	

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　　性	方　　法	事件处理程序
screen	showModelessDialog()	
screenLeft	sizeToContent()	
screenTop	stop()	
screenX		
screenY		
scrollbars		
scrollMaxX		
scrollMaxY		
scrollX		
scrollY		
self		
sidebar		
status		
statusbar		
toolbar		
top		
window		

† 见第 26 章。

†† 处理 IE4+和 W3C DOM 浏览器中其他对象的捕获或冒泡事件。

27.3.1　语法

创建窗口的语句如下：

```
var windowObject = window.open([parameters]);
```

访问 window 属性或方法的语句如下：

```
window.property | method([parameters])

self.property | method([parameters])

windowObject.property | method([parameters])
```

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

27.3.2　关于 window 对象

window 对象具有独一无二的地位：位于对象层次的最高层，甚至包括全能的 document 对象。这种极高的地位使 window 对象具有许多其他对象没有的属性和行为。

它具有许多独特性质，主要是因为所有活动都在窗口中进行，所以常常可以在对象引用中忽

更多资源请访问稀酷客(www.ckook.com)

略 window 对象。在前面的章节中调用文档方法，如 document.write()时，就已经采用了这种方式，它的完整引用是 window.document.write()。但因为活动在窗口中进行，而此窗口包含了运行脚本的文档，因此 window 是引用的一部分。对于单框架窗口而言，这个概念是很容易理解的。

如前所述，在 window 对象的一系列属性中有一个 self 属性，这个属性与 window 对象本身相同(所以它在层次结构表中显示为一个对象)。对象的属性与对象同名，这似乎令人难以理解，但这在面向对象环境中是很常见的。下面描述 self 属性时，将讨论为什么用 self 属性作为 window 对象的引用。

如前面的语法定义所示，不必总是在 JavaScript 代码中专门创建 window 对象。浏览器启动时常打开一个窗口，该窗口就是可用的 window 对象，即使它是空白的也同样如此。因此，用户把页面载入浏览器后将自动创建文档的 window 对象，以便脚本访问它。

一个需要避免的概念误区是：window 对象的事件处理程序或者自定义属性的生存期超出了创建它们的文档。窗口除了一些显而易见的物理属性之外，每个载入窗口的新文档都有全新的窗口属性和事件处理程序。

如果使用脚本控制现有(已打开)窗口的用户界面元素，其结果会随着运行应用程序的浏览器版本的不同而大不相同。在版本 4 以前的浏览器中，能对已打开窗口进行的唯一修改是修改浏览器窗口底部的状态栏。然而，版本 4 浏览器允许控制诸如位置、尺寸等属性和(在 Navigator 和 Mozilla 中使用签名脚本)窗框元素(例如工具栏和滚动条)的存在性。如果使用签名脚本(参见配书光盘中的第 49 章)或者用户允许进行这些改变，许多这类属性就可以修改，不受到特定的安全限制。

在所有浏览器中，window 对象的属性都非常活跃。用 window.open()方法生成新的窗口时，可以控制窗口的尺寸、工具栏或者其他视图选项。最近的浏览器版本还为新窗口提供了更多选项，包括窗口位置、是否显示窗口的标题栏等。同样，如果选项可以用于蒙蔽访问者(例如，自动隐藏一个窗口，该窗口在监控另一个窗口的活动)，就必须有签名脚本或者用户的许可。

在 window 对象上，脚本可以要求浏览器显示三种对话框(普通的警告对话框、OK/Cancel 确认对话框、用户文本输入对话框)中的一种。尽管对话框在为用户提供调试工具时非常有用(参见配书光盘中的第 48 章)，但它们会使浏览 Web 站点的访问者感到困惑。因为大多数 JavaScript 对话框是模态对话框(也就是说，在关闭对话框之前，不能在浏览器中做其他任何事情)，所以除非确有必要，务必谨慎使用。许多用户也许会在自己的计算机上创建宏来访问不能访问的站点，这种对站点的自动访问如果遇到了模态对话框，就会停止在该页面上，直到有人干预为止。

所有由 JavaScript 生成的对话框都把自己标示为由 JavaScript 生成，这主要是一种安全策略：防止具有欺骗性的脚本创建系统对话框或应用程序风格的对话框，它们会诱使访问者输入私人信息，也会影响对话框在网页设计中的应用。但有时对话框常常使用户感到烦恼。

除了 Safari 专用的模态对话框、IE 专用的模态和无模态对话框(请参见 window.showModalDialog()和 window.showModeless()方法)外，JavaScript 对话框只能输入基本的文本或者图像元素，没有太大的灵活性。事实上，甚至无法改变对话框按钮的内容或者添加按钮。在支持 DHTML 的浏览器上，可以用定位的 div 或者 iframe 元素以跨浏览器方式模拟对话框的行为。

至于 W3C DOM，在 DOM Level 2 之前，窗口一直没有收录到标准中。标准中最接近窗口的是 document.defaultView 属性，其值是目前浏览器(主要是 Mozilla)中的 window 对象，但正式的 DOM 标准没有为此视图对象指定属性或方法。

更多资源请访问稀酷客(www.ckook.com)

27.3.3　属性

appCore，components[]，content，controllers[]，prompter，sidebar

值：参见正文，　　　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

NN6+/Mozilla 允许通过脚本访问 xpconnect(xp 表示跨浏览器)包中的许多服务，xpconnect 服务是更大的 NPAPI(Netscape Plugin Application Programming Interface，Netscape 插件应用编程接口)的一部分。这些 xpconnect 服务允许脚本使用 COM 对象和 mozil1a.org XUL(基于 XML 的用户接口语言)工具(这是一个很宽泛的主题，超出了本书的讨论范围)。在基于 Mozilla 的浏览器和脚本编程中可以研究这个主题，参见 http://www.mozilla.org/scriptable/。

clientInformation

值：navigator 对象，　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari 1.2+，Opera-，Chrome+

为了提供对浏览器级的属性的脚本访问，同时避免对 Navigator 浏览器商标的引用，Microsoft 提供了 clientInformation 属性。它的值与 navigator 对象的值相同，是一个在 IE 中也可用的对象名。尽管 Safari 1.2 采用了 clientInformation 属性，但应为跨浏览器应用程序使用 navigator 对象(参见配书光盘中的第 42 章)。

相关主题：navigator 对象。

clipboardData

值：对象，　　　　读/写

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

可以使用 clipboardData 对象在脚本控制下传送数据，完成诸如剪切、复制和粘贴等动作。该对象包括与传送操作有关的一种或者多种数据类型。只有通过 Edit 菜单(或者组合键)或者由脚本控制(一般是使用与编辑相关的事件处理程序)的上下文菜单来执行编辑过程，才使用这一属性。

使用 clipboardData 对象需要了解表 27-1 中的三个方法，并熟悉与编辑相关的事件处理程序(剪切、复制、粘贴的前后版本，见第 26 章)。

表 27-1　window.clipboardData 对象方法

方　法	返 回 值	说　明
clearData([*format*])	无	从剪贴板中删除数据。如果没有提供格式参数，就清除所有数据。数据格式可以是以下的一个或多个字符串：Text、URL、File、HTML、Image
getData(*format*)	String	从剪贴板上获取指定格式的数据。format 是以下某个字符串：Text、URL、File、HTML、Image。在获取数据时，不会清空剪贴板，以便在多个顺序操作中获取数据
setData(*format*，*data*)	Boolean	将字符串数据存储在剪贴板中。format 是以下某个字符串：Text、URL、File、HTML、Image。对于非文本数据格式，data 必须是指定内容的路径或 URL 的字符串。如果向剪贴板传输数据成功，就返回 true

更多资源请访问稀酷客(www.ckook.com)

如果这些页面来自不同的域或者经由不同的协议(http 与 https)到达，则不能用 clipboardData 对象在某些页面之间传输数据。

示例

参见第 26 章的程序清单 26-36 和程序清单 26-44，查看 clipboardData 对象如何用于各种与编辑相关的事件处理程序。

相关主题：event.dataTransfer 属性；onbeforecopy、onBeforeCut、onbeforepaste、oncopy、oncut、onpaste 事件处理程序。

closed

值：Boolean，　　只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

用 window.open()方法创建子窗口时，必须从该子窗口中访问对象属性，例如设置文本域的值。为了访问子窗口，应使用 window.open()方法返回的 window 对象引用，如下所示：

```
var newWind = window.open("someURL.html","subWind");
...
newWind.document.entryForm.ZIP.value = "00000";
```

在这个例子中，变量 newWind 不是指向窗口的活动链接，而只是该窗口的引用。如果用户关闭窗口，变量 newWind 仍然包含指向已关闭窗口的引用。因此，引用关闭窗口中的任何对象都可能产生脚本错误。在访问子窗口中的项之前，必须知道窗口是否是打开的。

如果脚本或用户已经关闭了 window 对象，closed 属性就返回 true。总是可以在关闭窗口之前触发一个脚本语句来检查 closed 属性的值。

示例

程序清单 27-4 是一个基本的打开和关闭窗口示例。脚本首先初始化一个全局变量 newWind 用于存储第二个窗口的对象引用。该值必须是全局的，这样其他函数才能引用窗口完成相关任务，如关闭窗口。

在此示例中，新窗口包含一些动态写入的 HTML 代码而不是载入现有的 HTML 文件。因此，window.open()方法的 URL 参数是空字符串。接下来是一段短暂的延时以允许 Internet Explorer(尤其是版本 3 和 4)打开窗口，写入内容。这个延时(使用本章稍后说明的 setTimeout()方法)调用 finishNewWindow()函数来实现，该函数使用全局变量 newWind 来引用窗口，执行写入操作。document.close()方法关闭了对文档的写入，这与关闭窗口不同，另一个函数 closeWindow()负责关闭子窗口。

在最后一个测试中，if 语句检查两个条件：(1) window 对象是否用 null 之外的值进行了初始化(以防在创建新窗口前，单击窗口关闭按钮)；(2)窗口的 closed 属性是 null 还是 false。如果有一个条件为真，就给第二个窗口调用 close()方法。

注意：

这是一个简单例子，所以其中的事件处理属性赋值技术使用了绑定事件的现代方式：addEventListener() (NN6+/Moz/W3C)或 attachEvent()(IE5+)方法。这个现代的跨浏览器事件处理

更多资源请访问稀酷客(www.ckook.com)

技术详见第 32 章。本章及本书大部分代码所使用的事件处理技术是有意简化了的，以使代码更便于阅读。通常最好采用更现代的方式来绑定事件，如本例所示。

程序清单 27-4 在关闭窗口前进行检查

HTML: jsb27-04.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.closed Property</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb27-04.js"></script>
   </head>
   <body>
      <h1>window.closed Property</h1>
      <hr />
      <p>The new window might open behind this window.</p>
      <form id="newWinForm" name="newWinForm">
         <input id="openNewWin" name="openNewWin" type="button"
                value="Open Window" />
         <br />
         <input id="closeNewWin" name="closeNewWin" type="button"
                value="Close it if Still Open" />
      </form>
   </body>
</html>
```

JavaScript: jsb27-04.js

```
// initialize when the page has loaded
addEvent(window, "load", initialize);

// global variables
   var oNewWinForm;
   var oOpenNewWin;
   var oCloseNewWin;

// initialize global var for new window object
// so it can be accessed by all functions on the page
var newWind;
function initialize ()
{
   // get IE4+ or W3C DOM reference for an ID
   if (document.getElementById)
   {
      oNewWinForm = document.getElementById("newWinForm");
      oOpenNewWin = document.getElementById("openNewWin");
      oCloseNewWin = document.getElementById("closeNewWin");
   }
   else
```


更多资源请访问稀酷客(www.ckook.com)

```
        {
            oNewWinForm = document.newWinForm;
            oOpenNewWin = document.oNewWinForm.openNewWin;
            oCloseNewWin = document.oNewWinForm.closeNewWin;
        }
        addEvent(oOpenNewWin, 'click', newWindow);
        addEvent(oCloseNewWin, 'click', closeWindow);
    }

    // make the new window and put some stuff in it
    function newWindow()
    {
        newWind = window.open("","subwindow","height=200,width=200");
        setTimeout("finishNewWindow()", 100);
    }
    function finishNewWindow()
    {
        var output = "";
        output += "<html><body><h1>A Sub-window</h1>";
        output += "<form><input type='button' value='Close Main Window'";
        output +="onclick='window.opener.close()'></form></body></html>";
        newWind.document.write(output);
        newWind.document.close();
    }

    // close subwindow
    function closeWindow()
    {
        if (newWind && !newWind.closed)
        {
            newWind.close();
        }
        else
        {
            alert("The window is no longer open");
        }
    }
```

为了完成打开和关闭窗口的示例，给子窗口提供一个按钮，其 onclick 事件处理程序(在 initialization()函数中绑定)关闭主窗口/选项卡。在一些现代浏览器中，会给用户显示一个警告框，以确认关闭主浏览器窗口/选项卡。其他浏览器则忽略这个请求，有时仅在错误控制台上显示一个警告。Safari 在关闭主窗口/选项卡时不显示任何警告。

相关主题： window.open()，window.close()方法。

components

(参见 appCore)

更多资源请访问稀酷客(www.ckook.com)

controllers

(参见 appCore)

crypto，pkcs11

值：对象引用，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

crypto 和 pkcsll 属性返回浏览器对象的引用，这些对象与内部公钥的加密机制有关。这些主题超出了本书的讨论范围，但可以在站点 http://www.mozilla.org/projects/security/中了解到 Netscape 公司在这一领域的最新成果。

defaultStatus

值：字符串　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome-

文档加载到窗口或者框架之后，状态栏的消息域可显示一个字符串，只要鼠标的光标不在比状态栏还重要的对象(例如链接对象或者图像地图)上，这个字符串就是可见的。window.defaultStatus 属性通常是一个空字符串，但任何时候都可以设置这个属性。用户将光标移动到链接对象上时，这个属性设置就会被临时覆盖(参见 window.status 属性，它用于自定义临时的状态栏消息)。

window.defaultStatus 属性最常在文档载入窗口时设置。可以把它作为实时脚本语句加入文档的 head 或者 body 部分，或者执行为文档的 onload 事件处理程序的一部分。

示例

除非准备在用户浏览网页时更改默认状态栏文本，否则最好在文档载入时设置此属性。注意，程序清单 27-5 也在 onmouseout 事件处理程序中读取此属性，来重置状态栏。将 status 属性设置为空，也会将状态栏重置为 defaultStatus 设置。测试时，注意现代浏览器支持 status 属性，但其中一些浏览器不会显示用脚本编写的、与链接相关的状态栏文本，以避免链接诈骗。

程序清单 27-5　设置默认状态消息

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.defaultStatus property</title>
      <script type="text/javascript">
         window.defaultStatus = "Welcome to my Web site.";
      </script>
   </head>
   <body>
      <h1>window.defaultStatus property</h1>
      <hr />
      <p><a href="http://www.microsoft.com"
          onmouseover="window.status = 'Visit Microsoft\'s Home page. '; return true;"
```


更多资源请访问稀酷客(www.ckook.com)

```
            onmouseout="window.status = '';return true;">Microsoft</a>
        </p>
        <p><a href="http://mozilla.org"
            onmouseover="window.status = 'Visit Mozilla\'s Home page. '; return true;"
            onmouseout="window.status = window.defaultStatus;return true;">Mozilla</a>
        </p>
    </body>
</html>
```

如果需要在状态栏中显示单引号或双引号(如程序清单 27-5 中的两个链接)，可将转义序列(\"和\")作为赋予这些属性的字符串的一部分。

相关主题：window. status 属性。

dialogArguments

值：可变，　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera-，Chrome+

dialogArguments 属性只能用于由 showModalDialog()或 IE 特定的 showModelessDialog()方法生成的窗口。这些方法允许把一个参数传递到对话框窗口中，而且 dialogArguments 属性允许对话框窗口中的脚本访问该参数值，该值可以是字符串、数字或者 JavaScript 数组(这样便于传递多个值)。

示例

参见用于 window.showModalDialog()方法的程序清单 27-36，查看如何通过 dialogArguments 属性在对话框中传递和提取参数。

相关主题：window.showModalDialog()，window.showModelessDialog()方法。

dialogHeight，dialogWidth

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

如果文档位于由 showModalDialog()或 showModelessDialog()产生的模态对话框或者 IE 特定的非模态对话框中，文档中的脚本就可以通过 dialogHeight 和 dialogWidth 属性来读取或改变对话框窗口的高度和宽度。只有非模态对话框中的脚本才能通过主窗口访问这些属性，当用户控制主窗口元素时，这些脚本依然可见。

这些属性的值是字符串，其度量单位为像素(px)。

示例

对话框有时包含一些按钮或图标，为高级用户提供更多细节或更复杂的设置。可创建一个函数来切换两种尺寸的对话框。以下函数假设对话框中的文档有一个按钮，其标签在 Show Details 和 Hide Details 间切换。按钮的 onclick 事件处理程序通过 toggleDetails(this)语句调用函数：

```
function toggleDetails(btn)
{
   if (dialogHeight == "200px")
```


更多资源请访问稀酷客(www.ckook.com)

```
    {
       dialogHeight = "350px";
       btn.value = "Hide Details";
    }
    else
    {
       dialogHeight = "200px";
       btn.value = "Show Details";
    }
}
```

实际上，还必须将额外信息的 display 样式表属性在 none 和 block 之间切换，以确保在较小的对话框版本中不显示滚动条。

相关主题：window.dialogLeft 属性，window.dialogTop 属性。

dialogLeft，dialogTop

值：字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

如果文档位于由 showModalDialog()或 showModelessDialog()产生的模态对话框或者 IE 特定的非模态对话框中，文档中的脚本就可以通过 dialogLeft 和 dialogTop 属性来读取或改变对话框窗口的 left 和 top 坐标。只有非模态对话框中的脚本才能通过主窗口访问这些属性，当用户控制主窗口元素时，这些脚本依然可见。

这些属性的值是字符串，度量单位为像素(px)。如果更改这些值，使对话框窗口的任意部分在视频监视器之外，浏览器就会覆盖这些设置，以使整个窗口可见。

示例

可以重新定位已由脚本(或者用户，假定允许调整对话框的大小)调整过大小的对话框窗口，但这可能会影响用户，通常不太好。在对话框窗口文档的脚本中，以下语句将对话框窗口重新居中：

```
dialogLeft = (screen.availWidth/2) - (parseInt(dialogWidth)/2) + "px";
dialogHeight = (screen.availHeight/2) - (parseInt(dialogHeight)/2) + "px";
```

注意，parseInt()函数用于读取 dialogWidth 和 dialogHeight 属性的数值部分，以用于计算。

相关主题：window.dialogHeight 属性，window.dialogTopWidth 属性。

directories，locationbar，menubar，personalbar，scrollbars，statusbar，toolbar

值：对象， 读/写(用带签名的脚本)

兼容性：WinIE-，MacIE-，NN4+，Moz+，Safari-，Opera-，Chrome-

在窗口元素的方形区域(文档在这个区域中)外部，Netscape 浏览器窗口显示工具栏和其他窗框控件。创建新窗口时，所有浏览器都可以删除这些窗框项(作为 window.open()方法第三个参数的一部分)，但直到签名脚本可以在 Navigator 4 中使用，这些项才能在主浏览器窗口或者任何已有的窗口中打开和关闭。

更多资源请访问稀酷客(www.ckook.com)

Navigator 4 把这些元素提升为 window 对象包含的一级对象，Navigator 6 还增加了另一个特征，称为目录条，它类似于框架，可以从浏览器窗口的左边缘打开或者隐藏，同时，NN6+/Mozilla 浏览器不再允许隐藏和显示浏览器窗口的滚动条。窗框对象只有一个属性 visible，通过读取这个属性的 Boolean 值(没有签名脚本)，可为当前使用的元素检查访问者的浏览器窗口。

改变这些项的可见性将改变浏览器窗口内部和外部尺寸的关系。如果必须仔细控制窗口的大小才能显示内容，则应该先调整窗框元素，再调整窗口的大小。在改变窗框的可见性之前，应仔细权衡这个决定。因为富有经验的用户会把浏览器窗口设置成自己喜欢的外观，如果弄乱了页面的外观，则将失去许多访问者。幸好，对窗框元素可见性的改变不会保存到用户的参数首选项中。然而，这些改动会在页面卸载时保留下来。所以如果要改变这些设置，应确保先保存最初的设置，再用 onunload 事件处理程序恢复。

提示：

Macintosh 菜单栏不是浏览器窗口窗框的一部分，因此，它的可见性不能通过脚本进行调整。

示例

在程序清单 27-6 中，可以试着打开、关闭窗框元素来调整浏览器窗口的外观。要运行此脚本，必须对脚本签名或者打开 codebase principals(参见配书光盘中的第 49 章)，还必须启用 Java 以使用签名脚本语句。

页面在载入时，存储了每个窗框元素的当前状态。每个窗框元素的按钮都会触发 toggleBar() 函数，此函数对作为参数传递给函数的窗框对象的 visible 属性取反。最后，Restore 按钮将 visible 属性恢复为其原始设置。注意，restore()函数也由文档的 onunload 事件处理程序调用。

程序清单 27-6 控制窗口的窗框

HTML: jsb27-06.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Bars Bars Bars</title>
      <script type="text/javascript" src="jsb27-06.js"></script>
   </head>
   <body onunload="restore()">
      <h1>Bars Bars Bars</h1>
      <hr />
      <form>
         <b>Toggle Window Bars</b>
         <br />
         <input type="button" value="Location Bar"
                onclick="toggleBar(window.locationbar)" />
         <br />
         <input type="button" value="Menu Bar"
                onclick="toggleBar(window.menubar)" />
```


更多资源请访问稀酷客(www.ckook.com)

```
        <br />
        <input type="button" value="Personal Bar"
               onclick="toggleBar(window.personalbar)" />
        <br />
        <input type="button" value="Scrollbars"
               onclick="toggleBar(window.scrollbars)" />
        <br />
        <input type="button" value="Status Bar"
               onclick="toggleBar(window.statusbar)" />
        <br />
        <input type="button" value="Tool Bar"
               onclick="toggleBar(window.toolbar)" />
        <br />
        <hr />
        <input type="button" value="Restore Original Settings"
               onclick="restore()" />
      </form>
   </body>
</html>
```

JavaScript: jsb27-06.js

```
// store original outer dimensions as page loads
var originalLocationbar = window.locationbar.visible;
var originalMenubar = window.menubar.visible;
var originalPersonalbar = window.personalbar.visible;
var originalScrollbars = window.scrollbars.visible;
var originalStatusbar = window.statusbar.visible;
var originalToolbar = window.toolbar.visible;

// generic function to set inner dimensions
function toggleBar(bar)
{
   netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
   bar.visible = !bar.visible;
   netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite");
}
// restore settings
function restore()
{
   netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
   window.locationbar.visible = originalLocationbar;
   window.menubar.visible = originalMenubar;
   window.personalbar.visible = originalPersonalbar;
   window.scrollbars.visible = originalScrollbars;
   window.statusbar.visible = originalStatusbar;
   window.toolbar.visible = originalToolbar;
   netscape.security.PrivilegeManager.revertPrivilege("UniversalBrowserWrite");
}
```

相关主题：window.open()方法。

更多资源请访问稀酷客(www.ckook.com)

document

值：对象，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

将 document 属性列在此处主要是为了完整性。每个 window 对象都包含一个 document 对象。document 属性的值是 document 对象，这不是一个可显示的值。但在构造 document 对象属性和方法的引用，以及文档包含的其他对象(例如表单及其元素)的引用时，可使用 document 属性。要将不同文档载入窗口，可使用 location 对象(见第 28 章)。document 对象详见第 29 章。

相关主题：document 对象。

event，Event

值：对象，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4+，MacIE4+，NN+，Moz+，Safari 1+，Opera+，Chrome+

IE4+把 event 对象处理为 window 对象的一个属性，而 W3C DOM 把 Event 对象的实例作为参数传递给事件处理函数。event 对象与 window 对象的连接相当不合逻辑，因为所有与 event 对象相关的动作都发生在事件处理函数中，唯一的区别是当一个事件处理函数调用另一个事件处理函数时，event 对象会处理为更全局的对象。函数能直接访问 event 对象(在引用中可以使用 window 前缀，也可以不用)，它不必向下一个函数传递 event 对象参数。

在所有浏览器中使用 event 对象的细节请参阅第 32 章。

相关主题：event 对象。

external

值：对象，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE4+，MacIE-，NN-，Moz+，Safari-，Opera-，Chrome-

仅当浏览器窗口是另一个应用程序的组件时，external 属性才是有效的。此属性提供了当前浏览器窗口与其应用程序宿主之间的通道。

若 WinIE4+是主操作系统的一个组件，则 external 属性可用来访问多个方法来影响浏览器外部的行为。对于常规 Web 页面的脚本编写人员而言，最有效的 3 个方法是 AddDesktop Component()、AddFavorite()和 NavigateAndFind()。用户在浏览器或桌面菜单上做出选择后，前两个方法将显示同一个警告对话框，因此未经访问者的许可，不能擅自将 Web 站点加入他们的桌面或者 Favorites 列表。IE7 和 Firefox2+允许 AddSearchProvider()安装搜索插件。与前两个方法一样，在安装搜索插件时，会显示一个警告框，请求用户批准。OpenSearch 和传递给方法的 XML 文件格式请参见 http://www.opensearch.org/Specifications/OpenSearch/1.1。表 27-2 描述了这些方法的参数。

表 27-2　流行的 window.external 对象方法

方　　法	说　　明
AddDesktopComponent("URL", "type"[, left, top, width, height])	添加一个网站或图像到活动桌面(假定它在用户的 Windows 版本中打开)。type 参数值为 website 或 image，尺寸参数(可选)都是整数值

更多资源请访问稀酷客(www.ckook.com)

(续表)

方　法	说　明
AddFavorite("URL"[, "title"])	将指定的 URL 添加到用户的收藏夹列表中。可选的标题字符串参数决定 URL 如何在菜单中列出(如果没有此参数，URL 就显示在列表中)
AddSearchProvider("URL")	添加搜索提供程序，该提供程序在 URL 中用一个 OpenSearchDescription XML 文件指定
NavigateAndFind("URL", "findString", "target")	导航到第一个参数中的 URL，并在 target 框架中打开页面(若第一个参数是空字符串，则在当前框架中打开)。findString 是要在该页面上搜索并在页面载入时突出显示的文本

示例

第一个示例询问用户是否允许将一个网站添加到活动桌面上。如果未启用活动桌面，就给用户提供启用它的选项：

```
external.AddDesktopComponent("http://www.nytimes.com","website", 200,100,
400, 400);
```

在下一个示例中，询问用户是否允许将一个 URL 添加到收藏夹列表中。用户可以按照正常操作程序，将 URL 项放在列表的文件夹中：

```
external.AddFavorite("http://www.dannyg.com/support/update13.html",
   "JSBible 6 Support Center");
```

最后一个示例假定用户在一个 select 列表中进行了选择。select 列表的 onchange 事件处理程序调用以下函数，导航到一个虚构页面上，并在页面中查找所选运动队的列表：

```
function locate(list)
{
   var choice = list.options[list.selectedIndex].value;
   external.NavigateAndFind("http://www.collegesports.net/scores.html",
      choice, "scores");
}
```

frameElement

值：frame 或 iframe 对象引用，　　只读

兼容性：WinIE5.5+，MacIE-，NN7+，Moz1.0.1+，Safari 1.2+，Opera+，Chrome+

如果当前窗口因<frame>或者<iframe>标记而存在，窗口的 frameElement 属性就返回其宿主元素的引用。如本章后面的 frame 元素对象所述，frame 或 iframe 元素对象的引用可以访问某些属性，这些属性反映了 HTML 元素对象的特性。对于不属于框架集的窗口，frameElement 属性返回 null。

把单个文档加载到多个框架集时，这种属性的简便性表现得比较明显。甚至元素的 ID 从一个框架集变到另一个框架集时，文档中的脚本仍然指向 frame 包含元素。也可以通过 frameElement 属性的 parentElement 属性，来访问 frameset 元素：

更多资源请访问稀酷客(www.ckook.com)

```
var frameSetObj = self.frameElement.parentElement;
```

frameset 元素的引用允许调整框架的尺寸。

相关主题：frame，iframe 对象。

frames

值：数组，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在多框架窗口中，top 或者 parent 窗口包含任意多个独立的框架，每个框架都是一个完整的 window 对象。若语句必须引用另一个框架中的对象，就可以使用 frames 属性(注意属性名字的复数形式)。例如，若一个框架中的按钮把文档加载到另一个框架中，按钮的事件处理程序就必须能准确告诉 JavaScript 在哪里显示新 HTML 文档，此时就应使用 frames 属性。

为了利用 frames 属性在框架之间通信，它必须是以 parent 或者 top 属性开头的引用的一部分。这让 JavaScript 在层次结构中选择正确的路径，通过所有当前加载的对象，找到期望的对象。为了确定窗口中有多少个活动框架，可以使用以下表达式：

```
parent.frames.length
```

这个表达式的返回值表示父窗口定义了多少个框架，然而，如果在环境中定义了第三代框架，则这个嵌套框架并不记入这个值。换句话说，如果存在多代框架，就不能用单个属性来确定浏览器窗口中的框架总数。

浏览器在一个数字索引的数组中保存了所有可见框架的信息，其中第一个框架的索引(在框架集中定义的最外层的<frame>标记)为 0：

```
parent.frames[0]
```

因此，如果窗口显示了三个框架(其索引分别为 frames[0]、frames[1]和 frames[2])，则检索第二个框架中文档的 title 属性的引用是：

```
parent.frames[1].document.title
```

这个引用是一个路径图，它开始于父窗口，延伸到第二个框架的文档及其 title 属性。在 IE4 和 NN6/Moz/W3C 之前，除了在父窗口中定义的框架数和每个框架的名称(top.frames[i].name)之外，不能通过脚本直接从框架对象中访问框架定义中的其他值(见本章后面的 frame 元素对象)。在这些浏览器中，各个 frame 元素对象包含几个表示<frame>标记特性的属性。

然而，使用框架引用的索引值未必是最安全的策略，因为框架集的设计可能随时间而改变，此时，索引值也将改变。因此，最好利用<frame>标记的 name 特性，并为每个框架指定唯一的描述性名称。赋予 name 特性的值也是链接的 target 特性名称，以使链接的页面载入没有包含该链接的框架。还可以利用框架名替代索引引用，例如，程序清单 27-7 为两个框架指定了不同的名称。要访问 JustAKid2 框架中文档的标题，完整的对象引用是：

```
parent.JustAKid2.document.title
```

可用区分大小写的框架名代替 frames [1]数组引用，或者为了保持 JavaScript 的灵活性，可

更多资源请访问稀酷客(www.ckook.com)

使用对象名替代数组索引位置：

```
parent.frames["JustAKid2"].document.title
```

在引用中使用框架名的最大优点是，不管框架集的结构是否随时间而变化，对指定框架的引用总是能找到该框架，尽管其索引值(其在框架集中的位置)可能会改变。

示例

程序清单 27-7 和程序清单 27-8 演示了 JavaScript 如何处理来自框架内对象的框架引用值。将同一文档载入到每个框架中，该文档中的脚本提取当前框架和整个框架集的信息。

程序清单 27-7 程序清单 27-8 的框架集文档

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>window.frames property</title>
   </head>
   <frameset cols="50%,50%">
      <frame name="JustAKid1" src="jsb27-08.html" />
      <frame name="JustAKid2" src="jsb27-08.html" />
   </frameset>
</html>
```

对于当前引用的框架而言，框架的数量(长度)是 0，因为这里的每个框架都是一个没有内嵌框架的窗口。但在引用中增加 parent 属性后，就应考虑由父窗口文档生成的所有框架。

程序清单 27-8 显示不同窗口属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Window Revealer II</title>
      <script type="text/javascript">
         function gatherWindowData()
         {
            var msg = "";
            msg += "<p><b>From the point of view of this frame:</b><br />";
            msg += "window.frames.length: " + window.frames.length + "<br />";
            msg += "window.name: " + window.name + "</p>";
            msg += "<p><b>From the point of view of the framesetting ↵
                     document:</b><br />";
            msg += "parent.frames.length: " + parent.frames.length + "<br />";
            msg += "parent.frames[0].name: " + parent.frames[0].name + "</p>";
            return msg;
         }
      </script>
   </head>
```


更多资源请访问稀酷客(www.ckook.com)

```
   <body>
      <script type="text/javascript">
         document.write(gatherWindowData());
      </script>
   </body>
</html>
```

示例中的最后一条语句说明了如何通过数组语法(方括号)来引用特定框架。数组下标从 0 开始，0 表示第一项。由于文档需要第一个框架的名称(parent.frames[0])，因此对两个框架的响应都是 JustAKidl。

相关主题： frame 对象，frameset 对象；window.parent 属性，window.top 属性。

fullScreen

值： Boolean， 只读

兼容性： WinIE-，MacIE-，NN7.1+，Moz1.4+，Safari-，Opera-，Chrome-

fullScreen 属性用于表明浏览器是否处于全屏模式，在 Mozilla 浏览器中，全屏模式可以通过 View 菜单的 Full Screen 命令来设置。可惜，此属性并不可靠(到 Mozilla 1.8.1 为止)，无论浏览器的全屏设置如何，它总是返回 false。

history

值： 对象， 只读

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

参见第 28 章讨论的 history 对象。

innerHeight，innerWidth，outerHeight，outerWidth

值： 整型， 读/写(参见正文)

兼容性： WinIE-，MacIE-，NN4+，Moz+，Safari+

基于 NN4+/Moz/WebKit 和基于 Presto 的浏览器允许用脚本设置属性，来调整窗口的高度和宽度，包括主浏览器窗口。这种调整有助于在设置为特定宽度和高度的浏览器窗口中，页面以最佳效果显示。预先指定窗口的大小，就不要求用户设置浏览器窗口的大小，而获得页面的最佳效果(尽管用户通常习惯于手工设置主窗口的大小)。另外，还可以通过 navigator 对象(见配书光盘中的第 42 章)来检查访问者的操作系统，从而设置窗口的大小，使其根据不同的平台调整字体和表单元素。

支持这些属性的浏览器提供了两种不同的引用 inner 和 outer，来测试窗口的宽度和高度，两者都以像素为单位。inner 测试的是窗口的活动文档区域(有时称为窗口的内容区域)。如果文档的最佳显示效果取决于文档显示区域的像素高度或者宽度，就设置 innerHeight 和 innerWidth 属性。

相反，outer 测试的是整个窗口的外边界，包括显示在窗口中的所有窗框元素，如滚动条、状态栏等。outerHeight 和 outerWidth 属性的设置通常与 screen 对象属性(见配书光盘中的第 42 章)一致。outer 属性最常见的用法是设置浏览器窗口，使其填满用户显示器的可用屏幕区域。

要改变窗口的两个外部尺寸，更有效的方式是利用 window.resizeTo()方法，这个方法也可用于 IE4+。这个方法把像素宽度和高度(整数值)作为参数，在一条语句中实现窗口大小的改变。

更多资源请访问稀酷客(www.ckook.com)

注意，改变窗口的尺寸并不调整窗口的位置，因此，将窗口的外部尺寸设置为 screen 对象返回的可用空间，并不意味着窗口就立即占满显示器的可用空间。使用 window.moveTo()方法可以确保窗口的左上角在屏幕的(0, 0)坐标位置。

尽管这些属性给页面设计者提供了灵活性，Netscape 和基于 Mozilla 的浏览器仍为未加密签名的脚本指定了最小的尺寸限制，例如不能使窗口的高和宽均小于 100 像素。这种限制可防止未签名的脚本设置几乎不可见的小窗口，来监控其他窗口的活动。然而，如果用户许可，则可以使用签名的脚本把窗口设置为小于 100×100 像素。IE4+设置了一个更小的最小尺寸，来防止窗口的大小重新设置为 0。

警告：

用户可能不喜欢脚本随意更改其浏览器窗口的大小和位置。基于 NN7+/Moz/WebKit 和 Presto 的浏览器不允许脚本调整窗口的大小，除非脚本加上了签名。

示例

在程序清单 27-9 中，几个按钮会显示设置 innerHeight、innerWidth、outerHeight 和 outerWidth 属性的结果。Safari 会忽略脚本对这些属性的调整，而 Mozilla 用户可以设置首选项，来防止脚本移动窗口和调整窗口的大小。

程序清单 27-9　设置窗口的高度和宽度

HTML：jsb27-09.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Window Sizer</title>
      <script type="text/javascript" src="jsb27-09.js"></script>
   </head>
   <body>
      <h1>Window Sizer</h1>
      <hr />
      <form>
         <b>Setting Inner Sizes</b><br />
         <input type="button" value="600 Pixels Square"
                onclick="setInner(600,600)" /><br />
         <input type="button" value="300 Pixels Square"
                onclick="setInner(300,300)" /><br />
         <input type="button" value="Available Screen Space"
                onclick="setInner(screen.availWidth, screen.availHeight)" /><br />
         <hr />
         <b>Setting Outer Sizes</b><br />
         <input type="button" value="600 Pixels Square"
                onclick="setOuter(600,600)" /><br />
         <input type="button" value="300 Pixels Square"
                onclick="setOuter(300,300)" /><br />
         <input type="button" value="Available Screen Space"
```


更多资源请访问稀酷客(www.ckook.com)

```
                onclick="setOuter(screen.availWidth, screen.availHeight)" /><br />
         <hr />
         <input type="button" value="Cinch up for Win95"
                onclick="setInner(273,304)" /><br />
         <input type="button" value="Cinch up for Mac"
                onclick="setInner(273,304)" /><br />
         <input type="button" value="Restore Original"
                onclick="restore()" /><br />
      </form>
   </body>
</html>
```

JavaScript: jsb27-09.js

```
// store original outer dimensions as page loads
var originalWidth = window.outerWidth;
var originalHeight = window.outerHeight;
alert("original dimensions: \nwindow.outerWidth="
      + window.outerWidth
      + ". \nwindow.outerHeight ="
      + window.outerHeight
      + ". \nwindow.innerWidth = "
      + window.innerWidth
      + ". \nwindow.outerHeight = "
      + window.outerHeight);

// generic function to set inner dimensions
function setInner(width, height)
{
   window.innerWidth = width;
   window.innerHeight = height;
}

// generic function to set outer dimensions
function setOuter(width, height)
{
   window.outerWidth = width;
   window.outerHeight = height;
}

// restore window to original dimensions
function restore()
{
   window.outerWidth = originalWidth;
   window.outerHeight = originalHeight;
}
```

相关主题： window.resizeTo()方法，window.moveTo()方法；screen 对象，navigator 对象。

location

值： 对象，　　　　读/写

更多资源请访问稀酷客(www.ckook.com)

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

详见第 28 章讨论的 location 对象。

locationbar

参见 directories。

name

值：字符串，　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

所有 window 对象都有指定的名称。使用框架时，名称特别有用，因为在多框架环境中，优秀的命名机制有助于从其他框架引用中精确地定位当前使用的框架。

然而，主浏览器窗口没有默认的名称，其 name 的值是空字符串。没必要为该窗口指定名称，因为 JavaScript 和 HTML 提供了大量的其他方式来指向 window 对象(top 属性、target 特性的_top 常数和子窗口的 opener 属性)。

如果想给主窗口指定名称，可以随时设置 window.name 属性。但注意因为这个属性的生存期超出了任何给定文档的载入和卸载时间，可能脚本只在一个文档或者框架集中使用该引用。除非恢复默认的空字符串，否则所指定的窗口名将显示在后来载入的其他文档中。在这方面，最好在窗口或者框架集的 onload 事件处理程序中设定名字，再在对应的 onunload 事件处理程序中将它重置为空字符串：

```
<body onload="self.name = 'Main'" onunload="self.name = ''">
```

在程序清单 27-15 中显示了这种方式的应用示例，设置父窗口的名称将有助于弄清父子窗口之间的关系。

相关主题：top 属性；window.open()方法，window.sizeToContent()方法。

navigator

值：对象，　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

尽管在现代浏览器中，navigator 对象是 window 对象的一个属性，但它从一开始就存在(请参见配书光盘中的第 42 章)。在以前的浏览器中，navigator 对象引用为独立的对象。因为对 window 对象属性的引用可以忽略 window 对象，所以即使使用不带 window 的引用语法，也可以确保所有脚本浏览器的兼容性(至少对于所有浏览器都支持的 navigator 对象属性是这样的)，这是引用 navigator 对象的推荐方法。

示例

本书中有许多使用 navigator 对象的示例，主要用于浏览器检测。navigator 对象属性的示例可参见配书光盘上的第 42 章。

相关主题：navigator 对象。

更多资源请访问稀酷客(www.ckook.com)

netscape

值：对象，　　只读

兼容性：WinIE-，MacIE-，NN3+，Moz+，Safari-，Opera-，Chrome-

顾名思义，用户可能认为 netscape 属性会与 navigator 对象一起使用，其实并非如此。netscape 属性是 NN/Moz 浏览器特有的，它允许访问 Netscape 浏览器系列的特有功能，如权限管理器。

示例

netscape 属性通常用于访问 NN/Moz 特有的 PrivilegeManager 对象，以启用或禁用安全权限。下面是进行这种访问的示例：

```
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");
```

offscreenBuffering

值：Boolean 或字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari 1.2+，Opera-，Chrome+

IE4+/Safari 1.2+默认先将页面置于缓冲区(一块内存)，然后将其传输到显示屏。通过修改 window.offscreenBuffering 属性，可以显式控制这种行为。

这个属性的默认值为字符串 auto，也可将该属性设置为 Boolean 值 true 或者 false，以禁止自动处理这种行为。

示例

如果要关闭整个页面的缓存功能，可在脚本语句的开头位置包括以下语句：

```
window.offscreenBuffering = false;
```

onerror

值：函数，　　读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari-，Opera-，Chrome-

本书有一个规则：在对象引用部分，不将事件处理程序描述为属性，但 onerror 属性是一个例外。因为 onerror 事件带来了一些特别的属性，它们用于在脚本中控制事件处理属性的设置。

如果在加载或者处理页面时遇到了脚本错误，现代浏览器(IE5+、NN4+和 W3C)将防止脚本错误干扰用户。对于不了解 JavaScript 的用户来说，任何问题提示(状态栏中的消息或者图标)都令他们感到迷惑。其他人在页面上执行脚本时，JavaScript 允许关闭脚本错误窗口，或者阻止消息的显示。问题是：应何时关闭消息？

脚本错误通常意味着脚本出了问题，可能因为有编码错误，或者 JavaScript 中有错误(可能在不能检测错误的一个浏览器平台版本上)。如果发生这类错误，脚本常常不能继续执行期望的操作。在开发阶段隐藏脚本错误是非常愚蠢的，因为无法知道在代码中是否隐藏着不可见的错误。但对用户关闭错误对话框也是很危险的，因为用户相信页面能正常运作，而其实并非如此，一些数据值可能并没有正确计算或者显示。

如果希望将这些对话框窗口隐藏起来，这里可以列举一些例子。例如，如果知道一个与平

更多资源请访问稀酷客(www.ckook.com)

台相关的错误会打开错误消息窗口，但它不妨碍脚本的执行，就可以在 Web 站点的文件中隐藏该错误警告对话框。只有在经过大量测试后，才应隐藏该对话框，以确保脚本即使有错误，也能正确执行。

注意：

IE 仅为运行时错误触发 onerror 事件处理程序，这意味着即使脚本中存在语法错误，使浏览器在页面加载时出错，也不会触发 onerror 事件，而且不能捕获该错误信息。另外，如果用户安装了 IE 脚本调试器，则任何阻止显示浏览器错误消息的代码都无效。

浏览器启动时，window.onerror 属性是<undefined>。此时，所有错误都通过正常的 JavaScript 错误窗口或者消息显示出来。要隐藏错误警告，需要让 window.onerror 属性调用一个什么都不做的函数：

```
function doNothing() { return true; }
window.onerror = doNothing;
```

为了恢复错误消息的显示，应重载页面。

然而，可以给 window.onerror 属性指定一个自定义函数。在脚本控制下，这个函数将以更友好的方式来处理错误。只要打开了错误消息(默认)，脚本错误(或者 Java applet，或者 exception 类)都会调用指定给 onerror 属性的函数，并传递三个参数：

- 错误消息。
- 导致错误的文档的 URL。
- 错误的行数。

实际上，可以用自己的接口来捕获并处理所有错误(或者根本没有用户通知)。如果不想显示 JavaScript 脚本错误消息，这个函数的最后一个语句必须是 return true。

如果直接在脚本上利用 NPAPI 与 Java applet 通信，就可以使用相同的方式来处理 Java 抛出的异常。Java 异常未必是错误：许多方法假定，Java 代码会捕获异常，来处理特殊情况(例如，签名脚本对话框发出提示时，对拒绝用户的访问做出反应)。关于捕获特定 Java 异常的示例请参见配书光盘中的第 47 章；关于为 W3C DOM 兼容浏览器引入的 JavaScript 异常处理请参见第 21 章。

示例

在程序清单 27-10 中，一个按钮触发了一个包含错误的脚本。该脚本添加了错误处理函数，来处理错误，它会打开另一个窗口，并填写一个 textarea 表单元素(如图 27-4 所示)。该脚本还提供了一个 Submit 按钮，将错误信息发送到支持中心的电子邮件地址，这是一个如何处理脚本中错误的例子。

更多资源请访问稀酷客(www.ckook.com)

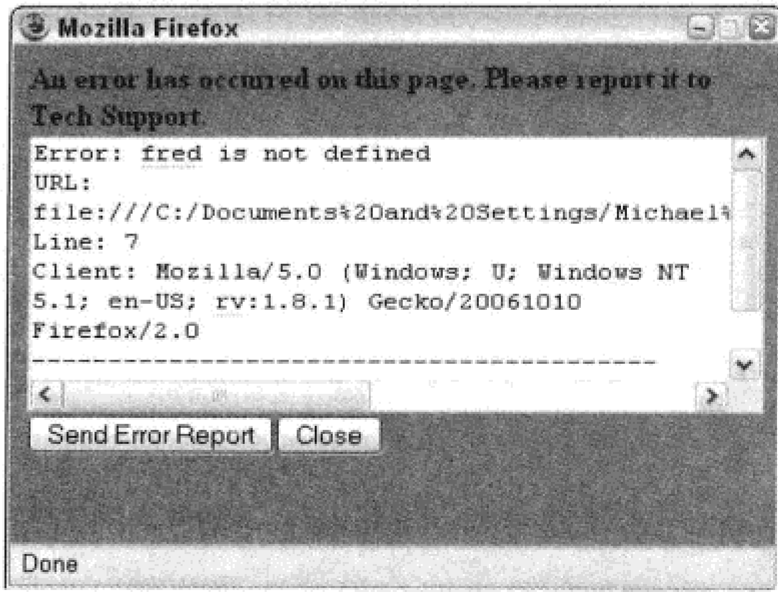


图 27-4　自行报告错误的窗口示例

程序清单 27-10　控制脚本错误

HTML: jsb27-10.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Error Dialog Control</title>
      <script type="text/javascript" src="jsb27-10.js"></script>
   </head>
   <body>
      <h1>Error Dialog Control</h1>
      <hr />
      <form name="myform">
         <input type="button" value="Cause an Error" onclick="goWrong()" />
         <p><input type="button" value="Turn Off Error Dialogs"
                   onclick="errOff()" />
            <input type="button" value="Turn On Error Dialogs"
                   onclick="errOn()" />
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb27-10.js

```
// function with invalid variable value
function goWrong()
{
   var x = fred;
}
// turn off error dialogs
function errOff()
{
   window.onerror = doNothing;
```


更多资源请访问稀酷客(www.ckook.com)

```
}
// turn on error dialogs with hard reload
function errOn()
{
   window.onerror = handleError;
}

// assign default error handler
window.onerror = handleError;

// error handler when errors are turned off...prevents error dialog
function doNothing() { return true; }

function handleError(msg, URL, lineNum)
{
   var errWind = window.open("","errors","height=270,width=400");
   var wintxt = "<html><body bgcolor=red>";
   wintxt += "<b>An error has occurred on this page. "
          + "Please report it to Tech Support.</b>";
   wintxt += "<form method=POST enctype='text/plain' "
          + "action=mailTo:support4@dannyg.com >";
   wintxt += "<textarea name='errMsg'cols=45 rows=8 wrap=VIRTUAL>";
   wintxt += "Error: "
          + msg + "\n";
   wintxt += "URL: "
          + URL + "\n";
   wintxt += "Line: "
          + lineNum + "\n";
   wintxt += "Client: "
          + navigator.userAgent + "\n";
   wintxt += "-----------------------------------------\n";
   wintxt += "Please describe what you were doing when the error occurred:";
   wintxt += "</textarea><br />";
   wintxt += "<input type=SUBMIT value='Send Error Report'>";
   wintxt += "<input type=button value='Close' onclick='self.close()' >";
   wintxt += "</form></body></html>";
   errWind.document.write(wintxt);
   errWind.document.close();
   return true;
}
```

本例提供了一个执行硬重载的按钮，该按钮将 window.onerror 属性重置为其默认值。在关闭错误对话框时，不会运行错误处理函数。

相关主题：location.reload()方法；JavaScript 异常处理(第 21 章)；调试脚本(配书光盘的第 48 章)。

opener

值：Window 对象引用，　　　　读/写

兼容性：WinIE3+，MacIE3+，NN3+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

许多脚本开发人员都误认为，由 window.open()方法创建的新浏览器窗口与原窗口存在父子关系，就像框架与其父框架的关系一样。但其实并非如此，创建新的浏览器窗口后，就仅通过 opener 属性与它的源窗口联系。opener 属性用于为新窗口中的脚本提供一个对源窗口的有效引用。例如，源窗口可能包含一些变量值或者通用函数，其 Web 站点的新窗口希望使用它们。源窗口也可能包括一些表单元素，其设值对新窗口有意义，或者通过新窗口中用户的交互进行设置。

因为 opener 属性的值是一般 window 对象的引用，所以可将该属性名作为引用的开头。或者，可使用更完整的 window.opener 或者 self.opener 引用。但是，该引用必须包括源窗口的某个对象或者属性，例如 window 方法，或者是对窗口文档中某些对象的引用。

如果一个子窗口打开了另一个子窗口，这种链接依然有效，虽然需要再加一步，但第三个窗口可用以 opener.opener 开头的引用到达主窗口。

```
Opener. Opener....
```

加载页面时，第三个窗口最好将 opener.opener 的值存储在全局变量中。接着，如果用户关闭第二个窗口，这个变量可以用来开始一个指向主窗口的引用。

当生成新窗口的脚本在框架内时，子窗口的 opener 属性就指向那个框架。因此，如果子窗口要与主窗口的父窗口或者主窗口的另一个框架通信，则建立一个到该远程对象的引用就必须非常小心。例如，如果子窗口需要得到一个复选框的 checked 属性，而这个复选框在创建该子窗口的框架的兄弟框架中，则引用为：

```
opener.parent.sisterFrameName.document.formName.checkboxName.checked
```

这种引用实际上很长，建立这种引用需要一步步地画出一条从脚本到目的地的路径。

示例

为说明 opener 属性的重要性，看看如何在主窗口的设置中定义新窗口(参见程序清单 27-11)。doNew()函数生成一个小的子窗口，并将程序清单 27-12 中的文件载入该窗口。注意，doNew () 中开头的条件语句可以确保，如果新窗口已经存在，就调用新窗口的 focus ()方法，将它显示在前端。

程序清单 27-11　生成第二个窗口的主窗口文档的内容

HTML: jsb27-11.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Master of all Windows</title>
      <script type="text/javascript" src="jsb27-11.js"></script>
   </head>
   <body>
      <form name="input">
         Select a color for a new window:
         <input type="radio" name="color" value="red" checked="checked" />Red
         <input type="radio" name="color" value="yellow" />Yellow
```


更多资源请访问稀酷客(www.ckook.com)

```
            <input type="radio" name="color" value="blue" />Blue
            <input type="button" name="storage" value="Make a Window" onclick="doNew()"/>
            <hr />
            This field will be filled from an entry in another window:
            <input type="text" name="entry" size="25" />
         </form>
      </body>
   </html>
```

JavaScript: jsb27-11.js

```
var myWind;
function doNew()
{
   if (!myWind || myWind.closed)
   {
      myWind = window.open("jsb27-12.html","subWindow","height=200, ↵
               width=350,resizable");
   }
   else
   {
      // bring existing subwindow to the front
      myWind.focus();
   }
}
```

程序清单 27-12 对 opener 属性的引用

HTML: jsb27-12.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>New Window on the Block</title>
      <script type="text/javascript" src="jsb27-12.js"></script>
      <script type="text/javascript">
         document.write("<body bgcolor='" + getColor() + "'>")
      </script>
   </head>
   <body>
      <form>
         <input type="button" value="Who's in the Main window?"
                onclick="alert(self.opener.document.title)" />
         <p>Type text here for the main window:
            <input type="text" size="25"
                   onchange="self.opener.document.forms[0].entry.value ↵
                   = this.value" />
         </p>
      </form>
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

JavaScript: jsb27-12.js

```
function getColor()
{
   // shorten the reference
   colorButtons = self.opener.document.forms[0].color;
   // see which radio button is checked
   for (var i = 0; i < colorButtons.length; i++)
   {
      if (colorButtons[i].checked)
      {
         return colorButtons[i].value;
      }
   }
   return "white";
}
```

在 getColor()函数中，对单选按钮数组的多个引用可能非常长。为了简化引用，getColor()函数首先将单选按钮数组赋给一个变量 colorButtons。该简写形式现在代表着以前冗长的引用，用于在单选按钮中循环，以确定选中了哪个按钮，并获取其 value 属性。

第二个窗口中的按钮只是获取 opener 窗口文档的标题。即使另一个文档也载入主窗口，opener 引用仍然指向主窗口，不过其 document 对象改变了。

最后，第二个窗口包含了一个文本输入对象。在其中输入想要的文本，然后按 Tab 键或在文本域外单击，onchange 事件处理程序就会更新 opener 文档中的域(假设仍然载入了文档)。

相关主题：window.open()，window.focus()方法。

outerHeight，outerWidth

参见本章前面的 innerHeight 和 innerWidth。

pageXOffset，pageYOffset

值：整数， 只读

兼容性：WinIE-，MacIE-，NN4+，Moz+，Safari+，Opera+，Chrome+

对于滚动文档来说，浏览器窗口的内容区域(内部的)左上角是一个重要位置。当文档滚动到窗口左上角时(或者文档足够小，浏览器窗口不需要滚动条)，文档的位置是(0, 0)，这意味着文档与浏览器窗口顶部和左边的距离都是 0 像素。如果要滚动文档，文档的其他一些坐标点将在左上角的下方。这种度量方法叫做页面偏移，pageXOffset 和 pageYOffset 属性表示文档相对于内部窗口左上角的像素值：pageXOffset 是横向偏移，pageYOffset 是垂直偏移。

如果在页面上设计导航按钮，来精确控制显示在窗口中的页面内容，这个度量方法的价值就明显地体现出来了。例如，假如有一个双框架页面，其中一个框架包含导航控件，而另一个框架显示主要内容。为了美观，在显示框架中关闭滚动条，使用导航控件来代替。与滚动按钮关联的脚本能确定文档的 pageYOffset 值，然后用 window.scrollTo()方法把文档精确定位到文档中的下一个逻辑分区，以进行显示。

IE 把这些值处理为 body 对象的属性：body.scrollLeft 和 body.scrollTop(见第 29 章)。

更多资源请访问稀酷客(www.ckook.com)

相关主题：window.innerHeight、window.innerWidth、body.scrollLeft、body.scrollTop 属性；window.scrollBy()、window.scrollTo()方法。

parent

值：Window 对象引用，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

parent 属性(和本节后面的 top 属性)主要用于将文档显示为多框架窗口的一部分。用户在多框架浏览器窗口的框架中见到的 HTML 文档与为整个窗口指定框架集的文档有区别。尽管后者也在浏览器的内存中(在浏览器的位置区域中显示为 URL)，但它对用户(除了在 Source View 中)是不可见的。

如果可见文档中的脚本需要引用框架集窗口的对象或者属性，则可以利用 parent 属性引用这些框架集窗口中的项。然而，不要在该引用之前加上 window 对象，例如 window.parent.propertyName，因为这在早期的浏览器中会出问题。简言之，parent 属性似乎违反了对象层次结构，因为对于单框架文档，此属性指向一个更高层次。如果没有指定 parent 属性，但在一个框架文档中指定了 self 属性，对象引用就仅指向该框架，而不是最外部的框架集 window 对象。

使用 parent 对象存储临时变量是一种完全合法的非传统方式。因此，可为单个变量值或者一组数据建立一个存储区域。此时，这些值就可以由加载到框架中的所有文档共享，包括框架中已改变的文档。然而，在 parent 中动态存储数据时(以响应用户在框架中的动作)，要特别小心。在早期浏览器中，如果用户重设窗口的尺寸，变量就恢复其默认值(即其父脚本设置的值)。

子窗口也能调用在父窗口中定义的函数，这种函数的引用是：

```
parent.functionName([parameters])
```

乍一看，好像 parent 和 top 属性指向同一个框架集 window 对象。在由一个框架集窗口和其直接子窗口组成的环境中，情况确实如此。但如果其中一个子窗口本身是另一个框架集窗口，这就是三代窗口环境。对于最“年轻”的子窗口而言(例如由第二个框架集定义的窗口)，parent 属性指向其直接的父窗口，而 top 属性指向这个链中的第一个框架集窗口。

另一方面，通过 window.open()方法创建的新窗口与源窗口之间不存在父子关系，新窗口的 top 和 parent 指向新窗口自身。这些关系可参见本章前面的 27.2 一节。

示例

为了演示各种 window 对象属性如何引用多框架环境中的窗口，用浏览器载入程序清单 27-13 中的文档。该文档又将两个相同大小的框架设置为同一文档：程序清单 27-14。此文档提取几个窗口属性的值，以及两个不同窗口引用的 document.title 属性。

程序清单 27-13　程序清单 27-14 的框架集文本

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>The Parent Property Example</title>
      <script type="text/javascript">
```


更多资源请访问稀酷客(www.ckook.com)

```
            self.name = "Framesetter";
        </script>
    </head>
    <frameset cols="50%,50%" onunload="self.name = ''">
        <frame name="JustAKid1" src="jsb27-14.html" />
        <frame name="JustAKid2" src="jsb27-14.html" />
    </frameset>
</html>
```

程序清单 27-14　显示各种与窗口相关的属性

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Window Revealer II</title>
        <script type="text/javascript">
            function gatherWindowData()
            {
                var msg = "";
                msg += "top name: "
                    + top.name + "<br />";
                msg += "parent name: "
                    + parent.name + "<br />";
                msg += "parent.document.title: "
                    + parent.document.title + "<br />";
                msg += "window name: "
                    + window.name + "<br />";
                msg += "self name: "
                    + self.name + "<br />";
                msg += "self.document.title: "
                    + self.document.title;
                return msg;
            }
        </script>
    </head>
    <body>
        <script type="text/javascript">
            document.write(gatherWindowData());
        </script>
    </body>
</html>
```

在这两个框架中，对 window 和 self 对象名的引用返回由框架集分配给框架的名称(左侧的框架为 JustAKidl，右侧的框架为 JustAKid2)。换句话说，对于每个框架而言，window 对象都是框架本身。对 self.document.title 的引用仅表示载入该窗口框架的文档。但对顶层和父窗口的引用(在此示例中是同一个窗口)，表明这些对象属性在两个框架之间共享。

还有两个要点。首先，框架集窗口的名称在程序清单 27-13 载入时设置，而不是响应<frameset>标记中的 onload 事件处理程序，因为名称只有在文档载入框架时设置，才能获得该

更多资源请访问稀酷客(www.ckook.com)

值。如果在框架集的 onload 事件处理程序中设置，则直到框架文档载入后，才设置名称；其次，在卸载框架集文档时，将父窗口的名称恢复为空字符串，以防将来的页面不理解窗口名。

相关主题： window.frames、window.self、window.top 方法。

personalbar

详见 directories。

returnValue

值： 任何数据类型，　　　　读/写

兼容性： WinIE4+，MacIE5+，NN-，Moz+，Safari+，Opera-，Chrome-

对于在 IE 特定的模态对话框中载入的文档，脚本可以使用 returnValue 属性。模态对话框由 showModalDialog()方法生成，它在关闭之前，其返回的数据赋给对话框窗口的 returnValue 属性，这是因为当模态对话框可见时，主窗口中的脚本停止执行。当对话框关闭时，可以把一个值返回到主窗口中调用模态对话框的脚本代码，然后主窗口的脚本继续执行下面的语句。

示例

参见用于 showModalDialog()方法的程序清单 27-36，从中了解如何获得对话框中的数据。

相关主题： showModalDialog()方法。

screen

值： screen 对象，　　　　只读

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

尽管 screen 对象在现代浏览器中是 window 对象的一个属性，但它在 NN4 中可用作单独的对象(见配书光盘中的第 42 章)。因为对于 window 对象的任何属性引用，都可以忽略 window 对象，所以可以使用不带 window 对象的引用语法，以便与支持 screen 对象的旧浏览器兼容。

示例

参见配书光盘中的第 42 章，学习如何用 screen 对象给运行浏览器的计算机确定显示器的特性。

相关主题： screen 对象。

screenLeft，screenTop

值： 整型，　　　　只读

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari 1.2+，Opera+，Chrome+

WinIE5+首次提供了 window 对象的 screenLeft 和 screenTop 属性，来确定浏览器窗口中 Microsoft 称为客户区的像素位置(相对于显示器左上角 0, 0 坐标点)。客户区不包括大多数窗口窗框元素，例如标题栏、地址栏和窗口标尺条。因此，当浏览器窗口最大化时(表示没有尺寸条)，窗口的 screenTop 属性随用户选择显示的工具栏而变化，而 screenLeft 属性始终为 0。对于非最大化窗口，如果窗口已定位，客户区的顶部或者左部未显示出来，这些属性值就是负整数。

这两个属性是只读的，可以通过 window.moveTo()和 window.moveBy()方法定位浏览器窗

更多资源请访问稀酷客(www.ckook.com)

口，但这些方法只定位整个浏览器窗口的左上角，不定位客户区。IE 7 及更早的版本都没有提供定位整个浏览器窗口的属性。

示例

使用 The Evaluator(第 4 章)试验 screenLeft 和 screenTop 属性。首先最大化浏览器窗口(如果使用 Windows)，再在顶部文本框中输入以下属性名：

```
window.screenLeft
```

单击 Evaluate 按钮，查看当前设置。取消窗口的最大化，并在屏幕上拖动它。每次结束拖动时，再次单击 Evaluate 按钮，查看当前值。对 window.screenTop 执行相同的操作。

相关主题：window.moveTo()方法，window.moveBy()方法。

screenX，screenY

值：整型， 读/写

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari 1.2+，Opera+，Chrome+

NN6+/Moz/W3C 的 screenX 和 screenY 属性表示浏览器窗口的外部边界相对于显示器左上角(0,0)的位置。浏览器窗口还包括 Win32 窗口周围 4 像素宽的窗口调整条。因此，在最大化 WinNN6+浏览器窗口时，screenX 和 screenY 的值为-4。NN/Moz/W3C 没有为浏览器窗口客户区提供与 IE5+中 screenLeft 和 screenTop 属性等价的属性。然而，可以确定各个工具栏在浏览器窗口中是否可见(参见 window.directories)。

尽管可为这两个属性赋值，但如果用户设置了首选项，阻止窗口移动和调整大小，则支持这两个属性的浏览器当前版本不会相应地调整窗口的位置。许多 Web 冲浪者认为，不应通过脚本移动窗口以及调整窗口的大小。

示例

使用 The Evaluator(第 4 章)试验 screenX 和 screenY 属性。首先最大化浏览器窗口(如果使用 Windows)，再在顶部文本框中输入以下属性名：

```
window.screenY
```

单击 Evaluate 按钮，查看当前设置。取消窗口的最大化，并在屏幕上拖动它，每次结束拖动时，再次单击 Evaluate 按钮，查看当前值。对 window.screenY 执行相同的操作。

相关主题：window.moveTo()方法，window.moveBy()方法。

scrollbars

参见 directories。

scrollMaxX，scrollMaxY

值：整型， 只读

兼容性：WinIE-，MacIE-，NN7.1+，Moz1.4+，Safari-，Opera-，Chrome-

NN7.1+/Moz1.4+的 scrollMaxX 和 scrollMaxY 属性可确定窗口的最大水平和垂直滚动范围。

更多资源请访问稀酷客(www.ckook.com)

只有窗口在所需方向上显示了滚动条，才能进行滚动。这两个属性的值是表示像素的整数。

相关主题：scrollX 属性，scrollY 属性。

scrollX，scrollY

值：整型， 只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

NN6+/Mozilla/Safari 的 scrollX 和 scrollY 属性可确定窗口的垂直和水平滚动。只有窗口在所需方向上显示了滚动条，才能进行滚动。这两个属性的值是表示像素的整数。

IE DOM 不提供类似的窗口属性，但可从 body.scrollLeft 和 body.scrollTop 属性中得到相同的信息。

示例

使用 The Evaluator(第 4 章)试验 scrollX 和 scrollY 属性。在顶部文本框中输入以下属性：

```
window.screenY
```

现在手动向下滚动页面，以看到 Evaluate 按钮。单击该按钮，查看窗口沿 Y 轴滚动了多远。

相关主题：body.scrollLeft 属性，body.scrollTop 属性。

self

值：Window 对象引用， 只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

window 对象引用是可选的，而当对象引用指向的窗口与包含该引用的窗口相同时，self 属性也是可选的。该结构似乎不寻常，self 属性表示与 window 相同的对象。例如，要获得一个单框架窗口中文档的标题，可使用以下三种结构：

```
window.document.title
self.document.title
document.title
```

尽管 self 是 window 对象的属性之一，但在单框架窗口脚本中，不应组合使用这两种引用(例如，引用不要以 window.self 开头，这会产生大量的脚本问题)。虽然对于单框架窗口来说，self 属性是可选的，但它有助于阅读代码的人搞清对象引用。在多框架窗口中，要特别注意这个属性。

JavaScript 可采用非常智能化的方式处理窗口引用，因此，对于同一个窗口的文档元素引用，self 部分基本可以忽略。但在多框架窗口中显示文档时，对象的完整引用(包括 self 前缀)会使阅读或者调试代码的用户更容易跟踪哪些人在执行哪些操作。可以访问任何窗口的 self 属性，其返回值是 window 对象的引用。

示例

程序清单 27-15 使用与程序清单 27-5 相同的操作，但用 self 属性代替所有的 window 对象引用。使用此引用完全是可选的，但如果 HTML 文档要显示在多框架窗口的某个框架中，尤其是此文档中的 JavaScript 代码引用其他框架中的文档时，self 属性有助于阅读和调试脚本。self

更多资源请访问稀酷客(www.ckook.com)

引用有助于阅读代码的人知道当前正在访问哪个框架。测试时要注意，即使现代浏览器支持 status 属性，其中一些浏览器也不会显示用脚本编写的、与链接相关的状态栏文本，以防止链接诈骗。

程序清单 27-15　使用 self 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>self Property</title>
      <script type="text/javascript">
         self.defaultStatus = "Welcome to my Website.";
      </script>
   </head>
   <body>
      <h1>self Property</h1>
      <hr />
      <p><a href="http://www.microsoft.com"
            onmouseover="self.status = 'Visit Microsoft\'s Home page. ';return true;"
            onmouseout="self.status = '';return true;">Microsoft</a>
      </p>
      <p><a href="http://mozilla.org"
            onmouseover="self.status = 'Visit Mozilla\'s Home page. ';return true;"
            onmouseout="self.status = self.defaultStatus;
            return true;">Mozilla</a></p>
   </body>
</html>
```

相关主题：window.frames，window.parent，window.top 属性。

sidebar

请参见 appCore。

status

值：字符串，　　　　读写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome-

浏览器窗口的底部是状态栏。状态栏包括一个区域，它一般显示文档加载过程或者某一时刻鼠标指向的链接的 URL。将一个文本字符串赋给 window 对象的 status 属性，可指定该区域的临时内容。最好只在响应具有临时效果的事件时，才调整 status 属性，例如链接或者图像地图区域对象的 onmouseover 事件处理程序。在这种情况下设置 status 属性，会覆盖状态栏上的其他值。如果用户从改变状态栏的对象上移开鼠标，状态栏就恢复其默认值(在一些页面中可能为空值)。然而，为了防止链接诈骗，并非所有现代浏览器都显示用脚本编写的、与链接相关的状态栏文本。

用户在页面上移动鼠标时，这个窗口属性可以作为显示链接的 URL 的更友好方式。例如，

更多资源请访问稀酷客(www.ckook.com)

若用状态栏来说明链接目标的特性，只需将该文本加入状态栏，以响应 onmouseover 事件处理程序。但注意，富有经验的 Web 冲浪者希望看到 URL，因此，可考虑为状态栏创建一个混合消息，其中包括 URL，在其后的括号中填上友好的描述信息。在多框架环境中，可以设置 window.status 属性，而不用考虑各个框架的引用。

示例

在程序清单 27-16 中，status 属性在一个事件处理程序中设置，该处理程序内嵌在两个 HTML 链接标记的 onmouseover 特性中。注意，该处理程序需要把 return true(或值为 true 的表达式)作为其最后一条语句。此语句是必需的，否则在所有浏览器中都不会显示状态消息。在测试时要注意，即使现代浏览器支持 status 属性，其中一些浏览器也不会显示用脚本编写的、与链接相关的状态栏文本，以防止链接诈骗。

程序清单 27-16　带有自定义状态栏消息的链接

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.status Property</title>
   </head>
   <body>
      <h1>window.status Property</h1>
      <hr />
      <a href="http://www.dannyg.com"
         onmouseover="window.status = 'Go to my Home page.(www.dannyg.com) '; ↵
         return true">
         Home</a>
      <p>
         <a href = "http://mozilla.org"
            onmouseover="window.status = 'Visit Mozilla Home page.(mozilla.org) '; ↵
            return true">
         Mozilla</a>
      </p>
   </body>
</html>
```

为避免出现某些平台特定的异常，从而影响 onmouseover 事件处理程序和 window. status 属性的行为，还应该为链接和客户端图像地图区域对象包括一个 onmouseout 事件处理程序。这类 onmouseout 事件处理程序应将 status 属性设置为空字符串，确保在鼠标指针从这些对象上移开时，状态栏消息恢复 defaultStatus 设置。如果要编写一个处理所有窗口状态变化的统一函数，可以这样做，但要谨慎使用 onmouseover 特性，以使事件处理程序返回 true。程序清单 27-17 给出了这样一个示例。测试时要注意，一些现代浏览器不会显示用脚本编写的、与链接相关的状态栏文本，以防止链接诈骗。

更多资源请访问稀酷客(www.ckook.com)

程序清单 27-17　处理状态消息的变化

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Generalizable window.status Property</title>
      <script type="text/javascript">
         function showStatus(msg)
         {
            window.status = msg;
            return true;
         }
      </script>
   </head>
   <body>
      <h1>Generalizable window.status Property</h1>
      <hr />
      <a href="http://www.dannyg.com"
         onmouseover="return showStatus('Go to my Home page. ')"
         onmouseout="return showStatus('')">Home</a>
      <p>
         <a href="http://mozilla.org"
            onmouseover="return showStatus('Visit Mozilla Home page. ')"
            onmouseout="return showStatus('')">Mozilla</a></p>
   </body>
</html>
```

注意该事件处理程序将 showStatus()方法的结果返回给事件处理程序，使该处理程序返回 true。

设置状态栏的最后一个示例(如程序清单 27-18 所示)还演示了如何在状态栏中创建一个简单的滚动横幅。

程序清单 27-18　创建滚动横幅

HTML：jsb27-18.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Message Scroller</title>
      <script type="text/javascript" src="jsb27-18.js"></script>
   </head>
   <body onload="scrollMsg()">
      <h1>Message Scroller</h1>
      <hr />
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

JavaScript: jsb27-18.js

```
var msg = "Welcome to my world...";
var delay = 150;
var timerId;
var maxCount = 0;
var currCount = 1;

function scrollMsg()
{
   // set the number of times scrolling message is to run
   if (maxCount == 0)
   {
      maxCount = 3 * msg.length;
   }
   window.status = msg;

   // keep track of how many characters have scrolled
   currCount++;

   // shift first character of msg to end of msg
   msg = msg.substring (1, msg.length) + msg.substring (0, 1);

   // test whether we've reached maximum character count
   if (currCount >= maxCount)
   {
      timerID = 0; // zero out the timer
      window.status = ""; // clear the status bar
      return; // break out of function
   }
   else
   {
      // recursive call to this function
      timerId = setTimeout("scrollMsg()", delay);
   }
}
```

由于状态栏由独立的函数(而不是 onmouseover 事件处理程序)设置，因此不必添加 return true 语句，来设置 status 属性。scrollMsg()函数使用更先进的 JavaScript 概念，如 window. setTimeout()方法(本章稍后介绍)和字符串方法(在第 15 章中介绍)。要加快文字在状态栏上的滚动速度，可减小 delay 的值。

许多 Web 冲浪者不关心这些永远在状态栏上滚动的消息。鼠标移过链接会干扰横幅的显示。少用滚动消息，或将其设计为仅在文档载入后运行数次。

提示：

在 Navigator 的各个版本中，用 onmouseover 事件处理程序设置 status 属性走过了曲折的道路。浏览器使用状态栏来报告载入进程，而设置状态栏的脚本总是与浏览器本身竞争。当页面上的热区位于框架边缘时，onmouseout 事件经常不能触发，从而使状态栏不能清除其中的信息。在宣布页面可供公众访问之前，一定要严格测试此类实现方案。

更多资源请访问稀酷客(www.ckook.com)

相关主题：window.defaultStatus 属性；onmouseover，onmouseout 事件处理程序；link 对象。

statusbar，toolbar

参见 locationbar 部分。

top

值：Window 对象引用，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

window 对象的 top 属性指向框架集对象层次结构中的最上层窗口。对于单框架窗口而言，该引用指向 window 对象本身(包括 self 和 parent 属性)，因此可以不将 window 作为引用的一部分。在多框架窗口中，top 窗口是定义第一个框架集的窗口(在嵌套的框架集中)。用户在多框架集环境下不会看到 top 窗口，但浏览器在内存中将其存储为对象。原因是 top 窗口拥有指向其他框架的路径图(如果一个框架需要另一个框架中的对象引用)，且其子框架可以调用它。引用如下：

```
top.functionName([parameters])
```

要了解 top 和 parent 属性之间的区别，可参见本章开头关于编写框架的深入讨论，还可参见 parent 属性的例子，其程序清单演示了 top 属性的值。

相关主题：window.frames，window.self，window.parent 属性。

window

值：Window 对象，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

将 window 属性作为单独的属性会使人更迷惑，而不是有所帮助。window 属性与 window 对象是同一对象，所以在使用引用时不必以 window.window 开头。尽管在许多引用中都忽略了 window 对象，但在脚本语句需要引用同一窗口或者框架中的项时，可将 window 作为引用的一部分；在引用更高层次(top 或者 parent)的项时，不要将 window 用作引用的一部分。

27.3.4　方法

alert("*message*")

返回值：无。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

警告对话框是一个模态对话框，它给用户提供了一条消息和一个用于关闭对话框的 OK 按钮。只要警告对话框显示出来，其他应用程序或者窗口就都处于非活动状态。因此用户只有首先关闭对话框，才能在浏览器中执行其他工作。

alert()方法的唯一参数可以是任何数据类型的值，包括一些不常用的数据类型，在 JavaScript 中通常不会用到这些值(例如完全对象)。所以警告对话框是调试 JavaScript 脚本的一个方便工具。只要想监视表达式的值，就可以在代码中将表达式作为临时 alert()方法的参数。脚本执行到该方法时，会将停下来显示表达式的值。有关调试脚本的更多提示信息，请参见配书光盘中

更多资源请访问稀酷客(www.ckook.com)

的第 48 章。

经常困扰应用程序设计者的一个问题是：由 JavaScript 创建的所有模态对话框(通过 alert()、confirm ()和 prompt()方法创建)都把自己看成由 JavaScript 或浏览器生成。这种身份认同是一种安全措施，它防止不讲道德的脚本开发人员欺骗系统或者浏览器的警告对话框，窃取用户的密码或者其他私人信息，脚本不能覆盖或者删除这些标识词语。也可以在跨浏览器方式中使用常规浏览器窗口来模拟模态对话框，但它不像真正的模态窗口那样可靠，该窗口可在 IE4+和其他一些现代浏览器中通过 window.showModalDialog()方法创建。

因为 alert()方法是全局的(也就是说，在多框架环境下，任何框架都不能从警告对话框中获益)，所以通常的做法是在调用该方法的语句中忽略所有的 window 对象引用，同时在 HTML 文档和站点设计中限制使用警告对话框。该模态窗口将中断用户在页面上的导航，此时要与用户通信，可以通过表单，也可以在独立的文档窗口框架中写入文本。当然，警告框仍然是非常便捷的快速调试手段。

示例

在程序清单 27-19 中，示例的参数是一个连接字符串，它连接两个固定字符串和浏览器的 navigator.appName 属性值。载入此文档将显示警告对话框，如图 27-5 中的几个配置所示。在旧浏览器中，不能从对话框中删除 JavaScript Alert:代码行，在稍后的浏览器中也不能更改标题栏。

程序清单 27-19　显示警告对话框

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.alert() Method</title>
   </head>
   <body>
      <h1>window.alert() Method</h1>
      <hr />
      <script type="text/javascript">
         alert("You are running the " + navigator.appName + " browser.");
      </script>
   </body>
</html>
```

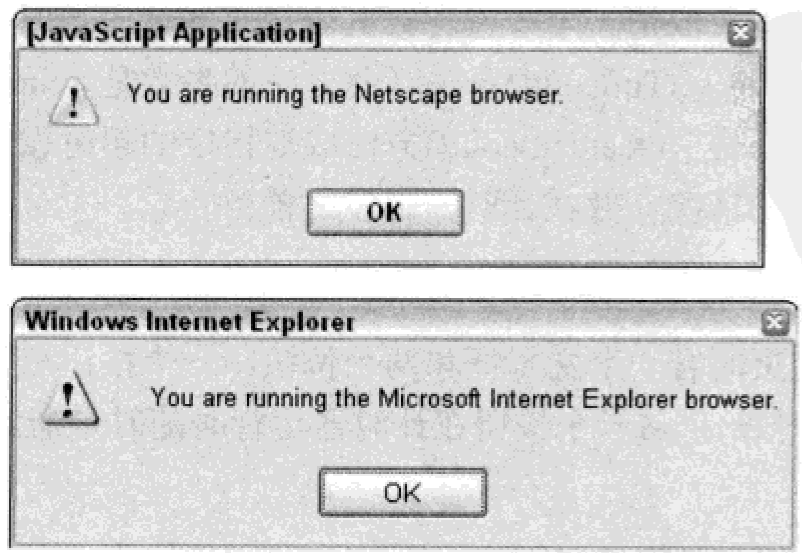


图 27-5　程序清单 27-19 中的 alert ()方法在 Firefox 和 Internet Explorer 中的结果

更多资源请访问稀酷客(www.ckook.com)

相关主题：window.confirm()，window.prompt()方法。

back()，forward()

返回值：无。

兼容性：WinIE+，MacIE-，NN4+，Moz+，Safari+，Opera+，Chrome+

window.back()和 window. forward()方法自从 NN4 开始就有了，它们提供全局前进和后退导航按钮的脚本化版本，并允许 history 对象在特定的窗口或者框架中严格控制导航。即使 IE 现在支持这些窗口方法，它们也没有流行起来(window 对象不在 W3C DOM Level 2 范围之内)，因此，查看浏览器的历史记录时，最好使用 history 对象的方法。版本兼容以及前后导航的内容请参见第 28 章。

相关主题：history.back()，history.forward()，history.go()方法。

clearInterval(*intervalIDnumber*)

返回值：无

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

可以使用 window.clearInterval()方法关闭用 window. setInterval()方法启动的间隔循环动作。其参数是 setInterval()方法返回的 ID 号。JavaScript 间隔机制常常应用于页面上对象的动画。如果有多个间隔操作，每个操作都在内存中有自己的 ID 值，就可以通过 ID 值关闭间隔操作。只要间隔循环停止，脚本就不能恢复这个间隔操作，而必须启动一个新的间隔操作，生成一个新的 ID 值。

示例

参见本章后面的程序清单 27-33 和程序清单 27-34，查看 setInterval()和 clearInterval()如何在页面中组合使用。

相关主题：window. setInterval()方法，window.setTimeout()方法，window.clearTimeout()方法。

clearTimeout(*timeoutIDnumber*)

返回值：无。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

如本章稍后所述，脚本要取消一个等待运行其表达式的计时器时，可以组合使用 window. clearTimeout()方法和 window.setTimeout()方法。此方法的参数是 window.setTimeout()方法在计时器开始计时时返回的 ID 号。clearTimeout()方法取消指定的超时设置。最好检查代码中用户动作不再需要运行计时器的情况，并在计时器触发之前停止它。

示例

程序清单 27-20 中的页面有一个文本框和两个按钮。一个按钮启动一个持续 1 分钟的倒计时器(很容易修改为其他时长)；另一个按钮在计时器运行时随时中断它。若过了 1 分钟，警告对话框就提醒用户。

更多资源请访问稀酷客(www.ckook.com)

程序清单 27-20　倒计时器

HTML: jsb27-20.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Count Down Timer</title>
      <script type="text/javascript" src="jsb27-20.js"></script>
   </head>
   <body>
      <form>
         <input type="button" name="startTime" value="Start 1 min. Timer"
                onclick="startTimer()" />
         <input type="button" name="clearTime"
                value="Clear Timer" onclick="stopTimer()" />
         <p><input type="text" name="timerDisplay" value="" /></p>
      </form>
   </body>
</html>
```

JavaScript: jsb27-20.js

```
var running = false;
var endTime = null;
var timerID = null;

function startTimer()
{
   running = true;
   now = new Date();
   now = now.getTime();
   // change last multiple for the number of minutes
   endTime = now + (1000 * 60 * 1);
   showCountDown();
}

function showCountDown()
{
   var now = new Date();
   now = now.getTime();
      if (endTime - now <= 0)
      {
         stopTimer();
         alert("Time is up. Put down your pencils.");
      }
      else
      {
         var delta = new Date(endTime - now);
         var theMin = delta.getMinutes();
         var theSec = delta.getSeconds();
```


更多资源请访问稀酷客(www.ckook.com)

```
            var theTime = theMin;
            theTime += ((theSec < 10) ? ":0" : ":") + theSec;
            document.forms[0].timerDisplay.value = theTime;
            if (running)
            {
               timerID = setTimeout("showCountDown()",1000);
            }
         }
   }

   function stopTimer()
   {
      clearTimeout(timerID);
      running = false;
      document.forms[0].timerDisplay.value = "0:00";
   }
```

注意，脚本在窗口中建立了三个全局变量：running、endTime 和 timerID。有多个函数需要这些值，因此在函数外部对它们进行初始化。

在 startTimer()函数中打开了 running 标志，这意味着计时器开始计时。使用一些日期函数(参见第 17 章)提取毫秒格式的当前时间，并加上下一分钟的毫秒数(额外乘 1，是为了将计时时长改为所需的分钟数)。将结束时间存储在全局变量中后，该函数就调用另一个函数，来比较当前时间和结束时间，并将时差显示在文本框中。

在 showCountDown()函数中，先检查计时器是否已到时间。如果是，就停止计时器，并提醒用户；否则，函数继续计算两个时间之差，并格式化为 mm:ss 形式。只要 running 标志设置为 true，函数就设置 1s 的超时计时器，再重复执行。要在计时器计时完成前停止它(在 stopTimer()函数中)，最重要的步骤是在浏览器中取消超时计时器的运行。clearTimeout()方法使用全局变量 timerID 的值来完成此操作，然后函数关闭 running 标志，并清零。

运行计时器时，时间可能偶尔会跳过 1s。这不是作弊，因为等待超时再显示下一秒的计算结果需要略多于 1s 的时间，用户看到的是当前时间和结束时间之间的真实时差。

相关主题：window. setTimeout()方法。

close()

返回值：无。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

window.close()方法可以关闭 window 对象指向的浏览器窗口。这个方法最常用于关闭主文档窗口创建的子窗口。如果关闭窗口的命令来自于窗口而不是新的子窗口，原始的 window 对象就必须包含一个子窗口对象的记录。为此，可将 window.open()方法返回的值存储在一个全局变量(例如，不在函数中初始化的变量)中，这个全局变量可由其他对象使用。另一方面，如果新子窗口中的对象调用 window.close()方法，使用 window 或者 self 引用就足够了。

务必要将 window 作为该方法引用的一部分，否则，JavaScript 会把该语句看作 document.close ()方法，从而得到不同的结果(见第 29 章)。只有 document.close()方法才能通过脚本关闭窗口。当然，如果关闭一个窗口，就会触发 onunload 事件处理程序，之后窗口从画面中消失；但在启动

更多资源请访问稀酷客(www.ckook.com)

window.close()方法后，就不能阻止它完成其任务。而且，如果 onunload 事件处理程序试图执行耗时的过程(例如使用正在关闭的窗口提交表单)，它就可能完不成任务，因为窗口可在处理完成前关闭，这个问题无法解决(但在 IE4+中，可以利用 onbeforeunload 事件处理程序来解决该问题)。

在“关闭窗口”这个主题中，存在一种特殊情况：子窗口试图(使用 self.opener.close()语句)关闭主窗口，而主窗口在其会话过程中有多个入口。为了防止脚本关闭一个没有创建的窗口，现代浏览器会询问用户是否希望关闭主窗口(通过浏览器创建的对话框)。不可能绕过这种安全措施，但在 NN4+/Moz 中可以通过签名脚本覆盖它。当然，这必须得到用户授予的浏览器控制许可。参见配书光盘中的第 49 章。

示例

参见程序清单 27-4(用于 window.closed 属性)，其中提供了在多个窗口上应用 window.close()方法的跨平台示例。

相关主题：window.open()，document.close()方法。

confirm(“message”)

返回值：Boolean

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

确认对话框可在一个模态对话框中提供消息，它带有 OK 和 Cancel 按钮。这个对话框可用于向用户提问，通常在脚本执行不能撤消的操作之前询问用户。在进行一般的 Web 页面导航之前，对用户进行提问，以响应用户在表单元素上的交互，通常会浪费用户不少时间，也会扰乱他们的注意力。但对于可能泄露用户身份的操作，或者将表单数据传递到服务器的操作，JavaScript 确认对话框就非常重要。用户也可能意外单击按钮，因此在执行操作之前应提供撤消操作的途径。

这个对话框返回一个 Boolean 值(OK=true; Cancel=false)，所以可以把这个方法用于比较表达式或者赋值表达式。在比较表达式中，可以把此方法嵌入需要 Boolean 值的其他任何语句中。例如：

```
if (confirm("Are you sure?"))
{
   alert("OK");
}
else
{
   alert("Not OK");
}
```

其中，确认对话框的返回值给 if ... else 结构提供了它需要的 Boolean 值类型(详见第 21 章)。这个方法也可以位于赋值表达式的右边，如下：

```
var adult = confirm("Do you certify that you are over 18 years old?");
if (adult)
{
   //statements for adults
}
```


更多资源请访问稀酷客(www.ckook.com)

```
else
{
   //statements for children
}
```

JavaScript 确认对话框窗口中的两个按钮不能使用其他警告图标或标签。

程序清单 27-21 给出的用户界面部分表示，在清除用户输入的数据表格前，通过确认对话框询问用户。在图 27-6 中，不能从对话框中删除标题栏中的文本。

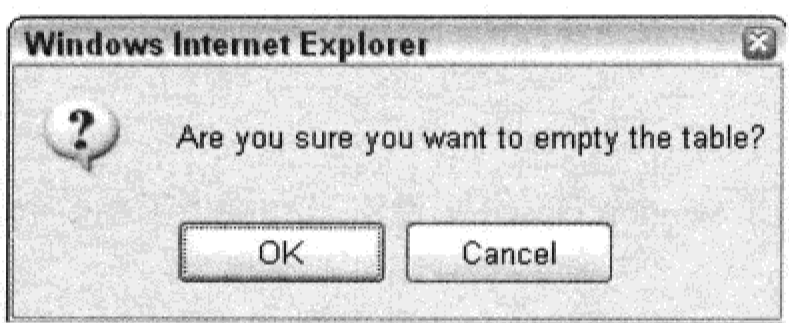


图 27-6　Internet Explorer 中的 JavaScript 确认对话框

程序清单 27-21　确认对话框

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.confirm() Method</title>
      <script type="text/javascript">
         function clearTable()
         {
            if (confirm("Are you sure you want to empty the table?"))
            {
               alert("Emptying the table..."); // for demo purposes
               //statements that actually empty the fields
            }
         }
      </script>
   </head>
   <body>
      <h1>window.confirm() Method</h1>
      <hr />
      <form>
         <h3>A large table with data would be here.</h3>
         <input type="button" name="clear" value="Reset Table"
                onclick="clearTable()" />
      </form>
   </body>
</html>
```

相关主题：window.alert()，window.prompt()，form.submit()方法。

更多资源请访问稀酷客(www.ckook.com)

createPopup()

返回值：Pop-up 对象引用。

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE 的弹出式窗口是一种与当前窗口重叠的无窗框矩形空间。与 showModalDialog()和 showModeless-Dialog()方法产生的对话框不同，弹出式窗口的所有内容必须由脚本明确控制，该窗口的尺寸和位置也是如此。通过 createPopup()方法产生的窗口仅在内存中创建对象，但不显示对象。接着可以用该方法返回的弹出式窗口的引用来定位窗口，填充其内容，并使其可见。详见本章后面的 popup 对象。

示例

有关 createPopup()方法的例子，请参见本章后面的程序清单 27-46。

相关主题：popup 对象。

dump("*message*")

返回值：无。

兼容性：WinIE-，MacIE-，NN7.1+，Moz1.4+，Safari-，Opera-，Chrome-

window.dump()是一个调试/诊断方法，可用于向标准输出(standard output)输出文本字符串，标准输出通常是操作系统的控制台窗口。与通过 alert()方法显示调试消息相比，dump()方法是一种干扰更少的方式。

execScript("*exprList* "[, *language*])

返回值：无。

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 特定的 window.execScript()方法执行一个或者多个传递为字符串表达式的脚本语句。该方法的第一个参数是一个或者多个脚本语句的字符串版本(多个语句必须用分号分开)；第二个可选参数是浏览器用来执行脚本语句的语言解释器。可接受的语言是 JavaScript、JScript、VBS 和 VBScript，其默认值是 JScript，因此提供 JavaScript 表达式时，可忽略第二个参数。

与 JavaScript 的核心语言 eval()函数(该函数也执行 JavaScript 语句的字符串版本)不同，execScript()方法没有返回值，尽管如此，该方法在拥有当前文档窗口的全局变量空间内执行。例如，如果文档的脚本声明了如下的全局变量：

```
var myVar;
```

execScript()方法可以读写此变量：

```
window.execScript("myVar = 10; myVar += 5");
```

在运行上述语句后，全局变量 myVar 的值为 15。

示例

使用 The Evaluator(第 4 章)试验 execScript()方法。The Evaluator 为小写字母 a~z 预先声明

更多资源请访问稀酷客(www.ckook.com)

了全局变量。在顶部文本框中输入以下各条语句，并观察结果。

```
a
```

在首次载入时，会声明该变量，但未赋值，因此它尚未定义：

```
window.execScript("a = 5")
```

该方法没有返回值，因此 The Evaluator 的内部机制认为，该语句未定义：

```
a
```

变量值现在是 5。

```
window.execScript("b = a * 50")
b
```

全局变量 b 的值是 250。继续试验其他脚本语句，用分号分隔开字符串参数中的多条语句。

相关主题：eval()函数。

find(["*searchString*" [, *matchCaseBoolean*, *searchUpBoolean*]])

返回值：非对话框搜索的 Boolean 值。

兼容性：WinIE-，MacIE-，NN4+，Moz1.0.1+，Safari+，Opera-，Chrome+

NN4 引入的 window.find()方法可以模拟浏览器 Find 对话框的功能，可通过工具栏的 Find 按钮访问该对话框。此方法在 NN6 中不可用，但在 NN7/Mozl 1.0.1 中可用。

如果未指定参数，执行该方法会显示浏览器的 Find 对话框，就像用户单击了工具栏的 Find 按钮一样。另外，这个函数没有返回值。

可给该函数指定一个搜索字符串作为参数。这个搜索只是简单的字符串匹配，而不是正则表达式类型的搜索(见配书光盘中的第 45 章)。如果找到匹配，浏览器就滚动到匹配的词，并高亮显示该词，就像使用浏览器的 Find 对话框一样。发现一个匹配后，该函数也返回布尔值 true；如果在文档中找不到匹配，或者没有在当前搜索路径上发现更多的匹配(默认路径是自上而下)，该函数就返回 false。

表 27-3 列出了可用的可选参数，所有参数的默认值都是 false。这些选项与 NN4+的 Find 对话框相同。

表 27-3　可通过脚本控制的 window.find()方法特性

属　性	说　明
caseSensitive	布尔值，true 表示该搜索是区分大小写的
backwards	布尔值，true 表示该搜索是向后执行的
wraparound	布尔值，true 表示该搜索到达底部后，再从文档开头开始
wholeWord	布尔值，true 表示该搜索是全词搜索
searchInFrames	布尔值，true 表示在框架内搜索
showDialog	布尔值，true 表示显示浏览器的 Find 对话框

更多资源请访问稀酷客(www.ckook.com)

一些现代浏览器(如 Firefox)已经放弃 Find 对话框，而支持显示在浏览器窗口底部的集成查找功能。此查找方式可以同时应用于整个页面，突出显示所有匹配的文本。

IE4+也有一个用脚本编写的文本搜索功能，但它是以完全不同的方式实现的(使用配书光盘中的第 33 章描述的 TextRange 对象)。其可视操作也不同，它不会突出显示并滚动到文本中的匹配字符串。

示例

对 window.find()方法的简单调用如下：

```
var success = window.find("contract");
```

如果要进行区分大小写的搜索，至少要添加两个可选参数中的一个：

```
success = window.find(matchString,caseSensitive);
```

在许多方面，window.find()方法都是 NN4 的遗留物。关于 body 文本搜索的更现代方式，请参见配书光盘中的第 33 章对 TextRange 和 Range 对象的讨论。

相关主题：TextRange. Range 对象(配书光盘中的第 33 章)。

forward()

(参见 window.back())

geckoActiveXObject("*progID*")

返回值：WMP 控件对象。

兼容性：WinIE-，MacIE-，NN7.1+，Moz1.4+，Safari-，Opera-，Chrome-

NN/IE 浏览器之战的一个有趣结果是，Microsoft 成功地将 Windows Media Player 确立为 Windows 操作系统上的 PC 媒体播放器。由于 WMP 实现为 ActiveX 控件，因此，NN/Moz 浏览器的脚本编程功能有些被冷落了。window.geckoActiveXObject()方法添加到 Moz 1.4 浏览器中，提供了把 WMP 作为 ActiveX 控件来访问的功能。尽管该方法的名称暗示它会支持大多数 ActiveX 控件，但它目前只能打开 WMP 控件。

geckoActiveXObject()的唯一参数是编程 ID，目前 MediaPlayer.MediaPlayer.1 表示 WMP。因此，要用 geckoActiveXObject()方法获取媒体播放器的 WMP 控件引用，可使用以下代码：

```
var player = new GeckoActiveXObject("MediaPlayer.MediaPlayer.1");
```

getComputedStyle(elementNodeRef, "pseudoElementName")

返回值：CSS 样式对象

兼容性：WinIE-，MacIE-，NN-，Moz+，Safari+，Opera+，Chrome+

window.getComputedStyle()方法允许访问与给定元素相关联的 CSS(层叠样式表)样式对象。该方法的第一个参数是元素节点引用，第二个参数是样式要应用的特定伪元素的可选名称。之所以说可选，是因为可以把一个空字符串作为第二个参数，来获得没有伪元素意义的样式对象。

更多资源请访问稀酷客(www.ckook.com)

注意：

尽管各个浏览器都支持 getComputedStyle()方法，但其返回值可以采用不同的格式，具体取决于为元素指定的样式。例如，如果给有背景色的元素指定样式，Opear 就会以 16 进制格式返回该颜色，而 Firefox、Safari 和 Chrome 以 rgb()格式返回它。

尽管 getComputedStyle()方法是为 window 对象定义的(且对 window 对象有效)，W3C DOM 却将 document.defaultView.getComputedStyle()作为访问元素的样式对象的标准方式。其实最终调用的就是这个方法，其访问方式是有区别的。

getSelection()

返回值：Selection 对象

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

此方法取代了 document 对象中已经过时的同名方法。该方法提供了一种脚本编程方式，来获得用户在页面中选择的文本。在选择并复制文档的 body 文本，以粘贴到其他应用程序文档时，这是一个常见的任务。window.getSelection()方法返回用户选择的文本串。如果用户未选择内容，该方法就返回一个空字符串。返回值只包括页面上的可见文本，而不包括其下的 HTML 或文本样式。

WinIE4+的对应功能是 document.selection 属性，它返回一个 IE selection 对象。要从该对象中获得文本，必须在其中创建一个 TextRange 对象，然后检查 text 属性：

```
var selectedText = document.selection.createRange().text;
```

示例

程序清单 27-22 中的文档提供了一个跨浏览器(但不包括 MacIE5)的解决方案，来获取用户在页面中选择的文本。所选的文本显示在文本区域中，脚本使用浏览器检测和分支功能，来处理 Mozilla 和 Internet Explorer 在事件处理上的差异。

程序清单 27-22　获取所选文本

HTML: jsb27-22.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Getting Selected Text</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript" src="jsb27-22.js"></script>
   </head>
   <body>
      <h1>Getting Selected Text</h1>
      <hr />
      <p>Select some text and see how JavaScript can capture the selection:</p>
      <h2>ARTICLE I</h2>
      <p>Congress shall make no law respecting an establishment of religion, or
         prohibiting the free exercise thereof; or abridging the freedom of
```


更多资源请访问稀酷客(www.ckook.com)

```
            speech, or of the press; or the right of the people peaceably to
            assemble, and to petition the government for a redress of grievances.
        </p>
        <form>
            <textarea name="selectedText" rows="3" cols="40" wrap="virtual">
            </textarea>
        </form>
    </body>
</html>
```

JavaScript: jsb27-22.js

```
addEvent(document, "mouseup", showSelection);
function showSelection()
{
    if (window.getSelection)
    {
        document.forms[0].selectedText.value = window.getSelection();
    }
    else if (document.selection)
    {
        document.forms[0].selectedText,value = document.selection.createRange().text;
        event.cancelBubble = true;
    }
}
```

相关主题： document.selection 属性。

home()

返回值： 无。

兼容性： WinIE-，MacIE-，NN4+，Moz+，Safari-，Opera+，Chrome-

像 Navigator 4 首次引入的许多 window 方法一样，window.home()方法也提供了一种脚本编程方式来代替工具栏上 Home 按钮的功能。这个动作将浏览器导航到首选项为主页位置设置的 URL 上。但该方法不能控制访问者浏览器的默认主页。

相关主题： window.back()方法，window.forward()方法；window.toolbar 属性。

moveBy(*deltaX*,*deltaY*)，moveTo(*x*,*y*)

返回值： 无。

兼容性： WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome-

版本 4 浏览器引入了一个功能，允许 JavaScript 调整屏幕上浏览器窗口的位置，这适用于主窗口或者由脚本生成的子窗口。NN/Moz 认为，窗口移出屏幕是一种潜在的安全漏洞，因此在 NN4+/Moz 中需要使用签名脚本将窗口移出屏幕。这些方法与 Chrome 兼容，但 Chrome 不使用它们。

可将窗口移到屏幕上的一个绝对位置，或者沿着水平和/或垂直轴将窗口调整到指定的像素位置(无论窗口的绝对像素位置在哪里)。*x*(水平)和 *y*(垂直)构成的坐标空间是整个屏幕，左上角

更多资源请访问稀酷客(www.ckook.com)

是(0, 0)。用 moveBy()和 moveTo()方法设置的坐标点是浏览器窗口外边界的左上角。因此，将窗口移动到(0, 0)点，就是使窗口与屏幕的左上角齐平。然而，在所有浏览器和操作系统中，这并不是真正意义上的最大化，因为最大化窗口的坐标可能是不太大的负像素值。

MoveTo()和 moveBy()方法的区别在于，一个是绝对移动，而另一个是相对于当前窗口位置的移动。为 moveTo()指定的参数是移动窗口后，窗口左上角在屏幕上的横坐标和纵坐标；相反，moveBy()的参数表示在每个方向上要将窗口移动多远。如果想将窗口向右移动 25 像素，仍必须包含两个参数，但 *y* 值为 0：

```
window.moveBy(25,0);
```

要向左移动，第一个参数必须为负数。

示例

程序清单 27-23 给出了使用 window.moveTo()和 window.moveBy()方法的一些示例。页面包含 4 个按钮，每个按钮执行一种不同的浏览器窗口移动操作。

程序清单 27-23　移动窗口

HTML: jsb27-23.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Window Gymnastics</title>
      <script type="text/javascript" src="jsb27-23.js"></script>
   </head>
   <body onload="init()">
      <h1>Window Gymnastics</h1>
      <hr />
      <form name="buttons">
         <ul>
            <li><input name="offscreen" type="button"
                      value="Disappear a Second"
                      onclick="moveOffScreen()" />
            </li>
            <li><input name="circles" type="button"
                      value="Swift and Tiny Circular Motion"
                      onclick="revolve()" />
            </li>
            <li><input name="bouncer" type="button"
                      value="Zig Zag"
                      onclick="zigzag()" />
            </li>
            <li><input name="expander" type="button"
                      value="Maximize"
                      onclick="maximize()" />
            </li>
         </ul>
      </form>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb27-23.js

```
// wait in onload for page to load and settle in IE
function init()
{
   // fill missing IE properties
   if (!window.outerWidth)
   {
      window.outerWidth = document.body.clientWidth;
      window.outerHeight = document.body.clientHeight + 30;
   }
}

// function to run when window captures a click event
function moveOffScreen()
{
   // branch for NN security
   if (window.netscape)
   {
      // will also get here if in Opera, but Opera will throw error, so...
      try
      {
        netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
      }
      catch (e)
      {
         null;
      }
   }
   var maxX = screen.width;
   var maxY = screen.height;
   window.moveTo(maxX+1, maxY+1);
   setTimeout("window.moveTo(0,0)",500);
   if (window.netscape)
   {
      // will also get here if in Opera, but Opera will throw error, so...
      try
      {
       netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserWrite");
      }
      catch (e)
      {
         null;
      }
   }
}

// moves window in a circular motion
function revolve()
{
   var winX = (screen.availWidth - window.outerWidth) / 2;
```


更多资源请访问稀酷客(www.ckook.com)

```
    var winY = 50;
    window.resizeTo(400,300);
    window.moveTo(winX, winY);
    for (var i = 1; i < 36; i++)
    {
        winX += Math.cos(i * (Math.PI/18)) * 5;
        winY += Math.sin(i * (Math.PI/18)) * 5;
        window.moveTo(winX, winY);
    }
}

// moves window in a horizontal zig-zag pattern
function zigzag()
{
    window.resizeTo(400,300);
    window.moveTo(0,80);
    var incrementX = 2;
    var incrementY = 2;
    var floor = screen.availHeight - window.outerHeight;
    var rightEdge = screen.availWidth - window.outerWidth;
    for (var i = 0; i < rightEdge; i += 2)
    {
        window.moveBy(incrementX, incrementY);
        if (i%60 == 0)
        {
            incrementY = -incrementY;
        }
    }
}

// resizes window to occupy all available screen real estate
function maximize()
{
    window.moveTo(0,0);
    window.resizeTo(screen.availWidth, screen.availHeight);
}
```

要在 NN/Moz 中成功运行此脚本，第一个按钮要求启用代码库委托(codebase principals)(参见配书光盘中的第 49 章)，以利用通常只有签名脚本才有的功能。moveOffScreen()函数暂时将窗口完全移出视野，Opera 能识别 window.netscape，但如果尝试启用代码库委托，它就会抛出一个错误。try 块可以捕获该错误。注意脚本先确定屏幕的大小，再决定向哪里移动窗口。在移出屏幕后，窗口将重新在屏幕左上角显示出来。

第二个函数 revolve()使窗口来回兜圈。在缩小窗口，并将其定位到屏幕上方中心附近后，脚本使用了一些数学方法，在标准圆上的 36 个位置(以 10 度为增量)定位窗口，这是一个基于数学计算来动态控制窗口位置的示例。IE 并未提供属性来表示浏览器窗口的外部尺寸，这使工作稍微复杂了一些。

为了演示 moveBy()方法，第三个函数 zigzag()使用了一个 for 循环来递增坐标点，使窗口在屏幕上以锯齿方式移动。*x* 坐标一直以线性方式递增，直至窗口到达屏幕边缘(这也是动态计算，以适应任意尺寸的监视器)，而 *y* 坐标必须随着窗口在屏幕上不时改变移动方向而增减。

更多资源请访问稀酷客(www.ckook.com)

第四个函数包含一些实用代码，演示了如何更好地模仿浏览器窗口的最大化操作，来填充访问者监视器上的全部可用屏幕空间。

相关主题：window.outerHeight 属性，window.outerWidth 属性；window.resizeBy()方法，window.resizeTo()方法。

navigate("*URL*")

返回值：无。

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera+，Chrome-

window.navigate()方法是 IE 专用的，它可将新文档载入窗口或者框架。这个方法的功能与为 location. href 属性指定 URL 一样，该属性在所有的脚本浏览器中都可用。如果用户只使用基于 IE 或 Opera 的浏览器，这个方法十分安全；否则最好用 location.href 来浏览文档。

示例

使用有效的 URL 作为方法参数，如下：

```
window.navigate("http://www.dannyg.com");
```

相关主题：location 对象。

open("*URL*", "*windowName*" [, "*windowFeatures*"][,*replaceFlag*])

返回值：表示新建窗口的 window 对象；如果方法失败，返回 null。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

脚本利用 window.open()方法给 Web 站点设计者提供了很多选项，以控制用户计算机屏幕上第二或者第三个 Web 浏览器窗口的外观。而且，这种控制可以用支持 JavaScript 的浏览器实现，而不需要签名脚本。因为新窗口的界面元素更直观，所以首先讨论 window.open()方法的参数。

1. 设置新窗口特性

可选的 windowFeatures 参数是一个字符串，它包含用逗号分隔的赋值表达式(其行为类似于 HTML 标记特性)。注意：为了获得最佳的浏览器兼容性，不要在逗号后加上空格。如果忽略第三个参数，JavaScript 创建的新窗口就与在 File 菜单中选择 New Web Browser 菜单项所打开的窗口相同，但可以通过第三个参数控制在新窗口中显示哪些窗口元素。记住这条重要的原则：如果只指定了第三个参数值的原始设置，则其他特性都是关闭的，除非该参数指定打开某项特性。表 27-4 列出了在所有浏览器中可以控制的新建窗口特性。在不指定第三个参数时，除非特别指明，否则所有的 Boolean 值默认为 yes。

表 27-4　可通过脚本控制的 window.open()方法特性

特　性	浏　览　器	说　　明
alwaysLowered[3]	NN4+/Moz+	(布尔值)总是在其他浏览器窗口之后
alwaysRaised[3]	NN4+/Moz+	(布尔值)总是在其他浏览器窗口之前
channelMode	IE4+	(布尔值)有频道栏的剧院模式(默认为无)
chrome[3]	NN7.2+/Moz1.7+	(布尔值)浏览器用户界面功能

更多资源请访问稀酷客(www.ckook.com)

(续表)

特　　性	浏　览　器	说　　明
close	NN4+/Moz+	(布尔值)对话框窗口顶部的系统关闭命令图标和菜单项
dependent[6]	NN4+/Moz+	(布尔值)如果关闭 opener 窗口，子窗口也关闭
directories[7]	NN2+/Moz+，IE3–6	(布尔值)显示 personal/bookmarks/links 工具栏
fullscreen[5]	IE4+	(布尔值)没有标题栏或菜单(默认为 no)
height	NN2+/Moz+，IE3+，Safari+，Opera+，Chrome+	(整数)以像素为单位的内容区域高度
innerHeight[4]	NN4+/Moz+，Safari+，Chrome+	(整数)内容区域高度，与旧的高度属性相同
innerWidth[4]	NN4+/Moz+，Safari+，Chrome+	(整数)内容区域宽度，与旧的宽度属性相同
left	NN6+/Moz+，IE4+，Safari+，Opera+，Chrome+	(整数)屏幕左上角的水平位置(Integer)
location[8]	NN2+/Moz+，IE3+，Safari	(布尔值)显示当前 URL 的域
menubar[1]	NN2+/Moz+，IE3+，Safari	(布尔值)窗口顶部的菜单栏
minimizable	NN7.1+/Mozl.2+	(布尔值)对话框窗口顶部的最小化命令图标
modal[3]	NN7.1+/Mozl.2+	(布尔值)窗口的模态，以阻止对主窗口的访问，直到关闭打开的窗口
outerHeight[4]	NN4+/Moz+	(整数)可见窗口的高度
outerWidth[4]	NN4+/Moz+	(整数)可见窗口的宽度
personalBar[7]	NN4+/Moz+	(布尔值)directories 特性的 Mozilla 特定版本
resizable[2, 9]	NN2+/Moz0-1.4、IE3+	(布尔值)允许通过拖动来调整大小的界面元素
screenX[4]	NN4+/Moz+	(整数)屏幕左上角的水平位置
screenY[4]	NN4+/Moz+	(整数)屏幕左上角的垂直位置
scrollbars	NN2+/Moz+，IE3+，Opera+	(布尔值)如果文档大于窗口，则显示滚动条
status	NN2+/Moz+，IE3+，Safari	(布尔值)窗口底部的状态栏
titlebar[3]	NN4+/Moz+	(布尔值)标题栏和所有其他边框元素
toolbar	NN2+/Moz+，IE3+，Safari	(布尔值)后退、前进和行中的其他按钮
top	NN6+/Moz+，IE4+，Safari+，Opera+，Chrome+	(整数)屏幕左上角的垂直位置
width	NN2+/Moz+，IE3+，Safari+，Opera+，Chrome+	(整数)以像素为单位的内容区域宽度
z-lock[3]	NN4+/Moz+	(布尔值)窗口层固定在浏览器窗口之下

1. 未在 Mac 中提供，因为浏览器窗口没有菜单栏；从 MacNN4 中退出时，将显示一个简化的 Mac 菜单栏。
2. Mac 窗口总是可以改变大小。
3. 要求签名脚本。
4. 要求签名脚本，在安全的范围内调整窗口的大小和位置。Left 和 Top 分别优先于这两个功能。
5. 其行为在 IE6+中有变化。不推荐使用它，除非在 kiosk 类型的浏览器中，否则使用它总是会令用户不舒服。

更多资源请访问稀酷客(www.ckook.com)

6. 其行为在 Moz5 中有变化。它没有在 Mac OS 上实现。它当前正在修订中。
7. 在基于 Mozilla 的浏览器中，只有 toolbar 也设置为 yes，这个功能才有效。
8. 在 Safari 中，如果选中了 toolbar，就总是选中 location。
9. 在 IE7+中，这个功能会禁用新窗口中的选项卡。在 FF3+中，新窗口总是可以调整大小。

Boolean 值的处理可能稍有不同，ture 值可以是 yes、1 或者属性名本身；而 false 值用 no 或者 0 表示。如果忽略 Boolean 特性，它们就指定为 false。因此，如果想创建一个新窗口，只显示工具栏和状态栏，并可改变窗口的尺寸，方法如下：

```
window.open("newURL","NewWindow", "toolbar,status,resizable");
```

这个新窗口没有指定宽度和高度，而是设置成浏览器窗口的默认值，浏览器窗口由浏览器用 File 菜单的 New Web Browser 命令创建。换句话说，新窗口不会自动继承调用 window.open() 方法所生成的窗口的大小。由脚本创建的新窗口的位置在某种程度上是随意的，除非使用现代浏览器提供的窗口定位特性。注意，每个浏览器的定位特性都不同(在 NN/Moz/W3C 中为 screenX 和 screenY；在 IE 中为 left 和 top)。可在单个参数字符串中包含两个特性，因为浏览器会忽略它不认识的特性。

注意：
通过窗口的 onload 和 onunload 事件处理程序调用 window.open()方法，会严重滥用不需要的广告弹出窗口。包括弹出窗口阻止程序的浏览器(如 IE6+和基于 Mozilla/WebKit/Presto 的浏览器)会阻止这些事件处理程序调用该方法。现在越来越多的浏览器和用户每天都使用弹出窗口阻止程序，所以不应向 Web 冲浪者弹出广告窗口。

2. 仅用于 Netscape/Mozilla 的签名脚本

许多 NN/Moz 专用的特性都存在安全隐患，因此在使用它们之前需要签名脚本和用户的许可。如果用户不授予权限，就会忽略这些存在安全问题的参数。

为了用签名脚本打开具有安全窗口特性的新窗口，必须激活 UniversalBrowserWrite 权限，就像其他签名脚本那样。下面代码段将生成一个 alwaysRaised 风格的窗口：

```
<script type="text/javaScript" archive="myJar.jar" id="1">
function newRaisedWindow()
{
   netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
   var newWindow = window.open("","","height=100,width=300,alwaysRaised=yes");
   netscape.security.PrivilegeManager.disablePrivilege("UniversalBrowserWrite");
   var newContent = "<html><body><b> "On top of spaghetti!"</b>";
   newContent += "<form><center><input type='button' value='OK'";
   newContent += "onclick='self.close()'></center></form></body></html>";
   newWindow.document.write(newContent);
   newWindow.document.close();
}
</script>
```

通过程序清单 27-24 中的脚本，可用任意属性组合来试验新窗口的外观和行为。这个页面

更多资源请访问稀酷客(www.ckook.com)

提供了新窗口的一组 Boolean 属性，并根据用户的选择创建了一个 300×300 像素的新窗口。如果使用 NN/Moz，则必须为签名脚本打开代码库委托(参见配书光盘中的第 49 章)。

关闭标题栏应小心。因为关闭标题栏时，窗口的内容就会浮动显示，因为没有显示边界。但热键总是打开的，所以可按 Ctrl+W 组合键关闭这个无边界窗口(但在 Mac 中，标题栏关闭时，热键总是禁用的)。在支持 titlebar 功能的浏览器中，为将计算机变成一个“亭子”，可以把窗口尺寸设置为屏幕尺寸，再将窗口选项设置为 titlebar=no。在 IE 中，还可在功能列表中添加 fullscreen=yes。

程序清单 27-24　新窗口练习

HTML: jsb27-24.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.open() Options</title>
      <style type="text/css">
         .colHeaders
         {
            background-color:yellow; text-align:center;
         }
      </style>
      <script type="text/javascript" src="jsb27-24.js"></script>
   </head>
   <body>
      <h1>Select new window options:</h1>
      <hr />
      <form>
         <table border="2">
            <tr>
               <td class="colHeaders" colspan="2">
                  Browser Features:
               </td>
            </tr>
            <tr>
               <td>
                  <input type="checkbox" name="toolbar" />
                  toolbar
               </td>
               <td>
                  <input type="checkbox" name="location" />
                  location
               </td>
            </tr>
            <tr>
               <td>
                  <input type="checkbox" name="directories" />
                  directories
               </td>
```


更多资源请访问稀酷客(www.ckook.com)

```
                <td>
                    <input type="checkbox" name="status" />
                    status
                </td>
            </tr>
            <tr>
                <td>
                    <input type="checkbox" name="menubar" />
                    menubar
                </td>
                <td>
                    <input type="checkbox" name="scrollbars" />
                    scrollbars
                </td>
            </tr>
            <tr>
                <td>
                    <input type="checkbox" name="resizable" />
                    resizable
                </td>
                <td>
                    <input type="checkbox" name="titlebar" checked="checked" />
                    titlebar
                </td>
            </tr>
            <tr>
                <td colspan="2" align="middle">
                    <input type="button" name="forAll" value="Make New Window"
                           onclick="makeNewWind(this.form)" />
                </td>
            </tr>
        </table>
    </form>
    <br />
  </body>
</html>
```

JavaScript: jsb27-24.js

```
function makeNewWind(form)
{
   if (window.netscape)
   {
      // will also get here if in Opera, but Opera will throw error, so...
      try
      {
      netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserWrite");
      }
      catch(e)
      {
         null;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
    }
    var attr = "width=300,height=300";
    for (var i = 0; i < form.elements.length; i++)
    {
        if (form.elements[i].type == "checkbox")
        {
            attr += "," + form.elements[i].name + "=";
            attr += (form.elements[i].checked) ? "yes" : "no";
        }
    }
    var newWind = window.open("bofright.html","subwindow",attr);
    if (window.netscape)
    {
        try
        {
            netscape.security.PrivilegeManager.revertPrivilege("CanvasAccess");
        }
        catch(e)
        {
            null;
        }
    }
}
```

3. 指定窗口名

现在分析一下 window.open()的其他参数，第二个参数是新窗口的名称。不要将这个参数与文档标题混淆，文档标题通常由决定窗口内容的 HTML 文本来设置，窗口名的样式必须与其他对象名和变量名的单字标识符一致。这个名称也不同于 open ()方法返回的 window 对象，脚本不会用到这个名称，这个名称最常用于链接和窗体的 target 特性。

4. 将内容载入新窗口

脚本通常在窗口中填入以下两类信息：

- 事先知道 URL 的已有 HTML 文档
- 立即创建的 HTML 页

为创建一个新窗口来显示已有的 HTML 文档，可将 URL 作为 window.open()方法的第一个参数。如果页面不能将 URL 载入新页，可以试着指定目标文档的完整 URL(而不只是文件名)。

将第一个参数设置为空字符串，会迫使窗口打开一个空白文档，脚本可以在其中写入 HTML (或者用另一个语句载入文档，这个语句将窗口地址设置为一个特殊的 URL)。如果想立即在窗口中写入内容，就可以将 HTML 内容组织为一个长字符串，然后用 document.write()方法将其粘贴到新窗口中。如果不想在页面中写入其他内容，就可在最后添加一个 document.close()方法，告诉浏览器，布局已完成(因此如果窗口有 Layout:Complete 或者 Done 消息，它们就显示在状态栏中)。

如果成功打开了窗口，window.open()方法就返回新 window 对象的引用。如果脚本需要定

更多资源请访问稀酷客(www.ckook.com)

位新窗口的元素(例如写文档时)，这个返回值就非常重要。

为了允许脚本中的其他函数引用子窗口，应将 window.open()方法的结果赋给一个全局变量。在首次写入新窗口之前，检测该变量，确认其不是 null，例如，窗口可能因为内存不足而打不开。如果一切顺利，就可以在新窗口中用此变量作为属性或者对象引用的开头。例如：

```
var newWindow
//...
function createNewWindow()
{
   newWindow = window.open("","");
   if (newWindow != null)
   {
      newWindow.document.write("<html><head><title>Hi!</title></head>");
   }
}
```

全局变量引用仍然可用于另一个函数——可能是(通过 close ()方法)关闭子窗口的函数。

子窗口中的脚本需要与源窗口中的对象和脚本通信时，如果访问者浏览器中的 JavaScript 没有自动提供 opener 属性，就必须确保子窗口具有该属性。见本章前面对 window.opener 属性的讨论。

在 Netscape 浏览器中，用同一个窗口名参数(第二个参数)调用多个 window.open()方法，并不能创建窗口的副本(但在 Internet Explorer 中可以)。JavaScript 将阻止用同一名称创建两个窗口，还要注意，window.open()方法不会将使用该名称的现有窗口放到窗口层的前面，而 window.focus()可以做到这一点。

5. Internet Explorer 特性

在 IE 中创建子窗口有时会比较复杂，因为浏览器会有意料不到的行为。试图用 document.write()将内容放入新建的窗口时，一个最常见的问题是：在 IE(包括一些最新的版本)中执行使用 document.write()的脚本语句之前，是不能打开窗口的。这将导致脚本错误，因为子窗口的引用是无效的。为解决这个问题，应将 HTML 和 document.write()语句放在一个单独的函数中，这个函数在窗口创建后通过 setTimeout()方法调用。参见程序清单 27-25 中的例子。

另一个影响 IE 的问题是脚本试图访问子窗口时，会偶然出现违反安全(拒绝访问)的行为警告。如果页面包括打开和访问子窗口的脚本，而页面由 HTTP 服务器提供服务，而不是从本地硬盘上提供服务，就不会出现这个问题。

示例

程序清单 27-25 显示的页面只有一个按钮，该按钮会生成一个特定大小的新窗口，其中只打开了状态栏。这里的脚本给出了创建新窗口所需的全部元素，在大多数平台上都有这些元素。新窗口对象的引用赋给全局变量 newWindow。在生成新窗口前，脚本检查以前是否未生成过窗口(在这种情况下 newWindow 是 null)；较新的浏览器则检查窗口是否关闭。如果有一个条件为 true，就用 open()方法创建窗口；否则，现有窗口用 focus()方法显示在最前面。

由于所有 IE 版本都有一个计时问题，所以把组装 HTML 和写入新窗口的操作放在不同的

更多资源请访问稀酷客(www.ckook.com)

函数中，函数在 50ms 的延时后调用(其他浏览器也采用这种方式，即使它们在组装和写入 HTML 时可以不延时也同样如此)。要构造最终写入文档的字符串，可以使用+=(加值)操作符，该操作符将其右侧的字符串添加到存储在左侧变量中的字符串之后。在此示例中，新窗口显示一个<h1>级的文本行。

程序清单 27-25　创建新窗口

HTML: jsb27-25.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>New Window</title>
      <script type="text/javascript" src="jsb27-25.js"></script>
   </head>
   <body>
      <h1>Creating a New Window</h1>
      <hr />
      <form>
         <input type="button" name="newOne" value="Create New Window"
                onclick="makeNewWindow()" />
      </form>
   </body>
</html>
```

JavaScript: jsb27-25.js

```
var newWindow;

function makeNewWindow()
{
   if (!newWindow || newWindow.closed)
   {
      newWindow = window.open("","","status,height=200,width=300");
      // force small delay for IE to catch up
      setTimeout("writeToWindow()", 50);
   }
   else
   {
      // window's already open; bring to front
      newWindow.focus();
   }
}

function writeToWindow()
{
   // assemble content for new window
   var newContent = "<html><head><title>One Sub Window<\/title><\/head>";
   newContent += "<body><h1>This window is brand new.<\/h1>";
   newContent += "<\/body><\/html>";
```


更多资源请访问稀酷客(www.ckook.com)

```
    // write HTML to new window document
    newWindow.document.write(newContent);
    newWindow.document.close(); // close layout stream
}
```

window.open()方法可能在 IE7+和 FF2+等浏览器中带来潜在问题，这些浏览器支持选项卡浏览，即多个页面打开为同一浏览器实例中的不同选项卡。对 window.open()方法的默认响应是为新窗口打开一个新选项卡，而不是打开一个新浏览器窗口，如果脚本希望打开一个全新的浏览器窗口，这可能出问题。

相关主题：window.close()方法，window.blur()方法，window.focus()方法，window.openDialog()方法；window.closed 属性。

openDialog("*URL*", "*windowName*" [, "*windowFeatures*"][, *arg1*][, *arg2*] . . .)

返回值：代表新建对话框(窗口)的窗口对象，如果方法失败，返回 null。

兼容性：WinIE-，MacIE-，NN7+，Moz1.0.1+，Safari-，Opera-，Chrome-

window.openDialog()是 Mozilla 专用的方法，是 window.open()方法的 XUL 对应方法。XUL 是 Mozilla 的基于 XML 用户界面描述语言，在 Mozilla 应用程序套件中使用。openDialog()方法提供了一些额外的窗口功能，可以传递数量可变的参数，在创建自定义窗口时这会很便捷。

openDialog()方法的参数与 open ()方法类似，但可选的 argl，arg2 等参数除外。一个明显的区别是 openDialog()增加了 all 窗口功能，它用于显示(all=yes)或隐藏(all=no)除 chrome、dialog 和 modal 外的所有窗口功能，chrome、dialog 和 modal 能单独显示或隐藏。

相关主题：window.open()方法。

print()

返回值：无。

兼容性：WinIE5+，MacIE5+，NN4+，Moz+，Safari+，Opera+，Chrome+

print()方法提供了将窗口或者框架从框架集传递到打印机的脚本方法。任何情况下都会显示 Print 对话框，让用户选择手工打印时常见的打印选项，这将防止 print()命令在未经用户允许的情况下连接打印机。但在将页面传送给打印机后，用户必须重载页面，才能看到页面的内容。

WinIE5 引入了一些打印专用的事件处理程序，可以由脚本打印和手工打印操作触发。在用户接受 Print 对话框中的选项后，这些事件就会激活。onbeforeprint 事件处理程序可用来打印不显示、但应出现在打印输出中的内容。在内容送到打印缓冲区后，onafterprint 事件可以恢复页面。

示例

程序清单 27-26 是一个框架集，它将程序清单 27-27 载入顶部框架，将 Bill of Rights 的副本载入底部框架。

程序清单 27-26　打印示例框架集

```
<!DOCTYPE html>
<html>
   <head>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>window.print() method</title>
    </head>
    <frameset rows="25%,75%">
        <frame name="controls" src="jsb27-27.html" />
        <frame name="display" src="bofright.html" />
    </frameset>
</html>
```

顶部控制面板中的两个按钮(参见程序清单 27-27)可以打印整个框架集(在支持它的浏览器和操作系统中)或只打印底部框架。要打印整个框架集，引用应包括父窗口；要打印底部框架，引用应指向 parent.display 框架。

程序清单 27-27　打印控制

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Print()</title>
    </head>
    <body>
        <h1>Print()</h1>
        <hr />
        <form>
            <input type="button" name="printWhole"
                   value="Print Entire Frameset"
                   onclick="parent.print()" />
            <p><input type="button" name="printFrame"
                   value="Print Bottom Frame Only"
                   onclick="parent.display.print()" />
            </p>
        </form>
    </body>
</html>
```

如果不喜欢打印输出的某些方面，要知道原因出在浏览器的打印引擎，而不是 JavaScript。print()方法只是调用了浏览器的常规打印例程。

相关主题：window.back()方法，window.forward()方法，window.home()方法，window.find()方法。

prompt("*message*", "*defaultReply*")

返回值：用户输入的文本字符串或 null。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

JavaScript 能显示的第三种对话框包括来自脚本作者的消息，一个用户输入域和两个按钮(OK 和 Cancel)。脚本作者可以提供一个事先写好的答案，这样用户在碰到提示对话框时，可以单击 OK 按钮(或者按 Enter 键)接受答案，而不用再输入。给 window.prompt()方法提供两个参

更多资源请访问稀酷客(www.ckook.com)

数是很重要的，即使不想提供默认答案，也可将空串作为第二个参数：

```
prompt("What is your postal code?","");
```

如果忽略了第二个参数，JavaScript 就将字符串 undefined 插入对话框的域中。这个消息会使大多数网页访问者惊慌失措。

用户单击 OK 按钮后，该方法的返回值是对话框的域中的一个字符串。如果要求用户输入数字，则这个方法的返回值是一个字符串，在数学计算中可能需要用 parseInt()或者 parseFloat()函数(见第 24 章)转换返回值的数据类型。

如果单击了提示对话框的 OK 按钮，而未在空白域中输入任何文本，则返回值是空串(" ")。然而，单击 Canel 按钮，此方法会返回一个空值。因此，脚本开发人员必须验证返回值的类型，以确认用户输入了数据，以便在后面的脚本中予以处理，如：

```
var entry = prompt("Enter a number between 1 and 10:","");
if (entry != null)
{
   //statements to execute with the value
}
```

这个脚本把提示对话框的返回值赋给一个变量，如果对话框的返回值非空(假设单击了 OK 按钮)，则执行嵌套语句。剩余的语句进行数据验证，保证输入的数值在期望的范围内(参见配书光盘中的第 46 章)。

将提示对话框作为方便的用户输入设备是非常吸引人的。但与其他的 JavaScript 对话框相同，提示对话框是模态的，会中断用户正常的文档操作过程，还会捕获自动运行的宏，一些用户激活这些宏，是为了捕获 Web 站点。在表单中，HTML 域是吸引用户输入文本的更好的界面元素。也许使用提示对话框的最安全方式是用户在页面上单击了按钮元素后，要求用户提供的信息能在单个提示对话框中显示，才显示提示对话框。显示一系列提示对话框，是非常令人讨厌的。

示例

在程序清单 27-28 中，函数从提示对话框(参见图 27-7 中的对话框)中接收值，再执行一些数据输入验证(这对于商业站点当然是不够的)。函数首先进行检查，以确保返回值既不是 null (Cancel)，也不是空字符串(用户没有输入任何值就单击 OK 按钮)。关于数据输入验证的更多内容，请参见配书光盘上的第 46 章。

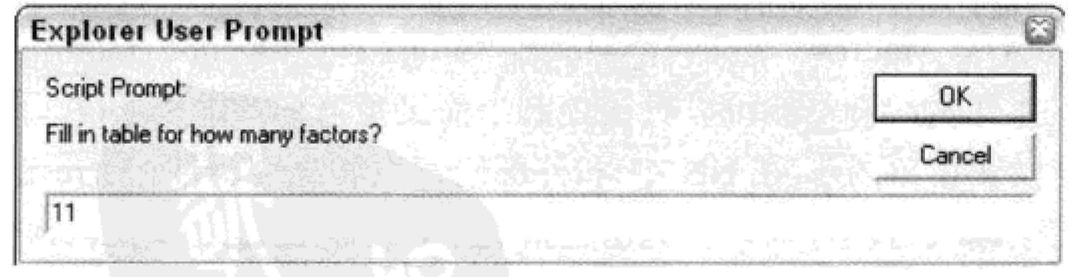


图 27-7 在 Internet Explorer 中显示程序清单 27-28 的提示对话框

程序清单 27-28 提示对话框

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.prompt() Method</title>
      <script type="text/javascript">
         function populateTable()
         {
            var howMany = prompt("Fill in table for how many factors?","");
            if (howMany != null && howMany != "")
            {
               alert("Filling the table for " + howMany); // for demo
               //statements that validate the entry and
               //actually populate the fields of the table
            }
         }
      </script>
   </head>
   <body>
      <h1>window.prompt() Method</h1>
      <hr />
      <form>
         <h3>Other statements that display and populate a large table
             would be here.</h3>
         <input type="button" name="fill" value="Fill Table..."
                onclick="populateTable()" />
      </form>
   </body>
</html>
```

注意程序清单 27-28 中一个重要的用户界面元素。由于单击按钮会打开一个需要用户输入更多信息的对话框，因此按钮标签以省略号结尾(或将三个句点作为省略号)，省略号常常表示，用户界面元素将打开某种对话框。与独立的应用程序一样，用户应能取消该对话框，并返回到单击按钮前的屏幕状态。

相关主题：window.alert()方法，window.confirm()方法。

resizeBy(*deltaX,deltaY*)，resizeTo(*outerwidth,outerheight*)

返回值：无。

兼容性：WinIE4+，MacIE4+，NN4，Moz+，Safari 1+，Opera-，Chrome-

从浏览器版本 4 开始，脚本就可以直接控制当前浏览器窗口的大小(在基于 Mozilla 的浏览器中不可用)。可以设置窗口内部(在 NN 中)和外部的宽度和高度属性，而 resizeBy()和 resizeTo()方法能在一个语句中调整这两个值。在这两种情况下，所有的调整都会影响窗口的右下角。如果要移动窗口的左上角，可使用 window.moveBy()或 window.moveTo()方法。

所有调整大小的方法都需要不同的参数，resizeBy()方法使窗口的大小在一个或两个坐标轴方向上改变一定的像素值。因此，它与窗口先前指定的大小无关，仅与每个坐标值的改变有关。例如，要增加窗口的尺寸，在横向增加 100 像素，纵向增加 50 像素，可使用如下语句：

更多资源请访问稀酷客(www.ckook.com)

```
window.resizeBy(100, 50);
```

必须设置这两个参数，但如果只想调整一个方向的尺寸，就将另一个参数设为 0。也可以给两个参数中的一个或者两个使用负值，来缩小窗口。

有时，在某个平台上，窗口需要调整到指定的宽度和高度，才能获得窗体元素的最佳显示效果，这时就应使用 resizeTo()方法。resizeTo()方法的参数是窗口外部的实际宽度和高度像素值，这与 NN4 的 window.outerWidth 和 window. outerHeight 属性一样。

要改变窗口的大小，使其占据所有的屏幕区域(Windows 的任务栏和 Macintosh 的菜单栏除外)，可用 screen 对象属性计算可用的屏幕区域：

```
window.resizeBy(screen.availWidth, screen.availHeight);
```

这个动作与 Windows 中的最大化窗口并不完全相同。为了最大化窗口，必须将窗口移动到(4, -4)，并对 resizeBy()的两个参数加 8：

```
window.moveTo(-4,-4);
window.resizeTo(screen.availWidth + 8, screen.availHeight + 8);
```

这隐藏了窗口自身的 4 个像素宽的边框，就像窗口在操作系统中最大化一样。关于操作系统特定的更多细节请参见 screen 对象的讨论(配书光盘中的第 41 章)。

在一些平台上，指定了窗口内部的宽度和高度，而不是窗口外部。如果必须指定外部尺寸，可改用 NN 专用的 window. outerHeight 和 window. outerWidth 属性。

Navigator 4 对窗口的最大化和最小化有严格的限制。这两个方法只能用于屏幕的可见区域以及可见区域的最小值，除非页面使用了签名脚本(参见配书光盘中的第 49 章)。如果有了签名脚本和用户许可，就可以将窗口调整到可用屏幕的边界之外。

示例

使用程序清单 27-29 中的页面可以试验调整窗口大小的方法。表单的两个文本框可以为每个方法输入值。window.resize()输入值的文本框还允许输入多个重复值，以更好地理解不同值的影响。输入 0 和负值，看看它们如何影响方法。还要测试一下不同浏览器的限制。

程序清单 27-29　调整窗口大小的方法

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Window Resize Methods</title>
      <style type="text/css">
         .instructions
         {
            font-weight:bold;
         }
      </style>
      <script type="text/javascript">
         function doResizeBy(form)
         {
```


更多资源请访问稀酷客(www.ckook.com)

```
            var x = parseInt(form.resizeByX.value);
            var y = parseInt(form.resizeByY.value);
            var count = parseInt(form.count.value);
            for (var i = 0; i < count; i++)
            {
               window.resizeBy(x, y);
            }
         }
         function doResizeTo(form)
         {
            var x = parseInt(form.resizeToX.value);
            var y = parseInt(form.resizeToY.value);
            window.resizeTo(x, y);
         }
      </script>
   </head>
   <body>
      <h1>Window Resize Methods</h1>
      <hr />
      <form>
         <div class="instructions">Enter the x and y increment,
            plus how many times the window should
            be resized by these increments:
         </div>
         Horiz:<input type="text" name="resizeByX" size="4" />
         Vert:<input type="text" name="resizeByY" size="4" />
         How Many:<input type="text" name="count" size="4" />
         <input type="button" name="ResizeBy" value="Show resizeBy()"
               onclick="doResizeBy(this.form)" />
         <hr />
         <div class="instructions">Enter the desired width
            and height of the current window:</div>
         Width:<input type="text" name="resizeToX" size="4" />
         Height:<input type="text" name="resizeToY" size="4" />
         <input type="button" name="ResizeTo" value="Show resizeTo()"
               onclick="doResizeTo(this.form)" />
      </form>
   </body>
</html>
```

相关主题：window.outerHeight 属性，window.outerWidth 属性；window.moveTo()方法，window.sizeToContent()方法。

scroll (*horizontalCoord*, *verticalCoord*)

返回值：无。

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

window.scroll()方法在 NN3 中引入，之后，所有脚本浏览器都实现了它。但同时，这个方法已被 window.scrollTo()所取代，后者在语法上与其他 window 方法的联系更密切。

更多资源请访问稀酷客(www.ckook.com)

window.scroll()方法有两个参数，分别是要定位在窗口或框架左上角的文档的横坐标(x)和纵坐标(y)。要知道，窗口和文档有两个相似但不同的坐标系，从窗口的角度来看，内容区域的左上角的像素是点(0, 0)，但所有文档的左上角也是(0, 0)。窗口的(0, 0)点不能移动，但文档的0, 0 点能通过手工或者脚本滚动而移动。尽管 scroll()是一个窗口方法，但其似乎更像一个文档方法，就像文档在窗口中重定位自身一样；同样，也可以认为是窗口在移动，使其(0, 0)点移动到文档的指定坐标。

尽管参数值可以设置为超过文档的最大尺寸或者负值，但结果因平台而异。此时，window.scroll()方法的最佳用法是把文档滚动到文档的顶部(window.scroll(0, 0))，作为文档的定位基点。对包含大量文本的文档进行垂直滚动时，HTML 锚是一种较好的选择(但它不会重新调整水平滚动位置)。

相关主题：window.scrollBy()方法，window.scrollTo()方法。

不期望的用户滚动

带有滚轮的鼠标越来越流行了。按下滚轮并转动它，就可以激活它。注意，即使页面在载入框架或者新窗口时有意锁住滚动条，但只要文档或者其背景图像大于窗口或者框架，都可以通过这个滚轮来滚动页面。用户可能没有意识到他们滚动了页面(因为没有可见的滚动条)。如果这影响了设计，就需要(通过 setTimeout())建立一个例程，定期将窗口滚动到(0, 0)。

scrollBy(*deltaX*,*deltaY*)，scrollTo(*x*,*y*)

返回值：无。

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

现代浏览器提供了一对相关的窗口滚动方法，window.scrollTo()方法是 window.scroll()方法的新版本。两者的工作方式相同，都将文档的指定坐标点定位在内部窗口区域的左上角。

而 window.scrollBy()方法允许对文档进行相对定位，其参数值表示文档在窗口中滚动的像素(水平和垂直)。如果想滚动到左边或者上边，可使用负参数值。如果希望隐藏窗口或者框架的滚动条，并为用户提供其他类型的滚动控制，就可以使用 scrollBy()方法。例如，使长文档向下滚动一整屏，可以用 window.innerHeight(在 NN/Moz 中)或者 document.body. clientHeight(在 IE 中)属性来确定决定从当前文档位置偏移的量：

```
// assign IE body clientHeight to window.innerHeight
if (document.body && document.body.clientHeight)
{
   window.innerHeight = document.body.clientHeight;
}
window.scrollBy(0, window.innerHeight);
```

要向上滚动，第二个参数应使用负值：

```
window.scrollBy(0, -window.innerHeight);
```

窗口滚动方法不能用来建立定位元素的滚动效果，这种动画效果可以通过调整 style 的 position 属性来获得(见配书光盘中的第 43 章)。

更多资源请访问稀酷客(www.ckook.com)

示例

要使用 scrollTo()方法，可使用程序清单 27-30、27-31 和 27-32。程序清单 27-34 中的代码包括了一个控制面板框架(参见程序清单 27-32)，它提供了输入，来试验 scrollBy()方法。

程序清单 27-30　ScrollBy 控制器的框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.scrollBy() Method</title>
   </head>
   <frameset rows="50%,50%">
      <frame src="jsb27-31.html" name="display" />
      <frame src="jsb27-32.html" name="control" />
   </frameset>
</html>
```

程序清单 27-31　要滚动的图像

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Arch</title>
   </head>
   <body>
      <h1>
         A Picture is Worth...
      </h1>
      <hr />
      <center>
         <table border="3">
            <caption align="bottom">
               A Splendid Arch
            </caption>
            <tr>
               <td>
                  <img alt="image" src="arch.gif" />
               </td>
            </tr>
         </table>
      </center>
   </body>
</html>
```

注意，在程序清单 27-32 中，对窗口属性和方法的所有引用都指向 display 框架。从文本域中获得的字符串值用 parseInt()全局函数转换为数字。

更多资源请访问稀酷客(www.ckook.com)

程序清单 27-32　ScrollBy 控制器

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>ScrollBy Controller</title>
      <script type="text/javascript">
         function page(direction)
         {
            var pixFrame = parent.display;
            var deltaY = (pixFrame.innerHeight) ? pixFrame.innerHeight :
               pixFrame.document.body.scrollHeight;
            if (direction == "up")
            {
               deltaY = -deltaY;
            }
            parent.display.scrollBy(0, deltaY);
         }
         function customScroll(form)
         {
            parent.display.scrollBy(parseInt(form.x.value), parseInt(form.y.value));
         }
      </script>
   </head>
   <body>
      <h1>ScrollBy Controller</h1>
      <hr />
      <form name="custom">
         Enter an Horizontal increment
         <input type="text" name="x" value="0" size="4" />
         and Vertical
         <input type="text" name="y" value="0" size="4" /> value.
         <br />
         Then
         <input type="button" value="click to scrollBy()"
               onclick="customScroll(this.form)" />
         <hr />
         <input type="button" value="PageDown" onclick="page('down')" />
         <input type="button" value="PageUp" onclick="page('up')" />
      </form>
   </body>
</html>
```

相关主题：window.pageXOffset 属性，window.pageYOffset 属性；window.scroll()方法。

scrollByLines(*intervalCount*)，scrollByPages(*intervalCount*)

返回值：无。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

更多资源请访问稀酷客(www.ckook.com)

window.scrollByLines()和 window.scrollByPages()方法分别将文档滚动指定的行数或页数。这些方法是单击浏览器垂直滚动条上的箭头(scrollByLines())和翻页(scrollByPages())区域的脚本实现方式。每个方法的参数都指定要滚动多少行或页，正值为向下滚动，负值为向上滚动。

setInterval("*expr*", *msecDelay* [, *language*])，
setInterval(funcRef, msecDelay [, funcarg1, . . . ,funcargn])

返回值：Interval ID 整数值。

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

理解 setInterval()和 setTimeout()方法的区别非常重要。在 setInterval()方法添加到 JavaScript 中之前，编程者使用 setTimeout()模拟该方法的行为，但通常要对脚本稍做改动。

当脚本需要延迟固定的时间，以重复调用一个函数或者执行某个表达式时，就可以使用 setInterval()方法。在某些语言中，延迟根本不像是在等待：因为延迟一个进程时，其他进程并不停止。典型应用程序包含的动画会以受控的速度在页面上移动对象(而不是让 JavaScript 解释器根据 CPU 的速度沿着路径移动对象)。在 kiosk 应用程序中，可以用 setInterval()加快幻灯片显示在其他框架上或者显示为图层的速度，可能每 10s 改变一次视图。时钟显示和倒计时器也可以使用这个方法(本书中还列举了用旧式 setTimeout ( )方法实现计时器和时钟功能的例子)。

而 setTimeout()最适用于将来只执行一次的函数或者表达式，即使这个将来只是两三秒之后。换句话说，setTimeout()提供了一个一次性计时器，而 setInterval()提供了一个循环计时器。此应用的细节请参见本章稍后讨论的 setTimeout()方法。

虽然 setInterval()方法的主要功能在所有浏览器中都相同，但 NN/Moz 和 IE 依据方法的参数，提供了一些额外的功能。在这个方法的简单调用中，同样的参数可以用在支持此方法的所有浏览器中。下面首先给出所有浏览器通用的参数。

setInterval()方法的第一个参数是在指定的时间过后要运行的函数名或者表达式，这一项必须是一个带引号的字符串。如果参数是函数，则不允许在函数的括号中出现参数，除非该参数是字面量字符串(参见下面的“传递函数参数”)。

这个方法的第二个参数是 JavaScript 用作函数或者表达式调用间隔的微秒数。尽管度量单位非常小，但时间间隔不会百分之百地准确，不同内部处理的延迟可能有轻微的差异。

与 setTimeout()一样，setInterval()也返回整数，这个整数是间隔处理程序的 ID 值。该 ID 值允许用 clearInterval()方法关闭此处理程序，ID 值是 clearInterval()方法的唯一参数。这种机制允许设置多个同时运行的间隔进程，脚本可以随时停止单个进程，而不影响其他进程。

IE4+用可选的第三个参数指定第一个参数调用的语句或者函数所使用的脚本语言。只要完全用 JavaScript 编程(JScript 与此相同)，就没必要包含这个参数。

6. 传递函数参数

NN4+/Moz 提供了一种机制，可将参数传递到 setInterval()调用的函数中。为了使用这种机制，setInterval()的第一个参数决不能是字符串，而必须是函数引用(没有括号)；第二个参数仍是延迟时间。但从第三个参数开始，可将函数的参数放在一个以逗号间隔的列表中。

```
intervalID = setInterval(cycleAnimation, 500, "figure1");
```


更多资源请访问稀酷客(www.ckook.com)

该函数的参数形式与其他函数相同：

```
function cycleAnimation(elemID) {...}
```

为在更多的浏览器版本中使用setInterval()，也可添加一个功能，将参数传递给由setInterval()调用的函数。因为对其他函数的调用是一个字符串表达式，所以也可以通过字符串连接操作，将计算出来的值作为字符串的一部分。例如，如果一个函数用事件处理程序确定用户单击了哪个元素(来启动某个动画序列)，该元素的 ID(由一个变量引用）就可以传递到由 setInterval ()调用的函数中：

```
function findAndCycle()
{
   var elemID;
   // statements here that examine the event info
   // and extract the ID of the clicked element,
   // assigning that ID to the elemID variable
   intervalID = setInterval("cycleAnimation(" + elemID + ")", 500);
}
```

如果在 setInterval()中每次调用函数时，都需要传递不断变化的参数，就可以在函数的末尾改用 setTimeout()，再次调用相同的函数。

示例

演示 setInterval()方法需要一个双框架环境，框架集文档见程序清单 27-33。

程序清单 27-33 SetInterval()演示框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>setInterval() Method</title>
   </head>
   <frameset rows="50%,50%">
      <frame src="jsb27-34.html" name="control" />
      <frame src="bofright.html" name="display" />
   </frameset>
</html>
```

在顶部框架中是一个控制面板，其中的几个按钮控制着底部框架中一个法案的文本文档的自动滚动。程序清单 27-34 给出了控制面板文档，其中的许多函数控制着间隔、滚动跳转大小和方向，它们演示了应用 setInterval()的几个方面。

注意，脚本首先建立了几个全局变量，其中三个变量是控制滚动的参数，最后一个变量用于存储 setInterval()方法返回的 ID 值。脚本要求该值是全局值，这样另一个函数可以用 clear Interval()方法停止滚动。

所有滚动都由 autoScroll()函数完成。为简洁起见，所有控制参数都是全局变量。在此应用程序中，将这些值存储在全局变量中，有助于页面用上次运行的参数重新开始自动滚动。

更多资源请访问稀酷客(www.ckook.com)

程序清单 27-34　SetInterval()控制面板

HTML: jsb27-34.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>ScrollBy Controller</title>
      <script type="text/javascript" src="jsb27-34.js"></script>
   </head>
   <body onload="startScroll()">
      <h1>AutoScroll by setInterval() Controller</h1>
      <hr />
      <form name="custom">
         <input type="button" value="Start Scrolling"
                onclick="startScroll()" />
         <input type="button" value="Stop Scrolling"
                onclick="stopScroll()" />
         <p>
            <input type="button" value="Shorter Time Interval"
                   onclick="reduceInterval()" />
            <input type="button" value="Longer Time Interval"
                   onclick="increaseInterval()" />
         </p>
         <p><input type="button" value="Bigger Scroll Jumps"
                   onclick="increaseJump()" />
            <input type="button" value="Smaller Scroll Jumps"
                   onclick="reduceJump()" />
         </p>
         <p><input type="button" value="Change Direction"
                   onclick="swapDirection()" />
         </p>
      </form>
   </body>
</html>
```

JavaScript: jsb27-34.js

```
var scrollSpeed = 500;
var scrollJump = 1;
var scrollDirection = "down";
var intervalID;

function autoScroll()
{
   if (scrollDirection == "down")
   {
      scrollJump = Math.abs(scrollJump);
   }
   else if (scrollDirection == "up" && scrollJump > 0)
   {
```


更多资源请访问稀酷客(www.ckook.com)

```
        scrollJump = -scrollJump;
     }
     parent.display.scrollBy(0, scrollJump);
     if (parent.display.pageYOffset <= 0)
     {
        clearInterval(intervalID);
     }
  }

  function reduceInterval()
  {
     stopScroll();
     scrollSpeed -= 200;
     startScroll();
  }
  function increaseInterval()
  {
     stopScroll();
     scrollSpeed += 200;
     startScroll();
  }
  function reduceJump()
  {
     scrollJump -= 2;
  }
  function increaseJump()
  {
     scrollJump += 2;
  }
  function swapDirection()
  {
     scrollDirection = (scrollDirection == "down") ? "up" : "down";
  }
  function startScroll()
  {
     parent.display.scrollBy(0, scrollJump);
     if (intervalID)
     {
        clearInterval(intervalID);
     }
     intervalID = setInterval("autoScroll()",scrollSpeed);
  }
  function stopScroll()
  {
     clearInterval(intervalID);
  }
```

在 startScroll()函数内调用 setInterval()方法，此函数先使页面显示一个 scrollJump 间隔时间，以便在 autoScroll()中测试页面是否一直滚动到顶部时，页面不会在滚动开始前就停止滚动。还要注意，函数还检查间隔 ID 是否存在，如果存在，就先清除它，再设置新的间隔 ID。这在示

更多资源请访问稀酷客(www.ckook.com)

例页面的设计中很关键，因为重复单击 Start Scrolling 按钮，会在浏览器中触发多个间隔计时器，只有最近触发的间隔计时器的 ID 会存储在 intervalID 中，且无法清除旧的间隔计时器，这个小分支可确保只有一个间隔计时器在运行。全局变量 scrollSpeed 用于为 setInterval()提供延时参数。要动态修改这个值，脚本必须停止当前间隔进程，更改 scrollSpeed 值，再开始一个新进程。

setInterval()方法很好地处理了此应用程序的高度重复性。

相关主题：window.clearInterval()，window.setTimeout()方法。

setTimeout("*expr*", *msecDelay* [, *language*])
setTimeout(*functionRef*, *msecDelay* [, *funcarg1*, . . . ,*funcargn*])

返回值：用于 window.clearTimeout()方法的 ID 值。

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

这个方法的名字容易让人误解，尤其是编写过其他类型的超时程序时。在 JavaScript 中，超时是表达式执行之前的一段时间(单位是微秒)。超时不是等待或者脚本延迟，而是告诉 JavaScript 延期一定的时间，再执行语句或脚本，setTimeout()语句后的其他语句会立即执行。

假定要设计一个网页，允许用户在一定的时间限制内与各种按钮或域进行交互(这是一个独立运行的网页)。此时可以打开窗口的超时功能，如果在两分钟(120 000ms)内没有与文档底部的特定按钮或者域进行交互，窗口就返回文档顶部或者显示帮助屏幕。若用户在指定时间内进行导航，就告诉窗口关闭超时功能，这时可以让用户单击的按钮调用 clearTimeout()方法(它与 setTimeout()正好相反)，来取消当前的计时器(本章前面解释了 clearTimeout()方法)。多个计时器可同时运行，而且相互独立。

虽然 setTimeout()方法的主要功能在所有浏览器中都相同，但 NN/Moz 和 IE 依据方法的参数，提供了一些额外功能。在这个函数的简单调用中，同样的参数可以用在支持此方法的所有浏览器中。下面首先给出所有浏览器共用的参数。

window.setTimeout()方法的第一个参数是一个带引号的字符串，它可以包含对任何函数、方法或者单个 JavaScript 语句的调用，该调用将在超时结束后执行。

超时并不终止脚本的执行，理解这一点是非常重要的。实际上，如果在脚本中间使用 setTimeout()方法，其后面的语句会立即执行；在延迟时间之后，就执行 setTimeout()中的表达式。因此，在脚本中设计超时的最好方法是将其放在函数的末尾：先执行所有其他的语句，然后用 setTimeout()方法停止执行脚本，直至超时为止。事实上，尽管一直处于超时状态，用户仍可以执行其他任务。在超时计时器运行后，不能调整超时的时间，但可以清除超时，然后打开一个新的计时器。

如果要在函数中使用 setTimeout()作为延迟机制，可将函数分成两部分，把 setTimeout()用作两者的桥梁。如程序清单 27-25 所示，IE 需要一点延迟时间来打开窗口，以便在其中写入内容。如果没有需要的延迟时间，组合 HTML 好内容写入操作将在打开新窗口的同一函数中完成。

使用 setTimeout()方法调用包含它的函数是很常见的。例如，假定编写一个 Java applet 程序，为页面执行某个额外的任务，且需要通过 NPAPI 来连接它，所以脚本必须等待 applet 加载和初始化完成。文档中的 onload 事件处理程序确保 applet 对象对脚本可见，但它不知道 applet 是否完成了初始化。JavaScript 函数每 500 微秒检查一次 applet，直到 applet 设置了某个表示已完成初始化的内部值为止，如下所示：

更多资源请访问稀酷客(www.ckook.com)

```
var t;
function autoReport()
{
   if (!document.myApplet.done)
   {
      t = setTimeout("autoReport()",500);
   }
   else
   {
      clearTimeout(t);
      // more statements using applet data //
   }
}
```

JavaScript 没有给 wait 命令提供内置的等效命令，最差的选择是自己设计一个循环函数，每隔固定的时间就检查一次脚本的执行情况。不过，这种方法会阻止其他进程的执行，因此应考虑改用 setTimeout()方法。

NN4+/Moz 为 setTimeout()调用的函数提供了一种传递参数的机制，这个主题参见 window.setInterval()的“传递函数参数”部分，这部分还讨论了其他浏览器版本的参数传递方式。

富有经验的程序员需要注意，setInterval()和 setTimeout()都不会生成新线程，来运行其调用的脚本。如果计时器结束后调用一个函数，这个进程就放在 JavaScrip 执行线程的脚本处理队列的末尾。

示例

载入程序清单 27-35 中的 HTML 页面时，会触发 updateTime()函数，该函数在状态栏中显示时间(格式是 hh:mm am/pm)。函数使最后一个字符交替显示为星号或不显示，就像可视心跳一样，而不是显示逐渐增加的秒数(这可能分散阅读页面的人的注意力)。

程序清单 27-35　显示当前时间

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Status Bar Clock</title>
      <script type="text/javascript" src="../jsb-global.js"></script>
      <script type="text/javascript">
         // initialize when the page has loaded
         addEvent(window, "load", updateTime);

         var flasher = false;
         // calculate current time, determine flasher state,
         // and insert time into status bar every second
         function updateTime()
         {
            var now = new Date();
            var theHour = now.getHours();
            var theMin = now.getMinutes();
            var theTime = "" + ((theHour > 12) ? theHour - 12 : theHour);
```


更多资源请访问稀酷客(www.ckook.com)

```
            theTime += ((theMin < 10) ? ":0" : ":") + theMin;
            theTime += (theHour >= 12) ? " pm" : " am";
            theTime += ((flasher) ? " " : "*");
            flasher = !flasher;
            window.status = theTime;
            // recursively call this function every second to keep timer going
            timerID = setTimeout("updateTime()",1000);
         }
      </script>
   </head>
   <body>
      <h1>Status Bar Clock</h1>
      <hr />
   </body>
</html>
```

在此函数中，setTimeout()方法按以下方式工作：当前时间(包括闪烁状态)显示在状态栏上时，函数将等待约 1 秒钟(1000ms)，之后再次调用同一函数。在此应用程序中不必清除 timerID 值，因为 JavaScript 每隔 1000ms 自动清除一次 timerID 值。

一个逻辑问题是：此应用程序是否应使用 setInterval 替代 setTimeout()？这两个方法都能完成任务。这里使用 setInterval()，就要求间隔进程在 updateTime()函数之外启动，因为只需要运行一个反复调用 updateTime()的进程。这是一个更简洁的实现方式，不像程序清单 27-35 那样会产生大量的超时进程。另一方面，应用程序不会像程序清单 27-35 那样在过时浏览器中运行，现在这可能不是一个问题，但这个应用程序仍然是使用 setTimeout()函数的一个很好示例。

为演示参数的传递，可修改 updateTime()函数，将它的调用次数显示到状态栏中。为此，函数必须有一个参数变量，setTimeout()的表达式在每次调用函数时，该变量都能获得一个新值。对于所有浏览器，函数的修改如下(未修改的行用省略号表示)：

```
function updateTime(i)
{
   //...
   window.status = theTime + " (" + i + ")";
   // pass updated counter value with next call to this function
   timerID = setTimeout("updateTime(" + i+1 + ")",1000);
}
```

如果只在 NN4+/Moz 中运行此脚本，可用更简便的方式给函数传递参数：

```
timerID = setTimeout(updateTime, 1000, i+1);
```

在这两种情况下，还必须修改 onload 事件处理程序，用一个初始参数启动它：

```
onload = "updateTime(0)";
```

相关主题：window.clearTimeout()，window. setInterval()，window.clearInterval()方法。

showHelp("*URL*",["*contextID*"])

返回值：无。

更多资源请访问稀酷客(www.ckook.com)

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 showHelp()方法允许脚本使用一种特殊的.hlp 文件打开 Winhelp 窗口，这种方法专用于 Win32 操作系统。

如果 Winhelp 文件在不同的地方指定了上下文标识符，就可将 ID 作为可选的第二个参数。这样，调用 showHelp()将会导航到.hlp 文件中适用于页面上指定元素的区域。

有关创建 Winhelp 文件的详细内容，请参见 Microsoft Visual Studio 创作环境。

showModalDialog("*URL*"[, *arguments*][, *features*]),
showModelessDialog("*URL*"[, *arguments*][, *features*])

返回值：returnValue(模态对话框)或 window 对象(非模态对话框)。

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari 2.01+，Opera-，Chrome+

IE、Safari、Chrome 和基于 Mozilla 的浏览器(如 Firefox)提供了多种打开模态对话框窗口的方法，这种窗口总是位于主浏览器窗口的前面，使用户不能访问主窗口。在 WinIE5 中，Microsoft 增加了非模态对话框，Safari、Chrome 和基于 Mozilla 的浏览器(如 Firefox)不支持它。非模态对话框也位于主窗口的前面，但允许用户访问主窗口中所有可见的内容。将 URL 作为第一个参数，该方法可以将任何 HTML 页面或者图片载入对话框窗口。另外，可以使用可选参数将数据传输到对话框中，还可以有效地控制窗口的外观。在 NN/Moz 中，可以通过 window.open Dialog()方法使用类似的对话框窗口。

用这两个方法生成的窗口几乎都是完整的 window 对象，它们具有一些有用的额外属性。其中最重要的属性是 window.dialogArgument，它允许脚本读入通过 showModalDialog()和 showModelessDialog()的第二个参数传递到窗口的数据。所传递的数据可以是任何有效的 JavaScript 数据类型，包括对象和数组。

对脚本来说，显示模态对话框会产生一些分支。特别是只要模态对话框可见，在主窗口中执行的脚本将在调用 showModalDialog()方法的语句处停止执行。在这期间，脚本会在对话框窗口中运行。用户关闭对话框后，脚本就继续在主窗口中执行。另一方面，显示非模态对话框的调用不会停止执行脚本，因为主页或者对话框窗口的脚本能与其他窗口进行实时通信。

7. 获得对话框数据

为在模态对话框窗口中将数据送回主窗口的脚本，对话框窗口的脚本可以将 window.returnValue 属性设置为任何 JavaScript 值。这个值应赋予一个变量，该变量接收 setModelDialog()方法的返回值，如下例所示：

```
var specifications = window.showModalDialog("preferences.html");
```

返回数据的组成和内容由脚本决定，数据不会自动返回。

因为非模态对话框与活动的主页窗口并存，所以数据的返回不像模态对话框那样直接。showModelessDialog()方法中第二个参数的任务不是将参数传递给对话框；而是，如果在主窗口的脚本中定义了全局变量或者函数，就可以将变量或者函数的引用作为第二个参数，来显示非模态对话框。这样，非模态对话框中的脚本就能利用那个引用，在对话框关闭之前(或者用户单击了 Apply 按钮或其他控件后)，将数据返回给主窗口。这个机制甚至允许将数据传回给主窗口

更多资源请访问稀酷客(www.ckook.com)

中的函数。例如，假如主窗口定义了一个函数，如下：

```
function receivePrefsDialogData(a, b, c)
{
   // statements to process incoming values //
}
```

然后在打开窗口时，将一个引用传递给这个函数：

```
dlog = showModelessDialog("prefs.html", receivePrefsDialogData);
```

对话框窗口文档中的脚本语句能得到此引用，这样其他语句也能使用该引用，例如将其用于 Apply 按钮的 onclick 事件处理函数：

```
var returnFunc = window.dialogArguments;
//...
function apply(form)
{
   returnFunc(form.color.value, form.style.value, form.size.value);
}
```

虽然这种方法在对话框打开时，似乎妨碍了向其中传递参数，但总可以在主窗口的脚本中引用对话框，直接设置表单或者变量的值：

```
dlog = showModelessDialog("prefs.html", receivePrefsDialogData);
dlog.document.forms[0].userName.value = GetCookie("userName");
```

注意，用这些方法中的一种打开的对话框不能通过 opener 属性与源窗口保持连接，这两种对话框都未定义 opener 属性。

8. 对话框窗口特征

这两个方法都提供了第三个可选属性，它可以指定对话框窗口的显示特征，忽略此属性会将所有特征设置为默认值。所有的参数都放在同一个字符串中，而且每个参数的名-值对都使用 CSS 的 attribute:value 语法。表 27-5 列出了两种窗口类型的所有窗口特征，如果设计目标是为了与 IE4 兼容，则只能使用模态对话框及窗口特征的一个子集，如表 27-5 所示。Boolean 值只有 4 种：yes、no、1、0。

这些特征的 CSS 类型语法允许在字符串中用分号将多个特征组合在一起，例如：

```
var dlogData = showModalDialog("prefs.html", defaultData,
"dialogHeight:300px; dialogWidth:460px; help:no");
```

表 27-5　IE 对话框窗口的特征

特　征	类　型	默 认 值	说　明
Center	Boolean	yes	对话框是否居中 (被 dialogLeft 和 / 或 dialogTop 覆盖)
dialogHeight	Length	Varies	对话框窗口的外部高度，IE4 默认的长度单位为 em，IE5+/Moz/Safari/Chrome 的单位是像素(px)

更多资源请访问稀酷客(www.ckook.com)

(续表)

特　征	类　型	默 认 值	说　明
dialogLeft	Integer	Varies	对话框与屏幕左边缘的距离，以像素为单位
dialogTop	Integer	Varies	对话框与屏幕顶边缘的距离，以像素为单位
dialogWidth	Length	Varies	对话框窗口的外部宽度，IE4 默认的长度单位为 em，IE5+/Moz/Safari/Chrome 的单位是像素
edge	String	raised | sunken	边框样式(仅用于 IE)
resizable	Boolean	no	对话框可以调整大小
scroll	Boolean	yes	显示滚动条
status	Boolean	Varies	在窗口底部显示状态栏(仅 IE5+/Safari/Chrome)，对于不信任的对话框默认为 yes，对于信任的对话框默认为 no

虽然未将滚动条明确列为窗口特征之一，但若内容超出了对话框的指定大小或者可用大小，滚动条通常就会显示在窗口中。如果不想显示滚动条，则可以在页面打开时，将对话框文档脚本的 document.body.scroll 属性设置为 false。

9. 对话框警告

一个潜在的用户问题是，一般情况下，只有加载对话框的 HTML 文件，才打开对话框窗口。因此，在加载复杂的文档之前，如果有很长时间的延迟，用户将看不到任何信息，指示正在加载文档。为此，可以试着设置 cursor 样式表属性，并在加载对话框文档时显示它。

示例

为了演示这两种对话框样式，本例为模态和非模态对话框实现了相同的功能(设置某些会话可见首选项)。这个策略说明了如何在主页面和两种样式的对话框窗口之间来回传递数据。

第一个示例演示了如何使用模态对话框。在该过程中，数据传递到对话框窗口中，并返回一些值。程序清单 27-36 是主页面的 HTML 和脚本，一个按钮的 onclick 事件处理程序调用了打开模态对话框的函数。对话框的文档(参见程序清单 27-37)包含了几个表单元素，用于输入用户名，为主页选择一些颜色样式。来自对话框的数据放在一个要发送回主窗口的数组中，在对话框关闭时，就把该数组赋给一个本地变量 prefs。如果用户取消对话框，就返回一个空字符串，因此 getPrefsData()中没有要执行的语句；但如果用户单击 OK 按钮，就会返回该数组。然后读取每个数组项，并赋予对应的表单值或样式属性，这些值也保存在全局数组 currPrefs 中，这样就可以在下次打开对话框时，把设置发送到模态对话框中(作为 showModalDialog()的第二个参数)。

程序清单 27-36　showModalDialog()的主页面

HTML: jsb27-36.html

```
<html>
   <head>
      <title>window.setModalDialog() Method</title>
      <script type="text/javascript" src="jsb27-36.js"></script>
   </head>
   <body bgcolor="#EEEEEE" style="margin:20px" onload="init()">
```


更多资源请访问稀酷客(www.ckook.com)

```
      <h1>window.setModalDialog() Method</h1>
      <hr />
      <h2 id="welcomeHeader">Welcome, <span id="firstName">.</span>!</h2>
      <hr />
      <p>Use this button to set style preferences for this page:
         <button id="prefsButton" onclick="getPrefsData()">Preferences</button>
      </p>
   </body>
</html>
```

JavaScript: jsb27-36.js

```
var currPrefs = new Array();
var nameHolder;

function getPrefsData()
{
   var prefs = showModalDialog("jsb27-37.html", currPrefs,
               "dialogWidth:400px; dialogHeight:300px;");
   if (prefs)
   {
      if (prefs["name"])
      {
         if (nameHolder.textContent == "friend")
         {
            nameHolder.textContent = prefs["name"];
         }
         else
         {
            //IE
            nameHolder.innerText = prefs["name"];
         }
         currPrefs["name"] = prefs["name"];
      }
      if (prefs["bgColor"])
      {
         document.body.style.backgroundColor = prefs["bgColor"];
         currPrefs["bgColor"] = prefs["bgColor"];
      }
      if (prefs["textColor"])
      {
         document.body.style.color = prefs["textColor"];
         currPrefs["textColor"] = prefs["textColor"];
      }
      if (prefs["h1Size"])
      {
         if (document.all)
         {
            document.all.welcomeHeader.style.fontSize = prefs["h1Size"];
         }
         else
```


更多资源请访问稀酷客(www.ckook.com)

```
            {
               document.getElementById("welcomeHeader").style.fontSize
               = prefs["h1Size"];
            }
            currPrefs["h1Size"] = prefs["h1Size"];
         }
      }
}
function init()
{
   nameHolder = document.getElementById("firstName");
   if (nameHolder.textContent == ".")
   {
      nameHolder.textContent = "friend";
   }
   else
   {
      //IE
      nameHolder.innerText = "friend";
   }
}
```

程序清单 27-37 中的对话框文档负责读取输入的数据(并相应地设置表单元素)，组装表单数据，以返回给主窗口的脚本。注意，在载入示例时，对话框文档的 title 元素会显示在对话框窗口的标题栏上。

页面加载到对话框窗口中时，init()函数就会检查 window.dialogArguments 属性。如果该属性包含数据，就使用这些数据来预置表单元素，以反映主页的当前设置。setSelected()实用函数预先选择 select 元素的选项，来匹配当前设置。

页面底部的按钮显式定位在窗口的右下角。每个按钮都调用一个函数，来执行关闭对话框所需的操作。在单击 OK 按钮时，handleOK()函数将 window.returnValue 属性设置为 getFormData()函数返回的数据。getFormData()函数读取表单元素的值，并将这些元素打包在一个数组中，用表单元素名称作为数组下标。这有助于这些元素在主窗口的脚本中有序排列，而脚本使用下标名称，就不必依赖表单元素在对话框窗口中的准确顺序。

程序清单 27-37　模态对话框的文档

HTML: jsb27-37.html

```
<html>
   <head>
      <title>User Preferences</title>
      <script type="text/javascript" src="jsb27-37.js"></script>
   </head>
   <body bgcolor="#EEEEEE" onload="init()">
      <h2>Web Site Preferences</h2>
      <hr />
      <form name="prefs" onsubmit="return false">
         <table>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <tr>
           <td>Enter your first name:
              <input name="name" type="text" value=""
                     size="20" onkeydown="checkEnter()" />
           </td>
        </tr>
        <tr>
           <td>Select a background color:
              <select name="bgColor">
                 <option value="beige">Beige</option>
                 <option value="antiquewhite">Antique White</option>
                 <option value="goldenrod">Goldenrod</option>
                 <option value="lime">Lime</option>
                 <option value="powderblue">Powder Blue</option>
                 <option value="slategray">Slate Gray</option>
              </select>
           </td>
        </tr>
        <tr>
           <td>Select a text color:
              <select name="textColor">
                 <option value="black">Black</option>
                 <option value="white">White</option>
                 <option value="navy">Navy Blue</option>
                 <option value="darkorange">Dark Orange</option>
                 <option value="seagreen">Sea Green</option>
                 <option value="teal">Teal</option>
              </select>
           </td>
        </tr>
        <tr>
           <td>Select "Welcome" heading font point size:
              <select name="h1Size">
                 <option value="12">12</option>
                 <option value="14">14</option>
                 <option value="18">18</option>
                 <option value="24">24</option>
                 <option value="32">32</option>
                 <option value="48">48</option>
              </select>
           </td>
        </tr>
     </table>
  </form>
  <div style="position:absolute; left:200px; top:220px">
     <button style="width:80px"
             onclick="handleOK()">OK</button>
       
     <button style="width:80px"
             onclick="handleCancel()">Cancel</button>
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb27-37.js

```
// Close the dialog
function closeme()
{
   window.close();
}

// Handle click of OK button
function handleOK()
{
   window.returnValue = getFormData();
   closeme();
}

// Handle click of Cancel button
function handleCancel()
{
   window.returnValue = "";
   closeme();
}

// Generic function converts form element name-value pairs
// into an array
function getFormData()
{
   var form = document.prefs;
   var returnedData = new Array();
   // Harvest values for each type of form element
   for (var i = 0; i < form.elements.length; i++)
   {
      if (form.elements[i].type == "text")
      {
         returnedData[form.elements[i].name] = form.elements[i].value;
      }
      else if (form.elements[i].type.indexOf("select") != -1)
      {
         returnedData[form.elements[i].name] =
           form.elements[i].options[form.elements[i].selectedIndex].value;
      }
      else if (form.elements[i].type == "radio")
      {
         returnedData[form.elements[i].name] = form.elements[i].value;
      }
      else if (form.elements[i].type == "checkbox")
      {
         returnedData[form.elements[i].name] = form.elements[i].value;
      }
```


更多资源请访问稀酷客(www.ckook.com)

```
        else
        {
            continue;
        }
    }
    return returnedData;
}

// Initialize by setting form elements from passed data
function init()
{
    if (window.dialogArguments)
    {
        var args = window.dialogArguments;
        var form = document.prefs;
        if (args["name"])
        {
            form.name.value = args["name"];
        }
        if (args["bgColor"])
        {
            setSelected(form.bgColor, args["bgColor"]);
        }
        if (args["textColor"])
        {
            setSelected(form.textColor, args["textColor"]);
        }
        if (args["h1Size"])
        {
            setSelected(form.h1Size, args["h1Size"]);
        }
    }
}

// Utility function to set a SELECT element to one value
function setSelected(select, value)
{
    for (var i = 0; i < select.options.length; i++)
    {
        if (select.options[i].value == value)
        {
            select.selectedIndex = i;
            break;
        }
    }
    return;
}

// Utility function to accept a press of the
// Enter key in the text field as a click of OK
function checkEnter()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
   if (window.event.keyCode == 13)
   {
      handleOK();
   }
}
```

对话框窗口中的最后一个简便特征是文本框中的 onkeypress 事件处理程序，它调用的函数需要回车键来启动。如果当焦点在文本框中时按下回车键，就会调用 handleOK()函数，就像用户单击了 OK 按钮一样。此功能使对话框的 OK 按钮变成自动默认按钮，就像真正的对话框一样。

为将模态方法转变为非模态方法，本例进行了一些重要的结构修改。程序清单 27-38 给出了主窗口的修改版本，以使用非模态对话框。它初始化另一个全局变量 prefsDlog，以存储 showModelessWindow()方法返回的非模态窗口引用。该变量不仅用于调用非模态对话框中的 init ()函数，也作为包含对话框生成语句的 if 结构中的条件。这样做是为了防止创建对话框的多个实例(在显示非模态窗口时，按钮仍然是活动的)。只要 prefsDlog 有值，且未关闭对话框窗口(检索对话框窗口的 window.closed 属性)，就不会再次创建该对话框。

showModelessDialog()方法的第二个参数是主窗口中更新主文档的函数引用。如后面所述，用户单击 OK 或 Apply 按钮时，就从对话框中调用该函数。

程序清单 27-38 showModelessDialog()的主页

HTML: jsb27-38.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.setModelessDialog() Method</title>
      <script type="text/javascript" src="jsb27-38.js"></script>
   </head>
   <body bgcolor="#EEEEEE" style="margin:20px" onload="init()">
      <h1>window.setModelessDialog() Method</h1>
      <hr />
      <h2 id="welcomeHeader">Welcome, <span id="firstName"> </span>!</h2>
      <hr />
      <p>Use this button to set style preferences for this page:
      <button id="prefsButton"
              onclick="getPrefsData()">Preferences</button>
      </p>
   </body>
</html>
```

JavaScript: jsb27-38.js

```
var currPrefs = new Array();
var prefsDlog;
function getPrefsData()
{
   if (!prefsDlog || prefsDlog.closed)
```


更多资源请访问稀酷客(www.ckook.com)

```
    {
      prefsDlog = showModelessDialog("jsb27-39.html", setPrefs,
                  "dialogWidth:400px; dialogHeight:300px");
      prefsDlog.init(currPrefs);
    }
}

function setPrefs(prefs)
{
    if (prefs["bgColor"])
    {
      document.body.style.backgroundColor = prefs["bgColor"];
      currPrefs["bgColor"] = prefs["bgColor"];
    }
    if (prefs["textColor"])
    {
      document.body.style.color = prefs["textColor"];
      currPrefs["textColor"] = prefs["textColor"];
    }
    if (prefs["h1Size"])
    {
      document.all.welcomeHeader.style.fontSize = prefs["h1Size"];
      currPrefs["h1Size"] = prefs["h1Size"];
    }
    if (prefs["name"])
    {
      document.all.firstName.innerText = prefs["name"];
      currPrefs["name"] = prefs["name"];
    }
}

function init()
{
    document.all.firstName.innerText = "friend";
}
```

为实现非模态版本(参见程序清单 27-39)，只会十分有限地修改对话框窗口文档。只是在屏幕底部添加了一个新的 Apply 按钮。与 Microsoft 产品中的许多对话框窗口一样，Apply 按钮允许将对话框中的当前设置应用于当前文档，而不关闭对话框，这种方法更便于试验各种设置。

Apply 按钮调用 handleApply()函数，除了不关闭对话框之外，其工作方式与 handleOK()相同。但这两个函数把数据从模态对话框返回给主窗口的方式不同。主窗口的处理函数传递为 showModelessDialog()的第二个参数，并可在对话框窗口的脚本中用作 window.dialogArguments 属性，在两个函数中，该函数引用都赋给一个本地变量，并调用远程函数，将 getFormData()函数的结果作为参数值传递回主窗口。

程序清单 27-39　非模态对话框的文档

HTML: jsb27-39.html

```
<!DOCTYPE html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>User Preferences</title>
      <script type="text/javascript" src="jsb27-39.js"></script>
   </head>
   <body bgcolor="#EEEEEE" onload="init()">
      <h2>Web Site Preferences</h2>
      <hr />
      <form name="prefs" onsubmit="return false">
         <table>
            <tr>
               <td>Enter your first name:
                  <input name="name" type="text" value=""
                         size="20" onkeydown="checkEnter()" />
               </td>
            </tr>
            <tr>
               <td>Select a background color:
                  <select name="bgColor">
                     <option value="beige">Beige</option>
                     <option value="antiquewhite">Antique White</option>
                     <option value="goldenrod">Goldenrod</option>
                     <option value="lime">Lime</option>
                     <option value="powderblue">Powder Blue</option>
                     <option value="slategray">Slate Gray</option>
                  </select>
               </td>
            </tr>
            <tr>
               <td>Select a text color:
                  <select name="textColor">
                     <option value="black">Black</option>
                     <option value="white">White</option>
                     <option value="navy">Navy Blue</option>
                     <option value="darkorange">Dark Orange</option>
                     <option value="seagreen">Sea Green</option>
                     <option value="teal">Teal</option>
                  </select>
               </td>
            </tr>
            <tr>
               <td>Select "Welcome" heading font point size:
                  <select name="h1Size">
                     <option value="12">12</option>
                     <option value="14">14</option>
                     <option value="18">18</option>
                     <option value="24">24</option>
                     <option value="32">32</option>
                     <option value="48">48</option>
```


更多资源请访问稀酷客(www.ckook.com)

```
                    </select>
                 </td>
              </tr>
           </table>
        </form>
        <div style="position:absolute; left:120px; top:220px">
           <button style="width:80px"
                   onclick="handleOK()">OK</button>
             
           <button style="width:80px"
                   onclick="handleCancel()">Cancel</button>
             
           <button style="width:80px"
                   onclick="handleApply()">Apply</button>
        </div>
    </body>
</html>
```

JavaScript: jsb27-39.js

```
// Close the dialog
function closeme()
{
   window.close();
}

// Handle click of OK button
function handleOK()
{
   var returnFunc = window.dialogArguments;
   returnFunc(getFormData());
   closeme();
}

// Handle click of Apply button
function handleApply()
{
   var returnFunc = window.dialogArguments;
   returnFunc(getFormData());
}

// Handle click of Cancel button
function handleCancel()
{
   window.returnValue = "";
   closeme();
}

// Generic function converts form element name-value pairs
// into an array
function getFormData()
{
```


更多资源请访问稀酷客(www.ckook.com)

```
    var form = document.prefs;
    var returnedData = new Array();
    // Harvest values for each type of form element
    for (var i = 0; i < form.elements.length; i++)
    {
       if (form.elements[i].type == "text")
       {
          returnedData[form.elements[i].name] = form.elements[i].value;
       }
       else if (form.elements[i].type.indexOf("select") != -1)
       {
          returnedData[form.elements[i].name] =
             form.elements[i].options[form.elements[i].selectedIndex].value;
       }
       else if (form.elements[i].type == "radio")
       {
          returnedData[form.elements[i].name] = form.elements[i].value;
       }
       else if (form.elements[i].type == "checkbox")
       {
          returnedData[form.elements[i].name] = form.elements[i].value;
       }
       else
       {
          continue;
       }
    }
    return returnedData;
}

// Initialize by setting form elements from passed data
function init(currPrefs)
{
    if (currPrefs)
    {
       var form = document.prefs;
       if (currPrefs["name"])
       {
          form.name.value = currPrefs["name"];
       }
       if (currPrefs["bgColor"])
       {
          setSelected(form.bgColor, currPrefs["bgColor"]);
       }
       if (currPrefs["textColor"])
       {
          setSelected(form.textColor, currPrefs["textColor"]);
       }
       if (currPrefs["h1Size"])
       {
          setSelected(form.h1Size, currPrefs["h1Size"]);
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
        }
    }

    // Utility function to set a SELECT element to one value
    function setSelected(select, value)
    {
        for (var i = 0; i < select.options.length; i++)
        {
            if (select.options[i].value == value)
            {
                select.selectedIndex = i;
                break;
            }
        }
        return;
    }

    // Utility function to accept a press of the
    // Enter key in the text field as a click of OK
    function checkEnter()
    {
        if (window.event.keyCode == 13)
        {
            handleOK();
        }
    }
```

这些窗口的最大设计挑战是：选择模态还是非模态对话框样式。一些设计人员坚持认为，在图形用户界面中不应使用模态对话框，其他人则认为，在进一步处理之前，有时需要将用户的注意力集中在一项特定任务上，此时就应使用模态对话框。

相关主题：window.open()方法。

sizeToContent()

返回值：无。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

NN6/Moz 的 window.sizeToContent()方法有助于确保把窗口(特别是子窗口)调整到最适合显示窗口内容的尺寸，但必须小心使用这个方法，否则将得不偿失。

可以调用 sizeToContent()方法重置窗口的大小，以显示所有的内容。在 OS 特定的显示环境下进行调整已经过时了。通常应只在一个窗口上执行这个任务，该窗口内容占用的空间很小，甚至比能运行代码的最小显示器还要小(最小显示器一般是 640×480 像素，但对于未来用在掌上计算机的浏览器版本而言，其空间显然更小)。

然而，如果在包含尺寸调整脚本的窗口中调用两次该方法，用户就会遇到问题，因为这可以扩展窗口，使其超出用户视频显示器的像素大小，而后续调用就不能再控制窗口的尺寸，来显示窗口的内容。然而，若调整大小的脚本语句在主窗口中，就可以在子窗口上安全地多次调用该方法。

更多资源请访问稀酷客(www.ckook.com)

示例

在 NN6+/Moz 中使用 The Evaluator(第 4 章)试验 sizeToContent()方法。假设从光盘的 Chap 13 目录(或原样复制到硬盘上的目录)上运行 The Evaluator，就可以用该目录中的其他某个文件打开一个子窗口，然后调整子窗口的大小。在顶部文本框中输入以下语句：

```
a = window.open("jsb13-02.html","")
```

根据光驱所在的磁盘，可能需要修改上述语句中的路径，如下所示：

```
a = window.open("../../Chap13/lst13-02.htm","")
```

根据浏览器版本和浏览器的配置设置，文件在新选项卡或新子窗口中打开。在顶部文本框中输入以下语句：

```
a.sizeToContent()
```

对于大小调整后的窗口(假定打开的是新选项卡)或子窗口，其宽度是浏览器窗口的最小推荐宽度，高度则足以显示文档中的一部分内容。

与改变浏览器窗口大小的方法一样，该方法在支持选项卡浏览(多个页面在同一浏览器实例的不同选项卡中打开)的浏览器(如 Firefox 2)中也会出问题。在支持选项卡的浏览器中，调整一个页面窗口的大小，不可能不改变其他窗口的大小。当然，打开多个浏览器实例可以解决这一问题，但对 window.open ( )的默认响应是打开一个新选项卡，而不是一个新浏览器窗口。

相关主题：window.resizeTo()方法。

stop()

返回值：无。

兼容性：WinIE-，MacIE-，NN4+，Moz+，Safari+，Opera+，Chrome-

NN/Moz 专用的 stop()方法提供了单击工具栏中 Stop 按钮的一个脚本化版本。使用它可在页面中创建自己的工具栏，然后(在使用签名脚本的主窗口或者子窗口中)隐藏它。例如，如果在页面上用图标代表 Stop 按钮，就可以用一个链接指向它，并将链接的动作设为停止加载，如下所示：

```
<a href="javascript: void stop()"><img src="myStop.gif" border="0"></a>
```

脚本不能停止加载包含它自己的文档，但可以停止加载其他框架或者窗口。同样，如果当前文档动态加载新图像或者多媒体 MIME 类型文件是一个独立的动作，stop()方法就能终止这个过程。虽然 stop()方法是 window 对象的一个方法，但它并不与任何窗口或者框架相连，Stop 意味着停止。

相关主题：window.back()、window.find ( )、window.forward()、window.home()、window.print()方法。

更多资源请访问稀酷客(www.ckook.com)

27.3.5　事件处理程序

onafterprint，onbeforeprint

兼容性： WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在用户单击 IE Print 对话框的 OK 按钮后，就会触发这两个事件处理程序，然后执行由手工(通过菜单和浏览器快捷按钮)和 window.print()方法调用的打印操作。

尽管打印效果通常是所见即所得，但有时希望文档的打印版本多于或者少于此刻文档的显示版本。例如，假设在页面送到打印机时，希望在每页的末尾处附加一个特殊的版权声明，则可以在页面加载时，把包含该版权内容的元素的 display 样式表属性设置为 none。而在文档送到打印机之前，脚本需要调整该样式属性，将元素显示为块项目；打印后，脚本再将该设置恢复为 none。

用户单击 Print 对话框中的 OK 按钮后，将会触发 onbeforeprint 事件处理程序。一旦页面送往打印机或者打印缓冲区，就触发 onafterprint 事件处理程序。

示例

以下脚本片断假设页面有一个 div 元素，在加载页面时，其样式表包括一个 display:none 设置。在 Head 部分的某个位置，将与打印相关的事件处理程序设置为属性：

```
function showPrintCopyright()
{
   document.all.printCopyright.style.display = "block";
}
function hidePrintCopyright()
{
   document.all.printCopyright.style.display = "none";
}
window.onbeforeprint = showPrintCopyright;
window.onafterprint = hidePrintCopyright;
```

onbeforeunload

兼容性： WinIE4+，MacIE5+，NN-，Moz+，Safari+，Opera-，Chrome+

只要用户或者脚本卸载或者替换当前页面，就会触发 onbeforeunload 事件处理程序。然而，与 onunload 事件处理程序不同，在真正执行卸载操作之前，onbeforeunload 允许复杂的脚本完成其任务。而且，在事件处理函数中，可以给 event.returnValue 属性指定字符串值。该字符串是警告窗口消息的一部分，在这个警告窗口上，用户可以选择继续留在页面上。如果用户同意留下，页面不会卸载，并会取消任何替换操作。

示例

程序清单 27-40 中的简单页面显示了如何给用户提供继续留在页面上的选项。

程序清单 27-40　使用 onbeforeunload 事件处理程序

```
<html>
   <head>
      <title>onbeforeunload Event Handler</title>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <script type="text/javascript">
            function verifyClose()
            {
                return "We really like you and hope you will stay longer.";
            }
        </script>
    </head>
    <body onbeforeunload="return verifyClose()">
        <h1>onbeforeunload Event Handler</h1>
        <hr />
        <p>Use this button to navigate to the previous page:
            <button id="go"
                    onclick="history.back()">Go Back</button>
        </p>
    </body>
</html>
```

相关主题：onunload 事件处理程序。

onerror

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari-，Opera-，Chrome-

(参见本章前面讨论的 window.onerror 属性)

onhelp

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

一般的 onhelp 事件处理程序在第 26 章中讨论。另外，用户在模态或非模态对话框中激活上下文帮助时，也会触发 onhelp。在后一种情况下，用户可单击对话框标题栏中的帮助图标，此时鼠标指针会变成问号，然后用户可单击窗口中的任何元素。在第二次单击时，会触发 onhelp 事件处理程序，event 对象包含了被单击元素的信息(event.srcElement 是对特定元素的引用)，允许脚本提供该元素的帮助信息。

为防止浏览器的内置帮助窗口显示出来，事件处理程序必须返回 false(IE4+)，或将 event.returnValue 属性设置为 false(IE5+)。

示例

以下脚本段可在对话框中提供上下文帮助。这里只给出了两个表单元素的帮助消息，但在实际应用程序中，可方便地添加更多消息。

```
function showHelp()
{
    switch (event.srcElement.name)
    {
        case "bgColor" :
            alert("Choose a color for the main window\'s background.");
            break;
        case "name" :
            alert("Enter your first name for a friendly greeting.");
```


更多资源请访问稀酷客(www.ckook.com)

```
            break;
        default :
            alert("Make preference settings for the main page styles.");
    }
    event.returnValue = false;
}
window.onhelp = showHelp;
```

由于此页面的帮助信息只关注表单元素，因此 switch 结构基于表单元素的 name 属性。对于其他类型的页面，id 属性可能更加合适。

相关主题：event 对象(第 32 章)；switch 结构(第 21 章)。

onload

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

文档加载完毕时(此时，所有文本和图像元素都已从源文件服务器传输到浏览器，所有插件程序和 Java applet 都已加载，开始运行)，就在当前窗口中触发 onload 事件处理程序。此时，浏览器的内存包含文档中浏览器能识别的所有对象和脚本组件。

在单框架文档中，onload 处理程序是<body>标记的一个特性，在多框架文档中，它是顶层窗口中<frameset>标记的一个特性。若该处理程序是<frameset>标记的一个特性，则只有在框架集定义的所有框架完全载入后，才触发该事件。

在下面的两个例子中，用旧方法将 onload 处理程序插入文档：

```
<html>
    <head>
    </head>
    <body [other attributes] onload="statementOrFunction">
        [body content]
    </body>
</html>

<html>
    <head>
    </head>
    <frameset [other attributes] onload="statementOrFunction">
        <frame [frame specification attributes] />
    </frameset>
</html>
```

另外，还可以使用 W3C 的 addEventListener()方法，将一个事件监听器绑定到 window 对象上。第 32 章介绍了现代跨浏览器的事件处理技术。

在框架集中定义这个处理程序时，有一种特殊的功能：只有框架集中所有子框架的 onload 事件处理程序都触发后，才触发这个 onload 处理程序。因此，如果某些初始化脚本依赖于其他框架中的成员，就在框架集的 onload 事件处理程序中触发它们。编写 JavaScript 时，有一个很好的通用规则：在文档加载期间执行的脚本应有助于生成文档及其对象的进程。为了立即操作那些对象，需要为该窗口设计由 onload 事件处理程序调用的额外函数。

适用于 onload 事件处理程序的操作可以快速运行，且不需要用户介入进来。用户在与页

PDG

更多资源请访问稀酷客(www.ckook.com)

面交互之前，并不需要先等待完成大量的加载后期操作。不应在 onload 处理程序中显示模态对话框。用户在自己的计算机上设计宏，来访问不能访问的站点时，可能会在自动显示警告、确认或者提示对话框的页面上挂起。另一方面，设置 window.defaultStatus 属性之类的操作非常适合于 onload 事件处理程序，将事件处理程序初始化为页面中元素对象的属性也适用于该处理程序。

注意：

安装了弹出窗口阻塞程序的浏览器会忽略在 onload 事件处理函数中调用的所有 window.open()方法。

相关主题：onunload 事件处理程序，window.defaultStatus 属性。

onresize

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

如果用户改变窗口的大小，就会触发 window 对象的 onresize 事件处理程序。如果为事件指定一个函数(例如 window.onresize = handleResizes)，NN/Moz 事件对象就传递 width 和 height 属性，它们表示整个窗口的外部宽度和高度。窗口的大小改变时不应重载文档，否则将触发 onload 事件处理程序(但一些早期的 Navigator 版本不会触发这个事件)。

相关主题：event 对象(见第 32 章)。

onscroll

兼容性：WinIE4+，MacIE4+，NN7+，Moz+，Safari 1.3+，Opera+，Chrome+

通过滚动条或导航键盘键手动滚动文档，或者通过 doScroll()方法、scrollIntoView()方法自动滚动文档，或者调整 body 元素对象的 scrollTop 和/或 scrollLeft 属性，都会触发 body 元素对象的 onscroll 事件处理程序。对于手动滚动和通过 doScroll()进行的滚动，该事件会连续触发两次。另外，即使 body 元素在处理 onscroll 事件处理程序，event.srcElement 属性也是 null。

示例

程序清单 27-41 说明了哪些工具可用于页面设计，这是一个人为痕迹非常明显的演示。假定有一个文档占据了窗口或框架，但即使无意中使用了鼠标滚轮，也不希望滚动它(注意如果用户在 Safari 和 Chrome 的一些 Windows 版本中用鼠标滚轮滚动页面，就不会触发 onscroll 事件，但该事件会在 Mac 版本上触发)。如果滚动内容会破坏内容的外观或价值，就需要确保页面始终位于顶部。程序清单 27-41 中的 onscroll 事件处理程序就实现了这一功能。注意，该事件处理程序设置为 window 对象的一个属性。

程序清单 27-41 防止页面滚动

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onscroll Event Handler</title>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <script type="text/javascript" src="../jsb-global.js"></script>
        <script type="text/javascript">
           addEvent(window, "scroll", zipBack);
           //window.onscroll = zipBack;

           function zipBack()
           {
              window.scroll(0,0);
           }
        </script>
     </head>
     <body>
        <h1>onscroll Event Handler</h1>
        <hr />
        <div>This page always zips back to the top if you try to scroll it.</div>
        <p>
           <iframe frameborder="0" scrolling="no" height="1000"
                   src="bofright.html"></iframe>
        </p>
     </body>
  </html>
```

相关主题：scrollBy()、scrollByLines()、scrollByPages()方法。

onunload

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera-，Chrome+

文档在当前窗口中消失之前，会触发 unload 事件。清空窗口的常见场合是载入新 HTML 文档时，用户单击 back 按钮或 reload 按钮时，用户按下重载组合键时，以及脚本开始为窗口或框架写入新 HTML 时。注意在 Safari 的一些 Windows 版本中，关闭选项卡或浏览器时，不会触发 unload 事件。

onunload 事件处理程序应只用于不禁止从一个文档转换到另一个文档的快速操作，不要在该处理程序中调用任何显示对话框的方法。在文档中，可在 onload 处理程序的位置，把 onunload 事件处理程序指定为单框架窗口的<body>标记特性，或者多框架窗口的<frameset>标记特性。onload 和 onunload 事件处理程序可以出现在同一个<body>或者<frameset>标记中，而不会出问题。onunload 事件处理程序仅仅是安全地隐藏在浏览器的内存中，在文档准备清空窗口时，等待触发 unload 事件。

有关 onunload 事件处理程序的一个警告是：尽管事件在文档消失之前触发，但该处理程序不应执行过于耗时的任务，例如创建新的对象或者提交窗体。文档可能在函数执行完毕之前消失，这样函数就会查找不再存在的对象或值。最好的防范措施是使 onunload 事件处理程序的工作最小化。

注意：

安装了弹出窗口阻塞程序的浏览器将忽略 onunload 事件处理函数调用的所有 window.open()方法。

更多资源请访问稀酷客(www.ckook.com)

相关主题：onload 事件处理程序。

27.4 frame 元素对象

关于 HTML 元素属性、方法和事件处理程序的详细内容，请参见第 26 章。

属　　性	方　　法	事件处理程序	属　　性	方　　法	事件处理程序
allowTransparency			marginHeight		
borderColor			marginWidth		
contentDocument			name		
contentWindow			noResize		
frameBorder			scrolling		
height			src		
longDesc			width		

27.4.1 语法

访问 frameset 中 frame 元素对象的属性或方法：

```
(IE4+)       document.all.frameID. property | method([parameters])
(IE5+/W3C) document.getElementById("frameID"). property |
             method([parameters])
```

在 frame 文档中访问 frame 元素的属性或方法：

```
(IE4+)       parent.document.all.frameID. property | method([parameters])
(IE5+/W3C) parent.document.getElementById("frameID"). property |
             method([parameters])
```

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

27.4.2 关于 frame 对象

如本章开头所述，frame 元素对象与文档层次结构中作为 window 对象的 frame 对象不同。只有所有的 HTML 元素都存在于对象模型中，frame 元素对象才可用于脚本。

因为 frame 元素对象是 HTML 元素，所以它也具有所有 HTML 元素都有的属性、方法和事件处理程序，如第 26 章所述。大体上，可以通过访问 frame 元素对象来设置或修改<frame>标记中的特性值。因此，若为标记的 id 特性指定标识符，就可以简化操作。如果脚本通过源对象模型指向框架(parent.frameName 引用)，则标记仍需要使用 name 特性。虽然对 name 和 id 特性使用同一个标识符并不违反规则，但最好使用不同的名称，以防止在能识别这两个特性的浏览器中，出现潜在的引用冲突。

为了改变框架的大小，必须访问 frameset 元素对象，它定义了框架集的 cols 和 rows 特性，这些属性在现代浏览器中可以随时改变。

更多资源请访问稀酷客(www.ckook.com)

27.4.3 属性

allowTransparency

值：Boolean，　　读/写

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

allowTransparency 属性指定框架背景是否透明。这个属性主要应用于 iframe 对象，因为框架集没有背景色，也没有通过透明框架显示的图像。

borderColor

值：三个十六进制值或颜色名字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

如果框架显示边框(由 frame 元素的 frameborder 特性，或者 frameset 元素的 border 特性决定)，边框就可以设置一个与框架中其他部分不同的颜色。初始颜色(如果与框架集中的其他部分不同)通常由＜frame＞标记的 bordercolor 特性设置，此后，脚本可以根据需要改变颜色设置。

有时修改单框架的边框颜色是很危险的，这取决于自定义颜色组合。实际上，协调冲突或者定义单框架的边框在边框空间中的宽度时，不同的浏览器遵循不同的规则。对单个框架边框颜色的修改不见得显示出来，要在尽可能多的浏览器和操作系统上测试该颜色组合。

示例

尽管在更改单个框架边框的颜色时可能会出问题，但如果脚本是在框架集文档内，W3C DOM 语法应如下所示：

```
document.getElementById("contentsFrame").borderColor = "red";
```

仅适用于 IE 的版本是：

```
document.all["contentsFrame"].borderColor = "red";
```

这些示例假设把框架名作为字符串传送给脚本函数。如果脚本在框架集的某个框架中执行，就在上面的语句中添加对 parent 的引用。

相关主题：frame.frameBorder 属性，frameset.frameBorder 属性。

contentDocument

值：document 对象引用，　　只读

兼容性：WinIE8+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

frame 元素对象的 contentDocument 属性只不过是框架所含文档的引用。这个属性将 frame 元素对象和 frame 对象联系起来。这两个对象包含相同的 document 对象，但在脚本编程中，引用常使用 frame 对象访问框架内的文档，而 frame 元素用于访问与 frag 标记特性等同的属性。但是，如果脚本有对 frame 元素对象的引用，就可以使用 contentDocument 属性获得文档的有效引用，因此获得框架的其他内容。

示例

框架集文档的脚本可以使用 frame 元素的 ID，来读取或调整某个元素属性，然后，通过其

更多资源请访问稀酷客(www.ckook.com)

document 对象对页面内容执行动作。可通过如下语句获得 document 对象的引用：

```
var doc = document.getElementById("Frame3").contentDocument;
```

然后，脚本可以访问文档中的表单：

```
var val = doc.mainForm.entry.value;
```

相关主题：contentWindow 属性；document 对象。

contentWindow

值：document 对象引用， 只读

兼容性：WinIE5.5+，MacIE-，NN7+，Moz1.0.1+，Safari+，Opera+，Chrome+

frame 元素对象的 contentWindow 属性就是框架所含文档的引用。这个属性可以访问框架的窗口，接着就可以使用该窗口访问框架内的文档。

示例

可以通过如下语句，来获得与框架相关联的 window 对象的引用：

```
var win = document.getElementById("Frame3").contentWindow;
```

相关主题：window 对象。

frameBorder

值：yes | no | 1 | 0 的字符串形式， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

frameBorder 属性可以通过脚本访问 frame 元素对象的 frameborder 特性设置。除 IE 外，其他浏览器都不能很好地响应页面载入后对这一属性的修改。

frameBorder 属性的值是具备 Boolean 意义的字符串。值 yes 或者 1 表示边框(假定是)打开；no 或者 0 表示关闭边框。各种浏览器对此没有一致的默认行为。

示例

frameBorder 属性的默认值是 yes，可以用这一设置来创建切换脚本(遗憾的是，在 IE 中这不能改变边框的外观)。W3C 兼容版本的形式如下：

```
function toggleFrameScroll(frameID)
{
   var theFrame = document.getElementById(frameID);
   if (theFrame.frameBorder == "yes")
   {
      theFrame.frameBorder = "no";
   }
   else
   {
      theFrame.frameBorder = "yes";
   }
}
```


更多资源请访问稀酷客(www.ckook.com)

相关主题： frameset.frameBorder 属性。

height，width

值： 整型，　　　　只读

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera-，Chrome+

这两个属性可以检索 frame 元素对象的宽度和高度，这些值不一定与 document.body.clientHeight 和 document.body.clientWidth 相同，因为框架尺寸考虑了与框架相关联的窗框，如滚动条。这些属性值是只读的。如果需要修改框架的尺寸，可使用 frameset 元素对象的 rows 和 cols 属性，但从框架的 height 和 width 属性中读取整数值，要比分析 rows 和 cols 字符串属性容易得多。

示例

以下脚本段假设，一个框架集把两个框架定义为框架集中的两列。下列语句位于框架集的文档中，它们获取左侧框架的当前宽度，并将该宽度增加 10%。这里给出的语法用于 W3C DOM，但很容易改为仅适用于 IE。

```
var frameWidth = document.getElementById("leftFrame").width;
document.getElementById("mainFrameset").cols =
   (Math.round(frameWidth * 1.1)) + ",*";
```

注意，现有框架的宽度值先增加 10%，然后连接到框架集的 cols 字符串属性的剩余部分。逗号后的星号表示，浏览器应该计算余下的宽度，并将其赋予右侧框架。

相关主题： frameset 对象。

longDesc

值： URL 字符串，　　　　读/写

兼容性： WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

longDesc 属性是<frame>标记中 longdesc 特性的脚本版本，这个 HTML 4 特性向浏览器提供一个文档的 URL，该文档包含元素的详细描述信息。未来的浏览器可以利用这个特性给有视觉障碍的网站访问者提供框架的信息。

marginHeight，marginWidth

值： 整型，　　　　读/写

兼容性： WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

浏览器常常在内容和框架之间添加页边距，以自动将内容插入框架。这些值用 marginHeight(上下边界)和 marginWidth(左右边界)属性来表示。尽管这些属性不是只读的，但在加载框架集后，改变这些值不会改变框架中文档的外观。如果要改变框架中文档的页边距，调整 document.body.style 页边距属性即可。

注意，尽管这些属性默认为空(即没有为<frame>标记设置 marginHeight 或者 marginWidth 特性)，但页边距是内置在页面中的。这些页边距的像素值因操作系统而异。

相关主题： style 对象(第 38 章)。

更多资源请访问稀酷客(www.ckook.com)

name

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

name 属性是与框架相关的标识符，用作一个引用。脚本可以通过 name 属性引用框架(例如 top.frames["myFrame"])，它通常用 name 特性来指定。

noResize

值：Boolean，　　　　读/写

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

Web 设计者通常固定框架集，这样用户就不能(通过拖动框架之间的分割条)改变框架的大小。noResize 属性可以在页面加载后，读取和调整框架的行为。例如，在页面上与用户交互的某些部分中，允许用户在某个模式下手工修改框架的大小，或者允许用户改变框架的大小。在触发 noResize 事件处理程序时，脚本将 frame 元素的 noResize 属性设置为 false。如果关闭改变框架大小的功能，则不管临近框架的 noResize 值如何设置，都不能改变框架的任何边界。关闭改变框架大小的功能后，脚本仍能通过 frameset 元素对象的 cols 或者 rows 属性改变框架的大小。

示例

以下语句关闭了调整框架大小的功能：

```
parent.document.getElementById("myFrame1").noResize = true;
```

由于属性名 noResize 包含否定词 no，因此其逻辑可能难以理解(将 noResize 设置为 true 意味着关闭调整大小的功能)。要注意该属性的 Boolean 值。

相关主题：frameset.cols 属性，frameset.rows 属性。

scrolling

值：yes | no | 1 | 0 的字符串形式，　　　　读/写

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

scrolling 属性允许脚本在框架集的单个框架中打开和关闭滚动条。在默认情况下，滚动条是打开的，除非<frame>标记的 scroll 特性覆盖了 scrolling 属性。

scrolling 属性的值是具备 Boolean 意义的字符串。yes 或者 1 表示滚动条可见(若内容很多，不滚动就不能显示所有内容)；no 或者 0 在框架中隐藏滚动条。IE 也允许使用(且设置为默认值)auto 值。

注意：

尽管此属性是读/写的，但利用脚本更改其值，不会改变框架在 WinIE、Mozilla 浏览器或 Safari 中的外观。

示例

程序清单 27-42 创建了一个包含 8 个框架的框架集，框架的内容由框架集中的脚本(通过 fillFrame()函数)生成。在每个框架的 Body 中，事件处理程序(onclick)都调用了 toggleFrameScroll()

更多资源请访问稀酷客(www.ckook.com)

函数。这里给出了引用 frame 元素对象的两种方式，而仅适用于 IE 的方式加上了注释。

在 toggleFrameScroll()函数中，if 条件检查 scrolling 属性是否设置为不是 no 的值。如果该属性设置为 auto(首次)或 yes(由函数设置)，条件就为 true。注意，在 IE、NN6+、Safari、Opera 或 Chrome 中，滚动条总是显示在框架中。

程序清单 27-42　控制 frame.scrolling 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>frame.scrolling Property</title>
      <script type="text/javascript">
         function toggleFrameScroll(frameID)
         {
            // IE5+/W3C version
            var theFrame = document.getElementById(frameID);

            if (theFrame.scrolling != "no")
            {
               theFrame.scrolling = "no";
            }
            else
            {
               theFrame.scrolling = "yes";
            }
         }

         // generate content for each frame
         function fillFrame(frameID)
         {
            var page = "<html><body onclick='parent.toggleFrameScroll(\""
                     + frameID
                     + "\")'><span style='font-size:24pt'>";
            page += "<p>This frame has the ID of:<\/p><p>"
                 + frameID
                 + ".<\/p>";
            page += "<\/span><\/body><\/html>";
            return page;
         }
      </script>
   </head>
   <frameset id="outerFrameset" cols="50%,50%">
      <frameset id="innerFrameset1" rows="25%,25%,25%,25%">
         <frame id="myFrame1" src="javascript:parent.fillFrame('myFrame1')" />
         <frame id="myFrame2" src="javascript:parent.fillFrame('myFrame2')" />
         <frame id="myFrame3" src="javascript:parent.fillFrame('myFrame3')" />
         <frame id="myFrame4" src="javascript:parent.fillFrame('myFrame4')" />
      </frameset>
      <frameset id="innerFrameset2" rows="25%,25%,25%,25%">
```


更多资源请访问稀酷客(www.ckook.com)

```
        <frame id="myFrame5" src="javascript:parent.fillFrame('myFrame5')" />
        <frame id="myFrame6" src="javascript:parent.fillFrame('myFrame6')" />
        <frame id="myFrame7" src="javascript:parent.fillFrame('myFrame7')" />
        <frame id="myFrame8" src="javascript:parent.fillFrame('myFrame8')" />
      </frameset>
    </frameset>
</html>
```

src

值：URL 字符串，　　读/写

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

frame 元素对象的 src 属性提供了在框架中导航到不同页面的另一种方式(意味着不需要为 frame 对象的 location.href 属性指定新的 URL)。然而，为了与旧的浏览器兼容，可继续使用 location.href 属性，通过脚本来导航。src 属性属于 frame 元素对象，不属于其代表的 window 对象，因此，src 属性的引用必须包含元素 ID 和/或节点层次结构。

示例

为了获得最好的结果，应使用完整的 URL 作为 src 属性的值，如下所示：

```
parent.document.getElementById("mainFrame").src = "http://www.dannyg.com";
```

相对 URL 和 javascript:伪 URL 在多数情况下都有效。

相关主题：location. href 属性。

27.5 frameset 元素对象

关于 HTML 元素的属性、方法和事件处理程序的详细内容，请参见第 26 章。

属　性	方　法	事件处理程序	属　性	方　法	事件处理程序
border			frameBorder		
borderColor			frameSpacing		
cols			rows		

27.5.1 语法

在 frameset 中访问 frameset 元素对象的属性或方法：

```
(IE4+)      document.all.framesetID. property | method([parameters])
(IE5+/W3C) document.getElementById("framesetID"). property |
            method([parameters])
```

在框架文档中访问 frameset 元素的属性或方法：

```
(IE4+)      parent.document.all.framesetID. property | method([parameters])
(IE5+/W3C) parent.document.getElementById("framesetID"). property |
```


更多资源请访问稀酷客(www.ckook.com)

```
method([parameters])
```

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

27.5.2　关于 frameset 对象

frameset 元素对象可以用脚本访问，等价于由＜frameset＞标记生成的元素。这一元素与源对象模型的父对象(窗口类型)不同，frameset 元素对象的一些属性和方法会影响 HTML 元素；而框架文档中的 window 对象通过 parent 或者 top 窗口来引用，它包含文档及其所有内容。

若框架集嵌套在另一个框架集中，则包含和被包含的框架集之间就存在节点父子关系。例如下面的嵌套框架集结构：

```
<frameset id="outerFrameset" cols="30%, 70%">
   <frame id="frame1">
   <frameset id="innerFrameset" rows="50%,50%">
      <frame id="frame2">
      <frame id="frame3">
   </frameset>
</frameset>
```

在这种结构的框架中，给文档写入脚本时，框架集窗口和框架的引用是比 HTML 更平的层次结构。所有框架中的脚本都通过 parent 来引用框架集窗口；而通过 parent.frameName 来引用其他框架。换句话说，文档中定义的框架集 window 对象都是兄弟关系，且拥有同一个父窗口。

但在 W3C 节点的术语中，上述结构就不是这样。父子关系由 HTML 元素的嵌套关系来决定，而不管浏览器创建了什么窗口。因此，frame2 只有一个兄弟 frame3，两者共享一个父框架 innerFrameset。frame1 和 innerFrameset 都是 outerFrameset 的子框架。如果脚本想利用 frame2 的引用改变 outerFrameset 的 cols 属性，则必须访问两代节点：

```
frame2Ref.parentNode.parentNode.cols = "40%,60%";
```

实际上，更糟的是，一个框架中的脚本只有用 window 对象跳出当前的 window 对象，才能转到为文档生成框架窗口的框架集上。换句话说，不能直接从文档跳到文档所在框架的 frame 元素对象上。文档的脚本要通过 parent.document 引用访问其框架集的节点层次结构，但这是对包含整个框架集结构的 document 对象的引用。幸好，W3C DOM 提供了 getElementByld ()方法，来提取嵌套在文档中的节点的引用。因此，其中一个框架的文档可以访问 frame 元素对象，就好像框架是文档中的元素一样。

```
parent.document.getElementById("frame2")
```

引用中不需要包括外层的 frameset 元素对象。要在一个框架窗口的脚本中改变列宽，应使用如下语句：

```
parent.document.getElementById("outerFrame").cols = "40%,60%";
```

也可以用相同的语法访问内部框架集。

更多资源请访问稀酷客(www.ckook.com)

27.5.3　属性

border

值：整型，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

frameset 元素对象的 border 属性可以读取框架集中框架之间的边框厚度(以像素为单位)。如果没有在框架集的标记中指定 border 特性，该属性就为空，而不是默认的实际边框厚度。

示例

这个属性可以读/写，但更改其值不会改变浏览器显示的边框厚度。如果需要确定边框的厚度，框架文档中的脚本引用应如下所示：

```
var thickness = parent.document.all.outerFrameset.border;
```

相关主题：f rameset.frameBorder 属性。

borderColor

值：三个十六进制值或颜色名字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

borderColor 属性可以读取赋予框架集标记中 bordercolor 特性的颜色值。该属性是可读写的，但通过脚本改变颜色不会改变显示在浏览器窗口中的边框颜色。如果这个特性值设为颜色名，则读取属性值时，就返回为三个十六进制值。

示例

为了检索框架集中的当前颜色设置，框架文档中的脚本引用应如下所示：

```
var borderColor = parent.document.all.outerFrameset.borderColor;
```

相关主题：frame.borderColor 属性，frameset.frameBorder 属性。

cols，rows

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

加载框架集后，frameset 元素对象的 cols 和 rows 属性就可以读取和修改框架的大小。这两个属性在 W3C DOM 中定义，它们的值都是字符串，其中可能包括百分号或者星号。因此，如果试图递增或者递减框架中列或者行的大小，应先在字符串中解析出原来的数值，再对它们执行数学操作(在 IE4+中，可以用 frame 元素对象的 height 和 width 属性获取当前框架的像素尺寸)。

调整这两个属性，可以完全改变框架集，还可以添加或者删除框架集表格中的行或列。因为改变框架集结构，会影响框架数组的大小，而这些框架集数组与父窗口相关，或者与正在卸载的、包含所需数据的文档相关，所以要用脚本检查框架集的状态。如果想从框架集视图中删除框架，将框架集中对应行或列的大小指定为 0 可能更安全。当然，尺寸为 0 仍会保留一个 1 像素框架，但如果 1 像素框架的边框未打开，且其背景颜色与其他框架相同，它就是不可见的。

更多资源请访问稀酷客(www.ckook.com)

使用这种技巧的另一个作用是可以恢复文档状态与其隐藏时相同的另一个框架。

在单个文档中定义嵌套的框架集时，一定要引用期望的 frameset 元素对象，因为一个对象可能指定了网格的列；而另一个(嵌套的)对象指定了网格的行。最好为每个 frameset 元素指定唯一的 ID，使引用准确地指向合适的对象。

示例

程序清单 27-43～程序清单 27-45 给出了框架集及其三个文档中的两个文档的 HTML，最后一个文档是一个法案的 HTML 版本，它在这里用作示例的内容框架。

框架集程序清单(参见程序清单 27-43)是一个三框架结构。左侧框架显示了目录(参见程序清单 27-44)，右侧框架分为两行，顶行是一个简单的控件，用于隐藏和显示目录框架(参见程序清单 27-45)。用户单击控件的热区时(位于 span 元素中)，onclick 事件处理程序就调用框架集中的 toggleTOC()函数。

程序清单 27-43　框架集和隐藏/显示框架的脚本

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Hide/Show Frame Example</title>
      <script type="text/javascript">
         var origCols;
         function toggleTOC()
         {
            if (origCols)
            {
               showTOC();
            }
            else
            {
               hideTOC();
            }
         }
         function hideTOC()
         {
            var frameset = document.getElementById("outerFrameset");
            origCols = frameset.cols;
            frameset.cols = "0,*";
         }
         function showTOC()
         {
            if (origCols)
            {
               document.getElementById("outerFrameset").cols = origCols;
               origCols = null;
            }
         }
      </script>
```


更多资源请访问稀酷客(www.ckook.com)

```
      </head>
      <frameset id="outerFrameset" frameborder="no" cols="150,*">
         <frame id="TOC" name="TOCFrame" src="jsb27-44.html" />
         <frameset id="innerFrameset1" rows="80,*">
            <frame id="controls" name="controlsFrame" src="jsb27-45.html" />
            <frame id="content" name="contentFrame" src="bofright.html" />
         </frameset>
      </frameset>
</html>
```

用户单击热区来隐藏框架时，脚本将原来的 cols 属性设置复制到一个全局变量中。该变量在 showTOC()中用于将框架集恢复至其原有比例。这样，设计人员就可修改框架集的 HTML，而不必在脚本中硬连接要恢复的尺寸。

程序清单 27-44 目录框架的内容

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Table of Contents</title>
      <style type="text/css">
         body
         {
          background-color:#EEEEEE;
         }
      </style>
   </head>
   <body>
      <h3>
         Table of Contents
      </h3>
      <hr />
      <ul style="font-size:10pt">
         <li><a href="bofright.htm#article1"
                target="contentFrame">Article I</a></li>
         <li><a href="bofright.htm#article2"
                target="contentFrame">Article II</a></li>
         <li><a href="bofright.htm#article3"
                target="contentFrame">Article III</a></li>
         <li><a href="bofright.htm#article4"
                target="contentFrame">Article IV</a></li>
         <li><a href="bofright.htm#article5"
                target="contentFrame">Article V</a></li>
         <li><a href="bofright.htm#article6"
                target="contentFrame">Article VI</a></li>
         <li><a href="bofright.htm#article7"
                target="contentFrame">Article VII</a></li>
         <li><a href="bofright.htm#article8"
                target="contentFrame">Article VIII</a></li>
         <li><a href="bofright.htm#article9"
```


更多资源请访问稀酷客(www.ckook.com)

```
                    target="contentFrame">Article IX</a></li>
            <li><a href="bofright.htm#article10"
                    target="contentFrame">Article X</a></li>
        </ul>
    </body>
</html>
```

程序清单 27-45　控制面板框架

```
<html>
    <head>
        <title>Control Panel</title>
        <style type="text/css">
            span
            {
                text-decoration:underline; cursor:pointer;
            }
        </style>
    </head>
    <body>
        <p><span id="tocToggle"
                onclick="parent.toggleTOC()">
            &lt;&lt;Hide/Show&gt;&gt;</span> Table of Contents
        </p>
    </body>
</html>
```

相关主题： frame 对象。

frameBorder

值： yes | no | 1 | 0 的字符串形式，　　读/写

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

frameBorder 属性可通过脚本访问 frameset 元素对象中的 frameborder 特性设置。IE4+不能很好地响应页面加载后对这个属性的修改。

frameBorder 属性的值是具备 Boolean 意义的字符串。值 yes 或者 1 表示(假定是)打开边框；而 no 或者 0 关闭边框。

示例

frameBorder 属性的默认值为 yes，此设置可用来创建一个切换脚本(而在 IE 中，这不能改变外观)。IE5+版本如下所示：

```
function toggleFrameScroll(framesetID)
{
    var theFrameset = document.getElementById(framesetID);
    if (theFrameset.frameBorder == "yes")
    {
        theFrameset.frameBorder = "no";
    }
```


更多资源请访问稀酷客(www.ckook.com)

```
    else
    {
        theFrameset.frameBorder = "yes";
    }
}
```

相关主题：frame.frameBorder 属性。

frameSpacing

值：整型， 读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

frameset 元素对象的 frameSpacing 属性可以读取框架集中框架的间隔(以像素为单位)。如果在框架集的标记中没有指定 framespacing 特性，该属性就为空，而不是默认的实际边框厚度(通常为 2)。

示例

如果需要修改框架的间隔，则可以使用如下脚本引用：

```
document.getElementById("outerFrameset").frameSpacing = "10";
```

相关主题：frameset.border 属性。

27.6 iframe 元素对象

关于 HTML 元素的属性、方法和事件处理程序的详情，请参见第 26 章。

属　性	方　法	事件处理程序	属　性	方　法	事件处理程序
align			marginHeight		
allowTransparency			marginWidth		
contentDocument			name		
contentWindow			noResize		
frameBorder			scrolling		
frameSpacing			src		
height			vspace		
hspace			width		
longDesc					

27.6.1 语法

在包含文档中访问 iframe 元素对象的属性或方法：

```
(IE4+)      document.all.iframeID. property | method([parameters])
(IE4+/NN6)  window.frames["iframeName"]. property | method([parameters])
```


更多资源请访问稀酷客(www.ckook.com)

```
(IE5+/W3C) document.getElementById("iframeID"). property |
           method([parameters])
```

从 iframe 元素的文档中访问 iframe 元素的属性和方法：

```
(IE4+)      parent.document.all.iframeID. property | method([parameters])
(IE5+/W3C) parent.document.getElementById("iframeID"). property |
           method([parameters])
```

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

27.6.2　关于 iframe 对象

iframe 元素允许 HTML 内容从一个独立的源载入另一个文档的 body。在某些方面，NN4 的 layer 元素是 iframe 概念的先导，但与 layer 的不同之处在于，iframe 元素本来是不可定位的，它的定位方式与其他的 HTML 元素一样：将定位特性指定为与 iframe 相关联的样式表。如果没有明确地定位，iframe 元素将按正常的源代码顺序显示在文档的 body 中。与框架集的框架不同，iframe 可随意放在任何文档中。如果 frame 通过脚本来改变大小，周围的内容就会向外或向内移动。

真正将 iframe 与其他 HTML 元素区分开的是：iframe 能载入和显示外部的 HTML 文件。利用脚本可将不同的页面载入 iframe，而不干扰主文档中的其他内容。载入 iframe 的页面也可以有脚本，还可以具备 HTML 文档(在 IE for Windows 中还包括 XML)中需要的其他特性。

iframe 元素有一个丰富的特性集，所以 HTML 作者可控制框架的外观、尺寸(height 和 width)，在某种程度上还可控制框架的行为。在脚本中，可将其中大多数特性作为 iframe 元素对象的属性来访问。

iframe 元素在许多方面都很像 frame 元素，这非常重要，特别是在窗口性质的关系中。如果在主窗口的文档中放置一个 iframe 元素，它就在主窗口的对象模型中显示为一个框架，可以通过普通的框架术语访问它：

```
window.frames[i]
window.frames[frameName]
```

这个 iframe 框架对象包含一个文档及其所有内容，iframe 中对 document 对象的所有引用都必须包含 iframe 框架。

相反，iframe 中文档的脚本可以通过 parent 引用与主文档通信。当然，不销毁源文档的 iframe，就不能用另一个 HTML 文档(例如使用 location.href)来代替主窗口中的内容。

27.6.3　属性

align

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

align 属性控制 iframe 元素与页面上其周围内容的对齐方式。该属性的两个值(left 和 right)把 iframe 定位在其包含元素(通常是 body)的左边界和右边界上。与图片相同，若 iframe 沿着容

更多资源请访问稀酷客(www.ckook.com)

器的左边界和右边界浮动，则其他内容围绕在该元素的周围。表 27-6 显示了该属性的所有值及其意义。

表 27-6 align 属性值

值	说 明
absbottom	将 iframe 的底部与沿围绕文本的下行字符延伸的假想线条对齐
absmiddle	将 iframe 的中间与围绕文本的 top 和 absbottom 之间的中心点对齐
baseline	将 iframe 的底部与围绕文本的基线对齐
bottom	将 iframe 的底部与围绕文本的底线对齐
left	使 iframe 与包含元素的左边缘平齐
middle	将包围文本的假想垂直中心线与 iframe 元素的垂直中心线对齐
right	使 iframe 与包含元素的右边缘平齐
texttop	将 iframe 元素的顶部与沿围绕文本的最高上行字符延伸的假想线条对齐
top	将 iframe 元素的顶部与包围元素的 top 对齐

当脚本改变 align 属性的值时，页面会自动根据新的对齐方式来重新编排内容。

示例

iframe 对齐方式的默认设置是 baseline。脚本可以用以下语句使 iframe 与包含元素的右边缘对齐：

```
document.getElementById("iframe1").align = "right";
```

相关主题： iframe. Hspace、iframe. vspace 属性。

allowTransparency

值： Boolean， 读/写

兼容性： WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

allowTransparency 属性表示框架背景是否透明。将此属性设为 true，背景色或图像就可以透过透明框架显示出来。

contentDocument

值： document 对象引用， 只读

兼容性： WinIE8+，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

iframe 元素对象的 contentDocument 属性就是框架所含文档的引用。如果脚本有一个 iframe 元素对象的引用，则可以使用 contentDocument 属性来获得文档的有效引用，于是也可以引用框架的其他内容。

示例

文档脚本可使用 iframe 元素的 ID 来读取或调整某个元素属性，然后通过其 document 对象对页面内容执行某些动作。使用如下语句可以获得对 document 对象的引用：

更多资源请访问稀酷客(www.ckook.com)

```
var doc = document.getElementById("Frame3").contentDocument;
```

然后脚本就可以访问文档中的表单：

```
var val = doc.mainForm.entry.value;
```

相关主题：contentWindow 属性；document 对象。

contentWindow

值：document 对象引用，　　只读

兼容性：WinIE5.5+，MacIE-，NN7+，Moz1.0.1+，Safari+，Opera+，Chrome+

iframe 元素对象的 contentWindow 属性用作由框架生成的 window 对象的引用，然后，该 window 对象可用作访问 document 对象和任何文档元素的一种方式。

相关主题：contentDocument 属性；window 对象。

frameBorder

(参见 frame.frameBorder()和 frameset.frameBorder())

frameSpacing

(参见 frameset.frameSpacing())

height，width

值：整型，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

可以通过这两个属性来访问 iframe 对象的高度和宽度，也可以修改框架的大小。这两个属性都以像素为单位。

hspace，vspace

值：整型，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

这些 IE 专用的属性允许在 iframe 元素的周围设置边框。一般而言，hspace 和 vspace 属性(及其 HTML 特性)已由 CSS 页边距和填充代替。另外，W3C 标准无法识别这些属性(包括 HTML 4)。

这些属性的值是整数，表示元素和周围内容之间的填充像素值。hspace 值为元素左边和右边指定相同的填充像素值；vspace 值为元素顶部和底部指定相同的填充像素值。在 WinIE5 中，脚本对这些属性的改变没有任何作用。

相关主题：style.padding 属性。

longDesc

值：URL 字符串，　　读/写

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

longDesc 属性是<iframe>标记的 longdesc 特性的脚本版本，这个 HTML 4 特性可以给浏

更多资源请访问稀酷客(www.ckook.com)

览器提供一个文档的 URL，该文档包含了元素的详细描述信息。未来的浏览器可以利用这个特性，为有视觉障碍的网站访问者提供框架的信息。

marginHeight，marginwidth

值：整型， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

浏览器在框架中插入内容时，会在框架的内容和边框之间自动加入空白。这些空白的大小由 marginHeight(顶边和底边)和 marginwidth(左边和右边)属性来表示。这些属性不是只读的，但在载入框架集后，对这些属性的改变不会影响框架中文档的显示。如需改变框架中文档的页边距，只需调整 document.body.style 页边距属性。

还要注意，尽管这些属性的默认值为空(即没有为<iframe>标记设置 marginheight 或 marginwidth 特性)，还是会在页面中设置页边距。这些页边距的像素值因操作系统而异。

相关主题：style 对象(第 38 章)。

name

值：字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

name 属性是与框架相关联的标识符，用作框架的引用。脚本可以通过 name 属性引用框架(例如 window.frames["myIframe"])，name 属性通常通过 name 特性来指定。

noResize

(参见 frame.noResize())

scrolling

值：yes | no | 1 | 0 的字符串形式， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

scrolling 属性允许脚本在 iframe 元素中显示和隐藏滚动条。在默认情况下，除非覆盖了<frame >标记的 scroll 特性，否则滚动条将自动打开。

scrolling 属性的值是具备 Boolean 意义的字符串。yes 或者 1 表示滚动条可见(假定内容很多，不滚动就不能显示所有内容)；no 或者 0 在框架中隐藏滚动条。IE4+也接受 auto 值(且设置为默认值)。

示例

下面的 toggleIFrameScroll()函数把 iframe 元素的 ID 字符串作为参数，在 iframe 中显示或不显示滚动条。if 条件检查 scrolling 属性是否设置为 no 之外的值，如果属性设置为 auto(首次)或 yes(由函数设置)，此测试的条件的求值结果为 true。

```
function toggleFrameScroll(frameID)
{
   // IE5 & NN6 version
```


更多资源请访问稀酷客(www.ckook.com)

```
    var theFrame = document.getElementById(frameID);

    if (theFrame.scrolling != "no")
    {
       theFrame.scrolling = "no";
    }
    else
    {
       theFrame.scrolling = "yes";
    }
}
```

相关主题：frame.scrolling 属性。

src

值：URL 字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

iframe 元素对象的 src 属性提供了另一种方式，允许在内嵌框架中浏览不同的页面(意味着不需要为 frame 对象的 location.href 属性指定新的 URL)。记住，src 属性属于 iframe 元素对象，而不是其代表的 window 对象，因此，src 属性的引用必须包含元素 ID 和/或节点层次结构。

示例

为了得到最好的结果，应使用完整的 URL 作为 src 属性的值，如下所示：

```
document.getElementById("myIframe").src = "http://www.dannyg.com";
```

相对 URL 和 javascript:伪 URL 在多数情况下都有效。

相关主题：location. href 属性。

27.7 popup 对象

属　性	方　法	事件处理程序
document	hide()	
isOpen	show()	

27.7.1 语法

创建 popup 对象：

```
var popupObj = window.createPopup()
```

从创建弹出式窗口的文档中访问 popup 对象的属性或方法：

```
popupObj.property | method([parameters])
```

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

更多资源请访问稀酷客(www.ckook.com)

27.7.2　关于 popup 对象

popup 对象是一种无窗框的窗口空间，它显示在生成它的文档窗口上。弹出窗口总是显示在对话框的前面。与 showModalDialog()和 showModelessDialog()方法创建的对话框窗口不同，脚本不仅可以创建弹出窗口，也可以在其中加入内容，然后指定它在屏幕上的位置及大小。因为 IE 创建的弹出窗口是一个窗口，而不是对话框，所以没有对应的 W3C 弹出窗口。但是使用 W3C DOM 可在节点树上添加和删除元素，因此能获得同样的功能。更多信息参见第 25 章。

弹出窗口没有窗框(即标题栏、大小控制柄等)，所以应该给它填充带边框和背景颜色的内容，使它不同于主窗口的内容。下面的语句演示了创建 popup 对象、填充并显示其内容的典型过程：

```
var popup = window.createPopup();
var popupBody = popup.document.body;
popupBody.style.border = "solid 2px black";
popupBody.style.padding = "5px";
popupBody.innerHTML = "<p>Here is some text in a popup window</p>";
popup.show(200,100, 200, 50, document.body);
```

注意：

尽管 innerHTML 很方便，但在严格遵循 W3C 的 JavaScript 中，不能使用 innerHTML 属性，因为它不是 W3C 标准的正式组成部分。不过，它非常强大，也很便捷，不能忽视它，本书中的许多代码就证明了这一点。一些示例给出了替代 innerHTML 的 W3C 节点操作方法。关于 innerHTML 的 W3C 替代方法的详细说明和示例，请参见第 29 章。

实际上，只有对其包含的文档而言，IE 创建的弹出窗口才是一个窗口。换句话说，popup 对象的属性和方法很少，但其文档的 parentWindow 属性是一个真正的 window 属性。尽管如此，这个弹出窗口也不会像 Windows 任务栏中的窗口那样显示为独立的窗口。如果用户在弹出窗口之外单击，或者转向别的应用程序，弹出窗口将消失，只有用脚本重新调用 show()方法(指定大小和位置参数)，才能使弹出窗口重新显示出来。

为弹出窗口指定内容时，必须确保该内容符合为弹出窗口指定的大小。如果内容超出了显示范围(在弹出窗口的矩形范围内，文本会自动换行)，不会出现滚动条。

27.7.3　属性

document

值：document 对象引用，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

document 属性可以用作访问弹出窗口中内容的通道，这个属性是在创建弹出窗口的脚本中访问弹出窗口的唯一访问点。这个属性最常见的应用是设置 document 属性，来控制弹出窗口的内容。例如，要给弹出窗口指定边框(因为弹出窗口本身没有窗框)，则创建窗口的脚本需要指定弹出窗口中文档的 style 属性值：

```
myPopup.document.body.style.border = "solid 3px gray";
```


更多资源请访问稀酷客(www.ckook.com)

注意，弹出窗口的 document 对象可能不具备主窗口 document 对象的所有灵活性，例如，不能为弹出窗口中的 document.URL 属性指定 URL。

示例

使用 The Evaluator(第 4 章)来试验 popup 对象及其属性。在顶部文本框中输入以下语句，第一条语句创建了一个弹出窗口，其引用赋给了全局变量 a。接下来，为了方便起见，变量 b 保存了对弹出窗口文档的 body 部分的引用，其他语句将使用这两个变量。

```
a = window.createPopup()
b = a.document.body
b.style.border = "solid 2px black"
b.style.padding = "5px"
b.innerHTML = "<p>Here is some text in a popup window</p>"
a.show(200,100, 200, 50, document.body)
```

有关该参数的详情，请参阅对 show()方法的描述。

相关主题：document 对象。

isOpen

值：Boolean，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

弹出窗口可见时，其 isOpen 属性返回 true；否则返回 false。因为浏览器中的任何用户操作都会隐藏弹出窗口，所以仅当弹出窗口可见后，在弹出窗口上运行的脚本语句才可使用这个属性。

示例

使用 The Evaluator(第 4 章)试验 isOpen 属性。在顶部文本框中输入以下语句，这些语句首先创建一个简单的弹出窗口，其引用赋给全局变量 a。注意，最后一条语句实际上是两条语句，这样在弹出窗口打开时，会执行第二条语句。

```
a = window.createPopup();
a.document.body.innerHTML = "<p>Here is a popup window</p>";
a.show(200,100, 200, 50, document.body); alert("Popup is open:" + a.isOpen);
```

此后在主窗口中单击，来隐藏弹出窗口。在顶部文本框中输入以下语句，会看到不同的结果：

```
alert("Popup is open:" + a.isOpen);
```

相关主题：popup.show()方法。

27.7.4　方法

hide()，show(left, top, width, height[, positioningElementRef])

返回值：无。

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

用 window.createPopup()方法创建 popup 对象，并填充内容后，必须明确调用 show()方法来

更多资源请访问稀酷客(www.ckook.com)

显示窗口。如果是因为用户在主浏览器窗口的某处单击而使其隐藏，就必须再次调用 show 方法(及其参数)。要用脚本隐藏窗口，为 popup 对象调用 hide()方法即可。

show()方法的前 4 个参数是必选的，它们定义了弹出窗口的像素位置和大小。默认情况下，left 和 top 参数的坐标空间是显示器的显示区域。因此，left 和 top 为 0，会将弹出窗口放在显示器的左上角。还可以添加可选的第 5 个参数，来定义另一个坐标空间。这个参数必须是页面上元素的引用。为将坐标空间限制为浏览器窗口的内容区域，可将 document.body 作为定位元素的引用。

示例

程序清单 27-46 演示了 popup 对象的 show()和 hide()方法。单击页面上的按钮，就会调用 selfTimer()函数，该函数是此页面的主例程。示例的目标是创建一个在显示 5s 后消失的弹出窗口。在此过程中，弹出窗口中的消息以秒为单位进行倒计时。

对弹出窗口的引用保存为全局变量 popup。创建 popup 对象后，initContent()函数为文档的 body 部分(文档的 body 部分在生成弹出窗口时自动创建)指定 style 属性和一些 innerHTML，并在弹出窗口中填充内容。再定义一个 span 元素，以便稍后另一个函数可以修改弹出窗口中的该部分文本内容。注意，为弹出窗口填充内容之前，弹出窗口应已初始化(给 popup 变量赋了值)，且没有显示出来。尽管不能在其他任何环境下调用 initContent()，但验证所需的条件仍然是一个良好的编程实践方式。

在 selfTimer()中，显示 popup 对象。定义所需的尺寸时需要反复试验，以确保弹出窗口正好能容纳 initContent()函数放入其中的文本。

弹出窗口显示出来后，就该调用 countDown()函数了。在函数执行任何动作之前，先验证弹出窗口已初始化并仍然可见。如果在计时器倒计时时，用户单击了主窗口，就将 isOpen 属性值改为 false，不执行 if 条件中的语句。

countDown()函数获取 span 的内部文本，并用 paresInt()提取整数(使用十进制，因为这里在处理以 0 开头的数字，它们也可以看作八进制值)。if 结构的条件将提取的整数值减 1。如果减后的值为 0，则时间已到，隐藏弹出窗口，同时全局变量 popup 恢复其原始值 null；否则，span 元素的内部文本就设置为减后的值(有前导 0)，并调用 setTimeout()方法，以便在 1s(1000ms)内重新调用 countDown()函数。

程序清单 27-46　隐藏和显示弹出窗口

HTML: jsb27-46.html

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>popup Object</title>
      <script type="text/javascript" src="jsb27-46.js"></script>
   </head>
   <body>
      <form>
         <input type="button" value="Impossible Mission"
                onclick="selfTimer()" />
```


更多资源请访问稀酷客(www.ckook.com)

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

JavaScript: jsb27-46.js

```
var popup;
function initContent()
{
    if (popup && !popup.isOpen)
    {
        var popBody = popup.document.body;
        popBody.style.border = "solid 3px red";
        popBody.style.padding = "10px";
        popBody.style.fontSize = "24pt";
        popBody.style.textAlign = "center";
        var bodyText = "<P>This popup will self-destruct in ";
        bodyText += "<span id='counter'>05<\/span>";
        bodyText += " seconds...<\/P>";
        popBody.innerHTML = bodyText;
    }
}
function countDown()
{
    if (popup && popup.isOpen)
    {
        var currCount = parseInt(popup.document.all.counter.innerText, 10);
        if (--currCount == 0)
        {
            popup.hide();
            popup = null;
        }
        else
        {
            popup.document.all.counter.innerText = "0" + currCount;
            setTimeout("countDown()", 1000);
        }
    }
}
function selfTimer()
{
    popup = window.createPopup();
    initContent();
    popup.show(200,200,400,100,document.body);
    setTimeout("countDown()", 1000);
}
```

在弹出窗口显示时，运行的脚本调用了这里的 hide()方法。如果用户单击了主窗口，弹出窗口会自动隐藏，因此 hide()方法不大可能自动调用，以响应主窗口中的用户动作。如果希望弹出窗口中的脚本关闭弹出窗口，可使用 parentWindow.close()。

相关主题：popup.isOpen 属性；window.createPopup()方法。

更多资源请访问稀酷客(www.ckook.com)

第28章

location 对象和 history 对象

> **本章包含哪些内容？**
> - 通过 location 对象载入新的页面和其他媒介类型
> - 框架间的安全限制
> - 在脚本控制下查看浏览器历史记录

并不是文档对象模型(DOM)中的所有对象都在浏览器窗口的内容区域中可见。每个浏览器窗口或者框架都包含一些信息，包括当前访问的页面和访问者所在的位置。窗口中页面的 URL 称为地址，这个信息存储在浏览器的 location 对象中。浏览网页时，浏览器把浏览过的网页的 URL 保存在 history 对象中。在浏览器菜单中，可以手动查看 history 对象包含的内容(以便跳到以前访问的页面中)。本章讲述了这两种几乎不可见但非常重要的对象。

这些对象不仅对浏览器有用，窥视者也可以使用它们编写脚本，来查看用户在另一个框架中浏览的网页的 URL，或者最近几次单击鼠标所访问的其他站点的 URL。然而，浏览器内置的安全限制禁止访问这些对象的一些属性(除非在 NN4+/Moz 中使用签名脚本)。而在旧版的浏览器中，脚本根本不能访问这些属性。

28.1 location 对象

属　性	方　法	事件处理程序
hash	assign()	无
host	reload()	
hostname	replace()	
href		
pathname		
port		
protocol		
search		

更多资源请访问稀酷客(www.ckook.com)

28.1.1　语法

把新文档加载到当前窗口中：

```
[window.]location.href = "URL";
```

访问 location 对象的属性或方法：

```
[window.]location.property | method([parameters])
```

28.1.2　关于 location 对象

在原始的文档对象层次结构中，location 对象比 window 类型的对象低一级，它表示当前打开的窗口或者指定框架的 URL 信息。要显示当前网页的 URL，可引用 location 对象，如下：

```
document.write(location.href);
```

在此示例中，href 属性计算为 URL，并完整地写入当前页面。location 对象还允许访问 URL 的各个部分，如后面所示。

在多框架窗口的框架集文档中引用 location 对象时，在浏览器位置(或地址)域中会显示其父窗口的 URL。每个框架也有与其相关联的位置信息，但在浏览器中看不到任何对框架 URL 的公开引用。要获得另一个框架中的文档的 URL 信息，则对 location 对象的引用必须包括窗口框架引用。例如，假设有一个窗口由两个框架组成，表 28-1 给出了对构成 Web 显示的所有框架的 location 对象引用。

注意：

脚本不能改变显示在浏览器位置/地址框中的 URL。出于安全和隐私方面的原因，该文本框只能显示当前页面的 URL 或正在传输的 URL。

表 28-1　在双框架浏览器窗口中的 location 对象引用

引　用	说　明
location(或 window.location)	显示文档的框架的 URL，该文档运行包括此引用的脚本语句
parent.location	定义<frameset>的父窗口的 URL 信息
parent.frames[0].location	第一个可见框架的 URL 信息
parent .frames[1].location	第二个可见框架的 URL 信息
parent.otherFrameName.location	同一框架集中另一个指定框架的 URL 信息

location 对象的大多数属性都处理面向网络的信息，这些信息涉及网络文档物理地址的各种数据，包括主机服务器、所用的协议以及其他 URL 组成部分。给定一个典型 WWW 页的完整 URL，window.location 对象会将属性名赋予 URL 的不同部分，如下所示：

```
http://www.example.com:80/promos/newproducts.html#giantGizmo
```


更多资源请访问稀酷客(www.ckook.com)

属　　性	值
protocol	"http："
hostname	"www.example.com"
port	"80"
host	"www.example.com:80"
pathname	"promos/newproducts.html"
hash	"#giantGizmo"
href	"http://www.example.com:80/promos newproducts.html#giantGizmo"

脚本需要提取 URL 的信息时，可以使用 window.location 对象，来获取一个基本引用，为其他要提取的文档创建 URL，作为用户动作的结果。Web 设计者在一台计算机上开发站点，然后上传到一个目录结构完全不同的服务器(可能是 Internet 服务提供商)上时，使用这个对象可以减少麻烦。利用脚本在当前文档的目录位置上构造基本引用，可为载入的文档构造完整的 URL。文档在计算机或者目录之间传递时，不需要在文档中手动更改基本引用数据。要提取 URL 的各段信息，并将其放入当前目录，可使用下面的语句：

```
var baseRef = location.href.substring(0,location.href.lastIndexOf("/") + 1);
```

警告：

为降低 Internet 安全漏洞和个人隐私被侵犯而带来的风险，脚本浏览器禁止一个框架中的脚本从其他域和服务器的框架中获得 location 对象属性(除非在 NN4+/Moz 中使用签名脚本，或者用户将 IE 浏览器设置为信任此站点)。这种限制妨碍了脚本设计者为 Web 关注者和访问者提供善意的设计和帮助。然而，如果企图访问这些属性，安全警告对话框就会显示“拒绝访问”(或者类似)信息。

设置 location 属性的值是控制将哪个文档载入窗口或者框架的首选方式。用户可能希望在 JavaScript 中找到名称中包含简单词汇 Go 或者 Open 的方法，来模拟浏览器菜单栏的功能，其实将 window.location.href 属性设置为 URL，就可以把浏览器指向另一个 URL。如：

```
window.location.href = "http://www.dannyg.com/";
```

在此类语句中，等号操作符(=)是一种强大的工具。实际上，将 location. href 对象设置为不同 MIME 类型的 URL，如某种声音和视频格式，浏览器就会将这些文件载入浏览器指定的插件程序或者辅助应用程序。location.assign()方法最初用于浏览器的内部，但脚本设计者也可以使用它(尽管本书不推荐将它用于导航)。Internet Explorer 对象模型包含的 window.navigate()方法也将一个文档载入窗口，但不能用在跨浏览器的应用程序中。

其他两个方法补足了 location 对象控制导航的能力，其中一个方法是单击 Reload 的脚本版本；另一个方法用脚本选择的下一个 URL 项替换历史记录中当前文档的项。

更多资源请访问稀酷客(www.ckook.com)

28.1.3 属性

hash

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

#是一个 URL 约定，它将浏览器指向文档中设置的锚点。在 URL 中(使用…标记对)为锚点指定的任意名称放在#的后面。location 对象的 hash 属性是当前 URL 中锚点的名称(它由#和名称组成)。

如果在 HTML 文档中应用了锚点，并将链接导航到这些锚点，就会发现，尽管目标地址将锚点显示为 URL 的一部分(例如在地址域中)，但当用户手工滚动到文档中其他锚点定义的位置时，窗口的锚点值不变。只有窗口导航到锚点定义的位置，并将其作为链接的一部分，或者响应调整 URL 的脚本，锚点才出现在 URL 中。

设置 window.location.href 属性可导航到任何 URL 上，同样，仅调整 location 的 hash 属性(不用#符号)也可以导航到同一文档中的另一个#(见下例)。

程序清单 28-1 说明了如何使用 hash 属性访问 URL 的锚点部分。载入程序清单 28-1 中的脚本后，调整浏览器窗口的高度，使一次只显示一部分文档。单击按钮，脚本就导航到序列中的下一个逻辑部分，并最终回到顶部。页面滚动到文档底部后就不再向下滚动了，因此，靠近页面底部的锚点可能不会显示在浏览器窗口的顶部。

程序清单 28-1　带有锚点的文档

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>location.hash Property</title>
      <script type="text/javascript">
      function goNextAnchor(where)
      {
         window.location.hash = where;
      }
      </script>
   </head>
   <body>
      <h1><a id="start" name="start">Top</a></h1>
      <form>
         <input type="button" name="next" value="NEXT"
         onclick="goNextAnchor('sec1')" />
      </form>
      <hr /><br /><br /><br /><br /><br />
      <h1><a id="sec1" name="sec1">Section 1</a></h1>
      <form>
         <input type="button" name="next" value="NEXT"
         onclick="goNextAnchor('sec2')" />
      </form>
```

更多资源请访问稀酷客(www.ckook.com)

```
        <hr /><br /><br /><br /><br /><br />
        <h1><a id="sec2" name="sec2">Section 2</a></h1>
        <form>
           <input type="button" name="next" value="NEXT"
           onclick="goNextAnchor('sec3')" />
        </form>
        <hr /><br /><br /><br /><br /><br />
        <h1><a id="sec3" name="sec3">Section 3</a></h1>
        <form>
           <input type="button" name="next" value="BACK TO TOP"
           onclick="goNextAnchor('start')" />
        </form>
    </body>
</html>
```

注意：

上例及本章使用的事件处理属性分配技术是有意简化了的，以使代码更容易理解。通常最好采用更现代的方式，用 addEventListener() (NN6+/Moz/W3C)或 attachEvent()(IE5+)方法来绑定事件。第 32 章将详细介绍现代的跨浏览器事件处理技术。其他几章也大量使用了这个现代技术。

锚点名称作为参数传递给每个按钮的 onclick 事件处理程序。这里没有添加 hash 标记和锚点值，在函数中组装好 window.location 值，而只是修改了当前窗口位置的 hash 属性，这是更简明的首选方式。

但如果试图在额外的脚本代码中读取 window.location.hash 属性，而窗口的实际 URL 可能尚未更新，此时浏览器似乎给脚本提供了错误信息。为了在这个函数的后续语句中防止这一问题，可利用设置 window.location.hash 属性的变量值来构造那些语句的 URL，而不要使用浏览器提供的值。

相关主题：location. href 属性。

host

值：字符串，　　　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

location.host 属性指定了 URL 的主机名和端口。只有 URL 明确指定了端口，端口才包含在属性值中。如果导航到一个 URL 上，但其端口号未在浏览器的 Location 域中显示，location.host 属性值就与 location.hostname 属性值相同。与 location.hostname 属性相同，如果在一些浏览器(如 IE 和 WebKit 浏览器 Chrome)中打开本地硬盘(localhost)上的页面，则 location.host 属性可能为空。

对于载入浏览器的文档，可以用 location.host 属性获取其 URL 的 hostname:port 部分。这项功能有助于创建指定文档的 URL，且脚本可随时访问它。

程序清单 28-2～程序清单 28-4 中的文档可帮助了解 window.location 属性的各种返回值。在浏览器中打开程序清单 28-2，此文件创建了一个双框架窗口，左侧框架包含的临时占位文档(参见程序清单 28-4)显示一些指示；右侧框架中的文档(参见程序清单 28-3)允许将 URL 载入左侧框架，并显示 3 个不同的可用窗口的信息：父窗口(它创建多框架窗口)、左侧框架和右侧框架。

更多资源请访问稀酷客(www.ckook.com)

程序清单 28-2　属性选择程序的框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>window.location Properties</title>
      </head>
      <frameset cols="50%,50%" border="1" bordercolor="black">
         <frame name="Frame1" src="jsb28-04.html" />
         <frame name="Frame2" src="jsb28-03.html" />
      </frameset>
</html>
```

程序清单 28-3　属性选择程序

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Property Picker</title>
      <script type="text/javascript">
      var isNav = (typeof netscape != "undefined") ? true : false;

      function fillLeftFrame()
      {
         newURL = prompt("Enter the URL of a document to show in the left
            frame:","");
         if (newURL != null && newURL != "")
         {
            parent.frames[0].location = newURL;
         }
      }
      function showLocationData(form)
      {
         for (var i = 0; i <3; i++)
         {
            if (form.whichFrame[i].checked)
            {
               var windName = form.whichFrame[i].value;
               break;
            }
         }
         var theWind = "" + windName + ".location";
         if (isNav)
         {
            netscape.security.PrivilegeManager.enablePrivilege(
               "UniversalBrowserRead");
         }
         var theObj = eval(theWind);
         form.windName.value = windName;
```


更多资源请访问稀酷客(www.ckook.com)

```
        form.windHash.value = theObj.hash;
        form.windHost.value = theObj.host;
        form.windHostname.value = theObj.hostname;
        form.windHref.value = theObj.href;
        form.windPath.value = theObj.pathname;
        form.windPort.value = theObj.port;
        form.windProtocol.value = theObj.protocol;
        form.windSearch.value = theObj.search;
        if (isNav)
        {
           netscape.security.PrivilegeManager.disablePrivilege(
              "UniversalBrowserRead");
        }
     }
   </script>
</head>
<body>
   Click the "Open URL" button to enter the location of an HTML document to
   display in the left frame of this window.
   <form>
      <input type="button" name="opener" value="Open URL..."
      onclick="fillLeftFrame()" />
      <hr />
      <center>
         Select a window/frame. Then click the "Show Location Properties"
         button to view each window.location property value for the desired
         window.
         <p><input type="radio" name="whichFrame" value="parent"
            checked="checked" />Parent window <input type="radio"
            name="whichFrame" value="parent.frames[0]" />Left frame <input
            type="radio" name="whichFrame" value="parent.frames[1]" />This
            frame</p>
         <p><input type="button" name="getProperties"
            value="Show Location Properties"
            onclick="showLocationData(this.form)" /> <input type="reset"
            value="Clear" /></p>
         <table border="2">
            <tr>
               <td align="right">Window:</td>
               <td><input type="text" name="windName" size="30" /></td>
            </tr>
            <tr>
               <td align="right">hash:</td>
               <td><input type="text" name="windHash" size="30" /></td>
            </tr>
            <tr>
               <td align="right">host:</td>
               <td><input type="text" name="windHost" size="30" /></td>
            </tr>
            <tr>
```


更多资源请访问稀酷客(www.ckook.com)

```
                    <td align="right">hostname:</td>
                    <td><input type="text" name="windHostname" size="30" /></td>
                 </tr>
                 <tr>
                    <td align="right">href:</td>
                    <td><textarea name="windHref" rows="3" cols="30" wrap="soft">
                       </textarea></td>
                 </tr>
                 <tr>
                    <td align="right">pathname:</td>
                    <td><textarea name="windPath" rows="3" cols="30" wrap="soft">
                       </textarea></td>
                 </tr>
                 <tr>
                    <td align="right">port:</td>
                    <td><input type="text" name="windPort" size="30" /></td>
                 </tr>
                 <tr>
                    <td align="right">protocol:</td>
                    <td><input type="text" name="windProtocol" size="30" /></td>
                 </tr>
                 <tr>
                    <td align="right">search:</td>
                    <td><textarea name="windSearch" rows="3" cols="30"
                       wrap="soft"></textarea></td>
                 </tr>
              </table>
           </center>
        </form>
     </body>
  </html>
```

程序清单 28-4　程序清单 28-2 的占位文档

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>Opening Placeholder</title>
   </head>
   <body>
      Initial placeholder. Experiment with other URLs for this frame (see
      right).
   </body>
</html>
```

为了得到最佳效果，可从载入程序清单的域和服务器上(可能是本地硬盘)打开网络上 Web 文档的 URL。如果从另一个域和服务器上打开 Web 文档的 URL，务必指定完整的 URL。尽可能载入一个包括锚点以便进行导航的长文档。选择 Left frame 单选按钮，然后单击显示所有属性的按钮，将所选窗口的全部可用 location 属性填入右侧框架的表格中。

更多资源请访问稀酷客(www.ckook.com)

相关主题：location.port、location. hostname 属性。

hostname

值：字符串，　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

URL 的主机名一般是网络上服务器的名称，该服务器存储了在浏览器上浏览的文档。大多数 Web 站点的服务器名不仅包括域名，还包括 www.前缀。然而，如果 URL 指定了端口号，主机名将不包括端口号。如果在一些浏览器(如 IE 和 WebKit 浏览器 Chrome)中打开本地硬盘(localhost)上的页面，则 hostname 属性可能为空。

参见程序程序清单 28-2～程序清单 28-4，了解如何用一组相关页面来查看各种其他页面的主机名数据。

相关主题：location.host、location.port 属性。

href

值：字符串，　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在 location 对象的所有属性中，href(超文本引用)在编写脚本时是最常用的属性。location.href 属性提供了指定 window 对象的完整 URL 字符串。

在赋值语句的左边使用这个属性是在窗口中打开要显示的 URL 的一种 JavaScript 方式，下面的两个语句都可在一个单框架浏览器窗口中载入 Web 站点的索引页：

```
window.location = "http://www.dannyg.com";
window.location.href = "http://www.dannyg.com";
```

有时，可能由于忽略了窗口引用，使 JavaScript 错误地引用 document.location 属性。为防止这种错误，document.location 属性已废弃(放在禁止列表中)，而用 document.URL 属性取而代之。另外，在引用中总是指定窗口就不会犯错了。

注意：

将新的 URL 赋给 location 对象时，可以忽略 href 属性名(如 location="http://www.dannyg.com")。多数情况下，这在大多数浏览器中是可行的，但在一些早期的浏览器中，将 URL 明确地赋给 location.href 属性会更加可靠。为安全起见，应总是使用 location.href 属性。

有时，只有在脚本中获取当前目录的准确名称，才能使另一个语句将已知文档名添加到 URL 的末尾，再把该文档载入到窗口中。尽管其他的 location 对象属性会生成 URL 的一部分，但它们都不能提供当前 URL 目录的完整 URL。这个任务可以用 JavaScript 字符串处理技巧完成。程序清单 28-5 演示了这个技巧。

根据所用的浏览器，location.href 属性的值可能用非字母符号的 ASCII 值来编码，这样的 ASCII 值包括%符号和 ASCII 数字值。URL 中最常用的编码字符是空格：%20。如果需要提取 URL，并在文档中把它显示为字符串，可将这些编码字符串传递到 JavaScript 的 unescape()函数中。例如，如果 URL 是 http://www.example.com/product%201ist，可以将它传递到 unescape()函

更多资源请访问稀酷客(www.ckook.com)

数中进行转换，如下：

```
var plainURL = unescape(window.location.href);
// result = "http://www.example.com/product list";
```

此函数的反函数是 escape()，它可将编码字符串发送到服务器应用程序，例如 CGI 脚本。这些函数详见第 24 章。

程序清单 28-5 说明了如何使用 href 属性查看当前页面的目录 URL。此示例的脚本在捕获 URL 之前包括了 unescape()函数，该函数的作用是转换显示格式，对于通常显示 ASCII 编码的浏览器，该函数在警告对话框中显示路径名。

注意：

尽管程序清单 28-5 使用了 unescape()全局函数，以确保向后兼容性，但从版本 3 起，该函数(及其反函数 escape())就从 ECMAScript 标准中删除了，这些函数被更现代的版本 decodeURI() 和 encodeURI()所取代，详见第 24 章。

程序清单 28-5 提取当前文档的目录

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Extract pathname</title>
      <script type="text/javascript">
      // general purpose function to extract URL of current directory
      function getDirPath(URL)
      {
         var result = unescape(URL.substring(0,(URL.lastIndexOf("/")) + 1));
         return result;
      }
      // handle button event, passing work onto general purpose function
      function showDirPath(URL)
      {
         alert(getDirPath(URL));
      }
      </script>
   </head>
   <body>
      <form>
         <input type="button" value="View directory URL"
         onclick="showDirPath(window.location.href)" />
      </form>
   </body>
</html>
```

相关主题：location. Pathname、document.location 属性；String 对象(第 15 章)。

更多资源请访问稀酷客(www.ckook.com)

pathname

值：字符串，读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

URL 的路径名部分是相对于服务器的根卷标的目录结构。换句话说，根目录(即 http:连接中服务器的名称)不包含在路径名中。如果 URL 的路径指向根目录中的一个文件，location. pathname 属性就是单个斜杠(/)字符。而其他路径名以斜杠字符开始，表示根目录中嵌套的目录。另外，location. pathname 属性的值也包括文档名。

查看本章前面程序清单 28-2～程序清单 28-4 中的多框架示例，了解各种 URL 的 location. pathname 属性。

相关主题：location. href 属性。

port

值：字符串，读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

目前，友好的 Web 站点很少需要将端口号作为 URL 的一部分。端口号大多用于不大流行的协议、个人开发的站点 URL、没有指定域名的站点 URL 等。使用 location.port 属性可以得到端口号。如果想从 URL 中得到端口值，然后用其构建另一个 URL，应在服务器的 IP 地址和端口号之间使用冒号。

如果可以访问包含端口号的 URL，就使用程序清单 28-2～程序清单 28-4 中的文档来查看 location. port 属性的输出。

相关主题：location. host 属性。

protocol

值：字符串，读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

URL 的第一部分是进行特定类型的通信的协议。万维网页面的标准协议是超文本传输协议(Hypertext Transfer Protocol，HTTP)；浏览器中的其他常用协议包括 HTTP-Secure(https)、文件传输协议(File Transfer Protocol，FTP)、File(file)和 Mail(mailto)。从本地硬盘上打开的网页使用 file 协议。location.protocol 属性的值不仅包括协议名称，也包括尾部的冒号分隔符。因此，对于典型网页的 URL，locationon.protocol 属性是：

```
http:
```

注意，通常在 URL 中，location.protocol 属性值不包含协议后面的斜杠。在 location 对象的所有属性中，只有完整的 URL (location.href)才在协议和其他部分之间加上斜杠。

查看程序清单 28-2～程序清单 28-4 中的多框架示例，来检查各种 URL 的 location.protocol 属性。注意，协议最初显示为 file:，表示左侧框架中的第一个页面存储在本地，并可通过 File 协议访问。还可以试着载入一个 FTP 站点，看一下该类 URL 的 location.protocol 值。

相关主题：location. href 属性。

更多资源请访问稀酷客(www.ckook.com)

search

值：字符串， 读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

只要在关键字域中输入字符，并使用 WWW 搜索服务寻找匹配项，就会在浏览器的 Location/Address 域中显示很长的、晦涩难懂的 URL。这个 URL 以常规的内容开始，如协议、主机和路径名，但在这些传统的 URL 部分之后是提交给搜索引擎(一般是在服务器上运行的CGI 程序)的搜索命令。location.search 属性可以用于获取或者设置这个尾部的搜索查询。

每个搜索引擎都有各自的查询提交格式，它们基于 HTML 表单的设计，这些表单用来从用户处获得信息。这些搜索查询采用编码格式，而不是纯语言。如果想编写搜索查询脚本，应完全理解搜索引擎的格式，才能开始组合字符串，并赋给窗口的 location.search 属性。

搜索数据最常用的格式是一系列名/值对，名称和值用等号(=)分开，多个名值对之间用(&)符号隔开。应该用 escape()函数将数据转换成 URL 适用的格式，在数据内容包含空格时尤其如此。

location.search 属性也可用于 URL 中文件名后的任何部分，包括传送到服务器上 CGI 程序的参数。

1. 通过 URL 在页面间传递数据

通常最好保留第一个页面中的一些数据，这样第二个页面的脚本才能得到第一个页面的脚本处理后的数据。可以采用下列三种方法在页面之间持久地保存数据，且不需要在服务器上编写任何程序：document.cookie(参见第 29 章)、框架集文档中的变量以及 URL 的搜索字符串。当访问搜索站点，进行电子交易，并将信息返回浏览器时，情况确实如此。它们不是把服务器上的搜索条件等保存下来，而是作为 URL 的一部分返回给浏览器。下次激活这个 URL 时，数据值将送到服务器上进行处理(例如，在另一个页面中发送回某个查询的搜索结果)。

在页面间传输数据并不仅限于客户端/服务器通信，也可在客户端使用搜索字符串进行页面间的数据传输。除非在服务器上编写了 CGI 进程，来处理搜索字符串，否则 Web 服务器就会将搜索字符串作为地址数据的一部分，与页面一起返回。新载入页面中的脚本可以(通过 location.search 属性)检测搜索字符串，然后从中获得相关数据，并将其赋给脚本变量。程序清单 28-6～程序清单 28-8 演示了此技术的一个应用。

如第 27 章所述，给定的 HTML 页面可在为其设计的框架集内强制打开。但通过搜索字符串可以重用同一个框架集文档，使其包含某个框架中任意数量的内容页面(而不是为框架集中每种可能的页面组合指定不同的框架集)。本节的程序清单创建了一个简单的示例，演示了如何给框架集传递某些页面信息，强制将页面载入到框架集中。这样，如果用户有某个内容框架的 URL(可能是右击框架，给该 URL 添加书签，或者是搜索引擎的结果)。用户下次访问页面时，页面就显示在其指定的框架集中。

本示例执行两个基本任务。第一是在每个内容页面中，脚本检查页面是否载入到框架集中。如果没有框架集，就构造搜索字符串，并添加到框架集文档的 URL 后面。框架集文档的简短脚本用于确定搜索字符串是否存在，如果存在，脚本就提取搜索字符串中的数据，并用它将指定的页面载入到框架集的内容框架中。

程序清单 28-6 是框架集文档。getSearchAsArray()函数的功能比这个简单示例所需要的更完

更多资源请访问稀酷客(www.ckook.com)

整，在其他情况下，可以用它把搜索字符串中任意数量的名称/值对(传统格式为 namel=Value&name2=Value2&etc.)转换为下标为名称的数组，以便于脚本提取传递过来的数据的特定部分。

程序清单 28-6　智能框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Example Frameset</title>
      <script type="text/javascript">
      // Convert location.search into an array of values
      // indexed by name.
      function getSearchAsArray()
      {
         var results = new Array();
         var input = unescape(location.search.substr(1));
         if (input)
         {
            var srchArray = input.split("&");
            var tempArray = new Array();
            for (var i = 0; i < srchArray.length; i++)
            {
               tempArray = srchArray[i].split("=");
               results[tempArray[0]] = tempArray[1];
            }
         }
         return results;
      }

      function loadFrame()
      {
         if (location.search)
         {
            var srchArray = getSearchAsArray();
            if (srchArray["content"])
            {
               self.content.location.href = srchArray["content"];
            }
         }
      }
      </script>
   </head>
   <frameset cols="250,*" onload="loadFrame()">
      <frame name="toc" src="jsb28-07.html" />
      <frame name="content" src="jsb28-08.html" />
   </frameset>
</html>
```

程序清单 28-7 是目录框架的 HTML，其内容并不复杂，但演示了正常的导航功能如何用于

更多资源请访问稀酷客(www.ckook.com)

这一简化的框架集，还说明了如何在此示例的基础上为有多个部分或页面的站点提供简便的目录功能。

程序清单 28-7 目录

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>Table of Contents</title>
   </head>
   <body bgcolor="#EEEEEE">
      <h3>Table of Contents</h3>
      <hr />
      <ul>
         <li><a href="jsb28-08.html" target="content">Page 1</a></li>
         <li><a href="jsb28-08a.html" target="content">Page 2</a></li>
         <li><a href="jsb28-08b.html" target="content">Page 3</a></li>
      </ul>
   </body>
</html>
```

程序清单 28-8 是一个内容页面。载入页面时，要调用 checkFrameset()函数。如果没有在框架集内载入窗口，脚本就导航到框架集页面，将当前内容的 URL 作为搜索字符串传递。注意，此页面的载入没有记录到浏览器的历史信息中，所以如果用户单击“后退”按钮，将不能访问该页面。

程序清单 28-8 内容页面

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Page 1</title>
      <script type="text/javascript">
      function checkFrameset()
      {
         if (parent == window)
         {
            // Use replace() to keep current page out of history
            location.replace("jsb28-06.html?content=" + escape(location.href));
         }
      }

      // Invoke the function
      checkFrameset();
      </script>
   </head>
   <body>
      <h1>Page 1</h1>
```


更多资源请访问稀酷客(www.ckook.com)

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

在实际中，最好将 checkFrameset()函数的代码及其调用放在一个外部.js 库中，并将该库链接到框架集的每个内容文档上。因此，函数将 location.href 通用属性赋给搜索字符串，以便在任何内容页面上使用它。

程序清单 28-6～程序清单 28-8 的代码建立了一个框架集，它包含两个框架。左侧框架是目录，允许用户在三个不同的页面上导航，第一个页面最初显示在右侧框架中。这个示例的有趣之处在于，可在搜索属性的 content 参数中指定一个新页面，然后在框架集中打开该页面。例如，下面的 URL 在右侧框架中打开 hello.html 页面：

```
jsb28-06.html?content=hello.html
```

由于此示例的 URL 包括了文件 jsb28-06.html，所以首先打开框架集页面，而 hello.html 文件指定为 content 参数的值。

注意：

这个框架集示例非常便于演示各种 location 对象属性的用法，但注意使用框架集不再是常见的编程实践方式。首先，它不支持可访问性，其次，CSS 定位功能是把内容放在网页上的标准方式。

相关主题： location.href 属性。

28.1.4 方法

assign("*URL*")

返回值： 无

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

前面讨论 location 对象时提到，将一个新的 URL 赋给 location 对象或者 location. href 属性，就可以导航到另一页。location.assign()方法的作用与此相同。实际上，将 location 对象设置为 URL 时，JavaScript 会自动使用 assign()方法。assign()方法除了代码更易理解以外，没有什么特别的优缺点。

相关主题： location. href 属性。

reload(*unconditionalGETBoolean*)

返回值： 无

兼容性： WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

location.reload()方法的名称可能不大合适，因为它可能与浏览器工具栏中的 Reload/Refresh 按钮混淆。实际上，reload()方法比 Reload/Refresh 按钮更强大，它会清除表单控件的值，而按下 Reload/Refresh 按钮将保留它们。注意，即使是软重载，Mac IE、Opera、WebKit 浏览器(如 Safari 和 Chrome)也不保留表单控件的设置。

更多资源请访问稀酷客(www.ckook.com)

在单击 Reload/Refresh 按钮之后，大多数表单元素的屏幕状态会保持不变，Text 和 textarea 对象的文本保持不变；单选按钮和复选框的选中状态保持不变；select 对象的选中项也会保持不变。可能 Reload/Refresh 清除的唯一项是全局变量值，以及可设置但不可见的属性(例如，隐藏的 input 对象的值)，因此这种加载叫做软重载。另一方面，硬重载则重置与页面关联的所有数据，包括默认的表单选择。

浏览器在内存缓存中重载文档的方式并不规范，这令人沮丧。理论上，如果页面仍然在缓存中，使用 location.reload()方法就应能从缓存中得到页面，不过 history.go(0)方法可能更好，它可以保存表单元素的设置。为该方法指定 true 参数，会强制将一个无条件的 GET 命令传到服务器，而忽略页面的缓冲版本。然而，当应用程序需要从缓冲区(为了提高速度)或服务器(为了获得刷新的版本)中获得页面时，浏览器的行为却完全背离了期望。Meta 标记有意设计成阻止页面缓存，但很少起作用。一些脚本设计者成功地从服务器中重载页面，方法是将 location.href 设置为页面的 URL，再加上一个稍微不同的搜索字符串(例如，基于 Data 对象的字符串表示)，这样缓存中就没有匹配的 URL。

必须准备尝试不同的方案，以获得理想的效果，当然也可能得不到需要的结果。换句话说，要学会接受一个事实：无法精确地控制获取新页面的过程。

程序清单 28-9 提供了一种方式，来测试软重载和硬重载的不同结果。在浏览器中打开此示例页面，选择单选按钮，然后输入一些新文本，在 select 对象中进行选择。单击 Soft Reload/Refresh 按钮会调用一个方法，该方法重载文档，就像用户单击了浏览器的 Reload/Refresh 按钮一样，它还保留了表单元素的可见属性。单击 Hard Reload 按钮则调用 location.reload()方法，将所有对象重置为其默认设置。

程序清单 28-9 硬重载与软重载

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Reload Comparisons</title>
      <script type="text/javascript">
      function hardReload()
      {
         location.reload(true);
      }
      function softReload()
      {
         history.go(0);
      }
      </script>
   </head>
   <body>
      <form name="myForm">
         <input type="radio" name="rad1" value="1" />Radio 1<br />
         <input type="radio" name="rad1" value="2" />Radio 2<br />
         <input type="radio" name="rad1" value="3" />Radio 3
         <p><input type="text" name="entry" value="Original" /></p>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <p><select name="theList">
              <option>Red</option>
              <option>Green</option>
              <option>Blue</option>
           </select></p>
        <hr />
        <input type="button" value="Soft Reload" onclick="softReload()" />
        <input type="button" value="Hard Reload" onclick="hardReload()" />
     </form>
  </body>
</html>
```

相关主题：history.go()方法。

replace("*URL*")

返回值：无

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

在较复杂的 Web 站点中，有些页面不应显示在用户的历史记录列表中。例如，注册步骤可能包含一个或多个对用户没有意义的中间 HTML 文档，某个一次性的介绍页面应只在用户第一次访问站点时显示。尤其没必要让用户在单击 Back 按钮，返回以前的 URL 时，再看见这些页面。可以使用 location.replace()方法导航到另一页，但它不允许当前页面保存在可以通过 Back 按钮访问的页面序列中。

当用户浏览页面时，不能阻止文档在历史记录列表中显示出来，但浏览器可将另一个文档载入窗口中，用新文档项替代当前的历史记录项。使用这个技巧可以不清空历史记录列表，但在下一个 URL 载入之前，可从历史记录列表中删除当前项。这样就可以防止用户单击 Back 按钮时，再看到此页。

程序清单 28-10 演示了如何使用 replace()方法将 Web 浏览器定向到新的 URL 上。调用 location.replace()方法会导航到另一个 URL 上，这类似于给 location 赋予 URL，区别在于新文档载入后，执行调用的文档不会显示在历史记录列表中。要验证这一点，可在单击程序清单 28-10 中的 Replace Me 按钮后，再试着单击 Back 按钮，返回到该页面。Back 按钮是灰显的，因为在浏览器历史记录中不再包含该页面。在单击 Replace Me 按钮前后，也要查看一下历史记录列表(在浏览器的通常位置上查看此信息)。

程序清单 28-10 调用 location. replace()方法

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>location.replace() Method</title>
      <script type="text/javascript">
      function doReplace()
      {
         location.replace("jsb28-01.html");
      }
```


更多资源请访问稀酷客(www.ckook.com)

```
    </script>
  </head>
  <body>
    <form name="myForm">
      <input type="button" value="Replace Me" onclick="doReplace()" />
    </form>
  </body>
</html>
```

相关主题：history 对象。

28.2　history 对象

属　　性	方　　法	事件处理程序
current	back()	(无)
length	forward()	
next	go()	
previous		

28.2.1　语法

访问 history 对象的属性或方法：

```
[window.]history.property | method([parameters])
```

28.2.2　关于 history 对象

用户浏览网页时，浏览器会保存一个最近访问页面的 URL 列表。在脚本对象模型中，这个列表用 history 对象表示。除非使用签名脚本(在 NN4+/Moz 中，参见配书光盘中的第 49 章)且用户许可，否则脚本不能提取列表中的 URL。如果没有签名，脚本就可以(通过相对编号或者每次后退一个 URL)有条不紊地导航到历史记录列表中的每个 URL，此时，用户会看到浏览器在自动导航，就像被幽灵控制一样。根据网络礼仪，没有用户明确授予的许可，就不能将用户导航到站点之外。

history 对象及其 back()或者 go()方法的一个应用是，在 HTML 文档中提供 Back 按钮的功能。这个按钮会触发脚本，检测历史记录列表中的项，然后返回上一页。文档不需要知道用户登录到页面的 URL 信息；它将导航细节传送回浏览器。

也可通过一对窗口方法 window.back()和 window. forward()来实现 Back 和 Forward 按钮的功能。history 对象方法是引用中的框架所特有的，parent.frameName.history.back()方法执行到该框架的历史记录的结尾时，就忽略对该方法的调用。

IE 的 history 机制不仅可用于框架集中的特定框架，其 history.back()和 history.forward()方法还可模拟单击工具栏按钮时的行为。如果想确保框架集中仍可以执行跨浏览器的操作，而不是跨浏览器版本的操作，最好将 history.back()和 history.forward()方法的引用传给父窗口。

更多资源请访问稀酷客(www.ckook.com)

Opera 的历史记录机制也很有趣。假定正在开发一个网页，将其加载到 Opera 中，观察其行为。接着编辑该页面，保存它，再返回 Opera，观察其行为。如果单击 Back 按钮，或者使用本章前面介绍的方法，就会返回刚才编辑的网页的前一个版本。

使用 history 对象及其方法时要特别小心。其应用程序的设计必须非常智能，能“观察”用户对页面所做的处理(例如，在用这些方法进行导航之前，检测当前的 URL)；否则，用户就有可能导航到意想不到的页面。此时脚本也可能出问题，因为在历史记录中，它检测不到当前文档在 Back/ Forward 序列中的位置。脚本也受浏览器历史记录机制的控制。

28.2.3 属性

current，next，previous

值：字符串，　　　　只读

兼容性：WinIE-，MacIE-，NN4+，Moz+，Safari-，Opera+ (只有 current 属性)，Chrome-

为了确定单击 Back 和 Forward 按钮后会到达哪个页面，浏览器保存了一个访问过的 URL 列表。如果想窥探隐私，想知道用户经常访问的页面和站点，这些信息是非常有用的。所以三个属性指定，历史记录列表中实际的 URL 只能在有签名脚本的页面中显示(NN4+/Moz)。只有获得许可的访问者才能访问浏览器的敏感数据(参见配书光盘中的第 49 章)。

有了签名脚本和用户的许可，就可以遍历任意框架和窗口的所有 history 数组项。因为该列表是数组，所以可通过索引值提取各个项。例如，如果数组有 10 项，则可以利用常规的数组索引方法查看第 5 项：

```
var fifthEntry = window.history[4];
```

没有属性和方法能直接显示当前载入的 URL 的索引值，但比较 history 对象的 current、next 和 previous 属性值与整个列表，就可以推测出来。

我们都不喜欢在上网时，有不明实体监视自己，因而最好不要使用这些权力，除非给用户提供了足够的警告。签名脚本许可对话框没有提供足够的细节来说明暴露这一信息的后果，这意味着即使通过签名脚本拥有默认许可，也应该明确地告诉用户：自己正在访问其历史记录。

相关主题：history.length 属性。

length

值：数值，　　　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

history.length 属性可以用于计算历史记录列表中的项数。不过，如果脚本相对于当前地址进行导航，这个信息就没有什么帮助，因为脚本不能从历史记录序列的当前文档地址中提取任何信息。如果当前文档在列表的顶部(这表示该文档是最近加载的)，就可以计算出相对于该文档的位置。但用户可使用 Go/View 菜单在历史记录列表中跳转。各个项在历史记录列表中的位置不会因用户导航回文档而改变。然而，history.length 为 1，表示当前文档是自从启动浏览器软件以来，用户载入的第一个文档。

程序清单 28-11 说明了如何使用 length 属性告知用户已经访问了多少个页面。

更多资源请访问稀酷客(www.ckook.com)

程序清单 28-11　浏览器历史记录计数

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>History Object</title>
      <script type="text/javascript">
      function showCount()
      {
         var histCount = window.history.length;
         if (histCount > 5)
         {
            alert("My, my, you\'ve been busy. You have visited " + histCount +
               " pages so far.");
         }
         else
         {
            alert("You have been to " + histCount + " Web pages this
               session.");
         }
      }
      </script>
   </head>
   <body>
      <form>
         <input type="button" name="activity" value="My Activity"
         onclick="showCount()" />
      </form>
   </body>
</html>
```

相关主题：无。

28.2.4　方法

back()，forward()

返回值：无

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

这些方法的名称可能让人误以为，它们模拟了浏览器工具栏上按钮的行为，其实并非如此。history.back()方法是窗口/框架特有的，表示如果对框架集中的框架连续调用 back()方法，则到达载入该框架的第一个文档时，则忽略该方法的进一步调用。Back 按钮和 window.back()方法都会卸载框架集，并返回浏览器的全局历史记录。

如果故意将用户引向网站中的死胡同，就应确保 HTML 文档提供了返回某个可识别位置的方式。没有工具栏或菜单栏的新窗口是很容易创建的(非 Macintosh 浏览器)，但这可能会使用户陷入困境，因为用户在这样的窗口中无法走出死胡同。为此，应在文档中包含一个按钮，为用

更多资源请访问稀酷客(www.ckook.com)

户提供返回上一位置的方式。

除非在导航到上一位置前需要进行一些额外的处理，否则可将此方法作为按钮的事件处理属性的参数。为保证在所有浏览器上兼容，在框架集中使用此方法时，应在父文档中调用它。

与 history.back()操作相比，更不容易编程实现执行相反操作的方法：在浏览器的历史记录列表中向前导航一步。唯一可以放心使用 history.forward()方法的地方是，在使用了 history.back()方法的同一脚本中，脚本会严密地跟踪其在两个方向上前进了多少步。使用 history.forward()方法要特别小心，只有对网页进行了大量的用户测试后才使用它，以确保覆盖所有可能的用户操作。与通过 history.back()向后导航类似，使用 history.forward()的前进过程只包括指定窗口或框架的历史记录，不包括整个浏览器的历史记录。

程序清单 28-12 和程序清单 28-13 提供了一个小练习，用于测试各种形式的前进和后退导航在不同浏览器中的行为。

程序清单 28-12 导航练习框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>Back and Forward</title>
   </head>
   <frameset cols="45%,55%">
      <frame name="controller" src="jsb28-13.html" />
      <frame name="display" src="jsb28-01.html" />
   </frameset>
</html>
```

程序清单 28-13 导航练习控制面板

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Lab Controls</title>
   </head>
   <body>
      <b>Load a series of documents into the right frame by clicking some
      of these links (make a note of the sequence you click on):</b>
      <p><a href="jsb28-01.html" target="display">Listing 28-1</a><br />
         <a href="jsb28-05.html" target="display">Listing 28-5</a><br />
         <a href="jsb28-09.html" target="display">Listing 28-9</a><br /></p>
      <hr />
      <form name="input">
         <b>Click on the various buttons below to see the results in this
         frameset:</b>
         <ul>
            <li><tt>history.back()</tt> and <tt>history.forward()</tt> for
               righthand frame:<input type="button" value="Back"
              onclick="parent.display.history.back()" /><input type="button"
```


更多资源请访问稀酷客(www.ckook.com)

```
                value="Forward" onclick="parent.display.history.forward()" />
                </li>
            <li><tt>history.back()</tt> for this frame:<input type="button"
                 value="Back" onclick="history.back()" /></li>
            <li><tt>history.back()</tt> for parent:<input type="button"
                value="Back" onclick="parent.history.back()" /></li>
        </ul>
      </form>
   </body>
</html>
```

相关主题：history.go()方法。

go(*relativeNumber* | "*URLOrTitleSubstring*")

返回值：无

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

history.go()方法可用于对当前存储在浏览器中的历史记录列表编写导航脚本。然而，如果选择把 URL 作为参数，该 URL 必须存在于历史记录列表中。因此，不要把这个方法看成将 window.location 对象设置为新 URL 的替代方式。

要沿着历史记录列表在前后两个方向上浏览 n 步，可使用 history.go()方法的参数 relativeNumber。这个值是整数，它相对于当前位置，表示列表中要使用的项。例如，如果当前的 URL 在列表顶部(即工具栏中的 Forward 按钮灰显)，则可使用下面的方法在列表中向后倒退两步：

```
history.go(-2);
```

换句话说，当前的 URL 就是 history.go(0)(这是重载窗口的方法)，其参数为正表示在历史记录列表中向前跳 n 步。因此，history.go(-1)和 history.back()相同，而 history.go(1)和 history.forward()相同。

另外，可指定保存在浏览器历史记录中的一个 URL 或文档标题(标题显示在 Go/View 菜单中)。随着人们对安全和隐私的关注程度的增长，go()方法的这个变体已禁用，最好不要在脚本中使用字符串参数。实际上，目前的浏览器版本都不支持这个变体。

与 history 对象的大多数其他方法一样，脚本很难管理当前 URL 在历史记录列表或者序列中的位置，这使脚本更难确定在两个方向上导航多远，或者跳转到哪个指定的 URL 或匹配的标题上。这个方法只用于严格控制用户活动的网页(或者为自己设计的脚本，该脚本可以在固定的范围内自动浏览站点)。一旦用户可控制导航，就无法保证历史记录列表与预期一样，任何依赖 history 对象的脚本都会出问题。

实际上，这个方法通常用 0 作为参数，执行当前窗口的软重载。

提示：

如果为所有脚本浏览器开发网页，则在 IE 中用 history.go (0)重载页面，通常会返回到服务器上，以重载页面，而不是从缓存中重载。

程序清单 28-14 的示例代码演示了如何使用 go()方法在历史记录列表中导航。在程序清单 28-14 的页面中填写数字或文本域，然后单击相应的按钮，脚本就将对应类型的数据传递给 go()

更多资源请访问稀酷客(www.ckook.com)

方法。除非使用旧浏览器版本，否则使用匹配的字符串文本框及其按钮是无效的。访问历史列表中较早的页面时，一定要使用负数。

注意：

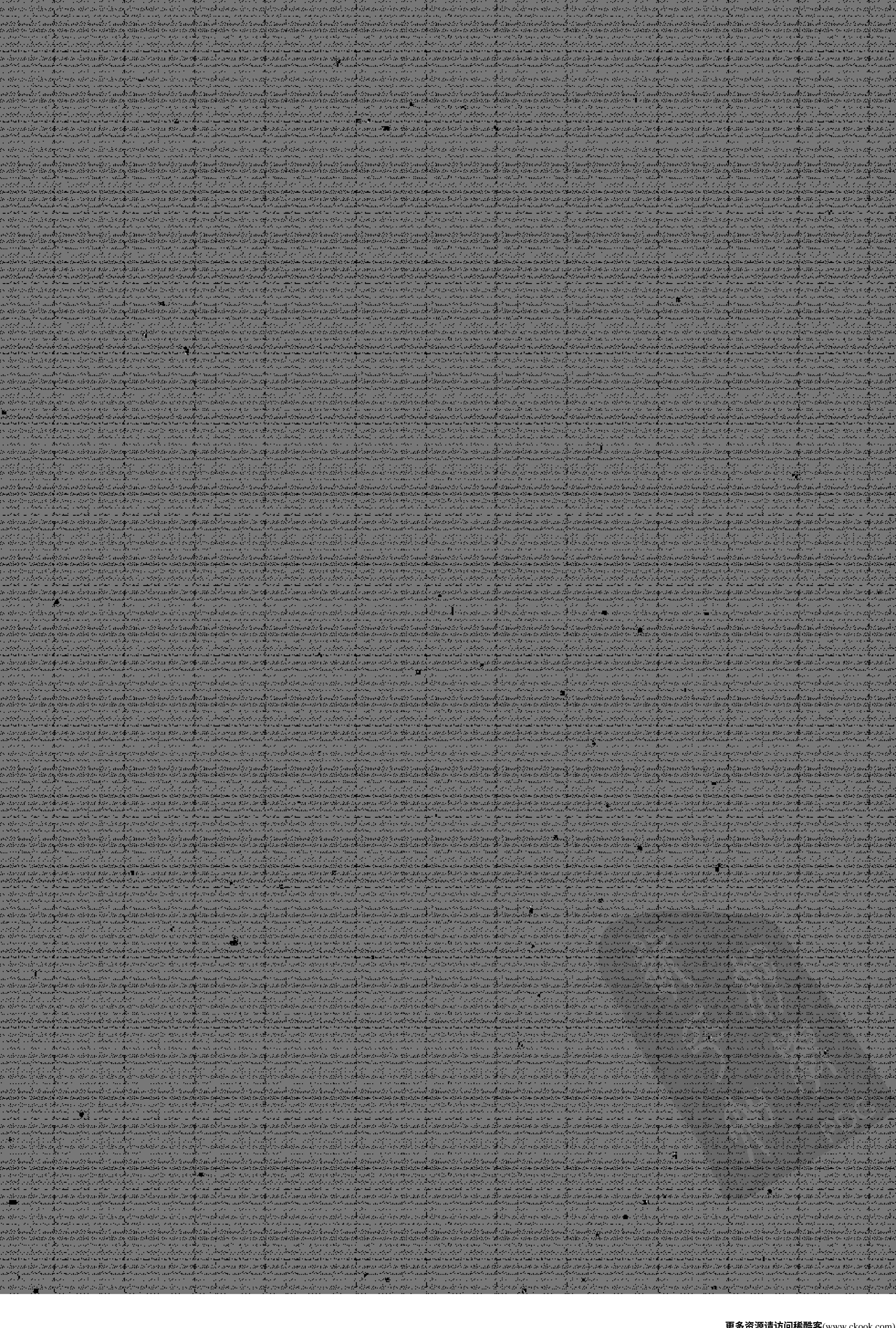
Mozilla 浏览器只响应使用 history.go()方法的整数偏移方式。

程序清单 28-14 浏览历史项

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>history.go() Method</title>
      <script type="text/javascript">
      function doGoNum(form) {
         window.history.go(parseInt(form.histNum.value));
      }
      function doGoTxt(form) {
         window.history.go(form.histWord.value);
      }
      </script>
   </head>
   <body>
      <form>
         <b>Calling the history.go() method:</b>
         <hr />
         Enter a number (+/-):<input type="text" name="histNum" size="3"
         value="0" /> <input type="button" value="Go to Offset"
         onclick="doGoNum(this.form)" />
         <p><div>Enter either a word found in a document title in your
                 history, or a word in a document URL in your history
                 (remember, this will only work if your browser version is
                 older):</div>
         <input type="text" name="histWord" />
            <input type="button" value="Go to Match"
            onclick="doGoTxt(this.form)" /></p>
      </form>
   </body>
</html>
```

相关主题：history.back()、history.forward()、location.reload()方法。

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第29章

document 对象和 body 对象

用户交互是客户端 JavaScript 脚本编程中一个至关重要的方面，脚本和用户之间的大多数交流都是通过 document 对象及其组件进行的。为成功地实现跨浏览器应用程序，就必须理解 document 对象在每种支持的对象模型中的作用域。

回顾一下 document 对象在原对象层次结构中的位置。如图 29-1 所示，document 对象是大部分对象的中枢点。在 W3C DOM 中，作为页面中所有元素对象的容器，document 对象具有更为重要的作用：document 对象是整个文档树的根。

本章包含哪些内容？

访问 document 对象包含的对象数组

在窗口或框架中写入新文档内容

使用 body 元素进行 IE 窗口度量

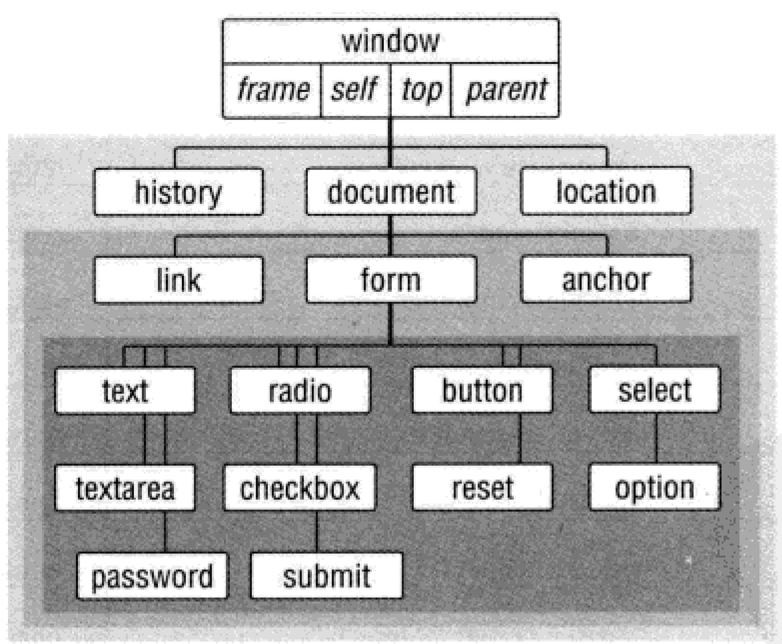


图 29-1　基本的文档对象模型层次结构

实际上，document 对象及其包含的全部内容非常广泛，所以对它的讨论分散在许多章节中，每一章都重点介绍相关的对象组。本章介绍 document 对象和 body 对象(它们有概念上的联系)，而本部分的后续章节则详述 document 对象包含的对象。

更多资源请访问稀酷客(www.ckook.com)

29.1 document 对象

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

属　性	方　法	事件处理程序
activeElement	attachEvent()†	onactivate†
alinkColor	captureEvents()	onbeforecut†
all†	clear()	onbeforedeactivate†
anchors[]	clearAttributes()†	onbeforeeditfocus†
applets[]	close()	onbeforepaste†
attributes†	createAttribute()	onclick†
baseURI	createCDATASection()	oncontextmenu†
bgColor	createComment()	oncontrolselect†
body	createDocumentFragment()	oncut†
charset	createElement()	Ondblclick†
characterSet	createElementNS()	ondrag†
childNodes†	createEvent()	ondragend†
compatMode	createEventObject()	ondragenter†
contentType	createNSResolver()	ondragleave†
cookie	createRange()	ondragover†
defaultCharset	createStyleSheet()	ondragstart†
defaultView	createTextNode()	ondrop†
designMode	createTreeWalker()	onhelp†
doctype	detachEvent()†	onkeydown†
documentElement	elementFromPoint()	onkeypress†
documentURI	evaluate()	onkeyup†
domain	execCommand()	onmousedown†
embeds[]	focus()†	onmousemove†
expando	getElementById()	onmouseout†
fgColor	getElementsByName()	onmouseover†
fileCreatedDate	getElementsByTagName()†	onmouseup†
fileModifiedDate	getElementsByTagNameNS()†	onpaste†
fileSize	hasFocus()	onpropertychange†
firstChild†	importNode()	onreadystatechange†
forms[]	mergeAttributes()†	onresizeend†

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	方　法	事件处理程序
frames[]	open()	onresizestart†
height	queryCommandEnabled()	onselectionchange
id†	queryCommandIndterm()	onstop
images[]	queryCommandState()	
implementation	queryCommandSupported()	
inputEncoding	queryCommandText()	
lastChild†	queryCommandValue()	
lastModified	recalc()	
layers[]	releaseCapture()†	
linkColor	releaseEvents()	
links[]	routeEvent()	
location	setActive()†	
media	write()	
mimeType	writeln()	
nameProp		
namespaces[]		
namespaceURI†		
nextSibling†		
nodeName†		
nodeType†		
ownerDocument†		
parentNode†		
parentWindow		
plugins[]		
previousSibling†		
Protocol		
readyState†		
Referrer		
scripts[]		
security		
selection		
strictErrorChecking		
styleSheets[]		

更多资源请访问稀酷客(www.ckook.com)

(续表)

属 性	方 法	事件处理程序
tags[]		
Title		
uniqueID†		
URL		
URLUnencoded		
vlinkColor		
width		
xmlEncoding		
xmlStandalone		
xmlVersion		

+参见第 26 章。

29.1.1 语法

访问 document 对象的属性或方法：

```
[window.]document.property | method([parameters])
```

29.1.2 关于 document 对象

document 对象包括浏览器窗口或者窗口框架的整个内容区域(不包括工具栏、状态栏等)。文档是内容和用于访问 Web 页面的界面元素的组合。在现代浏览器中，document 对象是页面节点层次树的根节点，所有其他节点都从根节点上生成。

在 HTML 文档中，document 对象没有明确地使用标记或其他符号表示出来，所以 JavaScript 和对象模型的设计者把 document 对象作为许多设置的入口，这些设置在 HTML 中属于 body 元素。于是，body 元素的标记包含文档范围的属性，例如背景色(bgcolor)和不同状态下的链接颜色(alink、link 和 vlink)。同时，body 元素也用作表单、链接和锚点的 HTML 容器，因此 document 对象具有 body 元素的大部分功能。尽管如此，document 对象仍非常便于绑定一些超出 body 元素范围的属性，例如 title 元素和把用户指向页面的链接的 URL。阅读 HTML 源代码时联系前后文，就会发现最初的 document 对象分为好几部分，即便如此，document 对象仍可作为原始对象模型中对象(如表单、图像和 applet)的引用基础。

当然，在所有 HTML 元素(包括 body 元素)在现代模型对象中用作对象之前是这样。令人惊讶的是，即使在 IE4+对象模型和 W3C DOM(两者都将 body 元素当作不同于 document 对象的对象)中，仍可轻松地兼容脚本与原始对象模型。document 对象有一种新的特点：一方面它位于原始对象模型中，另一方面，document 对象位于层次结构的根部，与其包含的 body 元素对象分离开来。document 对象知道自己在后面的脚本语句上将以何种“面目”出现，这意味着可以采用多种方法获得同一引用。例如，在所有的脚本浏览器中，可以用下面的语句获得文档中 form 对象的数目：

更多资源请访问稀酷客(www.ckook.com)

```
document.forms.length
```

在 IE4+中，也可以使用如下语句：

```
document.tags["form"].length
```

而在 IE5 和 NN6+/Moz/Safari/Opera/Chrome 实现的 W3C DOM 中，语句如下：

```
document.getElementsByTagName("form").length
```

现代浏览器提供了访问元素的通用方法，在 W3C DOM 中是 getElementsByTagName()方法，以便在对象模型中将每个 HTML(和 XML)元素当作对象。

将 body 元素提升为对象，为新对象模型的设计者提出了挑战。body 元素拥有的一些属性原本是原始 document 对象的默认属性。大多数属于原始 document 对象的属性在转为归属于 body 元素时进行了重命名，例如，原始 document.alinkColor 属性在新模型中为 body.aLink，但 bgColor 属性没有重命名。为使代码相互兼容，现代浏览器能识别两种属性，但 W3C DOM 给新 document 对象删除了旧版本属性。现代浏览器现在很普遍，所以应坚持使用新属性。

29.1.3 属性

activeElement

值：对象引用，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera+，Chrome+

在 IE4+中，脚本可检测 document.activeElement 属性，以确定哪个元素当前拥有焦点，返回值是一个元素对象引用。使用第 26 章的属性和方法可找到对象的更多细节。

尽管触发鼠标或者键盘事件的元素最有可能拥有焦点，但不要依靠 activeElement 属性确定哪个元素触发了事件，使用 IE 的 event.srcElement 属性更加可靠。

示例

在 IE4+中使用 The Evaluator(参见第 4 章)试验 activeElement 属性。在顶部文本框中输入下面的语句：

```
document.activeElement.value
```

按回车键后，Results 框就会显示用户刚才在文本框中输入的值(刚才输入的表达式)。如果接着单击 Evaluate 按钮，Results 框就会显示该按钮对象的 value 属性。

相关主题：event.srcElement 属性。

alinkColor，bgColor，fgColor，linkColor，vlinkColor

值：三个十六进制值或颜色名称字符串，　　一般可读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

这 5 个属性是同名<body>标记特性的脚本版本(但属性名是区分大小写的)。在现代浏览器中，这 5 个设置都可以通过 document.body 对象来读取。所有颜色属性的值都可以是常见的 HTML 十六进制三元组值(例如“#00FF00")，或标准颜色名称。

更多资源请访问稀酷客(www.ckook.com)

示例

随便选择一些颜色值，应用到程序清单 29-1 中难看颜色组的三个设置上。小窗口显示了一个哑元按钮，以便对比它们与颜色设置。注意，该脚本重写了整个窗口的 HTML 代码，来设置小窗口的颜色。更改颜色后，脚本在原窗口的文本区域中显示颜色值。即使有些颜色是用颜色常量设置的，属性也会恢复为十六进制三元组值。可随意更改程序清单中的颜色值，进行试验。每次更改脚本中的颜色值时，都要保存 HTML 文件，并在浏览器中重载它。

程序清单 29-1　调整页面元素的颜色

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Color Me</title>
    <script type="text/javascript">
      // may be blocked at load time by browser popup blockers
      var newWindow = window.open("","","height=150,width=300");

      function defaultColors()
      {
        return "bgcolor='#c0c0c0' vlink='#551a8b' link='#0000ff'";
      }

      function uglyColors()
      {
        return "bgcolor='yellow' vlink='pink' link='lawngreen'";
      }

      function showColorValues()
      {
        var result = "";
        result += "bgColor: " + newWindow.document.bgColor + "\n";
        result += "vlinkColor: " + newWindow.document.vlinkColor + "\n";
        result += "linkColor: " + newWindow.document.linkColor + "\n";
        document.forms[0].results.value = result;
      }

      // dynamically writes contents of another window
      function drawPage(colorStyle)
      {
        // work around popup blockers
        if (!newWindow || newWindow.closed)
        {
          newWindow = window.open("","","height=150,width=300");
        }
        var thePage = "";
        thePage += "<html><head><title>Color Sampler<\/title><\/head><body ";
        if (colorStyle == "default")
        {
```


更多资源请访问稀酷客(www.ckook.com)

```
        thePage += defaultColors();
    }
    else
    {
        thePage += uglyColors();
    }
    thePage += ">Just so you can see the variety of items and colors, <a ";
    thePage += "href='http://www.nowhere.com'>here\'s a link<\/a>, and <a
    href='http://home.netscape.com'> here is another link <\/a> you can
    use on-line to visit and see how its color differs from the standard
    link.";
    thePage += "<form>";
    thePage += "<input type='button' name='sample' value='Just a Button'>";
    thePage += "<\/form><\/body><\/html>";
    newWindow.document.write(thePage);
    newWindow.document.close();
    showColorValues();
}

// the following works properly only in Windows Navigator
function setColors(colorStyle)
{
    if (colorStyle == "default")
    {
        document.bgColor = "#c0c0c0";
    }
    else
    {
        document.bgColor = "yellow";
    }
}

// bind the event handlers
function addEvent(elem, evtType, func)
{
    if (elem.addEventListener)
    {
        elem.addEventListener(evtType, func, false);
    }
        else if (elem.attachEvent)
    {
        elem.attachEvent("on" + evtType, func);
    }
    else
    {
        elem["on" + evtType] = func;
    }
}
addEvent(window, "load", function()
{
```


更多资源请访问稀酷客(www.ckook.com)

```
            addEvent(document.getElementById("default1"), "click",
               function(evt) {drawPage("default")});
            addEvent(document.getElementById("weird1"), "click",
               function(evt) {drawPage("ugly")});
            addEvent(document.getElementById("default2"), "click",
               function(evt) {setColors("default")});
            addEvent(document.getElementById("weird2"), "click",
               function(evt) {setColors("ugly")});
         });
      </script>
   </head>
   <body>
      Try the two color schemes on the document in the small window.
      <form>
         <input type="button" id="default1" name="default" value='Default Colors'/>
         <input type="button" id="weird1" name="weird" value="Ugly Colors" />
         <p>
            <textarea name="results" rows="3" cols="20"></textarea>
         </p>
         <hr />
         These buttons change the current document.
         <p>
             <input type="button" id="default2" name="default"
                value='Default Colors' />
             <input type="button" id="weird2" name="weird" value="Ugly Colors" />
         </p>
      </form>
   </body>
</html>
```

注意：

本章的示例使用了现代的事件处理方法，包括 addEventListener()(NN6+/Moz/W3C)和 attach Event()(IE5+)方法。有关事件处理技术细节，请参阅第 32 章。

相关主题： body.aLink、body.bgColor、body.link、body.text、body.vLink 属性。

anchors[]

值： anchor 对象数组，　　　　只读

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在 HTML 文档中，anchor 对象(参见第 30 章)是用<a name="">标记的点。anchor 对象在 URL 中用页面的 URL 和锚点名之间的 hash 值来引用。与其他包含一系列嵌套对象的对象属性一样，document.anchors 属性也包含文档中锚点的索引数组。用数组引用定位指定的锚点，就可以获得锚的属性。

anchor 数组的索引值从 0 开始，因此，文档中的第一个锚点的引用是 document.anchors[0]。与内置的 array 对象一样，可以通过检测 length 属性来确定数组中有多少项。例如：

```
alert("This document has " + document.anchors.length + " anchors.");
```


更多资源请访问稀酷客(www.ckook.com)

document.anchors 属性是只读的。要通过脚本导航到特定的锚点，只需给 window.location 或者 window.location.hash 对象赋值即可，这与第 28 章中的 Location 对象一样。

示例

程序清单 29-2 给程序清单 28-1 额外添加了一段脚本，以演示如何确定文档中的准确锚点数量。该文档动态输出文档中的锚点数量。这类信息可能不需要提供给页面的用户，document.anchors 属性也不会经常被调用。在定义实际的锚点对象时，对象模型自动将其定义为文档属性。

程序清单 29-2　使用锚点在页面中导航

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.anchors Property</title>
      <script type="text/javascript">
         function goNextAnchor(where)
         {
            window.location.hash = where;
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener) {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            addEvent(document.getElementById("next1"), "click",
               function(evt) {goNextAnchor("sec1")});
            addEvent(document.getElementById("next2"), "click",
               function(evt) {goNextAnchor("sec2")});
            addEvent(document.getElementById("next3"), "click",
               function(evt) {goNextAnchor("sec3")});
            addEvent(document.getElementById("next4"), "click",
               function(evt) {goNextAnchor("start")});
         });
      </script>
   </head>
   <body>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <h1><a id="start" name="start">Top</a></h1>
        <form>
           <input type="button" id="next1" name="next" value="NEXT" />
        </form>
        <hr />
        <h1><a id="sec1" name="sec1">Section 1</a></h1>
        <form>
           <input type="button" id="next2" name="next" value="NEXT" />
        </form>
        <hr />
        <h1><a id="sec2" name="sec2">Section 2</a></h1>
        <form>
           <input type="button" id="next3" name="next" value="NEXT" />
        </form>
        <hr />
        <h1><a id="sec3" name="sec3">Section 3</a></h1>
        <form>
           <input type="button" id="next4" name="next" value="BACK TO TOP" />
        </form>
        <hr />
        <p>
           <script type="text/javascript">
              document.write("<i>There are " +
                             document.anchors.length +
                             " anchors defined for this document<\/i>")
           </script>
        </p>
    </body>
</html>
```

相关主题：anchor、location 对象；document.links 属性。

applets[]

值：applet 对象数组，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

applets 属性指向文档中用<applet>标记定义的 Java applet。applet 只有完全加载后，才是文档中真正的对象。

在 JavaScript 中，对 Java applet 的大部分处理都是通过 applet 中定义的方法和变量来完成的。尽管可根据 applet 在 applets 数组中的索引位置来引用它，但最好在引用中使用 applet 对象的名称，以避免混淆。

示例

浏览器为包含 applet 对象的文档创建对象模型时，会自动定义 document.applets 属性。此属性极少使用，除非像本例这样确定文档有多少个 applet 对象：

```
var numApplets = document.applets.length;
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：applet 对象。

baseURI

值：字符串，　　只读

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

baseURI 属性表示文档的绝对基准 URI。在 The Evaluator(参见第 4 章)中输入以下语句，可以查看文档的基准 URI：

```
document.baseURI
```

相关主题：document.documentURI 属性。

bgColor

详见 alinkColor。

body

值：body 元素对象，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

在现代对象模型中，document.body 属性是对 body 元素对象的快捷引用。在本章后面对 body 元素对象的讨论中提到，这个对象的许多重要属性可控制整个页面的外观。因为 document 对象是窗口或者框架中所有引用的根，所以可方便地通过 document. body 属性得到 body 属性，不需要通过更长的引用来访问 IE4+和 W3C 对象模型中的 HTML 元素对象。

示例

使用 The Evaluator(参见第 4 章)试验 body 元素对象的属性。首先，为了证明 document.body 就是较长引用返回的元素对象，在 IE5+、NN6+/Moz 或其他 W3C 浏览器中，在顶部文本框中输入下面的语句：

```
document.body == document.getElementsByTagName("body")[0]
```

接下来参阅本章稍后的 body 对象属性列表，将下面的代码输入到顶部文本框中，以便查看结果。例如：

```
document.body.bgColor
document.body.tagName
```

此例的要点是 document.body 引用提供了一种更简单、更直接的方式，来访问文档中的 body 对象，而不必使用 getElementsByTagName()方法。

相关主题：body 元素对象。

charset

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

charset 属性表示字符集，浏览器(IE4+)最初用它来显示当前文档(该属性的 NN6+/Moz 版本称为 characterSet)。访问如下地址，可查看这个属性的可能值：

```
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets
```

每个浏览器和操作系统都有自己默认的字符集，也可以使用<meta>标记来设置这个属性值。

示例

使用 The Evaluator(参见第 4 章)试验 charset 属性。要查看页面使用的默认设置，可在顶部文本框中输入下面的语句：

```
document.charset
```

如果运行的是 WinIE5+，并输入以下语句，浏览器就会给页面应用另一个字符集：

```
document.charset = "iso-8859-2"
```

如果 Windows 版本没有将该字符集安装在系统中，浏览器可能请求下载并安装该字符集。

相关主题：characterSet、defaultCharset 属性。

characterSet

值：字符串， 读/写

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

characterSet 属性表示浏览器显示当前文档所使用的字符集(这个属性的 IE 版本称为 charset)。访问如下地址，可查看这个属性的可能值：

```
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets
```

每个浏览器和操作系统都有各自默认的字符集，也可以使用<meta>标记来设置这个属性值。

示例

在 NN6+/Moz 中使用 The Evaluator(参见第 4 章)试验 characterSet 属性。要查看页面使用的默认设置，可在顶部文本框中输入下面的语句：

```
document.characterSet
```

相关主题：charset 属性。

compatMode

值：字符串， 只读

兼容性：WinIE6+，MacIE6+，NN7+，Moz+，Safari+，Opera+，Chrome+

compatMode 属性表示文档的兼容模式，该模式由 DOCTYPE 元素的内容决定。此属性的值可以是字符串常量 BackCompat 或 CSSlCompat。compatMode 属性默认设置为 BackCompat，这意味着文档与 W3C 标准不兼容；换言之，文档使用 Quirks 模式。否则，文档就使用 Strict 模式，与 W3C 标准兼容。与标准兼容是指 CSS1 标准。

更多资源请访问稀酷客(www.ckook.com)

示例

检查文档的兼容模式，以执行某种模式特有的处理。以下示例说明了如何通过分支语句来处理向后兼容的文档：

```
if (document.compatMode == "BackCompat") {
    // perform backward compatible processing
}
```

相关主题： 标准兼容模式(参见第 4 章)。

contentType

值： 字符串， 只读

兼容性： WinIE-，MacIE-，NN7+，Moz+，Safari-，Opera-，Chrome-

contentType 属性指定了文档的内容类型(MIME 类型)。对于普通 HTML 文档，此属性的值是 text/html。

cookie

值： 字符串， 读/写

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

Web 浏览器中的 cookie 机制允许在客户计算机上以相当安全的方式存储小段信息。换句话说，如果在载入不同的 HTML 文档，或者从一个会话转移到另一个会话时，需要在客户端保存一些信息，就可以使用 cookie 机制。在用密码保护的 Web 站点上，cookie 通常用作一种保存用户输入的用户名和密码的方法。首次将这些信息输入一个表单时，服务器端表单处理程序会让浏览器将信息写回到硬盘上的 cookie(通常在密码加密后)。为了避免下次访问站点时再次输入密码和用户名，服务器会搜索为这个服务器保存的 cookie 数据，提取用户名和密码，并在后台自动执行验证。

cookie 还可以用于存储用户上次访问站点时的首选项和信息，首选项可能包括字体风格或者字号，以及用户是否在框架集中查看内容。如配书光盘中的第 57 章所示，通过上次访问的时间标记，编码的 HTML 页面就会在自用户上次访问后改变的内容旁边高亮显示图片，即使期间多次刷新页面，也会显示该图片。同时，脚本还为访问者高亮显示新内容，而不是显示固定的 New 标志。

1. cookie 文件

允许外来的服务器程序在硬盘上读写数据，可能会使应用程序暂停，但浏览器的 cookie 机制不会打开驱动器目录，让别人来浏览(或者破坏)，而是只允许访问文本文件(Internet Explorer)或特殊的文本文件除 IE 外的其他浏览器，这些文件存放在驱动器上某个与平台相关的位置中。

例如，在基于 Mozilla 的浏览器中，cookie 文件命名为 cookies.txt，放在浏览器配置文件区的某个目录中(其名称以.slt 结尾)；在 Windows 中，该位置是 C:\\Windows\Application Data\Mozilla\Profiles\[profilename]\；在 Mac OSX 中，该位置是[user]/Library/Mozilla/Profiles/[profilename]/。WinIE 使用不同的文件系统：每个域的所有 cookie 都保存在 C:\\Windows\Temporary Internet Files\目录

更多资源请访问稀酷客(www.ckook.com)

的一个域专用文件中，该文件名以 Cookie:开头，包括用户名和写入 cookie 的服务器的域。Safari 的 cookie 放在[user]/Library/Cookies/目录的 XML 文件 Cookies.plist 中。Opera 的 cookies 放在[user]\Application Data\Opera\Opera 目录下的二进制文件 cookies4.dat 中。Google Chrome 的 cookies 放在[user]\Local Settings\Application Data\Google\Chrome\User Data\Default 目录下的 SQLite 数据库文件 Cookies 中。注意如果用 Google Chrome 进行测试，除非文件来自一个 HTTP 服务器，否则代码无效。Chrome 在 file:///文档中有意禁用了 cookies。

cookie 文件是文本文件。如果因为好奇而打开 cookie 文件，建议只操作保存在另一个目录或文件夹中的副本。对现有文件的任何更改都可能会搅乱为定期访问的站点保存的有效 cookie 数据。cookie 文件的数据格式在浏览器中各不相同，用于归档 cookie 的方法也不相同。Mozilla 文件(在警告用户不要手动更改文件的少量注释行之后)是用 Tab 键分隔的文本行，每个用回车分隔的行包含一个 cookie 信息。cookie 文件就像是数据库的文本列表。在 IE 的每个 cookie 文件中都存储了与 Mozilla cookie 相同的数据点，但各个项放在用回车分隔的列表中。这些文件的结构对 cookie 脚本编程并不重要，因为所有浏览器都使用相同的语法，通过 document.cookie 属性来读写 cookie。

注意：

试验浏览器的 cookie 时，在脚本给 cookie 写入一些数据后，不要查看 cookie 文件。cookie 文件通常不包含新写入的数据，因为在大多数浏览器中，cookie 文件仅在用户退出浏览器后才传送到磁盘；相反，cookie 文件在浏览器启动时才读入浏览器的内存。在浏览器会话中读写和删除 cookie 时，所有这些活动都在内存中进行(以加速进程)，然后保存。

2. cookie 记录

每个 cookie 记录的字段都包括以下内容(不一定是这个顺序)：

- 创建 cookie 的服务器域名。
- 是否需要安全的 HTTP 连接来访问 cookie 的信息。
- 可访问 cookie 的 URL 路径名。
- cookie 的截止日期。
- cookie 项名。
- 与 cookie 项有关的字符串数据。

注意，cookie 是域专用的，换句话说，如果一个域创建了 cookie，则另一个域不能通过浏览器的 cookie 机制访问它，所以在 cookie 中保存“用完即弃”密码(访问一些免费注册站点时需要的用户名/密码对)总是安全的。而且，在 cookie 中存储密码的站点通常使用加密的字符串，所以更难从没人照看的计算机上窃取 cookie 文件，也不能识别个人密码。

cookie 也有截止日期。因为一些浏览器不允许 cookie 的数目超过某个值(在 Firefox 中为 1000)，于是 cookie 文件就会逐渐被塞满。因此，如果 cookie 要在浏览器的当前会话结束后继续存在，其作者应该给它指定一个截止日期。浏览器将会自动清除到期的 cookie。

然而，并非所有的 cookie 都要在当前会话结束后继续存在。实际上，浏览 Web 站点时临时使用的 cookie 就是一个典型的例子。许多购物站点用一个或多个临时的 cookie 记录作为购物车，来记录访问者购买的物品，结账时这些 cookie 会复制到订单表中，但将订单表提交给服务

更多资源请访问稀酷客(www.ckook.com)

器后，这些客户端数据就失去价值了。此时，如果脚本不指定截止日期，浏览器就在内存中持续刷新 cookie，但不将其写入 cookie 文件。于是，退出浏览器时，cookie 数据就会丢失。

3. JavaScript 访问

在 JavaScript 中对 cookie 进行脚本访问时，只能(使用一些可选的参数)设置 cookie，获得 cookie 数据(但没有参数)。

原始对象模型将 cookie 定义为文档的属性，但这种描述有点误导性。如果用默认的路径设置 cookie(即脚本第一次设置 cookie 时的当前文档目录)，该服务器目录中的所有文档都可以读写 cookie。其优点是，如果一个脚本应用程序包含多个文档，则同一目录下的所有文档都可共享 cookie 数据。然而，在现代浏览器中，每个域至多有 20 个命名的 cookie 项(即一个名/值对)。如果 cookie 超过该限制，就需要改变连接 cookie 数据的方式。参见配书光盘的第 58 章中的 Decision Helper 应用程序。

4. 保存 cookie

要将 cookie 数据写入 cookie 文件，需要给 document.cookie 属性使用简单的 JavaScript 赋值操作符，但数据的格式对能否写入成功至关重要。给 cookie 赋值的语法如下(可选项在方括号中，数据的占位符使用斜体)：

```
document.cookie = "cookieName=cookieData
            [; expires=timeInGMTString]
            [; path=pathName]
            [; domain=domainName]
            [; secure]"
```

下面试验每个属性。

5. 名称/数据

每个 cookie 必须有一个名称和一个字符串(即使其值是空字符串)，这样的名/值对在 HTML 中非常常见，但它们在赋值语句中看起来很古怪。例如，如果想将字符串 Fred 保存在名为 userName 的 cookie 中，应使用下面的 JavaScript 语句：

```
document.cookie = "userName=Fred";
```

如果浏览器在当前域中没有找到这个名称的 cookie，将会自动创建 cookie 项；否则，浏览器就用新数据替换旧数据，此时检查 document.cookie，会得到下面的字符串：

```
userName=Fred
```

其他所有的 cookie 设置属性都可以忽略，此时浏览器使用默认值，如下一节所述。对于在当前浏览器会话结束后不需保存的临时 cookie，通常只需确定名/值对。

完整的名/值对必须是没有分号、逗号或者字符空格的单个字符串。为了防止词之间出现空格，应使用 JavaScriptescape()函数进行预处理，它用 URI 编码将空格变为%20，在以后提取 cookie 时，务必通过 decodeURIComponent()将该值恢复可读的空格。

更多资源请访问稀酷客(www.ckook.com)

不能在 cookie 中保存 JavaScript 数组或者对象，但使用 Array.join()方法可以将数组转换成字符串；在以后读取 cookie 后，再用 String.split()重建数组。

6. 截止日期

如果提供了截止日期，它就必须传递为格林尼治标准时间(Greenwich Mean Time, GMT)字符串(有关时间数据的信息，请参见第 17 章)。为了从当天日期开始计算截止日期，可使用 JavaScript 的 Date 对象，如下所示：

```
var exp = new Date();
var oneYearFromNow = exp.getTime() + (365 * 24 * 60 * 60 * 1000);
exp.setTime(oneYearFromNow);
```

由于 getTime()和 setTime()方法都把毫秒用作其单位，因此添加到当前日期上的年份必须转换为毫秒，计算完毕后，再将日期转换成可接受的 GMT 字符串格式：

```
document.cookie = "userName=Fred; expires=" + exp.toGMTString();
```

在 cookie 文件中，截止日期和时间都存储为数值(秒)，但设置它时，必须使用 GMT 格式的时间。将指定 cookie 文件的截止日期设置为早于当前日期和时间，就可以删除它。最安全的截止日期参数是：

```
expires=Thu, 01-Jan-70 00:00:01 GMT;
```

忽略截止日期，就表示 cookie 是临时的，浏览器不将其写入 cookie 文件，会在关闭浏览器后删除它。

7. 路径

对于客户端的 cookie，默认路径设置(当前目录)通常是最佳选择。当然，可以在另一个路径(和域)上创建 cookie 的一个副本，使同一数据可用于站点(或者 Web)其他地方的文档。

8. 域

为了使 cookie 数据与某个文档(或者文档组)同步，浏览器会匹配当前文档的域与 cookie 文件中 cookie 项的域值。因此，如果想显示 document.cookie 属性中的所有 cookie 数据，必须比较其域参数与当前文档的域参数，然后从两者相同的 cookie 文件中返回所有的名/值 cookie 对。

除非希望在域的另一个服务器中复制文档，否则在保存 cookie 时，通常可以忽略 domain 参数。默认行为将自动为 cookie 文件项提供当前文档的域。注意，域设置至少有两个句点，如下：

```
.google.com
.hotwired.com
```

也可以给域写入一个完整的 URL，包括 http://协议。

更多资源请访问稀酷客(www.ckook.com)

9. 安全

如果在保存 cookie 时忽略 SECURE 参数，就意味着 cookie 数据可由站点中匹配其他域和路径属性的任何文档或服务器端程序访问。对于客户端的 cookie 脚本，应在保存 cookie 时忽略这个参数。

10. 获取 cookie 数据

通过 JavaScript 获取的 cookie 数据通常放在一个字符串中，其中包括完整的“名-数据”对。即使 cookie 文件为每个 cookie 存储了其他参数，使用 JavaScript 也只能得到“名-数据”对。而且，若两个或多个(最多 20 个) cookie 满足当前域的条件，这些 cookie 也会组合到该字符串中，用分号和空格分隔开。例如，document.cookie 字符串如下所示：

```
userName=Fred; password=NikL2sPacU
```

换句话说，不能将指定的 cookie 当作对象；而必须解析整个 cookie 字符串，从期望的“名-数据”对中提取数据。

仅处理 cookie(其他数据不会添加到域中)时，可根据已知数据(如 cookie 名)来定制提取操作。例如，若某个 cookie 名有 7 个字符，就可以用如下语句提取数据：

```
var data = decodeURIComponent(document.cookie.substring
    (7,document.cookie.length));
```

substring()方法的第一个参数是将名称与数据分开的等号，它在代码中位于第 7 个位置。这个例子处理一个 cookie，只是因为它假设 cookie 在 cookie 文件的起始位置，如果文件包含多个 cookies，就不是这样。

提取 cookies 的更好方法是创建一个通用函数，它可以处理单项或者多项 cookie。例如：

```
function getCookieData(labelName) {
   var labelLen = labelName.length;
   // read cookie property only once for speed
   var cookieData = document.cookie;
   var cLen = cookieData.length;
   var i = 0;
   var cEnd;
   while (i < cLen) {
      var j = i + labelLen;
      if (cookieData.substring(i,j) == labelName) {
         cEnd = cookieData.indexOf(";",j);
         if (cEnd == -1) {
            cEnd = cookieData.length;
         }
         return decodeURIComponent(cookieData.substring(j+1, cEnd));
      }
      i++;
   }
   return "";
}
```


更多资源请访问稀酷客(www.ckook.com)

调用这个函数时，将期望的 cookie 标签名作为参数进行传递。函数将分析整个 cookie 字符串，通过分号去除不匹配的项，直到找到 cookie 名为止。

如果仍不明白这些 cookie 代码，可使用一个函数集，它由 hIdaho Design 的经验丰富的 JavaScript 开发人员和 Web 站点设计者 Bill Dortch 开发。他的 cookie 函数提供了所有 cookie 相关网页都可使用的常用 cookie 访问功能。程序清单 29-3 列出了 Bill 的 cookie 函数，它包括一些安全功能，以避免 Netscape Navigator 旧版本中的日期计算错误。这些代码使用现代 URL 编码和解码方法来更新。该程序清单看似很长，其实大部分是注释。

程序清单 29-3　Bill Dortch 的 Cookie 函数

```
<html>
   <head>
      <title>Cookie Functions</title>
   </head>
   <body>
      <script type="text/javascript">
      //
      // Cookie Functions -- "Night of the Living Cookie" Version (25-Jul-96)
      //
      // Written by: Bill Dortch, hIdaho Design
      // The following functions are released to the public domain.
      //
      // This version takes a more aggressive approach to deleting
      // cookies. Previous versions set the expiration date to one
      // millisecond prior to the current time; however, this method
      // did not work in Netscape 2.02 (though it does in earlier and
      // later versions), resulting in "zombie" cookies that would not
      // die. DeleteCookie now sets the expiration date to the earliest
      // usable date (one second into 1970), and sets the cookie's value
      // to null for good measure.
      //
      // Also, this version adds optional path and domain parameters to
      // the DeleteCookie function. If you specify a path and/or domain
      // when creating (setting) a cookie**, you must specify the same
      // path/domain when deleting it, or deletion will not occur.
      //
      // The FixCookieDate function must now be called explicitly to
      // correct for the 2.x Mac date bug. This function should be
      // called *once* after a Date object is created and before it
      // is passed (as an expiration date) to SetCookie. Because the
      // Mac date bug affects all dates, not just those passed to
      // SetCookie, you might want to make it a habit to call
      // FixCookieDate any time you create a new Date object:
      //
      // var theDate = new Date();
      // FixCookieDate (theDate);
      //
      // Calling FixCookieDate has no effect on platforms other than
      // the Mac, so there is no need to determine the user's platform
```


更多资源请访问稀酷客(www.ckook.com)

```
// prior to calling it.
//
// This version also incorporates several minor coding improvements.
//
// **Note that it is possible to set multiple cookies with the same
// name but different (nested) paths. For example:
//
// SetCookie ("color","red",null,"/outer");
// SetCookie ("color","blue",null,"/outer/inner");
//
// However, GetCookie cannot distinguish between these and will return
// the first cookie that matches a given name. It is therefore
// recommended that you *not* use the same name for cookies with
// different paths. (Bear in mind that there is *always* a path
// associated with a cookie; if you don't explicitly specify one,
// the path of the setting document is used.)
//
// Revision History:
//
// "JavaScript Bible 6th Edition" Version (28-July-2006)
//    - Replaced deprecated escape()/unescape() functions with
//      encodeURI() and decodeURI() functions
//
// "Toss Your Cookies" Version (22-Mar-96)
//    - Added FixCookieDate() function to correct for Mac date bug
//
// "Second Helping" Version (21-Jan-96)
//    - Added path, domain and secure parameters to SetCookie
//    - Replaced home-rolled encode/decode functions with
//      new (then) escape/unescape functions
//
// "Free Cookies" Version (December 95)
//
//
// For information on the significance of cookie parameters,
// and on cookies in general, please refer to the official cookie
// spec, at:
//
// http://www.netscape.com/newsref/std/cookie_spec.html
//
//******************************************************************
//
// "Internal" function to return the decoded value of a cookie
//
function getCookieVal (offset)
{
   var endstr = document.cookie.indexOf (";", offset);
     if (endstr == -1)
     {
       endstr = document.cookie.length;
```


更多资源请访问稀酷客(www.ckook.com)

```
      }
   return decodeURI(document.cookie.substring(offset, endstr));
}

//
// Function to correct for 2.x Mac date bug. Call this function to
// fix a date object prior to passing it to SetCookie.
// IMPORTANT: This function should only be called *once* for
// any given date object! See example at the end of this document.
//
function FixCookieDate (date)
{
   var base = new Date(0);
   var skew = base.getTime(); // dawn of (Unix) time - should be 0
     if (skew > 0)
     { // Except on the Mac - ahead of its time
       date.setTime (date.getTime() - skew);
     }
}

//
// Function to return the value of the cookie specified by "name".
//    name - String object containing the cookie name.
//    returns - String object containing the cookie value, or null if
//       the cookie does not exist.
//
function GetCookie (name)
{
   var arg = name + "=";
   var alen = arg.length;
   var clen = document.cookie.length;
   var i = 0;
   while (i < clen)
   {
     var j = i + alen;
       if (document.cookie.substring(i, j) == arg)
       {
          return getCookieVal (j);
       }
     i = document.cookie.indexOf(" ", i) + 1;
       if (i == 0)
       {
          break;
       }
   }
   return null;
}
//
// Function to create or update a cookie.
//    name - String object containing the cookie name.
```


更多资源请访问稀酷客(www.ckook.com)

```
//    value - String object containing the cookie value. May contain
//      any valid string characters.
//    [expires] - Date object containing the expiration data of the
//      cookie. If omitted or null, expires the cookie at the end of the
//      current session.
//    [path] - String object indicating the path for which the cookie is
//      valid.
//      If omitted or null, uses the path of the calling document.
//    [domain] - String object indicating the domain for which the cookie
//      is valid. If omitted or null, uses the domain of the calling
//      document.
//    [secure] - Boolean (true/false) value indicating whether cookie
//      transmission requires a secure channel (HTTPS).
//
// The first two parameters are required. The others, if supplied, must
// be passed in the order listed above. To omit an unused optional
// field, use null as a place holder. For example, to call SetCookie
// using name, value and path, you would code:
//
//     SetCookie ("myCookieName", "myCookieValue", null, "/");
//
// Note that trailing omitted parameters do not require a placeholder.
//
// To set a secure cookie for path "/myPath", that expires after the
// current session, you might code:
//
//     SetCookie (myCookieVar, cookieValueVar, null, "/myPath", null,
//         true);
//
function SetCookie (name,value,expires,path,domain,secure)
{
   document.cookie = name + "=" + encodeURIComponent (value) +
       ((expires) ? "; expires=" + expires.toGMTString() : "") +
       ((path) ? "; path=" + path : "") +
       ((domain) ? "; domain=" + domain : "") +
       ((secure) ? "; secure" : "");
}

// Function to delete a cookie. (Sets expiration date to start of epoch)
//   name - String object containing the cookie name
//   path - String object containing the path of the cookie to delete.
//          This MUST be the same as the path used to create the
//          cookie, or null/omitted if
//          no path was specified when creating the cookie.
//   domain - String object containing the domain of the cookie to
//          delete. This MUST be the same as the domain used to
//          create the cookie, or null/omitted if no domain was
//          specified when creating the cookie.
//
function DeleteCookie (name,path,domain)
```


更多资源请访问稀酷客(www.ckook.com)

```
        {
            if (GetCookie(name))
            {
                document.cookie = name + "=" +
                    ((path) ? "; path=" + path : "") +
                    ((domain) ? "; domain=" + domain : "") +
                    "; expires=Thu, 01-Jan-70 00:00:01 GMT";
            }
        }
        //
        //   Examples
        //
        var expdate = new Date ();
        FixCookieDate (expdate); // Correct for Mac date bug (call only once)
        expdate.setTime (expdate.getTime() + (24 * 60 * 60 * 1000)); // 24 hrs
        SetCookie ("ccpath", "http://www.hidaho.com/colorcenter/", expdate);
        SetCookie ("ccname", "hIdaho Design ColorCenter", expdate);
        SetCookie ("tempvar", "This is a temporary cookie.");
        SetCookie ("ubiquitous", "This cookie will work anywhere in this ⮐
            domain",null,"/");
        SetCookie ("paranoid", "This cookie requires secure ⮐
            communications",expdate,"/",null,true);
        SetCookie ("goner", "This cookie must die!");
        document.write (document.cookie + "<br>");
        DeleteCookie ("goner");
        document.write (document.cookie + "<br>");
        document.write ("ccpath = " + GetCookie("ccpath") + "<br>");
        document.write ("ccname = " + GetCookie("ccname") + "<br>");
        document.write ("tempvar = " + GetCookie("tempvar") + "<br>");
        </script>
    </body>
</html>
```

11. 额外处理

对于给定的域，一个站点可能需要 20 多个 cookie。例如，在购物站点，无法预测消费者会在购物车 cookie 中添加多少物品。

因为每个命名的 cookie 都保存为纯文本文件，所以可创建基于文本的自定义数据结构，使每个 cookie 包含多个信息(但注意，在每个域的组合 cookie 中，最多只有 4000 个字符，每个“名/值”对最多只有 2000 个字符)。另一个技巧是确定 cookie 数据不使用的分隔符，例如，在 Decision Helper 中(参见配书光盘中的第 58 章)，可用句点隔开存储在 cookie 中的多个数据。

确定了分隔符后，必须编写函数，将这些“子 cookie”连接成一个 cookie 字符串，然后在另一端提取它们。如果希望持久保存客户端的数据，这个工作量虽然大一点，但绝对值得。

示例

试着用程序清单 29-3 中的最后一组语句来创建、读取和删除 cookie，还可以用 The Evaluator 来试验，将名/值对字符串赋给 document.cookie，然后查看 cookie 属性的值。

更多资源请访问稀酷客(www.ckook.com)

相关主题：String 对象方法(第 28 章)。

defaultCharset

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera-，Chrome+

defaultCharset 属性表示浏览器显示当前文档所使用的字符集。可在以下网址找到这个属性的可能值：

```
http://www.iana.org/assignments/character-sets
```

每个浏览器和操作系统都有各自的默认字符集，其值也可以通过<meta>标记设置。DefaultCharset 和 charset 属性之间的区别并不明显，主要原因是，两者都是可读写的(但修改 defaultCharset 属性不会在页面上出现可见的效果)。然而，如果脚本临时修改了 charset 属性，可用 defaultCharset 属性返回原来的字符集：

```
document.charset = document.defaultCharset;
```

示例

使用 The Evaluator(参见第 4 章)试验 defaultCharset 属性。要查看页面使用的默认设置，可在顶部文本框中输入以下语句：

```
document.defaultCharset
```

相关主题：charset、characterSet 属性。

defaultView

值：window 或 frame 对象引用，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

defaultView 属性返回对文档查看器对象的引用。查看器可显示文档，在 Mozilla 中，defaultView 属性返回包含文档的 window 或 frame 对象。此 W3C DOM Level 2 属性可以访问应用于任何 HTML 元素(通过 document.defaultView.getComputedStyle()方法)的计算过的 CSS 值。

相关主题：window 和 frame 属性；window.getComputedStyle()方法。

designMode

值：字符串，　　读/写

兼容性：WinIE5+，MacIE-，NN7.1，Moz1.4+，Safari+，Opera+，Chrome+

只有将 WinIE5+技术用作另一个应用程序的组成部分，才能在 IE 中使用 designMode 属性。这个属性确定浏览器模块是否用于 HTML 编辑。在 IE5+浏览器的 HTML 页面中修改这个属性没有任何作用。但在 Mozilla 上，这个属性可将 iframe 元素的 document 对象转换为 HTML 可编辑文档。有关详细信息和例子，请参见 http://www.mozilla.org/editor。

更多资源请访问稀酷客(www.ckook.com)

doctype

值：DocumentType 对象引用，　　　　只读

兼容性：WinIE+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

doctype 属性来自于 W3C Core DOM，它返回一个表示文档 DTD 信息的 DocumentType 对象。DocumentType 对象(假设在源代码中明确定义)是根文档节点的第一个子节点(因此也是 HTML 元素的兄弟节点)。在 IE 中，这个属性只能用于 XML 文档；在 HTML 文档中，它总是返回 null。

对于一般的 HTML 页面，表 29-1 列出了典型 DocumentType 对象的属性和值。未来的 DOM 规范将允许读写这些属性。

表 29-1　NN6+/Moz 中的 DocumentType 对象

属　　性	值
baseURI	http://www.dannyg.com/index.html
entities	null
internalSubset	null
name	HTML
nodeName	HTML
nodeType	10
notations	null
publicId	-//W3C//DTD XHTML 1.0 Transitional//EN
systemId	http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd

相关主题：Node 对象(第 25 章)。

示例

在 The Evaluator 网页(参见第 4 章)的底部文本域中输入下面的代码，来查看 document.doctype 对象：

```
document.doctype
```

注意，publicId 属性实际上设置为-//W3C//DTD HTML 4.0l Transitional //EN，不同于表 29-1 中的值，这表明 The Evaluator 页面将自己声明为 HTML 4.01 文档。

documentElement

值：HTML 或 XML 元素对象引用，　　　　只读

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

documentElement 属性返回一个 HTML(或 XML)元素对象的引用，该对象包含当前文档的所有内容。这个属性的名称容易让人误解，因为根文档节点不是元素，但其唯一的子节点是页面的 HTML(或 XML)元素。最多可以认为此属性给脚本提供了一个元素界面(element face)，其中的 document 对象和文档节点与当前载入浏览器的页面相关联。

更多资源请访问稀酷客(www.ckook.com)

document.documentElement 对象表示页面的 html 元素，而 docoument.body 表示 body 元素，所以 document.body 是 document.documentElement 对象的子元素。

示例

使用 The Evaluator(参见第 4 章)查看 documentElement 属性的行为。在 IE5+/W3C 中，给顶部文本框输入如下语句：

```
document.documentElement.tagName
```

结果得到 HTML。

相关主题：ownerDocument 属性(第 26 章)。

documentURI

值：字符串，　　只读

兼容性：WinIE-，MacIE-，NN8+，Moz1.7+，Safari+，Opera+，Chrome+

documentURI 属性表示文档的位置，它包含在 W3C DOM Level 3 中，对应于非 W3C DOM 的 location. href 属性。使用 The Evaluator(参见第 4 章)输入如下语句，可查看文档 URI：

```
document.documentURI
```

相关主题：document.baseURI 属性。

domain

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，　Safari+，Opera+，Chrome+

安全限制会妨碍某些在其域中包含多个服务器的站点，因为一些对象(特别是 location 对象)禁止访问在其他框架中显示的其他服务器的属性，所以无法对这些属性进行合法访问。例如，受欢迎的网站通常会将其经常访问的站点放在 www.popular.com 服务器上。如果这个服务器上的某个页面包含一个前端，该前端指向位于 search.popular.com 的站点搜索引擎，那么用户使用具有这种域安全限制的浏览器，其访问将被拒绝。

为防止出现这种情况，可让两个服务器文档中的脚本指示浏览器把两个服务器看成一个。在前面的例子中，应在两个文档中将 document.domain 属性设置为 popular.com。若没有专门设置这个属性，则其默认值包括服务器名，导致主机名不匹配。

如果不能访问其他服务器，注意只能将 document.domain 属性设置为某些服务器(遵循“两点”规则)，这些服务器位于设置该属性的文档所在的域中。因此，只有源自 xxx.popular.com 的文档可将其 document.domain 属性设置为 popular.com 服务器。

相关主题：window.open 方法；window.location 对象；安全性(参阅配书光盘中的第 49 章)。

embeds[]

值：embed 元素对象数组，　　只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

尽管现在<embed>标记已被<object>标记替代，但<embed>可以加载必须用插件程序

更多资源请访问稀酷客(www.ckook.com)

运行或显示的数据。document.embeds 属性是文档中的一个 embed 元素对象数组：

```
var count = document.embeds.length;
```

相关主题：embed 元素对象(配书光盘中的第 44 章)。

expando

值：Boolean，　　读/写

兼容性：WinIE4+、 MacIE4+、 NN-、Moz-、Safari-、Opera-、 Chrome-

Microsoft 将不是 document 对象固有的自定义属性称为 expando 属性。默认情况下，在最新浏览器中，大多数对象允许脚本添加新的对象属性，来临时存储数据，而不需要明确定义全局变量。例如，如果想用一个独立的计数器记录函数的调用次数，可创建 document 对象的自定义属性，并将其用作存储工具：

```
document.counter = 0;
```

在 IE4+中，可以控制 document 对象是否接受 expando 属性。document.expando 属性的默认值为 true，允许使用自定义属性。但这种许可存在潜在的隐患，特别是在页面的创建阶段，document 对象可以接受无意中写错的固有属性名。如果将新的字符串赋给 document.Title 属性，就可能无法找出浏览器窗口的标题栏并不改变的原因(因为 JavaScript 区分大小写，该属性与固有的 document.title 属性有区别)。

示例

使用 The Evaluator(参见第 4 章)在 IE4+中试验 document.expando 属性。首先验证 document 对象可以正常接受自定义属性。在顶部文本框输入如下语句：

```
document.spooky = "Boo!"
```

现在设置了此属性，其值将一直保持到重载或卸载页面为止。

用如下语句冻结 document 对象的属性：

```
document.expando = false
```

如果试图添加一个如下的新属性，就会出错：

```
document.happy = "tra la"
```

有趣的是，即使关闭了 document.expando，仍然可访问和修改第一个自定义属性。

相关主题：自定义对象的 prototype 属性(参见第 23 章)。

fgColor

参见 alinkColor。

fileCreatedDate，fileModifiedDate，fileSize

值：字符串，整型(文件大小)，　　只读

更多资源请访问稀酷客(www.ckook.com)

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

这三个 IE 专用的属性返回包含当前文档的文件的信息，前两个属性(MacIE 没有实现它们)显示了当前文档文件的创建和修改日期。对于未修改的文件，创建和修改日期是相同的。fileSize 属性显示了文件的字节数。

前两个属性返回的日期值采用了 mm/dd/yyyy 格式。注意，这个值只包含日期，不包含时间。无论如何，都可以将这些值用作 new Date()构造函数的参数，还可以利用这些日期信息计算出上次修改与当日的间隔天数。

并非所有服务器都提供了文件的日期或者大小信息，或者采用 IE 能解释的格式，所以需要在部署服务器上测试这些属性，来确保兼容性。

还要注意，对于载入浏览器的文件，这些属性是只读的；对于存在于服务器上但未载入浏览器的文件，JavaScript 自身不能获得文件的这些信息。

示例

程序清单 29-4 动态生成了几个与文件创建和修改日期及大小有关的内容，还演示了如何将文件日期属性返回的值转换为可用于日期计算的真正日期对象。程序清单 29-4 计算了文件创建日期与某人查看文件的日期之间的完整天数。注意，在 body 内容的 span 元素中，innerText 属性添加了动态生成的内容。

程序清单 29-4　Web 页面的文件信息

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>fileCreatedDate and fileModifiedDate Properties</title>
      <script type="text/javascript">
         function fillInBlanks()
         {
            var created = document.fileCreatedDate;
            var modified = document.fileModifiedDate;
            document.getElementById("created").innerText = created;
            document.getElementById("modified").innerText = modified;
            var createdDate = new Date(created).getTime();
            var today = new Date().getTime();
            var diff = Math.floor((today - createdDate) / (1000*60*60*24));
            document.getElementById("diff").innerText = diff;
            document.getElementById("size").innerText = document.fileSize;
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
```


更多资源请访问稀酷客(www.ckook.com)

```
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            fillInBlanks();
         });
      </script>
   </head>
   <body>
      <h1>fileCreatedDate and fileModifiedDate Properties</h1>
      <hr />
      <p>This file (<span id="size"> </span> bytes) was created on
         <span id="created"> </span> and most recently modified on
         <span id="modified"> </span>.
      </p>
      <p>It has been <span id="diff"> </span> days since this file was
         created.
      </p>
   </body>
</html>
```

相关主题：lastModified 属性。

forms[ ]

值：数组，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在专门介绍 form 对象的第 34 章中提到，HTML 表单(定义在<form>...</form>标记对中的元素)是指向其本身的 JavaScript 对象。根据表单名(通过表单的 name 特性指定)就可以创建表单的有效引用。例如，如果文档包括如下表单定义：

```
<form name="phoneData">
  input item definitions
</form>
```

脚本就可以通过如下名称指向 form 对象：

```
document.phoneData
```

document 对象还通过另一种方式跟踪表单：form 对象的数组。document.forms 数组的第一项是最先载入的表单(它是 HTML 代码顶部的第一个表单)。如果文档定义了一个表单，form 属性就是一个长度为 1 的数组；当文档有 3 个表单时，数组的长度为 3。

更多资源请访问稀酷客(www.ckook.com)

使用标准的数组符号就可以引用 document.forms 数组中的特定表单。例如，文档中的第一个表单(document.forms 数组的“第 0 个”表项)可以引用为：

```
document.forms[0]
```

将 form 对象的属性或方法的名称附在上述引用之后，就可以调用它们。例如，要从文档的第二个表单中获得输入文本域 homePhone 的值，引用如下：

```
document.forms[1].homePhone.value
```

使用 document.forms 属性(而不是实际表单名)来定位 form 对象或者元素的一个好处是可以创建一个通用脚本库，来遍历文档的所有可用表单，找出具有特殊元素和属性的表单。下面的脚本段(第 21 章描述的重复循环的一部分)用一个循环计数变量(i)来遍历文档中的所有表单：

```
for (var i = 0; i < document.forms.length; i++) {
    if (document.forms[i]. ... ) {
        statements
    }
}
```

forms 数组引用的另一个变体是用表单名(字符串)代替 forms 数组索引。例如，phoneData 表单的引用如下：

```
document.forms["phoneData"]
```

如果十分小心地命名对象，则引用表单的格式可以是 document.formName。本书介绍了索引数组和表单名样式的引用，名称引用的优点在于，即使重新设计了页面，改变了表单在文档中的顺序，对指定表单的引用仍然有效，而表单的索引号可能会改变。参见第 34 章对 form 对象的讨论，并了解如何将表单数据传递到函数中。

示例

程序清单 29-5 中的文档用于显示一个警告对话框，该对话框根据 blues 复选框的选择状态，模拟导航到特定的音乐站点。这里的用户输入放在两个表单中：一个表单有复选框，另一个表单有导航按钮。在两个表单之间是一块样本内容。单击底部按钮(在第二个表单中)会触发一个函数，该函数将 document.forms[i]数组用作地址的一部分，来获取 blues 复选框的 checked 属性。

程序清单 29-5　简单表单示例

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.forms example</title>
      <script type="text/javascript">
         function goMusic()
         {
            if (document.forms[0].bluish.checked)
            {
```


更多资源请访问稀酷客(www.ckook.com)

```
                alert("Now going to the Blues music area...");
             }
             else
             {
                alert("Now going to the Rock music area...");
             }
          }
          // bind the event handlers
          function addEvent(elem, evtType, func)
          {
             if (elem.addEventListener)
             {
                elem.addEventListener(evtType, func, false);
             }
             else if (elem.attachEvent)
             {
                elem.attachEvent("on" + evtType, func);
             }
             else
             {
                elem["on" + evtType] = func;
             }
          }
          addEvent(window, "load", function()
          {
             addEvent(document.getElementById("visit"), "click", goMusic);
          });
       </script>
    </head>
    <body>
       <form name="theBlues">
          <input type="checkbox" name="bluish" />Check here if you've got the
          blues.
       </form>
       <hr />
       M<br />
       o<br />
       r<br />
       e<br />
       <br />
       C<br />
       o<br />
       p<br />
       y<br />
       <hr />
       <form name="visit">
          <input type="button" id="visit" value="Visit music site" />
       </form>
    </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

相关主题：form 对象(第 34 章)

frames[]

值：数组，　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera+，Chrome-

document.frames 属性类似于 window.frames 属性，但其与 document 对象的联系有时似乎不合逻辑。该属性返回的数组包含 window 对象，这意味着它们是已定义 frame 元素(来自框架集文档)或者 iframe 元素(来自纯 HTML 文档)的 window 对象。将 window 对象和 iframe 对象区分开来非常重要，window 对象的属性和方法与 frame 和 iframe 元素对象不同，后者的属性一般表示元素标记的特性。如果文档不包含 iframe 元素，document.frames 数组长度就是 0。

使用典型的数组语法(例如 document.frames[0])可以访问单个框架对象，也可以用 Microsoft 为对象集合提供的另一种语法：把索引号放在圆括号中，如下：

```
document.frames(0)
```

另外，如果给框架的 name 特性赋值，则可以用这个名称(字符串)作为参数：

```
document.frames("contents")
```

如果框架集合中有多个同名的框架，就必须特别小心。用此名作为参数，引用就会返回一组同名的 frame 对象。但可以用第二个可选的参数来指定索引，把返回值限制为只包括同名框架的单个实例。例如，如果文档的两个 iframe 元素具有相同的名字 contents，则脚本可以引用第二个 window 对象，如下：

```
document.frames("contents", 1)
```

为了确保跨浏览器的兼容性，应使用 window.frames 属性引用框架窗口对象。

示例

参见程序清单 27-7 和程序清单 27-8，查看结合使用 frames 属性与 window 对象的例子。

相关主题：window.frames 属性。

height，width

值：整型，　　　　只读

兼容性：WinIE-，MacIE-，NN4+，Moz+，Safari+，Opera-，Chrome+

height 和 width 属性指定了当前窗口(或框架)内容的像素尺寸。如果文档的内容区域小于浏览器的内容区域，这些属性返回的尺寸就包含窗口内容区域右边或底部的空白区；但如果文档内容超出了内容区域的可见范围，这个尺寸也包含不可见的内容。在 IE 中，对应的属性是 document.body.scrollHeight 和 document.body.scrollWidth。大多数现代浏览器也支持这些 IE 属性。

示例

使用 The Evaluator(参见第 4 章)查看该文档的 height 和 width 属性。在顶部文本框中输入如下语句，并单击 Evaluate 按钮：

更多资源请访问稀酷客(www.ckook.com)

```
"height=" + document.height + "; width=" + document.width
```

调整窗口的大小，使浏览器窗口显示垂直滚动条和水平滚动条，并再次单击 Evaluate 按钮。如果一个或两个属性值都变小，Results 框中的值就是文档所占据空间的大小；但如果扩大窗口，使窗口不再需要显示滚动条，这两个属性值就变为窗口内容区域在每个方向上的像素尺寸。

相关主题：document.body. scrollHeight、document.body.scrollWidth 属性。

images[]

值：数组，　　　　只读

兼容性：WinIE4+，MacIE3+，NN3+，Moz+，Safari+，Opera+，Chrome+

从 NN3 和 IE4 开始，图像就是一级对象，文档包含页面上定义的所有图像标记是很自然的(如链接和锚点那样)。将图像看成对象的重要优点是可以随时修改其内容(与图像矩形区域相关联的源文件)。image 对象详见第 31 章。

通过图像数组引用来确定特定的图像，可以获取图像的属性，或者为 src 属性指定新的图像文件。图像数组的索引计数从 0 开始：文档中第一个图像的引用是 document.images[0]。而且，与其他数组对象一样，可以检查 length 属性来确定数组包含的图像数目。例如：

```
var imageCount = document.images.length;
```

图像也可以有自己的名称，可以通过名称来引用图像对象：

```
var imageLoaded = document.imageName.complete;
```

或

```
var imageLoaded = document.images[imageName].complete;
```

document.images 数组可用于确定浏览器是否支持可交换图像。任何将 img 元素处理为对象的浏览器，总是为页面建立 document.images 数组。即使没有在页面上定义图像，该数组也存在，但其长度为 0。因此，该数组是否存在是确定 image 对象兼容性的线索。因为 document.images 数组如果存在，就是一个 array 对象，所以这个表达式可以用作条件表达式，建立用于图像交换的分支。

```
if (document.images) {
    // image swapping or precaching here
}
```

没有这个属性的浏览器(旧浏览器以及移动设备)会将 document.images 设置为 undefined，因此条件语句的值为 false。

示例

浏览器为包含图像对象的文档创建对象模型时，将会自动定义 document.images 属性，引用示例请参见第 31 章中对 Image 对象的讨论。

相关主题：Image 对象(第 31 章)。

更多资源请访问稀酷客(www.ckook.com)

implementation

值：对象，　　只读

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

核心 W3C DOM 使用 document.implementation 属性帮助脚本确定在当前环境下实现了什么 DOM 特征(即 DOM 标准模块)。虽然该属性返回的 DOMImplementation 对象没有属性，但拥有一个 hasFeature()方法，可帮助脚本找出某些特征。例如，环境是支持 HTML 还是只支持 XML。hasFeature()方法的第一个参数是特征的字符串形式；第二个参数是版本号的字符串形式，该方法返回一个 Boolean 值。

W3C DOM 规范的“一致性”部分指出如何管理模块名(该标准也允许通过 hasFeature()方法测试浏览器的特定特征)，模块名包括 HTML、XML 和 MouseEvents 等字符串。

W3C DOM 模块的版本号对应于 W3C DOM 级别。因此，DOM Level 2 中的 XML DOM 模块版本为 2.0。但要注意，这个版本是指 DOM 模块，而不是独立的 HTML 标准。

示例

使用 The Evaluator(参见第 4 章)试验 document.implementation.hasFeature()方法。在顶部文本框中一次输入一条语句，并查看结果：

```
document.implementation.hasFeature("HTML","1.0")
document.implementation.hasFeature("HTML","2.0")
document.implementation.hasFeature("HTML","3.0")
document.implementation.hasFeature("CSS","2.0")
document.implementation.hasFeature("CSS2","2.0")
```

尝试其他任意值。由于某种原因，到 IE7 为止，Internet Explorer 对其支持的一些功能返回 false，如 CSS 2.0。换句话说，至少在目前，最好不要过于相信 hasFeature ()方法在 IE 上的结果。

inputEncoding

值：字符串，　　只读

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari+，Opera-，Chrome+

文档的输入编码是在解析文档时有效的字符编码。例如，inputEncoding 属性报告的一个常见字符编码是 ISO-8859-1。

lastModified

值：日期字符串，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

每个磁盘文件都包含一个修改时间戳，大多数(但不是所有)服务器都在访问文件的浏览器上显示这一信息，通过读取 document.lastModified 属性就可以获得它。如果服务器给客户端提供了这条信息，也可以用这个属性值给 Web 页面的访问者显示这个信息。脚本会自动更新其值，而不需要每次在修改主页时，手工编写 HTML 代码行。注意如果测试 Safari 或 Google Chrome，代码就无效，除非文件来自于 HTTP 服务器。

如果返回值显示的日期在 1969 年，就意味着当前处于 GMT(即格林尼治标准时间)的西边

更多资源请访问稀酷客(www.ckook.com)

(从 GMT 的 1970 年 1 月 1 日向西几个时区)，在这种情况下，服务器处理文件时，不会提供正确的数据。有时配置服务器可以解决这个问题，但并非总是这样。

该属性的返回值并不是 date 对象，而是一个由时间和日期组成的字符串，就像文档的文件系统记录的那样。字符串的格式随浏览器和版本而异。然而，通常可将日期字符串转换成 JavaScript 的 date 对象，然后用这个 date 对象的方法提取所需的元素，重新编译，得到可读的形式。程序清单 29-6 显示了一个例子。

甚至本地的文件系统都不总是提供每个浏览器能正确解释的数据。但将同一文件放在 UNIX 或者 Windows 网络服务器上，通过网络来访问时，日期会正确显示。

示例

用程序清单 29-6 试验 document.lastModified 属性。但根据文件的位置，可能会得到不正确的结果。

程序清单 29-6　在页面上放入时间戳

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Time Stamper</title>
   </head>
   <body>
      <center>
         <h1>GiantCo Home Page</h1>
      </center>
      <script type="text/javascript">
         update = new Date(document.lastModified);
         theMonth = update.getMonth() + 1;
         theDate = update.getDate();
         theYear = update.getFullYear();
         document.writeln("<I>Last updated:" +
                          theMonth + "/" + theDate +
                          "/" + theYear + "<\/I>");
      </script>
      <hr />
   </body>
</html>
```

第 17 章讨论 Date 对象时提到，各国的日期格式差别极大。某些格式使用不同的日期元素顺序。在硬编码日期格式时，日期形式可能对页面的其他用户很陌生。

相关主题：Date 对象(第 17 章)。

layers[]

值：数组，　　　　只读

兼容性：WinIE-，MacIE-，NN4，Moz-，Safari-，Opera-，Chrome-

layer 对象是给对象模型提供定位元素的 NN4 方式。因此，document.layers 属性是文档中

更多资源请访问稀酷客(www.ckook.com)

定位元素的数组。NN4 废弃了 layer 对象和 document.layers 属性，其功能由 Mozilla 实现，所以它们不再占据重要位置。配书光盘中的第 43 章列举了几个例子，说明了如何用标准 W3C DOM 实现与 document.layers 属性相同的功能。

相关主题：layer 对象(配书光盘中的第 43 章)。

linkColor

参见 alinkColor。

links[ ]

值：数组，　　只读

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

document.links 属性与 document.anchors 属性类似，但数组中包含的是用<a hre f = " ">标记创建的 link 对象。使用数组引用定位一个特定的链接，就可以获取链接属性，例如在链接的 HTML 定义中指定的目标窗口。

links 数组索引从 0 开始：文档中第一个链接的引用为 document.links[0]。而且，与其他数组对象一样，可以通过检测 length 属性来确定数组有多少个项。例如：

```
var linkCount = document.links.length;
```

document.links 属性中的项是真正的 location 对象，这意味着 links[]数组的每个成员都拥有与 location 对象同样的属性。

示例

浏览器为包含链接对象的文档创建对象模型时，会自动定义 document.links 属性。除了确定文档中链接对象的数量之外，极少需要访问该属性。

相关主题：link 对象；document.anchors 属性。

location，URL

值：字符串，读/写和只读(参见正文)

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

document.URL 属性与 window.location 属性类似。如第 28 章所述，location 对象包含与当前在窗口或框架中载入的文档相关的众多属性。为 location 对象(或 location. href 属性)指定一个新 URL，就告诉浏览器将该 URL 的页面载入框架中。另一方面，document.URL 属性只是一个字符串，它显示当前文档的 URL(在 Navigator、Mozilla 和 Safari 中是只读的)。该值对脚本很重要，但该属性没有 window.location 对象的功能。不能更改该属性的值，因为文档只有一个 URL：文件在网络上(或用户硬盘)的位置以及访问文档所需的协议。

这是一个很好的特性。使用哪个引用(window.location 对象或 document.URL 属性)取决于用脚本完成的工作。如果用脚本载入一个新的 URL，来改变窗口的内容，就只能给 window.location 对象赋值。另外，如果脚本与 URL 的组成部分相关，则 location 对象的属性提供了获得该信息的最简捷方式。要得到文档 URL 的字符串形式(不管是在当前窗口还是在另一个框架)，可以使

更多资源请访问稀酷客(www.ckook.com)

用 document.URL 属性或者 window.location.href 属性。

注意：
document.URL 属性替代了旧的 document.location 属性，大多数浏览器仍支持这个旧属性。

示例

程序清单 29-7～程序清单 29-9 中的 HTML 文档建立了一个测试练习，以尝试在多框架环境中查看不同窗口和框架的 document.URL 属性。结果显示在一个表格中，还提供了 document.title 属性列表，以帮助识别被引用的文档。获取 window.location 对象属性时有一些安全限制，从另一个窗口或框架中获取 document.URL 属性时也具有这些限制。

程序清单 29-7　URL 的简单框架集例子

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.URL Reader</title>
   </head>
   <frameset rows="60%,40%">
      <frame name="Frame1" src="jsb29-09.html" />
      <frame name="Frame2" src="jsb29-08.html" />
   </frameset>
</html>
```

程序清单 29-8　针对不同的上下文显示位置信息

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>URL Property Reader</title>
      <script type="text/javascript">
         function fillTopFrame()
         {
            newURL=prompt("Enter the URL of a document to show in the top frame:","");
            if (newURL != null && newURL != "")
            {
               top.frames[0].location = newURL;
            }
         }

         function showLoc(item)
         {
            var windName = item.value;
            var theRef = windName + ".document";
            item.form.dLoc.value = decodeURIComponent(eval(theRef + ".URL"));
            item.form.dTitle.value = decodeURIComponent(eval(theRef + ".title"));
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
        // bind the event handlers
        function addEvent(elem, evtType, func)
        {
           if (elem.addEventListener)
           {
              elem.addEventListener(evtType, func, false);
           }
           else if (elem.attachEvent)
           {
              elem.attachEvent("on" + evtType, func);
           }
           else
           {
              elem["on" + evtType] = func;
           }
        }
        addEvent(window, "load", function()
        {
           addEvent(document.getElementById("opener"), "click", fillTopFrame);
           addEvent(document.getElementById("parent"), "click",
              function(evt) {showLoc(document.getElementById("parent"));});
           addEvent(document.getElementById("upper"), "click",
              function(evt) {showLoc(document.getElementById("upper"));});
           addEvent(document.getElementById("this"), "click",
              function(evt) {showLoc(document.getElementById("this"));});
        });
     </script>
  </head>
  <body>
     Click the "Open URL" button to enter the location of an HTML document to
     display in the upper frame of this window.
     <form>
        <input type="button" id="opener" name="opener" value="Open URL..." />
     </form>
     <hr />
     <form>
        Select a window or frame to view each document property values.
        <p>
           <input type="radio" id="parent" name="whichFrame" value="parent" />
           Parent window
           <input type="radio" name="whichFrame" id="upper"
              value="top.frames[0]" />
           Upper frame
           <input type="radio" name="whichFrame" id="this" value="top.frames[1]" />
           This frame
        </p>
        <table border="2">
           <tr>
              <td align="right">document.URL:</td>
              <td><textarea name="dLoc" rows="3" cols="30" wrap="soft">
```


更多资源请访问稀酷客(www.ckook.com)

```
                </textarea>
            </td>
         </tr>
         <tr>
            <td align="right">document.title:</td>
            <td><textarea name="dTitle" rows="3" cols="30" wrap="soft">
                </textarea>
            </td>
         </tr>
      </table>
   </form>
</body>
</html>
```

程序清单 29-9 URL 占位符页面的例子

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Opening Placeholder</title>
   </head>
   <body>
      Initial place holder. Experiment with other URLs for this frame (see
      below).
   </body>
</html>
```

相关主题：location 对象；location. Href、URLUnencoded 属性。

media

值：字符串， 读/写

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

document.media 属性指定了格式化内容的输出媒介。实际上在 IE7 及更早的版本中，这个属性返回一个空字符串，其实其目的是提供使用脚本设置 CSS2@media 规则(一种所谓的 at 规则，因其中的@符号而得名)的方式。这个样式表规则允许浏览器为每种输出设备指定不同的页面显示风格(例如，在打印机和屏幕上可以使用不同的字体)。然而实际上，这个属性是不可改变的，至少到 IE8 是这样。

相关主题：无。

mimeType

值：字符串， 只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

这个属性在 WinIE5+中是可读的，但严格说来，其值并不是 MIME 类型，至少不是传统的 MIME 格式。而且在 IE 5、6、7、8 版本中，该属性的返回值并不一致。这个属性主要用于 XML，而不是 HTML 文档环境。无论如何，这个属性并不表示当前浏览器支持的 MIME 类型。

更多资源请访问稀酷客(www.ckook.com)

nameProp

值：字符串，　　只读

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

nameProp 属性返回一个包含文档标题的字符串，与 document.title 相同。如果文档没有标题，nameProp 就包含一个空字符串。

相关主题：title 属性。

namespaces[]

值：namespace 对象数组，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

namespace 对象可动态导入基于 XML 的 IE 元素行为。namespaces 属性返回一个数组，它包含在当前文档中定义的所有 namespace 对象。

相关主题：无。

parentWindow

值：window 对象引用，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera+，Chrome-

document.parentWindow 属性返回包含当前文档的 window 对象的引用，其值与当前窗口的引用相同。

示例

为证明 parentWindow 属性指向文档的窗口，可在 The Evaluator(参见第 4 章)的顶部文本框中输入如下语句：

```
document.parentWindow == self
```

只有两个引用指向同一对象，上述表达式的值才是 true。

相关主题：window 对象。

plugins[]

值：数组，　　只读

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

document.plugins 属性返回的数组与 document.embeds 属性返回的 embed 元素对象数组相同，但这个属性已被 document.embeds 取代。

相关主题：document.embeds 属性。

protocol

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 document.protocol 属性返回协议的纯语言版本，此协议用来访问当前文档。例如，

更多资源请访问稀酷客(www.ckook.com)

如果在 Web 服务器上访问文件，此属性就返回 Hypertext Transfer Protocol。这个属性不同于 location.protocol 属性，后者返回的 URL 部分包含更模糊的协议简写形式(例如 http:)。一般而言，应对 Web 应用程序的用户隐藏所有这些信息。

示例

如果用 The Evaluator(第 4 章)来测试 document.protocol 属性，结果就是 File Protocol，因为当前是在本地硬盘或光盘上访问程序清单。不过，如果将 The Evaluator 网页上传到 Web 服务器，并在服务器上访问它，结果就是预期的 Hypertext Transfer Protocol。

相关主题： location.protocol 属性。

referrer

值： 字符串，　　　　只读

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在 JavaScript 的控制下，链接从一个文档指向另一个文档时，第二个文档可以显示包含链接的文档的 URL。document.referrer 属性包含了该 URL 的字符串。在根据用户上次访问的地址定制页面的内容时，这一特性是非常有用的。只有用户通过链接到达当前页面，referrer 才包含有效值，而其他导航方法(如通过历史记录、书签或者手工输入 URL)都会将这个属性设置为空字符串。

警告：

document.referrer 属性通常返回空字符串，除非在 Web 服务器上获得了文件。

示例

这个示例需要两个文档(还需要在 Web 服务器上访问文档)。程序清单 29-10 中的第一个文档只包含一行文本，作为指向第二个文档的链接。在程序清单 29-11 的第二个文档中，脚本会验证用户通过链接访问的文档。如果脚本知道该链接，就显示用户在第一个文档中的体验消息。再试着用 File 菜单中的 Open File 命令，在新的浏览器窗口中打开程序清单 29-11，看看脚本是否识别 referrer。

程序清单 29-10　Referrer 页面示例

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.referrer Property 1</title>
   </head>
   <body>
      <h1><a href="jsb29-11.html">Visit my sister document</a></h1>
   </body>
</html>
```


更多资源请访问稀酷客(www.ckook.com)

程序清单 29-11　当页面被通过链接访问时确定 Referrer

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.referrer Property 2</title>
   </head>
   <body>
      <h1>
         <script type="text/javascript">
            alert(document.referrer.length + " : " + document.referrer);
               if (document.referrer.length > 0 &&
                  document.referrer.indexOf("jsb29-10.html") != -1)
               {
                  document.write("How is my brother document?");
               }
               else
               {
                  document.write("Hello, and thank you for stopping by.");
            }
         </script>
      </h1>
   </body>
</html>
```

相关主题： link 对象。

scripts[]

值： 数组，　　　　只读

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

最初，IE 专用的 document.scripts 属性返回包含当前文档中所有 script 元素对象的数组。引用单个 script 元素对象不仅可以读取它与所有 HTML 元素对象(参见第 26 章)共享的属性，还可以读取脚本特定的属性，如 defer、src 和 htmlFor 等。实际的脚本可以通过 innerText 或者 text 属性访问任何 script 元素对象。

document.scripts 数组是只读时，单个 script 元素对象的许多属性都是可以改变的。添加或者删除 script 元素会影响 document.scripts 数组的长度。但脚本需要访问特定的 script 元素对象时，可以为其指定 id 特性，直接引用该元素。

这一属性与 W3C 浏览器表达式 document.getElementsByTagName("script")相似，它返回同一个对象的数组。

示例

可以用 The Evaluator(参见第 4 章)试验 document.scripts 数组。例如，如果在顶部文本框中输入如下语句，则在 The Evaluator 页面中只有一个 script 元素对象：

```
document.scripts.length
```


更多资源请访问稀酷客(www.ckook.com)

如果要查看该 script 元素对象的所有属性，可在底部文本框中输入如下语句：

```
document.scripts[0]
```

结果中包含 innerText 和 text 属性。如果将空字符串赋给这两个属性，脚本就会从对象模型中清除，但不从浏览器中清除。脚本之所以消失，是因为脚本载入后会缓存在对象模型之外。因此，如果在顶部文本框中输入如下语句：

```
document.scripts[0].text = ""
```

脚本内容就会从对象模型中消失，但随后单击 Evaluate 和 List Properties 按钮(它们调用 script 元素对象的函数)仍是有效的。

相关主题： script 元素对象(配书光盘中的第 40 章)。

security

值： 字符串，　　只读

兼容性：WinIE5.5+、MacIE-、 NN-、 Moz-、 Safari-、 Opera-、Chrome-

如果当前文档有一个安全证书，security 属性就显示这个安全证书的信息。

相关主题： 无。

selection

值： 对象，　　只读

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera+，Chrome-

document.selection 属性返回一个 selection 对象，其内容在浏览器窗口中显示为选中的主体文本。用户可以通过单击和拖放来选择这些文本，或者在脚本的控制下通过 WinIE TextRange 对象创建选中的文本(参见配书光盘中的第 33 章)。因为脚本通过 TextRange 对象执行选择操作(例如，查找选定文本的下一个实例)，所以常使用 document.selection.createRange()方法将选择内容转换成 TextRange 对象。selection 对象详见配书光盘中的第 33 章。

注意，不能使用用户界面元素(如按钮)让脚本与所选文本交互。单击按钮，会使按钮具有焦点，同时取消对所选文本的选择。但可以用其他的事件对所选文本执行动作，如 document.onmouseup。

示例

参见第 26 章的程序清单 26-36 和 26-44，查看 document.selection 属性在通过脚本执行复制和粘贴操作中的作用(仅用于 WinIE)。

相关主题： selection、TextRange 对象。

strictErrorChecking

值： 字符串，　　只读

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari-，Opera-，Chrome-

strictErrorChecking 属性指定了文档的错误检查模式。具体而言，如果该属性设置为 true(默认值)，就报告与 DOM 操作有关的异常和错误；否则，不会抛出与 DOM 相关的异常，也不报

更多资源请访问稀酷客(www.ckook.com)

告错误。

styleSheets[]

值：数组，　　　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.styleSheets 数组包含文档中所有 style 元素对象的引用。样式表不包括在这个数组中，它可以通过标记中的 style 特性指定给元素，或者通过 link 元素来链接。styleSheet 对象详见第 38 章。

相关主题：styleSheet 对象(第 38 章)。

title

值：字符串，　　　　只读和读写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

文档的标题是在 HTML 文档中 head 部分的<title>...< / title>标记对之间的文本。标题通常显示在单框架窗口的浏览器窗口标题栏中，或多面板浏览器窗口的选项卡面板中。只有最上面的框架集文档的标题才显示为多框架窗口的标题。即使这样，脚本仍可以使用框架中单个文档的 title 属性。例如，如果有两个框架(UpperFrame 和 LowerFrame)，则 LowerFrame 框架文档中的脚本可以引用 UpperFrame 框架文档的 title 属性，如下：

```
parent.UpperFrame.document.title
```

document.title 属性是从原始文档对象模型中延续而来的。在最新的浏览器中，HTML 元素的 title 属性有一种完全不同的应用方式(见第 26 章)。在现代浏览器中(IE4+/W3C/Moz/Safari/Opera/Chrome)，使用 title 的元素对象可以直接获取文档标题。

相关主题：history 对象。

URL

详见 location。

URLUnencoded

值：字符串，　　　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz+，Safari+，Opera+，Chrome+

document.URL 属性返回 URL 编码字符串，即将 URL 中非字母数字字符转换成对 URL 友好的字符(例如，将空格变成%20)。总是可以在 document.URL 属性的返回值上使用 decodeURI() 函数，但在 IE 中可以通过 URLUnencoded 属性做到这一点。如果在 URL 中没有 URL 编码字符，则这两个属性返回同一个字符串。

相关主题：document.URL 属性。

vlinkColor

详见 alinkColor。

更多资源请访问稀酷客(www.ckook.com)

width

详见 height。

xmlEncoding，xmlStandalone，xmlVersion

值：字符串，　　只读

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari+，Opera-，Chrome+

这三个属性显示与 XML 有关的文档信息。具体而言，它们分别表示文档的 XML 编码、文档是否是独立的 XML 文档，以及文档的 XML 版本号。如果不能确定这些属性值，它们的值就是 null。

29.1.4　方法

captureEvents(*eventTypeList*)

返回值：无

兼容性：WinIE-，MacIE-，NN4+，Moz-，Safari-，Opera-，Chrome-

只有在 Navigator 4 中，事件的自然传播方向才是从 window 对象向下，通过 document 对象，最终到达目标。例如，若单击一个按钮，则 click 事件先到达 window 对象，然后到达 document 对象。如果该按钮在一个层中定义，click 事件也会穿过该层，最终到达按钮，此处 onclick 事件处理程序将处理 click 事件。

语法不同的事件捕获功能已在 W3C DOM 中标准化，并且已经在 W3C 浏览器中实现，例如 Firefox 和 Camino(Mozilla)。具体而言，W3C 事件捕获模型引入了事件监听器的概念，将事件处理函数绑定到事件上。参见第 26 章中的 addEventListener()方法，它是与 NN4 的 captureEvents()方法相对应的 W3C 方法。另外，有关 W3C COM 中事件捕获和事件冒泡的更多细节，请参见第 32 章。

clear()

返回值：无

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

自从 NN2 开始，document.clear()方法就可以从浏览器窗口中清除当前文档。这个方法非常不实用，因为一般情况下，清除文档后，脚本还需要执行一些操作，但如果脚本消失了，就什么也不会执行。

实际上，document.clear()方法从不像期望的那样工作(在早期的浏览器中，这很容易导致浏览器崩溃)。最好不要使用 document.clear()，包括在使用 document.write()生成新页内容之前。document.write()方法在添加新内容之前，会从窗口中清除原文档。如果真的要清空窗口或者框架，应使用 document.write()编写一个空白的 HTML 文档，或从服务器中载入空的 HTML 文档。

相关主题：document.close()、document.write()、document.writeln()方法。

close()

返回值：无

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

只要通过 document.open()方法或者文档的写入方法(它也打开布局流)打开窗口的布局流，就必须在文档写入后关闭该流，此时状态栏会显示 Layout:Complete 和 Done 消息(但一些平台上的状态消息可能有错误)。文档的关闭步骤非常重要，只有关闭文档，窗口才能继续显示用脚本编写的新 HTML。如果不关闭文档，后面写入的内容将追加到文档的底部。

只有调用 document.close()方法后，为窗口指定的一些或全部数据才能正常显示，文档流中的图像尤其如此。一个常见的问题是文档部分先显示出来，之后立即消失。如果遇到这个问题，可能是在最后一个 document.write()方法后漏掉了 document.close ()方法。

修改粘滞沙漏光标

各种浏览器常常不能在调用 document.write()和 document.close()方法(和其他修改内容的脚本)之后，将光标恢复正常。在所有处理都结束后，光标会顽固地处于等待状态，或者进度条一直在显示。一个虽不雅观但有效的解决方法是通过 javascript:伪 URL 强制添加一个额外的 document.close()(只在脚本中添加另一个 document.close()不能解决这个问题)。为在框架集中使用这种解决方法，javascript:URL 必须指向框架集层次结构的顶部，而 document.close()方法指向内容已变的框架。例如，如果改变 content 框架的内容，则创建如下函数：

```
function recloseDoc() {
   top.location.href =
      "javascript:void (parent.content.document.close())";
}
```

如果将这个函数放在框架集文档中，可在执行完防止光标正常显示的操作后，使用修改 content 框架的脚本来调用上面的脚本。

示例

在试验 document.close()方法之前，需要先理解本章后面的 document.write()方法。之后，为该方法的示例创建三个独立的文档(程序清单 29-14～程序清单 29-16，在另一个目录或文件夹中)。在 takePulse()函数中，注释掉 document.close()语句，如下所示：

```
msg += "<p>Make it a great day!</body></html>";
parent.frames[1].document.write(msg);
//parent.frames[1].document.close();
```

现在，在浏览器中试验该页面。每次单击上面的按钮，都会将文本添加到底部框架的末尾，而无需首先删除以前的文本。这是因为，以前的布局流从未关闭，文档认为用户仍在向它写入内容。另外，如果不正确关闭流，最后一行文本可能不会显示在最近写入的内容中。

相关主题： document.open()、document.clear()、document.write()、document.writeln()方法。

createAttribute("*attributeName*")

返回值： Attribute 对象引用

兼容性： WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createAttribute()方法会创建一个 attribute 节点对象(W3C DOM 术语中的 Attr 对象)，

更多资源请访问稀酷客(www.ckook.com)

并返回新建对象的引用。调用此方法仅需指定 attribute 名称。之后脚本需要给新建对象的 nodeValue 属性赋值，再通过元素的 setAttributeNode()方法(参见第 26 章)将新的特性插入现存元素。下面的程序创建了一个特性，并使其成为 table 元素的特性：

```
var newAttr = document.createAttribute("width");
newAttr.nodeValue = "80%";
document.getElementById("myTable").setAttributeNode(newAttr);
```

特性不一定是 HTML 标准的特性，因为该方法也可以用于 XML 元素，而 XML 元素有自定义特性。

示例

要创建 attribute 并查看其属性，可在 The Evaluator(参见第 4 章)的顶部文本框中输入如下内容：

```
a = document.createAttribute("author")
```

现在在底部文本框中输入 a，查看 Attr 对象的属性。

相关主题：setAttributeNode()方法(第 26 章)。

createCDATASection("*data*")

返回值：CDATA 段对象引用

兼容性：WinIE-，MacIE5，NN7+，Moz+，Safari+，Opera+，Chrome+

document.createCDATASection ()方法为传递为参数的字符串生成一个 CDATA 段节点，新节点的值就是所传递的字符串。

createComment("*commentText*")

返回值：注释对象引用

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createComment()方法创建注释节点的一个实例。之后，该节点保存在内存中，并可以通过节点的 appendChild()或 insertBefore()方法插入到文档中。

相关主题：appendChild()和 insertBefore()方法。

createDocumentFragment()

返回值：文档片段对象引用

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createDocumentFragment()方法创建空文档片段节点的一个实例。此节点用作一个容器，来组合内存中的一系列节点。在创建节点，并组合到文档片段中后，就可以将整个片段插入到文档树中。

脚本组合任意系列的元素和文本节点时，文档片段尤其有用。在组合内容的过程中，片段节点为其中所有的内容提供父节点，给 W3C DOM 节点方法如 appendChild()提供了必要的父节点上下文。如果随后将文档片段节点添加或插入到文档树的元素中，片段包装器就会消失，但

更多资源请访问稀酷客(www.ckook.com)

其内容仍留在文档中的所需位置。因此，文档片段的典型用法是首先(通过 createDocumentFragment()方法)创建一个空的片段节点，用新建的元素或文本节点或两者填充它，然后将片段节点用作源材料，在文档树的元素上使用适当的节点方法进行添加、插入或替换。

相关主题：无。

createElement("*tagName*")，createElementNS("*namespaceURI*", "*tagName*")

返回值：元素对象引用

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createElement()和 document.createElementNS()方法为传递为参数的 HTML(或XML)标记名生成元素对象。采用这种方式创建的对象不是当前文档对象节点的正式组成部分，因为它还没有放入文档。但这些方法是组装元素对象，并最终插入文档的一种方式。createElementNS()方法与 createElement()方法相同，只是后者有一个额外的参数，用来传递元素的命名空间 URI。另外，在创建元素时，使用 createElementNS()指定的标记名必须是完全限定的名称。然而，Internet Explorer 直到版本 8 还不支持 createElementNS()。

此方法的返回值是对象引用。这个对象的属性包含了浏览器对象模型为元素对象定义的所有属性，且设置为默认值。因此，脚本能通过这个引用找到对象，然后设置对象的属性。一般情况下，应在对象插入文档之前完成属性的设置，否则在元素插入文档之前，可以修改只读属性。

在对象插入文档后，原引用(如用来保存 createElement()方法返回值的全局变量)仍指向对象，甚至它已放在文档中或者已显示给用户。下面的语句演示了这一效果，它创建了一个简单的段落元素，其中包含 text 节点：

```
var newText = document.createTextNode("Four score and seven years ago...");
var newElem = document.createElement("p");
newElem.id = "newestP";
newElem.appendChild(newText);
document.body.appendChild(newElem);
```

此时，新段落在文档中是可见的。现在可以修改段落的样式，例如，定位文档对象模型中的元素或包含所建对象引用的变量：

```
newElem.style.fontSize = "20pt";
```

或者

```
document.getElementById("newestP").style.fontSize = "20pt";
```

这两个引用密切相关，总是指向同一个对象。因此，如果想用脚本创建一系列类似的元素(例如一系列 li 列表元素)，可以用 createElement()创建第一个元素，再设置所有元素的共有属性。然后用 cloneNode()创建一个新副本，它可以看成独立的元素(可能要为每个元素指定唯一的 ID)。

在 W3C DOM 环境中编写脚本时，常常需要使用 document.createElement()为页面或其中一部分创建新内容(除非使用方便的 innerHTML 属性，以 HTML 的字符串形式添加内容)。在严格的 W3C DOM 环境中，创建新元素不是组合 HTML 字符串，而是创建真正的元素(和 text 节点)对象。

更多资源请访问稀酷客(www.ckook.com)

示例

第 26 章包含的众多示例结合使用了 document.createElement()方法与给文档添加或替换内容的方法。参见程序清单 26-10、26-21、26-22、26-28、26-29 和 26-31。

相关主题：document.createTextNode()方法。

createEvent("*eventType*")

返回值：事件对象引用。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createEvent()方法会创建指定事件类型的 W3C DOM Event 对象的一个实例。在创建时，一般事件必须初始化为特定的事件类型，并设置事件的其他相关属性。在成功地初始化事件后，可调用 dispatchEvent()方法来触发它。

Mozilla 识别的事件类型有 KeyEvents、MouseEvents、MutationEvents 和 UIEvents。从 Mozilla 1.7.5 开始，还可以使用以下类型：Event、KeyboardEvent、MouseEvent、MutationEvent、MutationNameEvent、TextEvent 和 UIEvent。在初始化这些事件类型的过程中，需要在相关的 initEvent()方法中有自己的参数序列，详见第 32 章。

示例

以下示例说明了如何创建事件、将其初始化为特定的事件类型，并发送给指定的元素：

```
var evt = document.createEvent("MouseEvents");
evt.initEvent("mouseup", true, true);
document.getElementById("myButton").dispatchEvent(evt);
```

相关主题：createEventobject()方法；W3C DOM 事件对象(第 32 章)。

createEventObject([*eventObject*])

返回值：event 对象。

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 createEventObject()方法会创建 event 对象，然后该对象可以作为参数传递给元素对象的 fireEvent()方法。使用这个事件创建的 event 对象类似于用户或者系统操作创建的 event 对象。

此方法的可选参数可在现有的 event 对象上建立新的事件。换句话说，新建 event 对象的属性继承了传递为参数的 event 对象的所有属性，并允许用户修改所选的属性。如果未给该方法提供参数，则必须手动填入必要的属性。event 对象详见第 32 章。

示例

创建要在元素上触发的事件时，需要遵循的步骤请参见第 26 章讨论的 fireEvent()方法。

相关主题：createEvent()方法、fireEvent()方法(第 26 章)；event 对象(第 32 章)。

createNSResolver(nodeResolver)

返回值：XPath 名称空间解析器对象引用

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

createNSResolver()方法用来在 XPath 中更改节点，使其能解析命名空间。该方法必须是 XPath 表达式的一部分。它唯一的参数是要用作命名空间解析器基础的节点。

相关主题： evaluate()方法。

createRange()

返回值： Range 对象引用

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

document.createRange()方法会创建一个空的 W3C DOM Range 对象，其范围的边界点折叠成一个点，显示在 body 文本的第一个字符前。

相关主题： Range 对象。

createStyleSheet(["*URL*"[,*index*]])

返回值： styleSheet 对象引用

兼容性： WinIE4+，MacIE4，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 createStyleSheet()方法可以创建 styleSheet 对象，这类对象包括 style 元素对象，以及通过 link 元素导入文档的样式表。因此，可在页面载入后动态载入外部的样式表。

与 W3C DOM 使用的其他创建方法不同，createStyleSheet()方法不仅创建样式表，而且立即将该对象插入文档对象模型。因此，属于(或赋予)该对象的样式表规则会立即应用于页面。如果希望创建一个样式表但延迟其应用，就应该使用 createElement()方法和元素对象组合技术。

如果在 WinIE 中不给该方法指定任何参数，该方法就会创建空的 styleSheet 对象。此时需要使用 styleSheet 对象的方法(如 addRule())给样式表添加细节。要链接外部的样式表文件，可以将文件的 URL 作为方法的第一个参数。新导入的样式表就会添加到 styleSheet 对象的 document.styleSheets 数组末尾。此方法的第二个可选参数可以精确地指定新链接的样式表在样式表元素序列中的插入位置。任何给定选项的样式表规则，都会替换为在文档的样式表序列中后来出现的同一选项的样式表。

示例

程序清单 29-12 演示了如何给文档添加内部和外部样式表。添加内部样式表时，addStylel()函数会调用 document.createStyleSheet()，并给控制页面中的 p 元素添加一个样式规则。在 addStyle2()函数中载入外部文件，该文件包含以下两个样式规则：

```
h2 {font-size:20pt; color:blue;}
p {color:blue;}
```

注意，为导入的样式表指定位置 0，则所添加的内部样式表总是在 styleSheet 对象序列的后面。因此，除非仅部署了外部样式表，否则 p 元素的红色文本颜色会取代外部样式表的蓝色。如果删除 addStyle2()中 createStyleSheet()方法的第二个参数，外部样式表就添加到列表的结尾。如果这是要添加的最后一个样式表，p 元素的文本就显示为蓝色。不断单击此示例中的按钮，继续给文档添加样式表。

更多资源请访问稀酷客(www.ckook.com)

程序清单 29-12　创建和应用样式

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.createStyleSheet() Method</title>
      <script type="text/javascript">
         function addStyle1()
         {
            var newStyle = document.createStyleSheet();
            newStyle.addRule("P", "font-size:16pt; color:red");
         }

         function addStyle2()
         {
            var newStyle = document.createStyleSheet("jsb29-12.css",0);
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            addEvent(document.getElementById("addint"), "click", addStyle1);
            addEvent(document.getElementById("addext"), "click", addStyle2);
         });
      </script>
   </head>
   <body>
      <h1>document.createStyleSheet() Method</h1>
      <hr />
      <form>
         <input type="button" id="addint" value="Add Internal" /> 
         <input type="button" id="addext" value="Add External" />
      </form>
      <h2>Section 1</h2>
      <p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do
```


更多资源请访问稀酷客(www.ckook.com)

```
        eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
        adminim veniam, quis nostrud exercitation ullamco laboris nisi ut
        aliquip ex ea commodo consequat.</p>
     <h2>Section 2</h2>
     <p>Duis aute irure dolor in reprehenderit involuptate velit esse cillum
        dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
        proident, sunt in culpa qui officia deseruntmollit anim id est
        laborum.</p>
  </body>
</html>
```

相关主题：styleSheet 对象(第 38 章)。

createTextNode("*text*")

返回值：对象

兼容性：WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

text 节点是一个 W3C DOM 对象，它包含没有任何 HTML(或者 XML)标记的主体文本，但 HTML 元素(或者 XML)通常包含 text 节点(也就是为其子元素)。W3C DOM 没有 IE 的 innerText 属性，因此只能依靠文档的节点层次结构来修改元素的文本(Mozilla 优越于 W3C DOM 的地方在于，它提供了 innerHTML 属性，可以用来替换元素的文本)。如果用 W3C DOM 方式插入或替换 HTML 元素的文本，可创建 text 节点，然后用父元素的方法(如 appendChild()、insertBefore()和 replaceChild()，详见第 26 章)修改文档的内容。要创建新的 text 节点，可以使用 document.createTextNode()。

createTextNode()方法的唯一参数是一个字符串，其文本是方法返回的 text 节点对象的 nodeValue 值；也可以创建一个空的 text 节点(传递空字符串)，以后再将字符串赋给对象的 nodeValue。只要在文档对象模型中提供 text 节点，脚本就可以改变 nodeValue 属性，来修改已有元素的文本。text 节点在 W3C DOM 中的作用详见第 25 章。

示例

第 25 章和第 26 章(例如程序清单 26-21)提供了使用 createTextNode()方法的许多示例，但要查看此方法在 IE5+/W3C 中生成的内容，最好使用 The Evaluator(参见第 4 章)。在顶部文本框中输入如下语句，用 The Evaluator 的一个内置全局变量来保存对新建文本节点的引用：

```
a = document.createTextNode("Hello")
```

Results 框显示对象已创建。现在，在底部文本框中输入 a，查看对象的属性。该属性列表在 IE5+和 W3C 浏览器上有所不同，但它们共有的 W3C DOM 属性表明，该对象的节点类型为 3，节点名称为#text。现在它还没有父节点、子节点或兄弟节点，因为这里创建的对象在明确添加到文档中之前，还不是文档层次树的组成部分。

为了帮助了解插入操作如何进行，可在顶部文本框中输入如下语句，将文本节点添加到 myP 段落中：

```
document.getElementById("myP").appendChild(a)
```

词语 Hello 显示在页面下方的简单段落的结尾。现在可以通过 p 包含元素的引用，或通过

更多资源请访问稀酷客(www.ckook.com)

新建节点的全局变量引用，来修改该节点的文本：

```
document.getElementById("myP").lastChild.nodeValue = "Howdy"
```

或

```
a.nodeValue = "Howdy"
```

相关主题：document.createElement()方法。

createTreeWalker(*rootNode*, *whatToShow*, *filterFunction*, *entityRefExpansion*)

返回值：TreeWalker 对象引用

兼容性：WinIE-，MacIE-，NN7+，Moz1.4+，Safari+，Opera+，Chrome+

document.createTreeWalker()方法创建了一个可用于导航文档树的 TreeWalker 对象实例。该方法的第一个参数是文档中要作为根节点的节点，第二个参数是一个整型常量，它指定某个内置筛选器，用于选择要包括在树中的节点。表 29-2 列出此参数可以接受的值：

表 29-2　可以接受的值

NodeFilter.SHOW_ALL	NodeFilter.SHOW_ATTRIBUTE
NodeFilter.SHOW_CDATA_SECTION	NodeFilter.SHOW_COMMENT
NodeFilter.SHOW_DOCUMENT	NodeFilter.SHOW_DOCUMENT_FRAGMENT
NodeFilter.SHOW_DOCUMENT_TYPE	NodeFilter.SHOW_ELEMENT
NodeFilter.SHOW_ENTITY	NodeFilter.SHOW_ENTITY_REFERENCE
NodeFilter.SHOW_NOTATION	NodeFilter.SHOW_PROCESSING_INSTRUCTION
NodeFilter.SHOW_TEXT	

createNodeIterator()方法的第三个参数是对筛选函数的引用，与 whatToShow 参数相比，该函数可以更进一步地筛选节点。此函数只接受一个节点，并根据以下常量返回一个整数值：NodeFilter.FILTER_ACCEPT、NodeFilter.FILTER_REJECT、NodeFilter.FILTER_SKIP。

应编写一个函数，对每个节点进行测试，并返回一个指示值，以告诉节点迭代器，是否在树中包括该节点。该函数不遍历节点，TreeWalker 对象机制根据需要不断地调用函数，来确定要用作筛选器的特征是否存在。

该方法的最后一个参数是一个 Boolean 值，它决定实体引用节点的内容是否应看成层次节点，此参数主要用于 XML 文档。

相关主题：TreeWalker 对象。

elementFromPoint(*x*, *y*)

返回值：元素对象引用

兼容性：WinIE4+，MacIE4+，NN-，Moz+，Safari+，Opera+，Chrome+

最初，IE 专用的 elementFromPoint()方法返回一个元素对象的引用，这个对象所在位置的整数坐标是该方法的参数。坐标平面也就是文档的平面，其左上角是点(0,0)。在交互设计中需

更多资源请访问稀酷客(www.ckook.com)

要对定位对象或鼠标事件进行冲突检测时，这个坐标平面非常有帮助。

多个对象在同一个位置(例如，一个元素定位在另一个元素上)时，该方法返回 z 值最高的元素；当定位元素放在普通 body 层次的对象上时，该方法总是返回定位元素；如果多个重叠的定位元素具有相同的 z 值(或者默认为 none)，该方法就从坐标相同的元素中返回在源代码中排在最后的元素。

示例

程序清单 29-13 中的文档包含许多不同类型的元素，每个元素都指定了 ID 特性。document 对象的 onmouseover 事件处理程序调用一个函数，来确定事件触发时鼠标指针在哪个元素上。注意事件坐标是 event.clientX 和 event.clientY，它们使用与页面相同的坐标平面作为其参照点。在每个元素上滚动鼠标时，其 ID 就显示在页面上。某些元素(如 br 和 tr)在文档中不占空间，因此它们的 ID 不会显示出来。在典型的浏览器屏幕上，把定位元素放在某个段落元素上，就可以看到 elementFromPoint()方法如何处理重叠的元素。如果滚动页面，事件的坐标和页面元素就会保持同步。

程序清单 29-13　在鼠标移过元素时跟踪鼠标

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>document.elementFromPoint() Method</title>
      <script type="text/javascript">
         function replaceHTML(elem, text)
         {
            while(elem.firstChild)
            {
               elem.removeChild(elem.firstChild);
            }
            elem.appendChild(document.createTextNode(text));
            }

         function showElemUnderneath(evt)
         {
            var elem
            elem = document.elementFromPoint(evt.clientX, evt.clientY);
            replaceHTML(document.getElementById("mySpan"), elem.id);
         }
      </script>
   </head>
   <body id="myBody" onmousemove="showElemUnderneath(event)">
      <h1 id="header">document.elementFromPoint() Method</h1>
      <hr id="myHR" />
      <p id="instructions">Roll the mouse around the page. The coordinates
         of the mouse pointer are currently atop an element
         whose ID is: "<span id='mySpan' style='font-weight:bold; '></span>".</p>
      <br id="myBR" />
```


更多资源请访问稀酷客(www.ckook.com)

```
        <form id="myForm">
           <input id="myButton" type="button" value="Sample Button" /> 
        </form>
        <table border="1" id="myTable">
           <tr id="tr1">
              <td id="td_A1">Cell A1</td>
              <td id="td_B1">Cell B1</td>
           </tr>
           <tr id="tr2">
              <td id="td_A2">Cell A2</td>
              <td id="td_B2">Cell B2</td>
           </tr>
        </table>
        <h2 id="sec1">Section 1</h2>
        <p id="p1">Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed
           do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
           adminim veniam, quis nostrud exercitation ullamco laboris nisi ut
           aliquip ex ea commodo consequat.</p>
        <h2 id="sec2">Section 2</h2>
        <p id="p2">Duis aute irure dolor in reprehenderit involuptate velit esse
           cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
           cupidatat non proident, sunt in culpa qui officia deseruntmollit anim
           id est laborum.</p>
        <div id="myDIV"
             style="position:absolute; top:340px; left:300px; background-color:yellow;">
           Here is a positioned element.
        </div>
     </body>
</html>
```

相关主题：event.clientx、event.clientY 属性；定位对象(配书光盘中的第 43 章)。

Evaluate ("expression", contextNode, resolver, type, result)

返回值：XPath 结果对象引用

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

document.evaluate()方法对 XPath 表达式求值，并返回一个 XPath 结果对象(XPathResult)。此方法的第一个参数最重要，它包含字符串形式的实际 XPath 表达式；第二个参数是表达式要应用到的上下文节点；而第三个参数是命名空间解析器(参见 createNSResolver()方法)。只要在表达式中没有使用命名空间前缀，resolver 参数就可以指定为 null。

type 参数确定表达式结果的类型，指定了 XPath 结果的类型，如任意类型用 0 表示、数字用 1 表示、字符串用 2 表示等。最后，可以在最后一个参数中指定一个可重用的 result 对象，然后修改该对象，并把该对象返回为表达式的结果。

相关主题：createNSResolver()方法。

execCommand("*commandName*"[, *UIFlag*] [, *param*])

返回值：Boolean

更多资源请访问稀酷客(www.ckook.com)

兼容性：WinIE4+，MacIE-，NN7.1+，Moz1.3+，Safari1.3+，Opera+，Chrome+

execCommand()方法是到一组命令的 JavaScript 通路，这组命令不在对象模型为对象定义的方法中。一系列相关方法(queryCommandEnable()和其他方法)也用于管理这些命令。

execCommand()方法至少要有一个参数，即命令名的字符串版本。命令名不区分大小写。可选的第二个参数是一个 Boolean 标志，指示命令显示相关的用户界面元素，默认为 false；有些命令还要求用该方法的第三个参数传递特性值。例如，要设置文本范围的字号，语法为：

```
myRange.execCommand("FontSize", true, 5);
```

如果命令成功执行，execCommand 方法就返回 Boolean 值 true；反之返回 false。一些命令可以返回值(例如，确定某选项的字体)，可以通过 queryCommandValue ()方法访问这些返回值。

在 Internet Explorer 中，大多数命令都在 TextRange 对象的文本选择区上操作。如配书光盘中的第 33 章所述，TextRange 对象必须通过脚本来创建，但可以创建 TextRange 对象，以便响应用户在文档中选择的文本。因为 TextRange 对象独立于元素层次结构(实际上，TextRange 可以散布在多个节点中)，所以它不能响应样式表规范。因此，许多能操作 TextRange 对象的命令都用于格式化或者修改文本。只用于 TextRange 对象的命令列表，请参见配书光盘第 33 章中的 TextRange.execCommand()方法。

在 document 对象中调用时，许多用于 TextRange 的命令依然有效，但本节只讨论操作整个文档的命令。表 29-3 列出了一些处理文档的命令，也列出了只作用于文档中所选文本的命令，而不管这些文本是由用户手工选择的还是借助 TextRange 对象选择的(见配书光盘中的第 33 章)。

表 29-3　document.execCommand()命令

命　令	参　数	说　明
BackColor	Color String	用 font 元素封装当前所选内容，元素的 style 特性将 background-color 样式设置为参数值
CreateBookmark	Anchor String	用 anchor 元素封装当前选择的内容(或文本范围)，元素的 name 特性设置为参数值
CreateLink	URL String	用 a 元素封装当前选择的内容，元素的 href 特性设置为参数值
decreaseFontSize	无	用 small 元素封装当前选择的内容
Delete	无	从文档中删除当前选择的内容
FontName	Font Face(s)	用 font 元素封装当前选择的内容，元素的 face 特性设置为参数值
FontSize	Size String	用 font 元素封装当前选择的内容，元素的 size 特性设置为参数值
FontColor	ColorString	用 font 元素封装当前选择的内容，元素的 color 特性设置为参数值
FormatBlock	Block Element String	用指定的块元素封装当前选择的内容，IE 仅支持 h1 ~ h6、地址和每个块元素，其他浏览器支持所有块元素
increaseFontSize	无	用 big 元素封装当前选择的内容
Indent	无	缩进当前选择的内容
InsertHorizontalRule	Id String	用 hr 元素封装当前选择的内容，元素的 id 特性设置为参数值

更多资源请访问稀酷客(www.ckook.com)

(续表)

命　令	参　数	说　明
InsertImage	Id String	用 img 元素封装当前选择的内容，元素的 id 特性设置为参数值
InsertParagraph	Id String	用 p 元素封装当前选择的内容，元素的 id 特性设置为参数值
JustifyCenter	无	将当前选择的内容居中
JustifyFull	无	将当前选择的内容两端对齐
JustifyLeft	无	将当前选择的内容左对齐
JustifyRight	无	将当前选择的内容右对齐
Outdent	无	减少当前选择内容的缩进
Refresh	无	重新载入页面
RemoveFormat	无	删除当前选择内容的格式
SelectAll	无	选择文档的所有文本
UnBookmark	无	删除包围当前选择的锚标记
Unlink	无	删除包围当前选择的链接标记
Unselect	无	取消文档中任何位置的当前选择内容

Mozilla 1.4 和 Safari 1.3 增加了一项功能，允许脚本将 iframe 元素的文档对象变为可编辑的 HTML 文档。下面的示例说明了如何将选中的文本在 ID 为 msg 的 iframe 中居中显示：

```
document.getElementById("msg").contentDocument.execCommand("JustifyCenter");
```

注意，contentDocument 属性用于将 iframe 作为文档访问。document.execCommand()方法的更多细节和示例请访问 http://www.mozilla.org/editor。

示例

配书光盘的第 33 章列举了 TextRange 对象的 execCommand 方法的许多示例，也可以在 Internet Explorer 中用 The Evaluator(参见第 4 章)试验用于文档的命令。在顶部文本框中输入如下语句，并单击 Evaluate 按钮：

```
document.execCommand("Refresh")
document.execCommand("SelectAll")
document.execCommand("Unselect")
```

在 Results 框中，所有方法都返回 true。

由于在 The Evaluator 中执行语句会迫使 body 选中内容在执行语句前变为取消选择，因此不能用这种方式试验用于选中内容的命令。

相关主题： queryCommandEnabled()、queryCommandIndterm()、queryCommandState()、queryCommandSupported()、queryCommandText()、queryCommandValue()方法。

getElementById("*elementID*")

返回值： 元素对象引用

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.getElementById()方法是一个 W3C DOM 方法，它可以获取文档中 ID 特性具有唯一标识符的元素引用。如果文档包含多个相同的 ID 实例，该方法就返回源代码中拥有此 ID 的第一个元素的引用。对于要在脚本控制下修改的对象，这个方法是给对象写入引用的重要方式，所以为元素指定唯一 ID 非常重要。

对于脚本设计者来说，这个方法的名称相当麻烦，因为 IE4+使任何元素的引用都以对象的 ID 开始。但 getElementById()方法是 W3C DOM 兼容浏览器(包括 IE)获得元素引用的跨浏览器方式，输入时注意，方法名的最后一个字母是小写的 d。

其他面向元素的方法(例如 getElementsByTagName())能由文档中的任何元素调用，而 getElementById()方法只用于 document 对象。

示例

本书有许多使用此方法的示例，但在 The Evaluator(参见第 4 章)中进行实验，可以更仔细地观察其工作方式。The Evaluator 中的许多元素都指定了 ID，因此可以使用此方法来查看对象及其属性。在 The Evaluator 的顶部和底部文本框中输入如下语句，顶部文本框的结果是对象的引用，底部文本框的结果是该对象的属性列表。

```
document.getElementById("myP")
document.getElementById("myEM")
document.getElementById("myTitle")
document.getElementById("myScript")
```

相关主题： getElementsByTagName()方法(参见第 26 章)。

getElementsByName("*elementName*")

返回值： 数组

兼容性： WinIE5+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

document.getElementsByName()方法返回一个对象引用数组，对象的 name 特性是指定了元素的名称，而元素的名称传递为方法的特性。对于默认没有 name 特性的元素，其他浏览器仍能识别 name 特性，但 IE 不能。因此，为了最大化跨浏览器兼容性，此方法应只用于定位默认定义了 name 特性的元素，如表单控件元素。如果文档中不存在定义了 name 特性的元素，此方法就返回一个长度为 0 的数组。

在大多数情况下，最好使用元素的 ID 和 getElementById()方法来获得对单个对象的引用。但某些元素使用 name 特性将多个元素组合到一起，尤其是单选按钮类型的 input 元素。这种情况下，调用 getElementsByName()将返回一个包含所有同名元素的数组——便于 for 循环检查单选按钮组的 checked 属性。因此，提取数组时，不是使用包含表单对象的旧方法：

```
var buttonGroup = document.forms[0].radioGroupName;
```

而可以更直接地使用：

```
var buttonGroup = document.getElementsByName(radioGroupName);
```


更多资源请访问稀酷客(www.ckook.com)

在后一种情况下，操作时可以不使用包含表单对象的下标或名称。当然，这假设组名没有在页面的其他位置上使用，否则无疑会导致混乱。

示例

使用 The Evaluator(参见第 4 章)试验 getElementsByName()方法。页面上部的所有表单元素都有各自的名称。在顶部文本框中输入如下语句，并观察结果：

```
document.getElementsByName("output")
document.getElementsByName("speed").length
document.getElementsByName("speed")[0].value
```

在底部文本框中输入如下表达式，还可以查看文本域的所有属性：

```
document.getElementsByName("speed")[0]
```

相关主题： document.getElementsById()、getElementsByTagName()方法。

importNode(*node*, *deep*)

返回值： 节点对象引用

兼容性： WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

document.importNode()方法从另一个文档对象中把一个节点导入当前文档对象。导入节点时，会生成原始节点的一个副本，所以原始节点保持不变。该方法的第二个参数是一个 Boolean 值，该值决定是导入节点的整个子树(true)还是只导入节点本身(false)。

open(["mimeType"] [,"*replace*"])

返回值： 无

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

打开文档与打开窗口大不相同。打开窗口时，在屏幕和浏览器内存中都要创建一个新对象。而打开文档只是告诉浏览器，准备接收一些要在窗口中显示的数据，该窗口的名称要么已指定，要么隐含在 document.open()方法的引用中(例如，parent.frames[1].document.open()指向框架集中的另一个框架，而 document.open()指向当前窗口或者框架)。因此，该方法名可能会误导新手，因为 document.open()方法根本不从 Web 服务器或者硬盘中载入文档，而只是通过 document.write()或者 document.writeln()方法向窗口传输数据的前奏。在某种意义上，document.open()方法只不过打开了管道；而 document.write()或者 document.writeln()方法把数据像流一样送入管道，页面数据发送完后，document.close()就关闭该管道。

document.open()方法是可选的，因为 document.write()方法试图写入已关闭的文档时，会自动清除旧文档，打开新文档流。不管是否使用 document.open()方法，在所有写入操作执行完毕后，都必须使用 document.close()方法。

document.open()方法的可选参数指定了发送到窗口的数据类型，MIME 类型是在 Internet 上传输和描述多媒体数据的规范(最初用于邮件传输，现在适用于所有的 Internet 数据交换)。在浏览器参数设置的辅助应用程序列表中简述了 MIME，MIME 类型就是用斜杠分开的一对数据

更多资源请访问稀酷客(www.ckook.com)

类型名(如 text/html 和 image/gif)。MIME 类型指定为 document.open()方法的参数时，就告诉浏览器要接收的数据类型，因此浏览器知道如何显示数据。大多数浏览器接受的值如下：

- text/html
- text/plain
- image/gif
- image/jpeg
- image/xbm

如果忽略参数，JavaScript 就假定使用最常用的类型 text/html，它是在写入窗口之前，用脚本组合的常见数据类型。text/html 类型包括 HTML 引用的任何图像。指定图像类型意味着，可能拥有要在新文档中显示的图像的原始二进制表示。

另一种可能是将 write()方法的结果输出到插件程序中。对于 mimeType 参数，要指定插件程序的 MIME 类型(例如，用于 Shockwave 的 application/x-director)；同样，写入插件程序的数据必须是它可以处理的格式。这个机制也适用于直接将数据写入辅助应用程序。

注意：

IE 仅接受 text/html MIME 类型的参数。

现代浏览器支持这个方法的第二个可选参数 replace。document.open()方法的这个参数的作用类似于 location 对象的 replace()方法。document.open()方法的参数 replace 可以用新文档替换窗口或者框架中的旧文档，而且不记录被替换的窗口或者框架的历史信息。

示例

在本章稍后对 document.write()方法的讨论中，可以看到使用 document.open()方法为另一个框架动态创建内容的示例。

相关主题：document.close()、document.clear()、document.write()和 document.writeln()方法。

queryCommandEnabled("*commandName*")，queryCommandIndterm("*commandName*")，queryCommandState("*commandName*")，queryCommandSupported("*commandName*")，queryCommandText("*commandName*")，queryCommandValue("*commandName*")

返回值：多种类型的值

兼容性：WinIE4+，MacIE-，NN7.1，Moz1.3+，Safari+，Opera+，Chrome+

这 6 个方法加大了对 document 和 TextRange 对象中 execCommand 方法的支持。如果选用 execCommand()方法修改所选文本的样式，就可以用这些查询方法确保浏览器支持需要的命令，并得到返回值。表 29-4 概述了每个查询方法的作用和返回值。注意浏览器对这 6 个方法的支持是不同的。

更多资源请访问稀酷客(www.ckook.com)

表 29-4　查 询 命 令

查询命令	返回值	说　明
Enabled	Boolean	显示 document 或 TextRange 对象是否能调用
Indterm	Boolean	显示命令是否处于未定状态
CommandState	Boolean | null	显示命令是已完成(true)、仍在执行(false)还是处于未定状态(null)
Supported	Boolean	显示当前浏览器是否支持命令
Text	String	返回命令所返回的文本
Value	Varies	返回命令返回的值(如果有的话)

因为正载入的页面不能调用 execCommand()方法，所以，在调用命令之前，必须用 queryCommandEnabled()检测任何可能与页面载入冲突的调用。也可以验证运行脚本的浏览器版本是否支持命令。因此，可用下面的条件结构封装命令的调用：

```
if (document.queryCommandEnabled(commandName) &&
    document.queryCommandSupported(commandName)) {
    // ...
}
```

用命令读取所选文本的信息时，可以用 queryCommandText()或者 queryCommandValue()方法捕获该信息(前面介绍过，execCommand()方法在调用任何命令时都返回一个 Boolean 值)。

示例

TextRange 对象的这些方法的示例请参见配书光盘中的第 33 章。

相关主题：TextRange 对象(配书光盘中的第 33 章)；execCommand()方法。

recalc([*allFlag*])

返回值：无

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE5 引入了动态属性的概念。利用所有元素的 setExpression()方法和 expression()样式表值，可以建立起对象属性和潜在动态属性之间的依赖关系，例如窗口的尺寸或者可拖动元素的位置。建立了这些依赖关系后，document.recalc()方法会重新计算它们，以响应用户的动作，例如改变窗口的大小或者移动元素。

此方法的可选参数是 Boolean 值，默认为 false，表示浏览器仅重新计算自从上次重新计算以来出现了变化的表达式。然而，如果该参数指定为 true，就重新计算所有的表达式，不管它们是否已经改变。

Mozilla 1.4 允许脚本将 iframe 元素的文档对象转为可编辑的 HTML 文档。部分脚本代码使用了 document.execCommand()和相关方法，其当前信息和示例请访问 http://www.mozilla.org/editor。

示例

在演示 setExpression()方法的程序清单 26-32 中包含 recalc()的应用例子。在该示例中，当前时间与标准元素对象的属性之间存在依赖关系。

更多资源请访问稀酷客(www.ckook.com)

相关主题：getExpression()、removeExpression()、setExpression()方法(参见第 26 章)。

releaseEvents(*eventTypeList*)

返回值：无

兼容性：WinIE-，MacIE-，NN4+，Moz-，Safari-，Opera-，Chrome-

在 NN4+中，只要允许捕获文档中特定类型的事件，该捕获功能就一直有效，直到页面卸载或者明确禁用该功能为止。可以通过 releaseEvents()方法关闭每个事件的事件捕获功能。事件捕获和释放的内容参见第 32 章。

相关主题：captureEvents()，routeEvent()方法

routeEvent([*eventObject*])

返回值：无

兼容性：WinIE-，MacIE-，NN4+，Moz-，Safari-，Opera-，Chrome-

如果在 NN4 中捕获了某种类型的事件，脚本就可能需要在该事件上执行某些有效的处理操作，之后让事件到达其目的地。为了让事件通过对象层次结构到达其目的地，可使用 routeEvent()方法，并把当前函数处理的事件对象传送为参数。事件处理的的内容参见第 32 章。

相关主题：captureEvents()、releaseEvents()方法。

write("*string*1" [,"*string*2" . . . [,"*stringn*"]])，writeln("*string*1" [,"*string*2" . . . [,"*stringn*"]])

返回值：Boolean，如果成功返回 true

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

这两个方法将文本发送到文档中，并在文档窗口显示出来。它们唯一的区别在于，document.writeln()在发送给文档的字符串末尾添加了一个回车符，在用户通过浏览器的源视图窗口查看源代码时，这个回车符有助于格式化源代码。如果要用这些方法创建新行，并显示在 HTML 上，仍然需要写入一个
，来插入换行符。

许多 JavaScript 新手通常错误地认为：这些方法允许脚本修改现有文档的内容，其实这是不对的。一旦文档载入窗口(或者框架)，只有 text 或者 textarea 对象的内容是完全向后兼容的文本，修改它们不需要重载或者重写整个页面。在 IE4+中，使用任何元素的 innerHTML、innerText、outerHTML 和 outerText 属性都可以修改 HTML 和文本。在 W3C DOM 兼容的浏览器上，设置元素的 nodeValue 或者 innerHTML 属性，就可以修改元素的文本。修改节点的内容时，首选的做法是创建、插入或者替换新元素，此时应严格遵循 W3C DOM 的要求，如第 26 章、本章和本书其他章节的例子所示。

使用 document.write()和 document.writeln()方法最安全的两种方式是：

- 用文档中内嵌的脚本编写页面的部分或者全部内容。
- 将 HTML 代码发送到新窗口，或者发送到多框架窗口中的独立框架。

对于第一种情况，实际上是使脚本段与 HTML 交错在一起。脚本在文档载入时运行，并写入需要的 HTML 内容。第 3 章的 scriptl.html 就完成了这个任务。当特定类型的浏览器需要特殊的语法时，可以用一个页面产生浏览器特定的 HTML。

在后一种情况下，脚本可在一个框架中接收用户输入，然后通过算法确定另一个框架的布

更多资源请访问稀酷客(www.ckook.com)

局和内容。脚本将另一个框架的 HTML 代码组合成一个字符串变量(包含所有必要的 HTML 标记)。脚本在框架中写入内容之前，可以用 parent.frameName.document.open ()方法打开布局流(关闭框架中的当前文档)；接着，parent.frameName.document.write()方法将整个字符串写入另一个框架；最后，parent.frameName.document.close()方法确保所有数据流写入窗口。这个框架看似是由服务器上的源文档创建的，而不是在内存中即时创建的。这个窗口或者框架的 document 对象是一个完全标准的 document 对象。因此，甚至可将这些临时 HTML 页面的脚本作为 HTML 规范的一部分。

HTML 文档(包含要通过 write()或 writeln()方法进行写操作的脚本)完全载入后，页面的引入流就会自动关闭。如果使用一系列 document.write()语句，则第一个 document.write()方法会完全删除原文档，包括原文档的所有对象和脚本变量值。因此，如果用一系列 document.write()语句生成新页面，则在第二个 document.write ()语句执行之前，原页面中的脚本和变量就会消失。为消除这个潜在的问题，可将新内容组合成一个字符串变量，然后将该变量作为参数传递给一个 document.write ()语句，并确保脚本的下一行包含 document.close()语句。

在脚本中，通过 document.write()方法组合 HTML 常常需要连接字符串值和嵌套字符串，JavaScript 的许多 String 对象快捷方式有利于格式化带 HTML 标记的文本(详见第 15 章)。

如果写入其他框架或者窗口，则可以随意使用多个 document.write()语句。脚本是通过多个 document.write()方法传递许多小字符串，还是通过一个 document.write()方法发送一个大字符串，部分取决于环境，部分取决于脚本编写样式。就性能而言，完全标准的程序应在内存中做更多的准备工作，且尽量减少 I/O(输入输出)调用。另一方面，组合长字符串时，很容易犯一些难于察觉的错误。应使用最适合自己的方式。

给这些方法传递参数还有一种鲜为人知的方法。将字符串值组合在一起时，不是用加号(+)操作符连接字符串值，而是用逗号将字符串分开。这种情况下，字符串是 document.write()方法的参数。例如，下面两个语句返回相同的结果：

```
document.write("Today is " + new Date());
document.write("Today is ",new Date());
```

两种方式各有千秋。应使用适合自己编程风格的方法。

注意：

动态生成脚本需要另一个技巧，在 NN 中尤其如此。问题在于，如果使用诸如 document.write 的代码("<script></script>",)，浏览器会将结束的 script 标记解释为执行写入操作的脚本结尾，所以必须将结束标记分为多个组件，也可以使用转义斜杠符号(/)。例如，如果要为每类浏览器载入不同的.js 文件，代码如下：

```
// variable 'browserVer' is a browser-specific string
// and 'page' is the HTML your script is accumulating
// for document.write()
page += "<script type='text/javascript' src='" +
        browseVer + ".js'><" + "\/script>";■
```

使用 document.open()、document.write()和 document.close()在文档中显示图像时，需要一些额外的步骤。首先，通过 document.write()写入的 URL 值必须是完整(不是相对)的 URL 引用；

更多资源请访问稀酷客(www.ckook.com)

另外，还可以为动态创建的页面写入<base>标记，使其 href 特性值匹配写入页面的文件的 href 特性值。

显示图像的另一个窍门是通过脚本或者其他方法为每个图像指定 height 和 width 特性。文档的显示效果在所有平台上都有所提高，因为浏览器在载入元素的详细资料前，这些值能帮助设计元素的布局。

除下面 document.write()方法的例子(见程序清单 29-14～程序清单 29-16)外，还可以在许多应用程序中找到更完整的例子(参见配书光盘中的第 52~61 章)，它们用这个方法组合图像和图表。因为可将有效的 HTML 组合成字符串，并写入窗口或者框架，所以可以定制出非常复杂的 HTML 文档。

示例

程序清单 29-14～29-16 中的示例演示了使用 document.write()或 document.writeln()方法写入另一个框架的几个要点。首先，可在框架中写入任何 HTML 代码，浏览器接受该代码，就像这些源代码来自其他位置的 HTML 文件。该示例组装了一个完整的 HTML 文档，包括基本的 HTML 标记。

程序清单 29-14 文档写入示例的框架集

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Writin' to the doc</title>
   </head>
   <frameset rows="50%,50%">
      <frame name="Frame1" src="jsb29-15.html" />
      <frame name="Frame2" src="jsb29-16.html" />
   </frameset>
</html>
```

程序清单 29-15 根据用户输入在文档中写入内容

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Document Write Controller</title>
      <script type="text/javascript">
         function takePulse(form)
         {
            var msg = "<html><head><title>On The Fly with " +
                      form.yourName.value + "<\/title><\/head>";
            msg += "<body bgcolor='salmon'><h1>Good Day " +
                    form.yourName.value +
                    "!<\/h1><hr />";
            for (var i = 0; i < form.how.length; i++)
            {
```


更多资源请访问稀酷客(www.ckook.com)

```
                if (form.how[i].checked)
                {
                    msg += form.how[i].value;
                    break;
                }
            }
            msg += "<br />Make it a great day!<\/body><\/html>";
            parent.Frame2.document.write(msg);
            parent.Frame2.document.close();
        }
        function getTitle()
        {
            alert("Lower frame document.title is now:" + parent.Frame2.document.title);
        }

        // bind the event handlers
        function addEvent(elem, evtType, func)
        {
            if (elem.addEventListener)
            {
                elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
                elem.attachEvent("on" + evtType, func);
            }
            else
            {
                elem["on" + evtType] = func;
            }
        }
        addEvent(window, "load", function()
        {
            addEvent(document.getElementById("enter"), "click",
                     function(evt)
                     {takePulse(document.getElementById("enter").form)
                     });
            addEvent(document.getElementById("peek"), "click", getTitle);
        });
    </script>
</head>
<body>
    Fill in a name, and select how that person feels today. Then click "Write
    To Below" to see the results in the bottom frame.
    <form>
        Enter your first name:
        <input type="text" name="yourName" value="Dave" />
        <p>How are you today?
            <input type="radio" name="how"
                value="I hope that feeling continues forever."
```


更多资源请访问稀酷客(www.ckook.com)

```
                        checked="checked" />
                Swell
                <input type="radio" name="how"
                        value="You may be on your way to feeling Swell" />
                Pretty Good
                <input type="radio" name="how"
                        value="Things can only get better from here." />So-So
            </p>
            <p><input type="button" id="enter" name="enter" value="Write To Below" /></p>
            <hr />
            <input type="button" id="peek" name="peek" value="Check Lower Frame Title" />
        </form>
    </body>
</html>
```

程序清单 29-16　文档写入示例的占位页面

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="content-type" content="text/html;charset=utf-8">
        <title>Placeholder</title>
    </head>
    <body>
    </body>
</html>
```

注意，此示例根据用户输入来定制文档的内容，这种定制让用户使用网页的体验更具交互性，且没有使用任何服务器端程序。

第二个要点是：使用 document.write()方法在单个框架中创建的文档是真正的 document 对象。在此示例中，如果在顶部框架中更改 name 域的输入后，重新绘制底部的框架，则写入文档的<title>标记也会改变。如果在更新底部框架后，单击下面的按钮，document.title 属性就会改变，以反映在显示框架页面的过程中写入浏览器的<title>标记。可以人为动态地创建真正意义上的 JavaScript document 对象，这是 JavaScript 无服务器 CGI 脚本编程(为了将信息传递给用户)的最重要功能之一。只要有足够的想象力，这里有大量的功能可供利用。

注意，可以修改程序清单 29-15，将结果写入包含域和按钮的文档所在的框架。代码如下：

```
parent.frames[1].document.open();
parent.frames[1].document.write(msg);
parent.frames[1].document.close();
```

而不必指定底部的框架：

```
document.open();
document.write(msg);
document.close();
```

此代码用结果替换表单文档，且不需要先建立任何框架。由于代码将新文档的所有内容都组合到一个变量值中，因此该数据在调用 document.write()方法后继续存在。

更多资源请访问稀酷客(www.ckook.com)

框架集文档(参见程序清单 29-14)载入了一个空白文档(参见程序清单 29-16)，来创建空白框架。最好使用一个替代方法：让框架集文档用自己创建的空白文档来填充框架，此技术详见第 27 章的 27.2.10 节。

相关主题：document.open()、document.close()、document.clear()方法。

29.1.5　事件处理程序

onselectionchange

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

onselectionchange 事件可以由许多用户动作触发，但这些动作都发生在受 WinIE5.5+编辑模式影响的元素上。

相关主题：oncontrolselect 事件处理程序。

onstop

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在 WinIE5+中，用户单击浏览器的 Stop 按钮时，会触发 onstop 事件。使用这个事件处理程序可以停止执行页面上可能失控的脚本，因为 Stop 按钮在页面载入后并不能控制脚本。如果在运行过程中有一个死循环，可以临时使用这个事件处理程序来停止脚本的调试。

示例

程序清单 29-17 的简单脚本是一个有意构造的无限循环。万一用户在除 IE5+外的浏览器中载入此页面，可单击 Halt Counter 按钮停止循环。Halt Counter 按钮和 onstop 事件处理程序调用同一个函数。

程序清单 29-17　使用 onstop 事件处理程序停止脚本

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>onStop Event Handler</title>
      <script type="text/javascript">
         var counter = 0;
         var timerID;
         function startCounter()
         {
            document.forms[0].display.value = ++counter;
            //clearTimeout(timerID)
            timerID = setTimeout("startCounter()", 10);
         }
         function haltCounter()
         {
            clearTimeout(timerID);
            counter = 0;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
            // bind the event handlers
            function addEvent(elem, evtType, func)
            {
               if (elem.addEventListener)
               {
                  elem.addEventListener(evtType, func, false);
               }
               else if (elem.attachEvent)
               {
                  elem.attachEvent("on" + evtType, func);
               }
               else
               {
                  elem["on" + evtType] = func;
               }
            }
            addEvent(window, "load", function()
            {
               addEvent(document, "stop", haltCounter);
               addEvent(document.getElementById("start"), "click", startCounter);
               addEvent(document.getElementById("halt"), "click", haltCounter);
            });
         </script>
      </head>
      <body>
         <h1>onStop Event Handler</h1>
         <hr />
         <p>Click the browser's Stop button (in IE) to stop the script counter.</p>
         <form>
            <p><input type="text" name="display" /></p>
            <input type="button" id="start" value="Start Counter" />
            <input type="button" id="halt" value="Halt Counter" />
         </form>
      </body>
   </html>
```

相关主题： Repeat 循环(第 21 章)。

29.2 body 元素对象

兼容性： WinIE4+、MacIE4+、 NN6+、 Moz+、Safari+、 Opera+、Chrome+

关于 HTML 元素属性、方法和事件处理程序的详细内容，请参见第 26 章。

属　　性	方　　法	事件处理程序
alink	createControlRange()	onafterprint
background	createTextRange()	onbeforeprint

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　　性	方　　法	事件处理程序
bgColor	doScroll()	onscroll
bgProperties		
bottomMargin		
leftMargin		
link		
noWrap		
rightMargin		
scroll		
scrollLeft		
scrollTop		
text		
topMargin		
vLink		

29.2.1　语法

访问 body 元素对象的属性或方法：

```
[window.] document.body.property | method([parameters])
```

29.2.2　关于 body 对象

在表示 HTML 元素对象的对象模型中，body 元素对象是页面内容的主要容器，body 包含要显示的所有 HTML。在节点层次结构中，这个特殊位置给 body 对象提供了一些特别的功能，在 IE 对象模型中尤其如此。

为了说明其特殊关系，IE 和 W3C 对象模型都给 body 元素提供了同样的快捷引用：document.body。body 是最重要的 HTML 元素对象(第 26 章中属性、方法和事件处理程序的长列表可以证明这一点)，也可以使用其他语法来得到它。

几个熟悉的body元素特性控制着整个主体内容的外观，例如链接颜色(有三种状态)和背景(颜色或者图像)。但 IE、NN/Mozilla 和 W3C 对 body 元素在脚本文档中的作用有许多不同的解释。NN/Mozilla 认为，许多属性和方法都属于窗口(例如，窗口内的滚动等)，而 IE 认为它们属于 body 元素对象。因此，NN/Mozilla 滚动的是窗口(及其包含的内容)，而 IE 滚动的是 body(在其所在的窗口中)。而且因为 body 元素填满了浏览器窗口或者框架的整个可见区域，所以这个可见矩形的大小在 IE 中由 body 的 scrollHeight 和 scrollWidth 属性确定，而 NN4+/Moz 提供了 window.innerHeight 和 window.innerWidth 属性。用脚本编写整个窗口或者文档的外观时，这个区别十分重要，因为根据目标浏览器的不同，必须使用 window 或者 body 元素对象的属性和方法。

更多资源请访问稀酷客(www.ckook.com)

注意：

在页面载入过程中引用 document.body 对象时要十分小心，在页面载入完毕前，该对象并不存在。如果想通过脚本设置一些初始属性，应使它们响应<body>标记的 onload 事件处理程序。如果试图在 head 元素的直接脚本中设置 body 元素对象的属性，浏览器可能会显示“找不到对象”的错误消息。

29.2.3 属性

alink，bgColor，link，text，vLink

值：三个十六进制值或颜色名称字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

aLink、link 和 vLink 属性是 document 对象中 alinkColor、linkColor 和 vlinkColor 属性的新版本。bgColor 与旧的 document.bgColor 属性相同，而 text 属性是 document.fgColor 属性的新版本。这些属性是 body 元素中 HTML 特性的脚本版本——新特性名与 HTML 特性相关，而不是与旧属性名相关。

CSS 已经基本上废弃了这些属性。它们仍能工作，但 Web 开发人员逐渐把样式表作为在网页中更改颜色的首选方式。链接颜色通过样式表中的伪类选择器(body 元素的 style 特性)设置时，必须用 body 对象的 style 特性来访问。

background

值：URL 字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

background 属性能设置或获得 body 元素的背景图像(如果存在)的 URL。若设置了特性或属性，body 元素的背景图像就会覆盖背景色。要删除文档背景的图像，可将 document.body.background 属性设置为空字符串。

与用于访问页面颜色的属性一样，在现代网页中，应通过样式表(而不是使用 body.background 属性)来设置背景图像。在这种情况下，背景应通过 body 对象的 style 属性以编程方式访问。

bgColor

详见 aLink。

bgProperties

值：字符串常量， 读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 bgProperties 属性是调整背景的另一种方法，调整背景是指用户滚动文档时，背景图像是固定还是随文档一起滚动。这个属性的初始设置通过 CSS 特性 background- attachment 来完成，并通过 body 元素的 style.backgroundAttachment 属性在脚本的控制下修改。

无论采用什么方式引用这个属性，其唯一可用的值都是字符串常量 scroll(默认)或者 fixed。

更多资源请访问稀酷客(www.ckook.com)

示例

以下两条语句都更改了 IE4+中背景图像的默认滚动方式：

```
document.body.bgProperties = "fixed";
```

或

```
document.body.style.backgroundAttachment = "fixed";
```

使用样式表版本的另一个优点在于，它也可用于 NN6+/Moz 和其他 W3C 浏览器。

相关主题： body.background 属性。

bottomMargin，leftMargin，rightMargin，topMargin

值： 整型， 读/写

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

这 4 个 IE 专用的 margin 属性可为 body 元素设置 4 个样式表页边距属性(body.style. marginBottom 等)。样式表页边距属性表示元素的内容和其外层容器间的边距。对于 body 元素，这个容器是不可见的文档容器。

在 4 个属性中，如果内容不能填满窗口或者框架的垂直区域，则只有代表底部边界的属性可能出问题，因为其值不会自动增加，来容纳额外的空白区域。

示例

以下两条语句都更改了 IE4+中的默认左边距。使用样式表版本的另一个优点在于，它也可用于 NN6+/Moz 和其他 W3C 浏览器。

```
document.body.leftMargin=30;
```

或

```
document.body.style.marginLeft = 30;
```

相关主题： style 对象。

leftMargin

详见 bottomMargin。

link

详见 aLink。

noWrap

值： Boolean， 读/写

兼容性： WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

noWrap 属性可以修改 body 元素的行为，这些行为通常用 HTML 特性 nowrap 来设置，因为该属性名与其表示的含义相反，所以控制它的 Boolean 值常常令人难以理解。

更多资源请访问稀酷客(www.ckook.com)

noWrap 为默认值 false 时，body 元素的默认行为是文本在窗口或者框架的宽度内自动换行。将 noWrap 值设置为 true，文本行会一直向右延伸，超出窗口或框架的右边框，直到遇到 HTML 中的换行符(或者段落结尾)为止。如果文本行超出了窗口的右边框，窗口(或框架)会显示水平滚动条(当然，如果框架设置为不滚动，就不显示滚动条)。

通常，用户不喜欢在任何方向上进行不必要的滚动，所以除非确有必要使所有文本放在一行上，否则最好使用默认值。

示例

要更改默认的文本换行行为，语句为：

```
document.body.noWrap = true;
```

相关主题：无。

rightMargin

详见 bottomMargin。

scroll

值：字符串常量，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 专用的 scroll 属性可以通过脚本访问 body 元素的 scroll 特性。默认情况下，IE 的 body 元素在内容区的高度不足时显示垂直滚动条；仅当内容比窗口或者框架更宽时，才显示水平滚动条。要隐藏水平和垂直滚动条，可以将 scroll 特性设置为 no，或者通过脚本改变特性值。这个属性的值是字符串常量 yes 和 no。

除了 frame 属性和 NN4+/Moz 的签名脚本之外，其他浏览器都只能在脚本的控制下关闭滚动条。但可以(通过 window.open()方法)创建一个新窗口，然后隐藏其滚动条。

示例

要更改默认的滚动条外观，语句为：

```
document.body.scroll = "no";
```

相关主题：window.scrollbars 属性；window.open()方法。

scrollLeft，scrollTop

值：整型，　　读/写

兼容性：WinIE4+，MacIE4+，NN7+，Moz+，Safari+，Opera+，Chrome+

body 对象的 scrollLeft 和 scrollTop 属性与通用 HTML 元素对象的一样，但它们在确定定位元素的位置时仍具有重要的作用(详见配书光盘中的第 43 章)。因为鼠标事件和元素定位属性常常与浏览器窗口的可见内容区域相关，所以在指定绝对位置时，必须考虑 document.body 对象的滚动值。这两个属性的值是以像素为单位的整数。

更多资源请访问稀酷客(www.ckook.com)

示例

程序清单 29-18 是一个不寻常的示例，它创建了一个框架集，并为其中的两个框架创建内容，这些都在同一个 HTML 文档中完成。在框架集的左侧框架中，有两个域显示了右侧框架的 xOffset 和 yOffset 属性的像素值，右侧框架是一个宽度固定(800 像素)的 30 行表格。鼠标单击事件在文档级上捕获(参见第 32 章)，这样，单击表格或单元格边框，或在表格之外单击，就可以触发右侧框架中的 showOffsets()函数。该函数是一个简单脚本，它在左侧框架的相应域中显示页面偏移值。

程序清单 29-18　决定滚动值

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Master of all Windows</title>
      <script type="text/javascript">
         function leftFrame()
         {
             var output = "<html><body><h3>Body Scroll Values<\/h3><hr />\n";
             output += "<form>body.scrollLeft:<input type='text' ";
             output += "name='xOffset' size=4 /><br />\n";
             output += "body.scrollTop:<input type='text' ";
             output += "name='yOffset' size=4 /><br />\n";
             output += "<\/form><\/body><\/html>";
             return output;
         }
         function rightFrame()
         {
            var output = "<html><head><script type='text/javascript'>\n";
            output += "function showOffsets() {\n";
            output += "parent.readout.document.forms[0].xOffset.value ";
            output += "= document.body.scrollLeft\n";
            output += "parent.readout.document.forms[0].yOffset.value ";
            output += "= document.body.scrollTop\n}\n";
            output += "document.onclick = showOffsets\n";
            output += "<\/script><\/head><body><h3>Content Page<\/h3>\n";
            output += "Scroll this frame and click on a table border to view ";
            output += "page offset values.<br /><hr />\n";
            output += "<table border=5 width=800>";
            var oneRow = "<td>Cell 1<\/td><td>Cell 2<\/td><td>Cell ";
            oneRow += "3<\/td><td>Cell 4<\/td><td>Cell 5<\/td>";
            for (var i = 1; i <= 30; i++)
            {
               output += "<tr><td><b>Row " + i + "<\/b><\/td>" + oneRow + "<\/tr>";
            }
            output += "<\/table><\/body><\/html>";
            return output;
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
      </script>
   </head>
   <frameset cols="30%,70%">
      <frame name="readout" src="javascript:parent.leftFrame()" />
      <frame name="display" src="javascript:parent.rightFrame()" />
   </frameset>
</html>
```

相关主题：window.pageXOffset、window.pageYOffset 属性。

text

详见 aLink。

topMargin

详见 bottomMargin。

vLink

详见 aLink。

29.2.4　方法

createControlRange()

返回值：数组

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

此方法在 WinIE5+浏览器中创建一个控件范围。控件范围用于基于控件的选择，而基于文本的选择是通过文本范围实现的。该方法仅适用于处于编辑模式下的文档。在通常的文档查看模式中，createControlRange()方法返回一个空数组。

createTextRange()

返回值：对象

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在 IE4+中，body 元素对象常常用于创建 TextRange 对象，当要操作的文本是文档 body 文本的一部分时尤其如此。createTextRange()方法返回的初始 TextRange 对象包含了 body 元素的 HTML 和 body 文本，对返回对象的进一步操作需要设置文本范围的开始点和结束点。更多内容请参见配书光盘中的第 33 章对 TextRange 对象的讨论。

示例

createTextRange()方法示例请参见程序清单 30-10。

相关主题：TextRange 对象(配书光盘中的第 33 章)。

更多资源请访问稀酷客(www.ckook.com)

doScroll (["*scrollAction*"])

返回值：无

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

在包含当前文档的窗口或者框架中，可以用 doScroll()方法模拟用户在滚动条上的动作。如果想创建自己的滚动条，来代替标准的系统滚动条，就可以使用这个方法。然而，即使将 Display 控制面板设置为自动滚动，滚动也只是瞬时的，不是连续的。该方法的参数是表 29-5 中的字符串常量值。实际上，更长的滚动动作名会更真实地模拟滚动条元素的实际单击动作，而简短的滚动动作名可能只滚动一点点。

表 29-5　document. body.doScroll()的参数

长参数	短参数	模拟的滚动动作
scrollbarDown	down	单击下箭头
scrollbarHThumb	n/a	单击水平滚动条上的滚动块(没有滚动动作)
scrollbarLeft	left	单击左箭头
scrollbarPageDown	pageDown	单击向下翻页区域或按下 PgDn 键(默认)
scrollbarPageLeft	pageLeft	单击向左翻页区域
scrollbarPageRight	pageRight	单击向右翻页区域
scrollbarPageUp	pageUp	单击向上翻页区域或按下 PageUp 键
scrollbarRight	right	单击右箭头
scrollbarUp	up	单击上箭头
scrollbarVThumb	n/a	单击垂直滚动条上的滚动块(没有滚动动作)

doScroll()方法不能(通过设置 body 元素的 scrollTop 和 scrollLeft 属性)将文档滚动到特定的像素位置，它完全依赖于主体内容和窗口或者框架尺寸之间的空间关系。另外，doScroll()方法会触发 body 对象元素的 onScroll 事件处理程序。

注意通过脚本修改 body 内容，可以改变这些空间关系。IE 在内容改变后，会慢慢更新其所有的内部尺寸。如果在这个布局修改后调用 doScroll()方法，滚动操作可能不会像预期的那样执行，但可以使用一个常见技巧：用 setTimeout()延迟一点时间，再调用 doScroll()方法。

示例

使用 The Evaluator(参见第 4 章)在 IE5+中试验 doScroll()方法。调整浏览器窗口的大小，至少显示出垂直滚动条(意味着它有滚动区域)。在顶部文本框中输入如下语句，按几次 Enter 键，以便模拟单击 PageDown 键的操作。

```
document .body .doScroll()
```

回到页面顶部，执行同样的操作，以模拟单击滚动条下箭头的动作：

```
document .body .doScroll("down")
```

也可以试验向上滚动。在顶部文本框中输入所需的语句，并将文本光标放在域中，手动滚

更多资源请访问稀酷客(www.ckook.com)

动到页面底部，然后按 Enter 键，来激活命令。

相关主题：body.scroll、body.scrollTop、body.scrollLeft 属性；window.scroll()、window.scrollBy()、window.scrollTo()方法。

29.2.5 事件处理程序

onafterprint，onbeforeprint，onscroll

详见第 27 章中 window 对象的这些事件处理程序。

29.3 TreeWalker 对象

属　性	方　法	事件处理程序
currentNode	firstChild()	
expandEntityReference	lastChild()	
filter	nextNode()	
root	nextSibling()	
whatToShow	parentNode()	
	previousNode()	
	previousSibling()	

29.3.1 语法

创建 TreeWalker 对象：

```
var treewalk = document.createTreeWalker(document, whatToShow,
   filterFunction, entityRefExpansion);
```

访问 TreeWalker 对象的属性和方法：

```
TreeWalker.property | method([parameters])
```

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

29.3.2 关于 TreeWalker 对象

TreeWalker 对象用作一列节点的容器，这些节点符合 document.createTreeWalker()方法定义的标准，该方法用于创建 TreeWalker 对象。TreeWalker 对象包含的节点列表与引用它们的文档具有相同的层次结构。TreeWalker 对象允许根据其内在的树型结构，浏览此节点列表。

在某种程度上，可将 TreeWalker 对象看作遍历器对象，因为其主要作用是提供一种方式，来遍历列表中的节点。不过在这里，列表是层次树，而不是线性列表。TreeWalker 对象在节点列表中保存了一个总是指向当前节点的指针。只要使用 TreeWalker 对象在列表中导航，该导航就总是相对于指针。例如，调用 previousNode()或 nextNode()方法来引用前一个或后一个节点时，

更多资源请访问稀酷客(www.ckook.com)

该引用取决于节点指针在树中的当前位置。

可以使用 document.createTreeWalker()方法为文档创建 TreeWalker 对象。此方法需要一个用户函数作为筛选器，来选择节点，作为树的一部分，对此用户函数的引用是方法的第三个参数。此用户函数的返回值可以是表示当前节点状态的三个常量：NodeFilter.FILTER_ ACCEPT、NodeFilter.FILTER_REJECT 和 NodeFilter.FILTER_SKIP。NodeFilter.FILTER_REJECT 和 NodeFilter.FILTER_SKIP 之间的区别在于：被跳过节点的后代仍然可以成为树的组成部分，而被拒绝节点及其后代都不能成为树的组成部分。如下代码是可用于创建 TreeWalker 对象的用户函数：

```
function ratingAttrFilter(node)
{
   if (node.hasAttribute("rating"))
   {
      return NodeFilter.FILTER_ACCEPT;
   }
   return NodeFilter.FILTER_REJECT;
}
```

在此示例函数中，只有包含 rating 特性的节点才能通过筛选，这意味着只有这些节点才能添加到列表(树)中。准备好这个函数后，调用 document.createTreeWalker()方法来创建 TreeWalker 对象：

```
var myTreeWalker = document.createTreeWalker(document, NodeFilter.SHOW_ELEMENT,
                   ratingAttrFilter, false);
```

创建 TreeWalker 对象后，就可以使用其属性和方法来访问单个节点，在列表中导航。

29.3.3　属性

currentNode

值：节点引用，　　读/写

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

currentNode 属性返回对当前节点的引用，当前节点位于树的节点指针处。currentNode 属性可以用于访问当前节点，还可以设置当前节点。

示例

要将一个节点赋给树的当前位置，只需使用 currentNode 属性创建一个赋值语句：

```
myTreeWalker.currentNode = document.getElementById("info");
```

相关主题：root 属性。

expandEntityReference，filter，root，whatToShow

值：参见正文，　　只读

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

这些属性反映了创建 TreeWalker 对象时，传递到 document.createTreeWalker()方法中的参数值。

更多资源请访问稀酷客(www.ckook.com)

相关主题：document.createTreeWalker()方法。

29.3.4 方法

firstChild()，lastChild()，nextSibling()，parentNode()，previousSibling()

返回值：节点引用

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

这些方法返回 TreeWalker 对象所包含的树型节点列表层次结构中的节点引用。树中的所有节点之间都存在父子关系，这些函数可根据此关系获取节点引用。只要使用其中某个方法，导航到指定的节点，树中的节点指针就移动到新节点，这意味着在调用这类导航方法后，可将新节点作为当前节点来访问。

示例

下面的代码说明了如何在 TreeWalker 对象中获取当前节点的父节点的标记名称：

```
if (myTreeWalker.parentNode())
{
   var parentTag = myTreeWalker.currentNode.tagName;
}
```

相关主题：nextNode()、previousNode()方法。

nextNode()、previousNode()

返回值：节点引用

兼容性：WinIE-，MacIE-，NN7+，Moz+，Safari+，Opera+，Chrome+

nextNode()和 previousNode()方法可在 TreeWalker 对象所包含的节点列表中向后和向前导航。注意，这些方法在节点列表上的操作就像列表从树型变为线性节点序列一样。这两个方法分别将内部节点指针移到下一个或上一个节点。

示例

下面的代码演示了节点筛选函数和另一个函数，该函数可以(在一系列警告窗口中，可能是为了调试)显示 body 中所有指定了 id 特性的元素的 ID。首先调用 nextNode()方法，将 TreeWalker 的节点指针移到集合中的第一个节点，然后在 do-while 结构中循环，获取通过节点筛选测试的下一个节点。

```
function idFilter(node)
{
   if (node.hasAttribute("id"))
   {
      return NodeFilter.FILTER_ACCEPT;
   }
   return NodeFilter.FILTER_SKIP;
}
function showIds()
```


更多资源请访问稀酷客(www.ckook.com)

```
{
    var tw =
    document.createTreeWalker(document.body, NodeFilter.SHOW_ELEMENT,
        idFilter, false);
    // make sure TreeWalker contains at least one node, and go to it if true
    if (tw.nextNode())
    {
        do
        {
            alert(tw.currentNode.id);
        } while (tw.nextNode());
    }
}
```

相关主题： parentNode()方法。

更多资源请访问稀酷客(www.ckook.com)

第30章

link 和 anchor 对象

本章包含哪些内容？
link、anchor 和 a 元素对象之间的区别
通过脚本编程，用链接来调用脚本函数
通过脚本编程，用链接在鼠标滚动时交换图像

Web 的目标是世界上的信息可以通过超链接(hyperlink)连接到一起，超链接是可单击的文本块或图像，好奇的读者可以利用它们导航到详细的解释或相关材料。在 JavaScript 使用的所有文档对象中，link(链接)就是进行这种连接的对象，anchor(锚)也提供了到文档中特定位置的路径。

在第一代支持脚本编程的浏览器中，链接和锚是较为简单的对象，但这种简单掩盖了它们在整个 Web 方案中的重要性。在脚本的控制下，链接的功能远不止是连接到 Web 上。

在现代浏览器中，不再把链接和锚区分为两个类似但明显不同的对象。较新浏览器将 link 元素看成脚本对象，可能会混淆 link 与对象的关系，link 元素有着完全不同的作用(参见配书光盘上的第 40 章)。HTML 元素对象取代了锚和链接对象，该 HTML 元素表示一个由<a>标记创建的元素。a 元素对象拥有现代对象模型中所有 HTML 元素对象都有的全部属性、方法和事件处理程序。为帮助你了解这些转变，本章介绍了这三种对象类型。

锚、链接和 a 元素对象

关于 HTML 元素的属性、方法和事件处理程序，请参见第 26 章。

属　性	方　法	事件处理程序
charset		
coords		
hash		
host		
hostname		
href		

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　　性	方　　法	事件处理程序
hreflang		
methods		
mimeType		
name		
enameProp		
pathname		
port		
protocol		
rel		
rev		
search		
shape		
target		
type		
urn		

语法

访问 link 对象的属性：

```
(all) [window.]document.links[index].property
```

访问 a 元素对象的属性：

```
(IE4+)       [window.]document.all.elemID.property | method([parameters])
(IE5+/W3C)  [window.]document.getElementById("elemID").property |
    method([parameters])
```

兼容性： WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

关于 link 对象

了解脚本编程的历史，有助于了解 link 和 anchor 对象的来龙去脉，以及 a 元素对象如何从它们进化而来。

在原始对象模型的术语中，anchor 和 link 对象都在<a>标记中创建。区分链接和锚点的是标记是否有 href 特性，如果没有 href 特性，元素就是 anchor 对象，它在现代浏览器中只有一个 name 属性。另一方面，link 更像一个对象，因为它包含 href 特性，这个特性通常指向窗口或框架的 URL。

对象模型将 HTWM 元素处理为对象时，会将 anchor 和 link 对象都包括在 a 元素对象中。

更多资源请访问稀酷客(www.ckook.com)

即便如此，原对象依然有一个重要的特征：像 link 对象(有 href 特性)那样执行操作的所有 a 元素对象都是 document.links 属性数组的成员。因此，如果脚本需要在页面上检测或修改所有 link 对象的属性，可以使用 for 循环来遍历 link 对象数组。即使脚本只适用于现代浏览器，要改变所有链接的 style 属性(例如，将它们的 style.textdecoration 属性从 none 改变为 underline)，这也是可行的。a 元素对象根据是否存在 href 特性而具有两种行为，所以被当成两个不同的东西。这两者的区别十分重要，所以对 link 和 anchor 对象的引用是分开的。

脚本编程新手想用脚本编写链接，使其作用于不同的框架或窗口时，常常弄不清楚 a 元素的 target 特性。在纯 HTML 中，target 特性指向要加载新文档的框架或窗口(赋予 href 特性)，而当前窗口或框架保持不变。但如果想要(通过设置 location.href 属性)用事件处理程序来导航，target 特性就不能应用在脚本编制的操作中，而必须为期望框架或窗口的 location.href 属性指定新的 URL。例如，如果框架包含完全由链接构成的目录，这些链接的 onclick 事件处理程序可将 parent.main.location.href 属性设置为 URL，将其他页面载入 main 框架。还必须取消链接的默认行为，如第 26 章中的通用 onclick 事件处理程序所示。

如果想单击链接(不管链接是由文本还是图像组成)来启动某个动作，而不导航到另一个 URL 上，可使用一个特殊技巧：javascript:伪 URL，将 URL 指向 JavaScript 函数。URL javascript: functionName()是 href 特性的一个有效参数(且不仅仅用于 link 对象)。也可以增加一个特殊的 void 运算符来确保所调用的函数不触发任何链接动作(href = "javascript: void someFunction()")。为 href 特性指定空字符串，会为客户机生成一个 FTP 样式的文件列表，这是一个不符合需要的结果。还要注意，如果 URL 指向初始化浏览器帮助程序的文件类型(例如播放 QuickTime 电影)，帮助程序或插件会载入并运行，而不改变浏览器窗口中的页面。

注意：

JavaScript:伪 URL 的用法是有争议的，尽管大多数浏览器都支持它，但没有支持它的公开行业标准。在禁用了脚本或无法使用脚本的浏览器(例如为存在视觉障碍的用户设计的浏览器)上，它对访问页面的用户也不友好，因为链接不起作用，这会让用户感到失望。还要注意，搜索引擎在访问站点时，不会跟踪此类链接。

利用 JavaScript，单个链接可一次改变多个框架的内容。如果希望只有支持 JavaScript 的浏览器操作这个链接，可以用 javascript:伪 URL 调用一个函数来改变多个框架的 location.href 属性。例如，下面的函数改变两个框架的内容：

```
function navFrames(url1, url2)
{
   parent.product.location.href = url1;
   parent.accessories.location.href = url2;
}
```

然后用 javascript:伪 URL 调用这个多功能函数，将链接的细节作为参数传递给它：

```
<a href="javascript: void navFrames('products/gizmo344.html',
  'access/access344.html')">Deluxe Super Gizmo</a>
```

或者，如果希望某个链接对每个人都执行某个操作，但为支持 JavaScript 的浏览器执行另

更多资源请访问稀酷客(www.ckook.com)

一个操作(这种方式适用于为可访问性设计页面)，则可将标准链接行为和 onclick 事件处理程序组合在一起来处理这两种情况：

```
function setAccessFrame(url)
{
   parent.accessories.location.href = url;
}
...
<a href="products/gizmo344.html" target="product"
onclick="setAccessFrame('access/access344.html')">Deluxe Super Gizmo</a>
```

注意：

前面示例中使用的属性分配事件处理技术是有意简化了的，以便提高代码的可读性。通常最好采用更现代的方式，即用 addEventListener() (NN6+/Moz/W3C)或 attachEvent() (IE5+)方法来绑定事件。第 32 章将详细介绍现代的跨浏览器事件处理技术。

注意，在这里，标准链接行为需要使用 target 特性，而脚本将一个 URL 赋给框架的 location.href 属性。

另一个技巧允许单个链接标记在脚本和非脚本浏览器中操作。在非脚本浏览器中，需要建立一个真正的 URL，在链接中导航，然后确保链接的 onclick 事件处理程序返回 false，或者取消默认动作。在单击时，脚本浏览器执行事件处理程序，并忽略 href 特性；非脚本浏览器忽略事件处理程序，而进行链接。请参见第 26 章，来了解通用 onclick 事件处理程序的详情。

属性

charset

值： 字符串， 读/写

兼容性： WinIE6+，MacIE6+，NN6+，Moz+，Safari+，Opera+，Chrome+

charset 属性表示 a 元素的 HTML 4 charset 特性，它建议浏览器，文档使用 href 特性表示的字符集。该值为字符集编码字符串，这个编码在 http://www.iana.org/assignments/character-sets 注册。在 Web 上，最常用的字符集是 ISO-8859-5。

coords，shape

值： 字符串， 读/写

兼容性： WinIE6+，MacIE6+，NN6+，Moz+，Safari+，Opera+，Chrome+

当链接包含图像时，HTML 4 为容纳不同形状(方形、圆形和多边形)和坐标的 a 元素提供了规范。尽管所有 W3C DOM 兼容浏览器中的 a 元素对象都存在 coords 和 shape 属性，NN6 却不支持这个特性。

hash，host，hostname，pathname，port，protocol，search

值： 字符串， 读/写

兼容性： WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

这个庞大的属性集合与 location 对象的同名属性相同(见第 28 章)，所有属性都是 link 对象中 href 特性的 URL 的组成部分。尽管这些属性都不在 a 元素对象的 W3C DOM 规范中，但为了向后兼容，它们仍留在现代浏览器中。如果想用脚本改变链接的目标，可修改对象的 href 属性值，而不是 URL 的单个部分。

相关主题：location 对象。

href

值：字符串，　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

href 属性(包含在 W3C DOM 中)是 a 元素的目标 URL，用作链接。URL 可以是相对的，也可以是绝对的。

在兼容 W3C DOM 的浏览器中，可指定 href 属性的值，将 anchor 对象转换成 link 对象，尽管在从服务器载入的 HTML 中，a 元素没有 href 属性。自然，这一转换是临时的，它只持续到页面载入浏览器后。给包含文本的 a 元素指定 href 属性值后，文本就呈现出链接的外观(默认的外观，或者给链接指定的样式)。

相关主题：location 对象。

hreflang

值：字符串，　　读/写

兼容性：WinIE6+，MacIE6+，NN6+，Moz+，Safari+，Opera+，Chrome+

hreflang 属性给浏览器(如果浏览器使用它)推荐编写内容的语言，该语言用于 a 元素的 href 特性指向的内容，这个属性的值必须是标准语言编码的形式(例如美式英语是 en-us).

Methods

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

Methods 属性(注意大写的 M)表示 HTML4.0 中 a 元素的 methods 特性。这个特性的值建议浏览器使用哪个 HTTP 方法访问目的文档。HTML 4 特性很少不包含在 W3C DOM 中。尽管如此，IE4＋支持这个属性，但 IE 浏览器不对信息做任何处理。

mimeType

值：字符串，　　只读

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

此属性可以获取通过 a 元素链接的文档的 MIME 类型。HTML4 和 W3C DOM 规范定义了 type 特性和 type 属性，而没有定义 mimeType。该属性是只读的，因此无法控制目标文档的 MIME 类型。有趣的是，a 对象的 IE8 MSDN 文档没有列出这个属性，但 IE8 仍支持它。

相关主题：a.type 属性。

更多资源请访问稀酷客(www.ckook.com)

name

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

name 特性对仅作为 link 对象的 a 元素来说是可选的，但它对于 anchor 对象是必须的。这个值通过 name 属性进行脚本编程。虽然不一定需要通过脚本改变其值，但可用这一属性在重复循环中识别 document.links 数组中的 link 对象。例如：

```
for (var i = 0; i < document.links.length; i++)
{
    if (document.links[i].name == "bottom"
    {
       // statements dealing with the link named "bottom"
    }
}
```

如果此代码将 name 属性放在 if 子句中，就可以找到 bottom 链接。

nameProp

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-

IE 专用的 nameProp 属性可以获得 href 的一部分(URL 最右边斜杠的右边部分)。通常这个值是 URL 中的文件名，但如果 URL 也包含端口号，端口号就作为 nameProp 值的一部分返回。

rel，rev

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

rel 和 rev 属性定义了 a 元素的目标文档前进和后退的方向。换句话说，它们描述了链接与其指向的文档的关系，以及文档与其内容的关系。例如，在目录页中，指向各章的链接将其 rel 特性设置为 chapter，而其 rev 特性设置为 contents。这些特性和属性的大多数潜力还有待开发。

这些属性根据 HTML 4 中相应的特性值来预定义其值。如果浏览器不处理这些值，就忽略这些值。另外，可在单个字符串中用空格分隔符将多个值串在一起。可接受的值如表 30-1 所示：

表 30-1　可接受的值

alternate	contents	index	start
appendix	copyright	Next	stylesheet
bookmark	glossary	Prev	subsection
chapter	help	section	

target

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

link 对象的一个重要属性是 target，其值是提供给 a 元素的 target 特性的窗口名。

该属性可以临时改变链接的目标，但是，与大多数临时对象属性相同，该设置在软重载时失效。与其用这种方式改变目标，不如让 htef 特性调用 javascript: functionName ()这个伪 URL(在其中，函数将文档赋予期望的 window.location)，来稳妥地改变目标。如果以前编写过大量的 HTML 代码，就很难抛弃依赖 target 特性的习惯。

target 特性的另一个缺点是严格的 XHTML DTD 不支持它。因此，如果开发的 XHTML 页面必须用严格的 DTD 进行验证，就不能在<a>标记中包含 target 特性。而应使用页面的 onload 事件处理程序或 a 元素的 onclick 事件处理程序，来调用一个函数，将所需的值赋给 target 属性。这里使用 JavaScript 属性来绕开 HTML 特性的限制。

相关主题：document.links 属性。

type

值：字符串，　　读/写

兼容性：WinIE6+，MacIE6+，NN6+，Moz+，Safari+，Opera+，Chrome+

type 属性表示 HTML4 中的 type 特性，它为目的文档(该文档用元素的 href 特性指定)的内容指定 MIME 类型。这个属性主要用于建议浏览器，根据链接另一端的内容类型，来显示不同的光标样式。到目前为止，浏览器不使用这一特征。然而，可以为特性指定 MIME 类型值(例如 video/mpeq)，让脚本读出这些值，这样就可以在页面载入后，改变链接文本的样式。IE4+用 mimeType 属性实现了一个类似的属性。

相关主题：a.mimeType 属性。

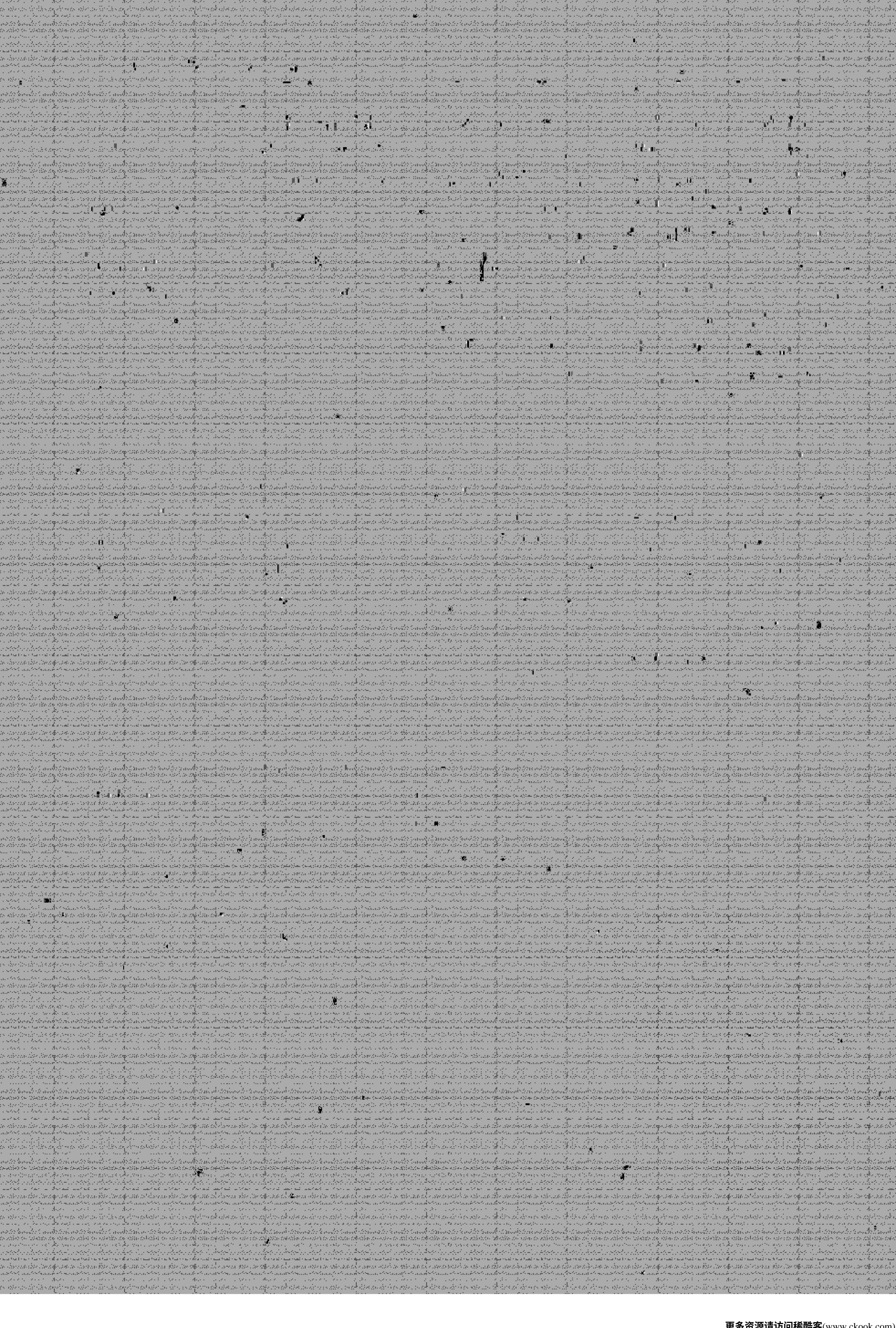
urn

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

urn 属性表示 IE 专用的 URN 特性，它允许程序员对 a 元素的目标使用 URN (Uniform Resource Name，详见 http://www.ietf.org/rfc/rfc2141.txt)。这个属性并不常用。

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第31章

image、area、map 和 canvas 对象

在现代 Web 浏览器中，由<img>和<area>标记创建的图像和区域是可以用脚本编程来增强交互性的一流对象。显示在<img>标记中的图像可与其他图像交换，把光标滚动到图标按钮上，图标按钮就会高亮显示。利用支持脚本的客户端区域映射，页面可以更智能地响应用户在图形区域上的单击。

对脚本设计者而言，另一个优点是载入页面时，图像可在浏览器的图像缓存中预先载入。由于图像已缓存，因此用户在第一次交换图像时不会有延迟。若连接的带宽较大，则对此功能的需求略低，但连接有速度限制的用户仍需要这个功能。

本章包含哪些内容？

- 预先缓存图像
- 文档加载后交换图像
- 创建交互式的客户端图像映射
- 使用画布绘制矢量图

在 JavaScript 的图形场景中，画布是一个新概念，它是可以通过 JavaScript 代码操作图形的一个图形区域。有几个浏览器支持画布，因此下面开始试用它们。

31.1 image 和 img 元素对象

关于 HTML 元素属性、方法和事件处理程序，详见第 26 章。

属　性	方　法	事件处理程序
align		onabort
alt		onerror
border		onload
complete		
dynsrc		

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　　性	方　　法	事件处理程序
fileCreatedDate		
fileModifiedDate		
fileSize		
fileUpdatedDate		
height		
href		
hspace		
isMap		
loop		
longDesc		
lowsrc		
mimeType		
name		
nameProp		
naturalHeight		
naturalWidth		
protocol		
src		
start		
useMap		
vspace		
width		
x		
y		

31.1.1　语法

创建 image 对象：

```
imageObject = new Image([pixelWidth, pixelHeight]);
```

访问 img 元素和 image 对象的属性或方法：

```
(NN3+/IE4+)  [window.]document.imageName. property | method([parameters])
(NN3+/IE4+)  [window.]document.images[index]. property | method([parameters])
(NN3+/IE4+)  [window.]document.images["imageName"]. property |
             method([parameters])
(IE4+)       [window.]document.all.elemID.property | method([parameters])
```


更多资源请访问稀酷客(www.ckook.com)

```
(IE5+/W3C)  [window.]document.getElementById("elemID").property |
            method([parameters])
```

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

31.1.2 关于 image 对象

在详细讨论 image 对象之前，需要先理解静态 image 对象的实例与 img 元素对象之间的区别。静态 image 对象仅存在于浏览器的内存中，不向用户显示任何信息；img 元素是页面上通过<img>(或者非标准但可接受的<image>)标记创建的元素。脚本使用 image 对象的实例为页面预先缓存图像，但显然，image 对象没有多少可用的属性、方法和事件处理程序，因为它既不显示在页面上，也不受标记特性的影响。

将 img 元素处理为对象的主要优点是，即使文档已载入，并显示了初始图像后，脚本仍可以改变在页面的 img 对象空间上显示的图像，这种脚本编程的关键是图像的 src 属性。

页面通常与初始图像一起加载。该图像的标记指定了额外的特性，例如 height 和 width(以加速页面的显示)，还指定了图像是否使用客户端图像映射进行交互(见本章后面的 area 对象)。浏览页面时，可在同一位置用新的图像替换原来的图像(可能响应用户动作或脚本中的定时事件)。在支持 img 元素对象的旧浏览器中，载入到元素中的初始图像的宽度和高度会为图像建立一个大小固定的矩形区域。在该区域中载入另一个大小不同的图像时，图像会自动缩放，以放在该矩形区域中。但在现代浏览器中(IE4+/Moz/W3C)，改变图像的大小，会在该图像的周围自动重新排列页面的内容。当然，也可以禁用这个功能，并预先使图像与原图像的大小相同，以免自动重新排列页面内容。

使用单个 Image 对象实例的优点是，脚本可以创建虚拟图像，来存储预先加载的图像(图像载入图像缓存，但浏览器不显示图像)。这样，用户在浏览页面或等待页面的下载时，会把一个或多个不可见的图像载入内存。然后，为了响应页面上的用户操作，图像可以即时改变，而用户不必等待下载所需的图像。

为了预先载入图像，需要先将一个新的空 image 对象赋给全局变量。新图像通过 image 对象的构造函数创建：

```
var imageVariable = new Image(width, height);
```

如果将预先加载的图像的像素宽度和高度作为参数提供给构造函数，就有助于浏览器为图像分配内存空间。上述语句只是在内存中创建一个属性全部为空的对象。为强制浏览器将图像载入缓存，需要给对象的 src 属性指定图像文件的 URL：

```
var oneImage = new Image(55,68);
oneImage.src = "neatImage.gif";
```

载入这个图像时，可在状态栏上查看载入的进度，这与载入任何图像一样。之后，将这个已存储图像的 src 属性赋给页面上 img 元素对象的 src 属性：

```
document.images["someImage"].src = oneImage.src;
```

根据图像的类型和大小，这种载入的响应速度可能非常快，图像会以小图片的形式立即显

更多资源请访问稀酷客(www.ckook.com)

示出来。

一种经常使用的用户界面技术是，用户把鼠标指针移动到可单击的图像按钮上时，图像的外观就会改变。这个动作假定，在一个与对象相关的元素上触发鼠标事件。图像滚动操作常常使图像在两种不同的状态(正常显示和高亮显示)下显示。也可以再增加几个状态，更逼真地模拟可单击按钮在应用程序中的活动方式。在一些情况下，第 3 种状态表示按钮被单击时的状态。例如，如果在框架中使用图像滚动，以进行导航，则用户单击按钮时，会导航到产品区域，此时按钮保持选中状态，但其样式不同于图像滚动时的高亮显示。一些脚本设计者还提供了第 4 种状态：在图像上按下鼠标时，这种状态会显示很短暂的时间。每种状态都要求下载另一图像，因此这个效果可能会延迟页面的载入。

图像交换的速度很快，似乎可以用这种方法来创建动画。尽管这种方式适用于短暂的动画，但在 Web 页面中引入动画的其他方法(例如通过 GIF89a 标准图像、Flash 动画、Java applet 和多种插件程序)可以产生更便于控制其速度的动画。实际上，在某些卡通片动画上交换预载的 JavaScript image 对象，其速度可能会过快。此时可以用 setInterval()方法建立延迟机制，但帧之间精确的时间间隔因客户处理器的性能而异。

实现 img 元素对象的浏览器也实现了 document.images 数组。在使用脚本语句处理 img 元素或 image 对象之前，可以(也应该)利用这个数组作为条件分支。其语句如下：

```
if (document.images)
{
    // statements working with images as objects
}
```

在早期的浏览器中，这个数组不存在，所以将 if 子句的条件语句处理为 false。

这里讨论的大多数属性都对应 HTML 元素 img 的特性。在所显示的内容上，这些特性值的含义详见 HTML 4.01 规范(http://www.w3.org/TR/html4/)、HTML 5 规范(http://www.w3.org/TR/ html5/)和 Microsoft 为 IE 所做的扩充(http://msdn.microsoft.com/en-us/library/ms535259%28VS. 85%29. aspx)。

31.1.3　属性

align

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

align 属性定义图像相对于其周边文本内容的方向。它有两个用途，可以根据它的值(和图像是否受 float 样式特性的影响)来控制垂直或水平对齐。其值为字符串常量，如下：

absbottom	middle
absmiddle	right
baseline	texttop
bottom	top
left	

图像的默认对齐方式是 bottom。另外，元素的对齐方式也由样式表控制。它对应于

更多资源请访问稀酷客(www.ckook.com)

HTML 4.01 中废弃的 alignment 特性，在 HTML 5 中不存在对应的特性。在现代网页中，设计人员应尽可能使用样式表，而不是元素特性，来调整显示细节，如对齐方式。

程序清单 31-1 允许从不同的 align 属性值中选择，对于其 HTML 内嵌到其他文本中的图像，它们将影响图像的布局。调整窗口的大小，查看页面上文本换行的情况，及其对对齐方式的影响。并非所有浏览器都为每个选项提供了相同的对齐方式，因此应在多个支持该属性的浏览器中进行试验。

程序清单 31-1　测试图像的 align 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>img align Property</title>
      <script type="text/javascript">
      function setAlignment(sel)
      {
        document.getElementById("myIMG").align =
           sel.options[sel.selectedIndex].value;
      }
      </script>
   </head>
   <body>
      <h1>img align Property</h1>
      <hr />
      <form>
         Choose the image alignment: <select onchange="setAlignment(this)">
            <option value="absbottom">absbottom</option>
            <option value="absmiddle">absmiddle</option>
            <option value="baseline">baseline</option>
            <option value="bottom" selected="selected">bottom</option>
            <option value="left">left</option>
            <option value="middle">middle</option>
            <option value="right">right</option>
            <option value="texttop">texttop</option>
            <option value="top">top</option>
         </select>
      </form>
      <hr />
      <p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit, sed do
         eiusmod tempor incididunt ut labore et dolore magna aliqua.
         <img alt="image" id="myIMG" src="desk1.gif" height="90" width="120" />
         Ut enim adminim veniam, quis nostrud exercitation ullamco laboris nisi ut
         aliquip ex ea commodo consequat.</p>
   </body>
</html>
```

相关主题：text-align，float 样式表特性。

更多资源请访问稀酷客(www.ckook.com)

alt

值：字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

在图像下载到客户端之前，alt 属性可以设置或修改浏览器显示在图像矩形区域(假定在标记中指定了宽和高)上的文本。另外，如果浏览器关闭了图像(或不能显示图像)，alt 文本可以帮助用户了解应在这个位置上显示的图像。在 IE 中，把鼠标放在图像上，会显示 alt 文本。在页面载入后，可以修改这个 alt 文本。

示例

使用 The Evaluator(第 4 章)将一个字符串赋给页面上 document.myIMG 图像的 alt 属性。首先将一个不存在的图像赋给 src 属性，以清除现有图像：

```
document.myIMG.src = "fred.gif"
```

向下滚动到图像，可看到原图像被一个空白区域替代，该空白表示不存在的新图像。现在，将一个字符串赋给 alt 属性：

```
document.myIMG.alt = "Fred\'s face"
```

上述语句的反斜杠是必需的，它用于转义字符串中的单引号。向下滚动，查看图像空间中新的 alt 文本。The Evaluator 中这个简单的测试在大多数浏览器中都有效。在有些浏览器中，如 Firefox，它是无效的。Firefox 等浏览器支持该属性，但因为新图像不存在，所以没有替代原图像；又因为原图像没有问题，所以看不到修改 alt 属性的效果。

相关主题：title 属性。

border

值：整型， 读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

border 属性定义了图像边框的像素厚度。如果在 a 元素中放入图像，以使用鼠标事件(用于图像滚动)，就要确保设置＜img＞标记的 border=0，以防浏览器给图像创建普通链接类型的边框。即使该特性的默认值为 0，也要把图像放在 a 元素中，或将图像与一个客户端图像映射关联起来，给图像生成一个边框。

与 alignment 属性相同，元素的边框也由样式表控制。它对应于 border 特性，但在 HTML 5 中不存在该属性。在现代网页中，设计人员应尽可能使用样式表(而不是元素特性)来调整显示细节，如边框。

示例

在 The Evaluator(第 4 章)中将不同的整数值赋给 document.myIMG.border 属性，试验图像的该属性。

相关主题：isMap，useMap 属性。

更多资源请访问稀酷客(www.ckook.com)

complete

值：Boolean， 只读

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

有时，需要在启动另一个过程前，确保当前图像载入完毕。这完全不同于在触发其他进程前等待图像的载入(可通过 image 对象的 onLoad 事件处理程序来实现)。要验证 img 对象是否显示了整幅图像，可检验 complete 属性的 Boolean 值；要验证某个图形文件是否已载入，首先检查 complete 属性是否为 true；然后比较 src 属性和期望的文件名。

即使只有指定的 lowsrc 图像载入完毕，图像的 complete 属性也变为 true，所以如果 src 和 lowsrc 特性都在<img>标记中指定，就不能仅依赖这个属性判断 src 图像是否载入完毕。

应用这个属性的最佳方式之一是用于 if 结构的条件语句：

```
if (document.myImage.complete)
{
    // statements that work with document.myImage
}
```

要练习 image.complete 属性，可运行程序清单 31-2。单击 Load NASA Image 按钮。图像载入后，单击 Is it loaded yet?按钮，查看 image 对象的 complete 属性值。在载入结束之前，其值为 false，载入结束后，其值变为 true。这里故意选择直接从 NASA 网站上载入一个大图像，以便有足够的时间单击几次 Is it loaded yet?按钮，查看 complete 属性的值。如果已经访问过 NASA 网站的图像库，就可能需要在试验期间退出并重新启动浏览器，以清除缓存中的图像(或清空浏览器的内存缓存)。

程序清单 31-2 image.complete 的脚本编程

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>The Complete Property</title>
      <script type="text/javascript">
      function loadIt()
      {
         document.getElementById("result").value = "";
         document.getElementById("theImage").src =
           "http://www.nasa.gov/images/content/402217main_ssc2008-01b_full.jpg" ;
      }
      function checkLoad()
      {
         document.getElementById("result").value =
            document.getElementById("theImage").complete;
      }
      </script>
   </head>
   <body>
      <img id="theImage" alt="image" width="120" height="90" />
```


更多资源请访问稀酷客(www.ckook.com)

```
        <form>
           <input type="button" value="Load NASA image"
                 onclick="loadIt();" />
           <p><input type="button" value="Is it loaded yet?"
                    onclick="checkLoad();" />
              <input id="result" type="text" />
           </p>
           <input type="reset" />
        </form>
     </body>
  </html>
```

注意：

前面示例中使用的属性分配事件处理技术是有意简化了的，以提高代码的可读性。通常最好采用更现代的方式，即用 addEventListener()(NN6+/Moz/W3C)或 attachEvent()(IE5+)方法来绑定事件。第 32 章将详细介绍现代的跨浏览器事件处理技术。

相关主题：img.src，img.lowsrc，img.readyState 属性；onload 事件处理程序。

dynsrc

值：URL 字符串，　　读/写

兼容性：WinIE4-6，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

dynsrc 属性是视频源文件的 URL，在 IE 中，它可以通过 img 元素来播放。为 image 元素对象的 dynsrc 属性指定有效视频源文件(例如.avi 或.mpg 文件)的 URL，可将静态图像的区域变成视频播放器。但与 image 对象的 src 属性不同，将 URL 赋给 dynsrc 属性不会预先缓存视频。

在 img 元素显示.gif 或.jpg 图像后，给图像的 dynsrc 属性指定一个值，就会在 IE 的不同版本中出错。在 WinIE5 中，即使下载已经完成，状态栏也指示视频文件仍在下载，单击 Stop 按钮也没用；WinIE5.5+甚至不载入视频文件，尽管该属性仍存储了 URL 的值，也仅在页面上显示一个空白区域；MacIE5 在静态和动态图像之间转换不存在任何问题，但多次播放视频文件将导致 img 元素在其矩形区域上显示黑屏。

相关主题：img.loop，img.start 属性。

fileCreatedDate，fileModifiedDate，fileUpdatedDate，fileSize

值：字符串，整型(文件大小)，　　只读

兼容性：WinIE4+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

这 4 个 IE 专用的属性返回在 img 元素中显示的文件的信息(不管是静态还是动态图像)。前 3 个属性分别显示当前图像文件的创建、修改和更新日期。未修改文件的创建和修改日期是相同的。图像的更新日期是图像文件上次上传到服务器的日期，只有 WinIE5.5+和 MacIE5 支持 fileUpdatedDate 属性；fileSize 属性显示了文件的字节数。

前两个属性返回的 date 值在 IE4 和 IE5 中有不同的格式，第一个属性提供了完整的天数和日期；第二个属性返回 mm/dd/yyyy 格式。然而，这些值仅包含日期，而不包含时间。在任何情况下，都可以用这些值作为 new Date()构造函数的参数，创建 Date 对象。之后就可以进行日

更多资源请访问稀酷客(www.ckook.com)

期计算，获得一些信息，诸如当天和最近一次修改之间间隔的天数。

并非所有服务器都提供了正确的文件日期或大小信息，也不一定使用 IE 可解释的格式，所以需要在部署服务器上验证这些信息，来确保兼容性。

另外，注意对于载入浏览器的文件来说，这些属性是只读的。JavaScript 本身不能从服务器上获得这些文件信息。

注意：

这些与文件相关的属性在 Mac 版本的 IE 中都存在，但值为空。

示例

这些属性类似于 document 对象的同名属性，程序清单 18-4 演示了这些属性。复制该程序清单，提供一个图像，然后将 document 对象引用改为 Image 对象引用，看看这些属性如何与 img 元素对象一起工作。

或在 The Evaluator(第 4 章)中使用现有图像，一次试验一个属性：

```
document.getElementById("myIMG").fileSize
```

相关主题：无。

height，width

值：整型，　　　　读/写(见正文)

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

height 和 width 属性返回并控制 image 对象的像素宽度和高度。在所有支持 img 元素对象的现代浏览器中，该属性是可读/写的。不过，更改这些属性的效果因浏览器而异。例如，如果在 Mozilla 中调整图像的 height 属性，该浏览器就会自动缩放图像，使其与原图像有相同的比例，但调整 width 属性对 height 属性没有影响。在 IE7 中，调整 width 和 height 的效果是相同的，只要动态缩放图像，图像中就可能出现多余的像素，因此，修改图像的大小要非常小心。

示例

使用 The Evaluator(第 4 章)试验 height 和 width 属性。首先在顶部文本框中输入如下语句，获取默认值：

```
document.myIMG.height
document.myIMG.width
```

将图像的高度从默认的 90 增加到 180：

```
document.myIMG.height = 180
```

然后增加宽度：

```
document.myIMG.width = 400
```

查看所得的图像效果。

相关主题：hspace，vspace 属性。

更多资源请访问稀酷客(www.ckook.com)

href

详见 src 属性。

hspace，vspace

值：整型，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

hspace 和 vspace 属性控制包围图像的透明页边距的像素宽度。具体而言，hspace 属性控制图像左边和右边的页边距；而 vspace 控制顶部和底部的页边距。默认情况下，图像的页边距是 0 像素。

示例

使用 The Evaluator(第 4 章)试验 hspace 和 vspace 属性。首先注意没有为页面底部的图像指定边距，该图像与页面左对齐。现在把水平边距指定为 30 像素：

```
document.myIMG.hspace = 30
```

图像向右移动 30 像素，图像右侧也存在着不可见的边距。

相关主题：height，width 属性。

isMap

值：Boolean，　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

isMap 属性可以指定图像是否作为服务器端的图像映射。图像设置为服务器端的图像映射时，单击点的像素坐标就作为参数传递给包围图像的链接 href；有关客户端图像映射的详情，请参见本章后面的 useMap 属性。

示例

The Evaluator 中的图像未定义为图像映射，因此，如果在顶部文本框中输入如下语句，该属性返回 false：

```
document.myIMG.isMap
```

相关主题：img.useMap 属性。

longDesc

值：URL 字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE6+，MacIE5+，NN6+，Moz+，Safari+，Opera+，Chrome+

longDesc 属性是一个文件的 URL，该文件提供了与 img 元素相关的图像的详细说明。当前浏览器能识别此属性，但不对信息进行任何操作——无论这些信息是用脚本指定还是用 longdesc 特性指定。

相关主题：alt 属性。

更多资源请访问稀酷客(www.ckook.com)

loop

值：整型，　读/写

兼容性：WinIE4-6，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

loop 属性表示通过 img 元素对象播放视频剪辑的次数。在视频播放指定的次数后，就在图像区域中显示视频的第一帧。该属性默认为 1；但如果该属性值设置为-1，视频就连续播放。不过，把 URL 指定给 dynsrc 属性之前，将 loop 属性设置为 0，视频也至少播放一次(除了在 Mac 上，如本章前面对 dynsrc 属性的讨论)。

相关主题：dynsrc 属性。

lowsrc，lowSrc

值：URL 字符串，　读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz-，Safari-，Opera-，Chrome-

对于加载时间较长的大图像，现代浏览器可在载入它时指定一个低分辨率图像或其他快速载入的占位符。这个替代图像可以通过<img>标记的 lowsrc 特性指定，该特性会反应在 image 对象的 lowsrc 属性中。所有现代浏览器都允许指定这个属性或其对应的 HTML 特性，但只有 IE 会显示指定的图像。

所有兼容的浏览器都识别这个属性的全小写版本，NN6 还识别大小写相间的 lowSrc 属性。

注意在一些浏览器中，如果将一个 URL 指定给 lowsrc 特性，complete 属性值就变为 true，在替代文件载入完毕后触发 onload 事件处理程序：浏览器不等待主 src 文件的载入。

示例

查看程序清单 31-4 中 Image 对象的 onload 事件处理程序，看看与源相关的属性如何影响事件的处理。

相关主题：img.src，img. complete 属性。

mimeType

值：字符串，　只读

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

mimeType 属性返回图像的 MIME 类型的纯语言描述，如 JPEG Image 或 GIF Image。

示例

在 Internet Explorer 中，可使用 mimeType 属性来确定图像的格式，如下：

```
if (document.myIMG.mimeType.indexOf("JPEG") != -1) {
   // Carry out JPEG-specific processing
}
```

在此示例中，index0f()方法检查 MIME 类型字符串中是否存在 JPEG，这是可行的，因为 JPEG 图像的 mimeType 属性返回"JPEG Image"字符串。

相关主题：无。

更多资源请访问稀酷客(www.ckook.com)

name

值：标识符字符串， 读/写

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+ Opera+，Chrome+

name 属性返回 img 元素的 name 特性值。现代浏览器可以使用元素的 ID(id 特性)，通过 document.all(IE)和 document.getElementById()引用 img 元素对象，但 document.imageName 和 document.images[imageName]形式的引用只能使用 name 特性的值。

在一些设计中，为 img 元素指定数字序列名是很方便的，如 imgl、img2 等。与脚本标识符一样，名称不能以数字开头。img 元素对象的名称很少需要改变。

示例

使用 The Evaluator(第 4 章)可以查看该页上图像的 name 属性的返回值，在顶部文本框中输入如下语句：

```
document.myIMG.name
```

当然，这是多余的，因为名称是对象引用的一部分。

相关主题：id 属性。

nameProp

值：文件名字符串， 只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

IE 中的 src 属性返回完整的 URL，而 IE 的 nameProp 属性仅返回文件名，不包括协议和路径。如果图像交换脚本需要读取当前指定给图像的文件名(以确定下一个显示的图像)，使用 nameProp 属性很容易获得实际的文件名，而不必解析 URL。

示例

可以用 The Evaluator Sr.(第 4 章)比较 src 和 nameProp 属性在 WinIE5+中的结果。在顶部文本框中输入如下语句：

```
document.myIMG.src
document.myIMG.nameProp
```

相关主题：img.src 属性。

naturalHeight，naturalWidth

值：整数， 只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera-，Chrome+

naturalHeight 和 naturalWidth 属性返回未缩放的图像高度和宽度，以像素为单位。若脚本代码或 img 元素特性已经缩放了图像，而用户想知道图像的原始大小，就可以使用这些属性。

示例

用 The Evaluator(第 4 章)在基于 Mozilla 或 WebKit 的浏览器中试验 naturalHeight 和

更多资源请访问稀酷客(www.ckook.com)

naturalWidth 属性。首先在顶部文本框中输入如下语句，获取默认值：

```
document.myIMG.width
```

将图像宽度从默认的 120 增加到 200：

```
document.myIMG.width = 200
```

如果向下滚动到图像，可看到图像已经按比例缩放了。现在可以查看 naturalWidth 属性，确定原始图像的自然宽度：

```
document.myIMG. naturalWidth
```

The Evaluator 显示图像的自然宽度为 120，但图像的宽度目前放大到 200。

相关主题：img. height，img.width 属性。

protocol

值：字符串， 只读

兼容性：WinIE4+，MacIE5+，NN-，Moz-，Safari-，Opera-，Chrome-

IE 的 protocol 属性仅返回完整 URL 的协议部分，而 src 属性返回完整的 URL。这允许脚本执行一些操作，例如辨别图像是来自于本地硬盘还是 Web 服务器。该属性的返回值并非实际的协议字符串，而是协议的描述信息：HyperText Transfer Protocol 或 File Protocol。

示例

使用 The Evaluator Sr.(第 4 章)可以查看页面上图像的 protocol 属性。在顶部文本框中输入如下语句：

```
document.myIMG.protocol
```

相关主题：img.src，img.nameProp 属性。

src

值：URL 字符串， 读/写

兼容性：WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

src 属性是(在内存的 image 对象实例中)预缓存图像和(在 img 元素对象中)执行图像交换的途径。将 URL 赋予内存中 image 对象的 src 属性，浏览器就将图像载入浏览器缓存(假定用户启用了缓存)；而将 URL 赋予 img 元素对象的 src 属性，该元素就会显示新图像。为了利用这个功能强大的组合，应将可交换图像的替代版本预先载入内存中的 image 对象，然后将 Image 对象的 src 属性赋予期望的 img 元素对象的 src 属性。

在旧浏览器中，初始 img 元素定义了图像的大小，来控制分配给图像的矩形区域。如果为 img 元素对象指定了另一个大小不同的图像，图像就会重新缩放，以放在矩形区域中(通常会导致图像变形)。但在所有现代浏览器中，img 元素对象可以重新改变其大小，来包含图像，页面内容也将在新尺寸的图像周围重新编排。

注意，读取 src 属性时，会返回图像文件的完整 URL，包含协议和路径。脚本利用文件名

更多资源请访问稀酷客(www.ckook.com)

来交换一系列图像常常很不方便。其他一些机制(如将当前文件名保存到全局变量中)可能更简单(见 WinIE5+的 nameProp 属性)。

示例

以下示例(参见程序清单 31-3)演示了图像对象的一些应用。最重要的是对比预缓存图像和普通图像给用户的感觉。另外，本例还说明了如何设置计时器，来自动更改显示在图像对象中的图像。在显示广告横幅或幻灯片的站点中，常常需要此功能。

载入页面时，把一个全局变量传递给图像对象数组，其数组项将字符串名称作为下标值(deskl、desk2 等)，这些名称最终用作数组项的地址。数组中的每个图像对象都指定了一个 URL，它预先缓存了图像。

页面(参见图 31-1)包括两个 img 元素：一个显示未缓存的图像，一个显示已经缓存的图像。在每个图像下是一个 select 元素，用来为每个元素选择 4 个可能的图像文件之一。每个 select 列表的 onchange 事件处理程序调用不同的函数，来更改未缓存的(loadIndividual())或缓存的(loadCached())图像，这两个函数都将包含 select 元素的表单引用作为其唯一参数。

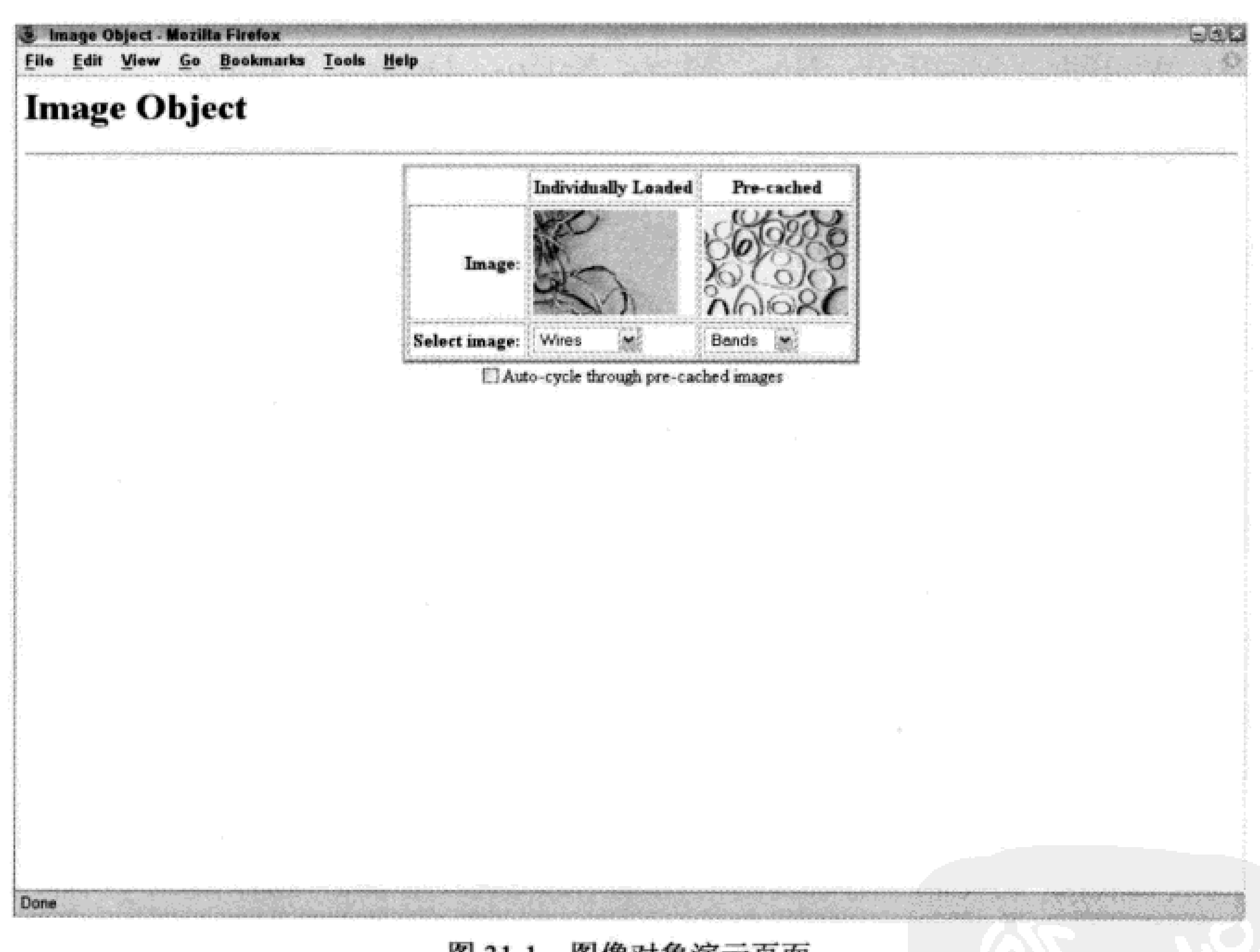


图 31-1　图像对象演示页面

要以 5s 的间隔循环显示图像，首先需要使用 checkTimer()函数查看是否选中计时器复选框。如果选中，就复制并递增缓存图像 select 控件的 selectedIndex 属性(如果下标已是最大值，就重置为 0)。调整 select 元素，以调用 loadCached()函数，读取当前选择的项，并相应地设置图像。

为了显示其他样式，<body>标记包括一个 onunload 事件处理程序，这个事件处理程序调用 resetSelects()函数，而此通用函数遍历页面上的所有表单和每个表单中的所有元素。对于每个 select 元素，selectedIndex 属性重置为 0。因此，如果用户重新载入页面，或者通过“后退”按钮返回页面，图像就按最初的顺序排列。onload 事件处理程序确保图像与 select 选项同步，checkTimer()函数在 5s 的延时后调用。不过，除非选中计时器复选框，否则不循环缓存的图像。

更多资源请访问稀酷客(www.ckook.com)

程序清单 31-3　通过脚本编程的图像对象和循环图像

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Image Object</title>
      <script type="text/javascript">
      // global declaration for 'desk' images array
      var imageDB;
      // pre-cache the 'desk' images
      if (document.images)
      {
          // list array index names for convenience
          var deskImages = new Array("desk1", "desk2", "desk3", "desk4");
          // build image array and pre-cache them
          imageDB = new Array(4);
          for (var i = 0; i < imageDB.length ; i++)
          {
              imageDB[deskImages[i]] = new Image(120,90);
              imageDB[deskImages[i]].src = deskImages[i] + ".gif";
          }
      }
      // change image of 'individual' image
      function loadIndividual(form)
      {
          if (document.images)
          {
              var gifName =
                  form.individual.options[form.individual.selectedIndex].value;
              document.getElementById("thumbnail1").src = gifName + ".gif";
          }
      }
      // change image of 'cached' image
      function loadCached(form)
      {
          if (document.images)
          {
              var gifIndex =
                  form.cached.options[form.cached.selectedIndex].value;
              document.getElementById("thumbnail2").src = imageDB[gifIndex].src;
          }
      }
      // if switched on, cycle 'cached' image to next in queue
      function checkTimer()
      {
          if (document.images & document.Timer.timerBox.checked)
          {
              var gifIndex = document.selections.cached.selectedIndex;
                 if (++gifIndex > imageDB.length - 1)
```


更多资源请访问稀酷客(www.ckook.com)

```
            {
                gifIndex = 0;
            }
            document.selections.cached.selectedIndex = gifIndex;
            loadCached(document.selections);
            var timeoutID = setTimeout("checkTimer()",5000);
        }
    }
    // reset form controls to defaults on unload
    function resetSelects()
    {
        for (var i = 0; i < document.forms.length; i++)
        {
            for (var j = 0; j < document.forms[i].elements.length; j++)
            {
                if (document.forms[i].elements[j].type == "select-one")
                {
                    document.forms[i].elements[j].selectedIndex = 0;
                }
            }
        }
    }
    // get things rolling
    function init()
    {
        loadIndividual(document.selections);
        loadCached(document.selections);
        setTimeout("checkTimer()",5000);
    }
    </script>
</head>
<body onload="init()" onunload="resetSelects ()">
    <h1>Image Object</h1>
    <hr />
    <center>
        <table border="3" cellpadding="3">
            <tr>
                <th></th>
                <th>Individually Loaded</th>
                <th>Pre-cached</th>
            </tr>
            <tr>
                <td align="right"><b>Image:</b></td>
                <td><img alt="image" src="cpu1.gif" id="thumbnail1"
                    height="90" width="120" /></td>
                <td><img alt="image" src="desk1.gif" id="thumbnail2"
                    height="90" width="120" /></td>
            </tr>
            <tr>
                <td align="right"><b>Select image:</b></td>
```


更多资源请访问稀酷客(www.ckook.com)

```
                <form name="selections">
                <td><select name="individual"
                   onchange="loadIndividual(this.form)">
                      <option value="cpu1">Wires</option>
                      <option value="cpu2">Keyboard</option>
                      <option value="cpu3">Discs</option>
                      <option value="cpu4">Cables</option>
                      </select></td>
                <td><select name="cached" onchange="loadCached(this.form)">
                   <option value="desk1">Bands</option>
                   <option value="desk2">Clips</option>
                   <option value="desk3">Lamp</option>
                   <option value="desk4">Erasers</option>
                </select></td>
                </form>
             </tr>
          </table>
          <form name="Timer">
             <input type="checkbox" name="timerBox"
             onclick="checkTimer()" />Auto-cycle through pre-cached images
          </form>
       </center>
    </body>
</html>
```

相关主题：img.lowsrc，img.nameProp 属性。

start

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4-6，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

在 IE4+中，start 属性用于通过 img 元素播放的视频剪辑。默认情况下，只要打开图像文件，视频剪辑就开始播放(除了在 Macintosh 中)，这符合 start 属性的默认设置 fileopen。另一个可识别的值是 mouseover，直到用户将鼠标光标移动到图像上，才播放视频剪辑。

相关主题：img.dynsrc，img.loop 属性。

useMap

值：标识符字符串，　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

useMap 属性表示 img 元素的 usrmap 特性，它指向页面上 map 元素的名称(见程序清单 31-6)，这个 map 元素包含客户端图像映射的信息(在本章后面描述)。useMap 属性的值必须包括＃号，它定义了页面上内部 HTML 的引用。如果需要为同一个 img 元素切换两个或多个图像映射(例如切换图像或用户在不同的模式下)，可定义多个名称不同的 map 元素，然后改变 img 元素对象的 useMap 属性值，将图像与不同的映射关联起来。

相关主题：isMap 属性。

更多资源请访问稀酷客(www.ckook.com)

vspace

详见 hspace。

width

详见 height。

x，y

值：整型，　只读

兼容性：WinIE-，MacIE-，NN4，Moz1+，Safari1+，Opera-，Chrome+

脚本可以通过 x 和 y 属性获取 img 元素的 x 和 y 坐标(图像所在矩形空间的左上角)，这些属性是只读的，通常不用，因为所有元素都有 offsetLeft 和 offsetTop 属性，IE 也支持这些属性。

相关主题：img.offsetLeft，img.offsetTop 属性；img.scrollIntoView()，window.scrollTo()方法。

31.1.4　事件处理程序

onabort，onerror

兼容性：WinIE4+，MacIE4+，NN3+，Moz-，Safari-，Opera-，Chrome-

脚本应该预先考虑到，在图像载入时，或者网络或服务器的问题导致图像传输失败时，用户会单击 Stop 按钮。此时，就需要使用 onabort 事件处理程序激活一个函数；对于意外传输错误，应使用 onerror 事件处理程序。注意，onerror 可用于所有浏览器，而 onabort 不是。

实际上，这些事件处理程序并未提供脚本需要的所有信息，例如此时载入的图像的文件名。如果此类信息对于脚本非常重要，脚本就需要在设置图像的 src 属性之前，将当前载入的图像名存储到一个变量中。虽然不知道触发 error 事件的错误性质，但可以强制重载脚本页面，或导航到 Web 站点中一个完全不同的地方，来解决这个问题。

示例

程序清单 31-4 包含一个 onabort 事件处理程序。如果图像已经在缓存中，就必须退出，并重启浏览器，以停止图像的载入。该示例为整个页面提供了一个重载选项。如何处理异常在很大程度上取决于页面设计，应尽力消除用户可能遇到的问题。

onload

兼容性：WinIE3+，MacIE3+，NN2+，Moz+，Safari+，Opera+，Chrome+

在以下三种情况下，会激活 img 对象的 onload 事件处理程序：图像的 lowsrc 图像载入完毕；若没有指定 lowsrc 图像，则 src 图像载入完毕；显示动态 GIF(GIF89a 格式)的每个帧。

如果在<img>标记中定义了 lowsrc 文件，则 src 图像载入后，img 对象就不接收其他信息，理解这一点非常重要。如果这些信息对脚本非常重要，应检测 image 对象的 src 属性，来验证当前的图像文件。

还要注意，img 元素的 onload 事件处理程序可能会在页面上其他元素载入完毕之前激活，因

更多资源请访问稀酷客(www.ckook.com)

此，如果事件处理函数引用页面上的其他元素，函数就应在定位这些元素之前验证其是否存在。

退出并重启浏览器，或者清空缓存，才能最有效地使用程序清单 31-4。在首次载入文档时，lowsrc 图像文件(铅笔擦图片)在 NASA 图像之前载入(并非所有浏览器都支持 lowsrc 属性，所以不一定在 NASA 图像之前看到铅笔擦图片)。这里故意选择直接从 NASA 网站上载入一个大图像，以便有较多的时间查看 onload 事件处理程序的行为。铅笔擦图片载入后，即使主图像(NASA 图像)尚未载入，onload 事件处理程序也在文本域中写入 done。

程序清单 31-4　图像的 onload 事件处理程序

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>The Image onload Event Handler</title>
      <script type="text/javascript">
      function loadIt(theImage,form)
      {
         if (document.images)
         {
            form.result.value = "";
            document.images[0].lowsrc = "desk1.gif";
            document.images[0].src = theImage;
         }
      }
      function checkLoad(form)
      {
         if (document.images)
         {
            form.result.value = document.images[0].complete;
         }
      }
      function signal()
      {
         if(confirm("You have stopped the image from loading. Do you want to try again?"))
         {
         location.reload();
         }
      }
      </script>
   </head>
   <body>
      <h1>The Image onload Event Handler</h1>
      <h3>Click the browser's STOP button to invoke the onabort event handler</h3>
      <img alt="image" width="120" height="90" lowsrc="desk4.gif"
           src="http://www.nasa.gov/images/content/406570main_s129e007575_hires_
                full.jpg"
           onload="if (document.forms[0].result) document.forms[0].result.value='done'"
           onabort="signal()" />
      <form>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <p><input type="button" value="Is it loaded yet?"
                onclick="checkLoad(this.form)" />
        <input type="text" name="result" />
        <input type="hidden" />
        </p>
    </form>
  </body>
</html>
```

相关主题：img.src，img.lowsrc 属性。

31.2　area 元素对象

关于 HTML 元素属性、方法和事件处理程序，请参见第 26 章。

属　　性	方　　法	事件处理程序
alt		
cords		
hash		
host		
hostname		
href		
noHref		
pathname		
port		
protocol		
search		
shape		
target		

31.2.1　语法

访问 area 元素对象的属性：

```
(NN3+/IE4+)  [window.]document.links[index].property
(IE4+)       [window.]document.all.elemID.property | method([parameters])
(IE4+)       [window.]document.all.MAPElemID.areas[index].property |
             method([parameters])
(IE5+/W3C)   [window.]document.getElementById("MAPElemID).areas
             [index].property | method([parameters])
(IE5+/W3C)   [window.]document.getElementById("elemID").property |
             method([parameters])
```


更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE4+，MacIE4+，NN3+，Moz+，Safari+，Opera+，Chrome+

31.2.2　关于 area 对象

文档对象模型将图像映射 area 对象处理为文档中的一个链接对象(a 元素，见第 30 章中的 anchor 对象)。这种处理根本没有违反逻辑，因为单击一个映射区域，通常会将用户引导到另一个文档，或定位到同一文档中的另一个锚点位置(超链接引用)。

尽管链接和映射区域的 HTML 定义完全不同，但这两种对象最早的脚本版本几乎具有相同的属性和事件处理程序。从 IE4、NN6/Moz 和符合 W3C 标准的浏览器开始，所有 area 元素特性都可以作为脚本属性来访问。而且，可以通过 map 元素对象来改变客户端图像映射区域的各个组成部分。map 元素对象包含一个嵌套的 area 元素对象数组，可以在 map 元素中删除、修改或添加 area 元素。

客户端图像映射使用起来十分有趣，自从 Netscape Navigator 2 引入这个特性以来，它们在 HTML 参考中就进行了很好的说明。本质上，可以根据形状和坐标在图像中定义任意多个区域。可以使用许多图形工具来获得图像的坐标，这些坐标需要放入<area>标记的 coords 特性中。

提示：
大多数 HTML 作者都会犯一个错误：<img>和<map>标记之间的微妙链接。必须给<map>指定一个名称；在<img>标记中，usemap 特性需要一个井号(#)和映射名称。如果忘记了#号，就不能创建图像及其映射之间的连接。

程序清单 31-5 包含的客户端图像映射示例允许在中东地区的不同地理位置间导航。在航测图上移动鼠标时，鼠标指针会在某些区域上改变，表示有一个链接与该区域相关联。单击区域将显示一个警告对话框，指出用户单击了哪个区域。

程序清单 31-5　简单的客户端图像映射

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Image Maps</title>
      <script type="text/javascript">
      function show(msg)
      {
         window.status = msg;
         return true;
      }
      function go(where)
      {
         alert("We're going to " + where + "!");
      }
      function clearIt()
      {
         window.status = "";
         return true;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
        </script>
    </head>
    <body>
        <h1>Sinai and Vicinity</h1>
        <img alt="image" src="nile.gif" width="320" height="240"
            usemap="#sinai" />
        <map id="sinai" name="sinai" >
          <area href="javascript:go('Cairo')" coords="12,152,26,161"
                shape="rect" onmouseover="return show('Cairo')"
                onmouseout="return clearIt()" />
          <area href="javascript:go('the Nile River')"
                shape="poly" onmouseover="return show('Nile River')"
                onmouseout="return clearIt()"
                coords="1,155,6,162,0,175,3,201,61,232,109,227,167,238,274,239,292,
                220,307,220,319,230,319,217,298,213,282,217,267,233,198,228,154,
                227,107,
                221,71,225,21,199,19,165,0,149" />
          <area href="javascript:go('Israel')" coords="95,69,201,91"
                shape="rect" onmouseover="return show('Israel')"
                onmouseout="return clearIt()" />
          <area href="javascript:go('Saudi Arabia')"
                coords="256,57,319,121"
                shape="rect" onmouseover="return show('Saudi Arabia')"
                onmouseout="return clearIt()" />
          <area href="javascript:go('the Mediterranean Sea')"
                coords="1,55,26,123" shape="rect"
                onmouseover="return show('Mediterranean Sea')"
                onmouseout="return clearIt()" />
          <area href="javascript:go('the Mediterranean Sea')"
                coords="27,56,104,103" shape="rect"
                onmouseover="return show('Mediterranean Sea')"
                onmouseout="return clearIt()" />
        </map>
    </body>
</html>
```

31.2.3　属性

alt

值：字符串，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

alt 属性表示 area 元素的 alt 特性，鼠标指针停留在 area 上时，Internet Explorer 就在 area 上的一个微型弹出窗口(工具提示)中显示 alt 文本。对于 Microsoft 在图像地图和普通图像中使用工具提示显示 alt 文本的做法，Web 开发人员们的看法存在分歧。

未来的浏览器可能实现这一特性，来提供与 area 元素相关联的链接信息。目前，Internet Explorer 是明确使用 alt 属性的唯一主流浏览器，但所有浏览器都能识别该属性。

更多资源请访问稀酷客(www.ckook.com)

相关主题：title 属性。

coords，shape

值：字符串，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

coords 和 shape 属性控制图像热区(由 area 元素控制)的位置、大小和形状。shape 属性值控制着 coords 属性值的格式，如表 31-1 所示。

表 31-1　形状、坐标和示例

形　状	坐　标	示　例
circ	center-x, center-y, radius	"30，30，31"
circle	center-x, center-y, radius	"30，30，31"
poly	x1，y1，x2, y2,…	"0，0，0，30，15，30，0，0"
polygon	x1, yl, x2, y2,…	"0，0，0，30，15，30，0，0"
rect	left, top, right, bottom	"10，31，60，40"
rectangle	left, top, right, bottom	"10，31，60，40"

area 的默认形状为 rectangle(矩形)。

相关主题：无。

hash，host，hostname，href，pathname，port，protocol，search，target

详见 link 对象的相应属性(第 30 章)。

noHref

值：Boolean，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

可以使用 noHref 属性来启用或禁用映射中的特定区域，其值默认为 true，表示启用区域。将此属性设置为 false 会阻止区域作为映射中的链接。该属性在浏览器中的实际行为可能无法预测，所以应仔细测试。

shape

详见 coords。

31.3　map 元素对象

有关 HTML 元素属性、方法和事件处理程序，请参见第 26 章。

更多资源请访问稀酷客(www.ckook.com)

属　性	方　法	事件处理程序
areas[]		onscroll+
name		

+参见第 29 章。

31.3.1　语法

访问 map 元素对象的属性：

```
(IE4+)       [window.]document.all.elemID.property | method([parameters])
(IE5+/W3C) [window.]document.getElementById("elemID").property |
             method([parameters])
```

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

31.3.2　关于 map 对象

map 元素对象是所有 area 元素的不可见 HTML 容器，每个 area 元素都为图像定义了一个“热区”。客户端图像映射将链接(和目标)与图像的矩形、圆形或多边形区域关联起来。

到目前为止，map 元素对象最重要的属性是 areas 数组和 name 属性。一般不改变 map 的名称(最好用不同的名称定义多个 map 元素，然后给 img 元素对象的 useMap 属性赋予希望的名称)，但可用 areas 数组改变给定客户端映射中 area 对象的组成。

31.3.3　属性

area[]

值：area 元素对象数组，　　　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

areas 数组可以用于迭代 map 元素中的所有 area 元素对象。基于 Mozilla 和 WebKit 的浏览器严格遵循 W3C DOM 的文档节点结构，而 IE4+可以更直接地访问嵌套在 map 中的 area 元素对象。如果想重写 map 中的 area 元素对象，可将 areas 数组的 length 属性设置为 0，清空先前的内容，然后在数组中插入 area 元素对象，来建立该数组。

程序清单 31-6 演示了如何使用脚本来替换 map 元素中的 area 元素对象。页面载入了一个计算机键盘图像，此图像链接到 keyboardMap 客户端图像地图上，该地图指定了图像上三个热区的详细信息。如果切换显示在该 img 元素中的图像，脚本就改变 img 元素对象的 useMap 属性，以指向第二个地图，该地图指定了第二个图像中台灯图像的详细信息。在图像上滚动鼠标指针，查看状态栏中与每个区域相关联的 URL(对于此示例，URL 不打开其他页面)。

页面上的另一个按钮调用 makeAreas()函数(不能用于 MacIE5)，该函数创建 4 个新的 area 元素对象，并通过 DOM 特有的途径将这些新区域的说明添加到图像中。在函数执行后，如果在图像上滚动鼠标，URL 就反映了新区域的 URL。还要注意添加了第 4 个区域，其状态栏消息如图 31-2 所示。

更多资源请访问稀酷客(www.ckook.com)

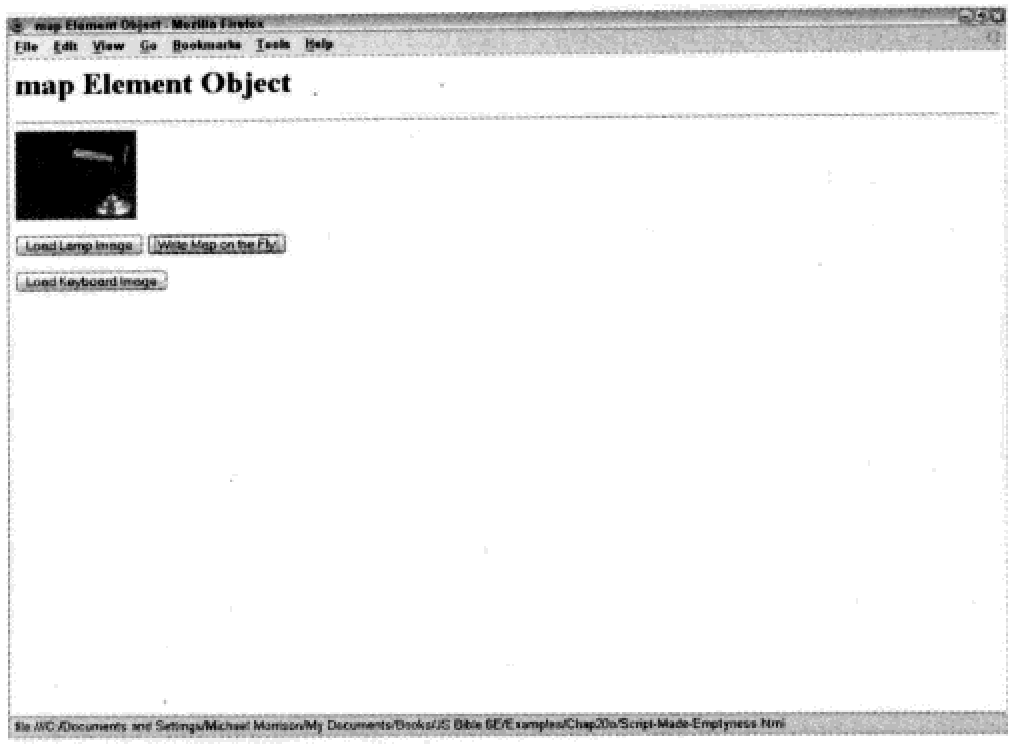


图 31-2　通过脚本为图像创建的客户端图像地图

程序清单 31-6　动态修改 area 元素

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>map Element Object</title>
      <script type="text/javascript">
      // generate area elements on-the-fly
      function makeAreas()
      {
         document.getElementById("myIMG").src = "desk3.gif";
         // build area element objects
         var area1 = document.createElement("area");
         area1.href = "Script-Made-Shade.html";
         area1.shape = "polygon";
         area1.coords = "52,28,108,35,119,29,119,8,63,0,52,28";
         var area2 = document.createElement("area");
         area2.href = "Script-Made-Base.html";
         area2.shape = "rect";
         area2.coords = "75,65,117,87";
         var area3 = document.createElement("area");
         area3.href = "Script-Made-Chain.html";
         area3.shape = "polygon";
         area3.coords = "68,51,73,51,69,32,68,51";
         var area4 = document.createElement("area");
         area4.href = "Script-Made-Emptyness.html";
         area4.shape = "rect";
         area4.coords = "0,0,50,120";
         // stuff new elements into MAP child nodes
         var mapObj = document.getElementById("lamp_map");
         while (mapObj.childNodes.length)
         {
```


更多资源请访问稀酷客(www.ckook.com)

```
            mapObj.removeChild(mapObj.firstChild);
         }
         mapObj.appendChild(area1);
         mapObj.appendChild(area2);
         mapObj.appendChild(area3);
         mapObj.appendChild(area4);
         // workaround NN6 display bug
         document.getElementById("myIMG").style.display = "inline";
      }
      function changeToKeyboard()
      {
         document.getElementById("myIMG").src = "cpu2.gif";
         document.getElementById("myIMG").useMap = "#keyboardMap";
      }
      function changeToLamp()
      {
         document.getElementById("myIMG").src = "desk3.gif";
         document.getElementById("myIMG").useMap = "#lampMap";
      }
      </script>
   </head>
   <body>
      <h1>map Element Object</h1>
      <hr />
      <img alt="image" id="myIMG" src="cpu2.gif" width="120" height="90"
           usemap="#keyboardMap" />
      <map id="keyboardMap" name="keyboardMap">
         <area href="AlpaKeys.htm" shape="rect" coords="0,0,26,42" />
         <area href="ArrowKeys.htm" shape="polygon"
               coords="48,89,57,77,69,82,77,70,89,78,84,89,48,89" />
         <area href="PageKeys.htm" shape="circle" coords="104,51,14" />
      </map>
      <map name="lampMap" id="lamp_map">
         <area href="Shade.htm" shape="polygon"
               coords="52,28,108,35,119,29,119,8,63,0,52,28" />
         <area href="Base.htm" shape="rect" coords="75,65,117,87" />
         <area href="Chain.htm" shape="polygon"
               coords="68,51,73,51,69,32,68,51" />
      </map>
      <form>
         <p><input type="button" value="Load Lamp Image"
                   onclick="changeToLamp()" />
            <input type="button" value="Write Map on-the-Fly"
                   onclick="makeAreas()" />
         </p>
         <p><input type="button" value="Load Keyboard Image"
                   onclick="changeToKeyboard()" />
         </p>
      </form>
   </body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

相关主题：area 元素对象。

name

值：字符串，读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

需要建立图像地图与图像之间的关系。为此应使用图像地图的 name 属性值：它匹配对应图像的 useMap 属性值。

相关主题：对应图像的 useMap 属性。

31.4　canvas 元素对象

关于 HTML 元素的属性、方法和事件处理程序，请参见第 26 章。

属　性	方　法	事件处理程序
fillStyle	arc()	
globalAlpha	arcTo()	
globalCompositeOperation	bezierCurveTo()	
lineCap	beginPath()	
lineJoin	clearRect()	
lineWidth	clip()	
miterLimit	closePath()	
shadowBlur	createLinearGradient()	
shadowColor	createPattern()	
shadowOffsetX	createRadialGradient()	
shadowOffsetY	drawImage()	
strokeStyle	fill()	
	fillRect()	
	getContext()	
	lineTo()	
	moveTo()	
	quadraticCurveTo()	
	rect()	
	restore()	
	rotate()	
	save()	

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	方　法	事件处理程序
	scale()	
	stroke()	
	strokeRect()	
	translate()	

31.4.1　语法

访问 canvas 元素对象的属性：

```
(W3C)        [window.]document.getElementById("canvasID").property |
                 method([parameters])
```

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

31.4.2　关于 canvas 对象

canvas 是个较新的结构，它允许在页面上创建一个矩形区域，并通过 JavaScript 以编程方式在此区域上绘图。与网页相比较，画布显示为一个图像，因为它占据一个矩形空间。不过与图像不同，画布的内容是使用 canvas 对象上定义的一系列方法以编程方式生成的。Safari 1.3 浏览器率先支持画布，后来 Mozilla 自从 1.8 版本开始也支持它，它对应于 Firefox 1.5。IE 的当前版本不支持这个对象，但第三方库允许在 IE 中使用它。

可以使用<canvas>标记在页面上创建和定位 canvas 对象，该对象只有两个独特的属性：width 和 height。创建画布后，与创建画布图形的工作就由 JavaScript 代码来完成。通常需要创建一个特殊的绘图函数，该函数在页面载入时在画布上绘图。绘图函数的工作是使用 canvas 对象的方法来显示画布图形。

程序清单 31-7 包含的轮廓页用于放置带边框的基本画布。注意 draw 函数用于接收显示画布图形的代码。

程序清单 31-7　轮廓画布

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>canvas Object</title>
      <script type="text/javascript">
      function draw()
      {
         // Draw some stuff
      }
      </script>
      <style type="text/css">
         canvas { border: 1px solid black; }
```


更多资源请访问稀酷客(www.ckook.com)

```
        </style>
    </head>
    <body onload="draw();">
        <h1>canvas Object</h1>
        <hr />
        <canvas width="350" height="250"></canvas>
    </body>
</html>
```

程序清单 31-8 包含一个更加有趣的画布示例，该示例以轮廓页面为基础，添加了一些实际的画布绘制代码。canvas 对象中的属性和方法可以完成一些令人称奇的操作，而此示例只触及了画布功能的冰山一角。

程序清单 31-8 包含简单图表的画布

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>canvas Object</title>
      <script type="text/javascript">
      function draw()
      {
         var canvas = document.getElementById('chart');
            if (canvas.getContext)
            {
               var context = canvas.getContext('2d');
               context.lineWidth = 20;
               // First bar
               context.strokeStyle = "red";
               context.beginPath();
               context.moveTo(20, 90);
               context.lineTo(20, 10);
               context.stroke();
               // Second bar
               context.strokeStyle = "green";
               context.beginPath();
               context.moveTo(50, 90);
               context.lineTo(50, 50);
               context.stroke();
               // Third bar
               context.strokeStyle = "yellow";
               context.beginPath();
               context.moveTo(80, 90);
               context.lineTo(80, 25);
               context.stroke();
               // Fourth bar
               context.strokeStyle = "blue";
               context.beginPath();
               context.moveTo(110, 90);
```


更多资源请访问稀酷客(www.ckook.com)

```
                context.lineTo(110, 75);
                context.stroke();
            }
        }
        </script>
        <style type="text/css">
           canvas { border: 1px solid black; }
        </style>
    </head>
    <body onload="draw();">
        <h1>canvas Object</h1>
        <hr />
        <canvas id="chart" width="130" height="100"></canvas>
    </body>
</html>
```

图 31-3 显示了此画布示例的结果，包括一个简单的条形图。注意，此条形图是用非常强大的矢量图形以编程方式显示的。

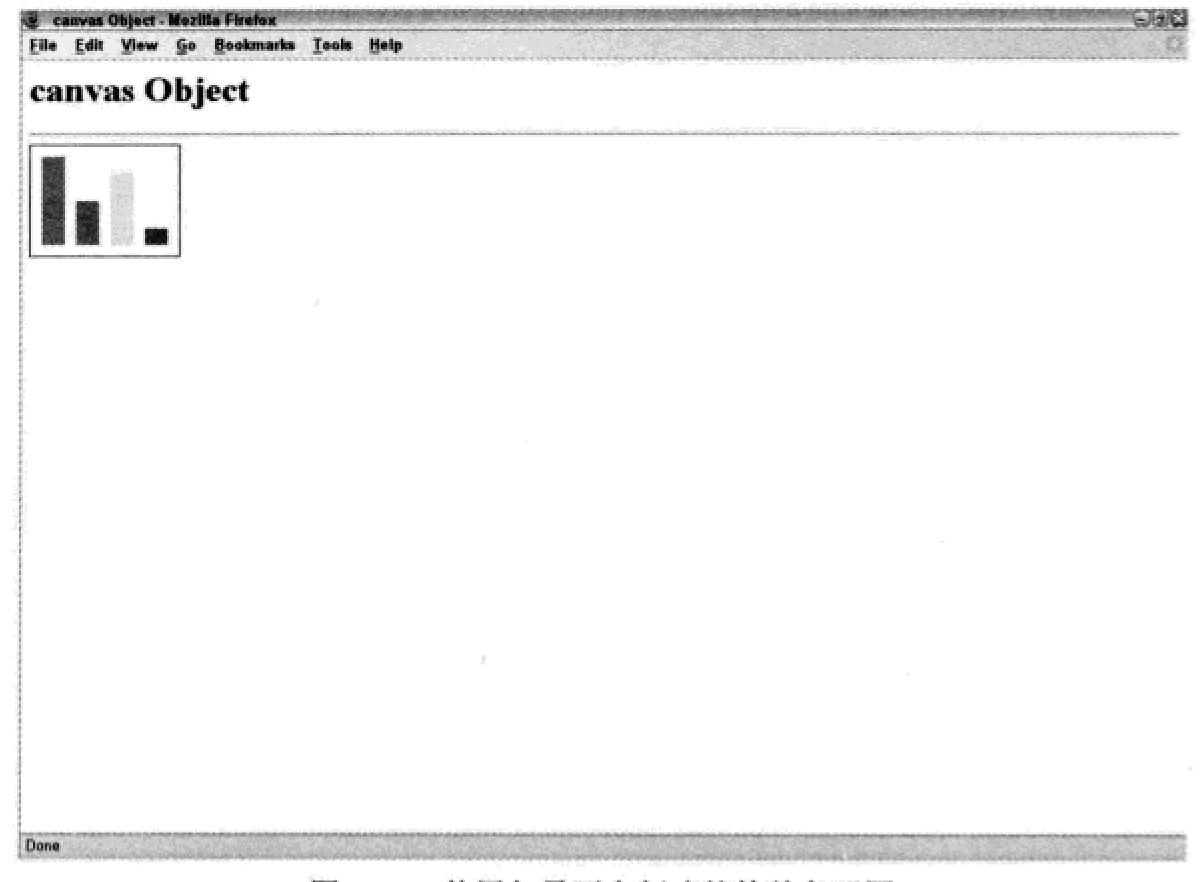


图 31-3　使用矢量画布创建的简单条形图

此示例显示了画布工作方式的更多细节。注意，要在画布上执行操作，首先必须获得一个上下文。实际上，图形操作是在画布上下文而不是画布元素上执行的，上下文的任务是提供一个虚拟表面，以进行绘图。在画布对象上调用 getContext()方法，并指定上下文类型，就可以获得上下文。目前，浏览器只支持 2d(二维)上下文。

有了上下文后，就可以相对于上下文来执行画布绘制操作。可任意设置笔画，填充颜色，创建路径和填充方式，以及执行矢量绘图的大多数常见图形操作。

在 http://developer.mozilla.org/en/docs/Canvas_tutorial 上可以找到介绍用画布元素进行绘图的完整教程和示例。

更多资源请访问稀酷客(www.ckook.com)

31.4.3 属性

fillstyle

值：字符串， 读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

fillstyle 属性可以设置在填充画布区域时使用的颜色或图案。填充形状时可以创建渐变颜色和其他有趣的图案，但 fillStyle 属性的最基本用法是设置一种 HTML 样式的颜色(#RRGGBB)，来创建纯色填充。所有遵循 fillstyle 设置的填充操作都将使用新的填充样式。

示例

要设置填充颜色，只需将一种 HTML 样式颜色赋给 fillstyle 属性：

```
context.fillStyle = "#FF00FF";
```

相关主题：strokeStyle 属性。

globalAlpha

值：浮点数， 读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

globalAlpha 是一个浮点属性，它确定在画布上绘制的内容的透明度(或不透明度，取决于用户如何看待它)。此属性的值在 0.0(完全透明)和 1.0(完全不透明)之间，默认设置为 1.0，这表示所有画布最初都是不透明的。

示例

要将画布的透明度设置为 50%，可将 globalAlpha 属性设为 0.5：

```
context.globalAlpha = 0.5;
```

相关主题：无。

globalCompositeOperation

值：字符串， 读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

globalCompositeOperation 属性确定画布相对于网页上的背景内容如何显示。这个强大的属性可极大地影响画布内容相对于下层网页内容的显示方式。默认设置为 source -over，这意味着显示画布的不透明区域，而不显示透明区域。其他常见设置包括 copy、lighter 和 darker。

示例

如果希望画布始终完全覆盖背景网页，而不管它是否包含透明区域，应将 globalComposite Operation 属性设置为 copy：

```
context.globalCompositeOperation = "copy";
```

相关主题：无。

更多资源请访问稀酷客(www.ckook.com)

lineCap，lineJoin，lineWidth

值：字符串、浮点数(lineWidth)，　　读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些属性都可以影响在画布上绘制直线的方式。lineCap 属性指定线条端点的外观(butt、round 或 square)。lineJoin 属性与 lineCap 类似，它决定线条如何相互连接(bevel、miter 或 round)。默认情况下，线条结束时没有特殊的端点(butt)，也不使用特殊的连接图形(miter)，而是直接连接。

lineWidth 属性指定线条的宽度，它是画布坐标空间中一个大于 0 的整数值。在绘制线条时，其宽度显示在线条坐标的中央。

示例

在绘制画布图形时，经常需要更改 lineWidth 属性，以获得不同的效果。下面是一个设置线宽(在这里是 10)的示例：

```
context.lineWidth = 10;
```

相关主题：miterLimit，strokeStyle 属性。

miterLimit

值：浮点数，　　读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

miterLimit 属性是一个浮点值，它更准确地确定线条如何连接到一起。miterLimit 属性和 lineJoin 属性一起使用，可以整齐连贯地连接路径中的线条。

相关主题：lineJoin 属性。

shadowBlur，shadowColor，shadowOffsetX，shadowOffsetY

值：整数，字符串(shadowColor)，　　读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些属性结合使用，可在画布内容的四周建立阴影。shadowBlur 属性指定阴影本身的宽度，shadowColor 将阴影颜色设置为 HTML 样式的 RGB 值(#RRGGBB)。最后，shadowOffsetx 和 shadowOffsety 属性精确指定阴影偏离图形多远。shadowBlur、shadow0ffsetX 和 shadowOffsetY 属性值都以画布坐标空间的单位来表示。

示例

以下代码创建了一个阴影，其宽度为 5 个单位，其颜色为浅灰(#BBBBBB)，在 X 和 Y 方向上都偏移 3 个单位：

```
context.shadowBlur = 5;
context.shadowColor = "#BBBBBB";
context.shadowOffsetX = 3;
context.shadowOffsetY = 3;
```

相关主题：无。

更多资源请访问稀酷客(www.ckook.com)

strokeStyle

值：字符串，　　　　　　　　　　　　　　　　　　　　　　　　　　　读/写

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

strokeStyle 属性控制用于在画布上绘图的笔划样式。使用更高级的画布功能可将笔划样式设置为渐变色或图案，但更简单的方式是只用 HTML 样式的颜色值(#RRGGBB)设置纯色笔划。默认笔划样式是纯黑笔划。

示例

要将笔划样式改为纯绿画笔，可将 strokeStyle 属性设置为绿色：

```
context.strokeStyle = "#00FF00";
```

相关主题：fillstyle 属性。

31.4.4　方法

arc(*x*, *y*, *radius*, *startAngle*, *endAngle*, *clockwise*)，arcTo(*x1*, *y1*, *x2*, *y2*, *radius*)，bezierCurveTo(*cp1x*, *cp1y*, *cp2x*, *cp2y*, *x*, *y*)，quadraticCurveTo(*cpx*, *cpy*, *x*, *y*)

返回值：无。

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些方法都以某种方式绘制曲线。如果此前曾绘制过矢量图形，你可能已经知道弧、Bezier 曲线和二次曲线之间的区别。

arc()方法根据中心点、半径以及起始角度和结束角度来绘制曲线，此方法其实是绘制从一个角度到另一个角度的圆弧。角度以弧度表示，而不是度数，因此需要将度数转化为弧度：

```
var radians = (Math.PI / 180) * degrees;
```

arc()方法的最后一个参数是 Boolean 值，它确定按顺时针方向(true)还是逆时针方向(false)绘制弧。

arcTo()方法根据圆的切线沿曲线绘制弧。直到版本 1.8.1，此方法才在 Mozilla 中实现。

最后，bezierCurveTo()和 quadraticCurveTo()方法分别根据与 Bezier 和二次曲线相关的参数绘制曲线，这超出了本书的讨论范围。要了解 Bezier 和二次曲线的更多内容，请访问 en.wikipedia.org/wiki/Bezier_curve。

beginPath()，closePath()

返回值：无。

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这两个方法都用于管理路径。调用 beginPath()方法会开始一个新路径，然后可在其中添加形状。最后，调用 closePath()来闭合路径。closePath()方法的唯一作用是关闭仍然开放的子路径，这意味着将一个开放的形状连接回其起点。如果所创建的路径是闭合的，就没必要调用 closePath()方法。程序清单 31-8 中的示例不需要 closePath()方法，因为条形图不需要闭合。

更多资源请访问稀酷客(www.ckook.com)

clip()

返回值： 无。

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

clip()方法根据当前路径和已有的裁剪路径，来重新计算裁剪路径，随后的绘制操作使用新计算的裁剪路径。

createLinearGradient(*x1*, *y1*, *x2*, *y2*)，createRadialGradient(*x1*, *y1*, *radius*1, *x2*, *y2*, *radius*2)，createPattern(*image*, *repetition*)

返回值： Gradient 对象引用，pattern 对象引用(createPattern())。

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些方法用于为填充操作创建特殊图案和渐变色。线性渐变指定为两个坐标之间的平滑颜色过渡，而径向渐变根据半径不同的两个同心圆来指定。在从 createLinearGradient()和 createRadialGradient()方法返回的 gradient 对象上调用 addColorStop()方法，来设置实际的渐变颜色范围。addColorStop()方法仅有的两个参数是一个浮点偏移值和一个字符串颜色值。

createPattern ( )方法可根据图像创建填充图案。该方法需要的参数是一个图像对象和一个重复选项，重复选项决定在填充区域时如何平铺图像，其值是 repeat、repeat-x, repeat-y 或 no-repeat。

drawImage(*image*, *x*, *y*), drawImage(*image*, *x*, *y*, *width*, *height*), drawImage(*image*, *srcX*, *srcY*, *srcWidth*, *srcHeight*, *destX*, *destY*, *destWidth*, *destHeight*)

返回值： 无。

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些 drawImage()方法都在上下文中绘制图像，它们之间的区别与绘制时是否以及如何缩放图像有关。第一个方法在某个坐标点上绘制图像，但不缩放图像；第二个方法将图像缩放为指定的目标宽度和高度，最后，第三个方法允许用缩放的宽度和高度将图像的一部分绘制到目标位置上。

fill()，fillRect(*x*, *y*, *width*, *height*)，clearRect(*x*, *y*, *width*, *height*)

返回值： 无。

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这些方法用于填充和清除区域。fill()方法填充当前路径内的区域，而 fillRect()方法则填充指定的矩形，这独立于当前路径。clearRect()方法用于清除(删除)矩形。

getContext(*contextID*)

返回值： Context 对象引用。

兼容性： WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

getContext( )方法在画布元素对象(而不是上下文)上操作，它用于为后面的图形操作获取上下文。在画布上绘图之前，必须调用此方法来获取上下文，因为所有画布绘制方法实际上都是相对于上下文(而不是画布)来调用的。对于标准的二维画布上下文，2d 是该方法的唯一参数。

更多资源请访问稀酷客(www.ckook.com)

lineTo(*x*, *y*)，moveTo(*x*, *y*)

返回值：无。

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

这两个方法用于绘制线条和调整笔划位置。moveTo()方法仅移动当前笔划位置，而不向路径添加任何内容；另一方面，lineTo()方法绘制一条从当前笔划位置到指定点的直线。注意，用 lineTo()方法绘制直线只是给当前路径添加一条直线，在调用 stroke()方法绘制实际的路径之前，该直线不会真正显示在画布上。

rect(*x*, *y*, *width*, *height*)

返回值：无。

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

rect()方法给当前路径添加一个矩形。与其他绘制方法类似，矩形只添加到当前路径上，在使用 stroke()方法显示它之前，矩形是不可见的。

restore()，save()

返回值：无

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

保存并还原图形上下文的状态，表示可以进行更改，然后返回到所需状态。上下文状态包括剪裁区域、线条宽度和填充颜色等信息。调用 save()和 restore()方法可以保存和还原上下文状态。

rotate(*angle*)，scale(*x*, *y*)，translate(*x*, *y*)

返回值：无

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

在画布上执行的所有绘图操作都以相对于画布坐标系统的单位来表示。这三个方法允许通过旋转、缩放或变换其原点来更改画布的坐标系统。在涉及角度的绘制操作中，旋转坐标系统会影响角度的表示，缩放坐标系统影响坐标系统中单位的相对大小，最后，变换坐标系统会改变原点位置，影响正值和负值在绘图表面上的相交位置。

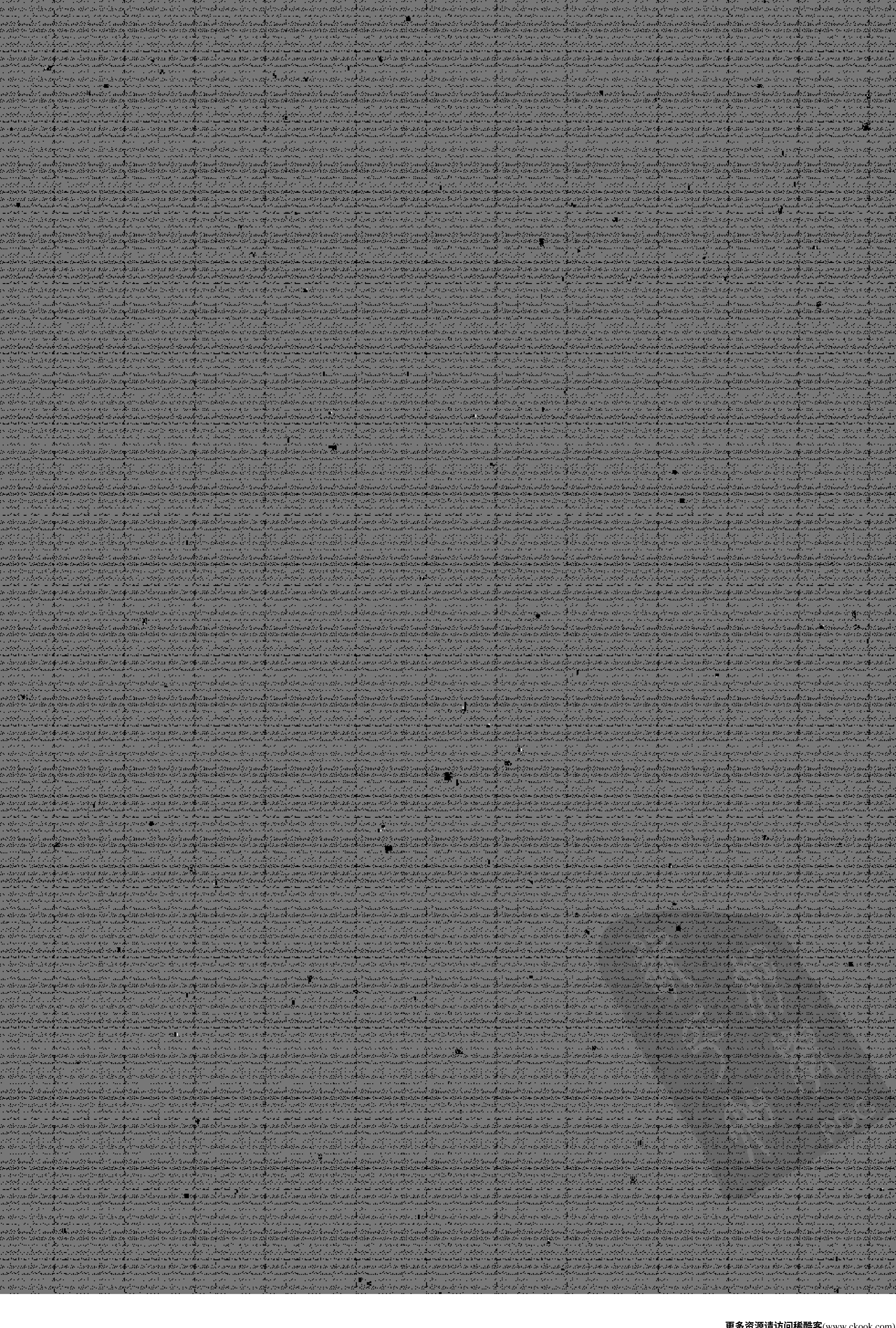
stroke()，strokeRect(x, y, width, height)

返回值：字符串

兼容性：WinIE-，MacIE-，NN-，Moz1.8+，Safari1.3+，Opera+，Chrome+

画布上的大多数绘图操作都影响当前路径，当前路径可以想像为已提交到内存但未放到纸上的图。可以调用 Stroke()方法将路径显示到画布上。如果想在画布上绘制矩形，而不涉及当前路径，应调用 strokeRect()方法。

更多资源请访问稀酷客(www.ckook.com)



更多资源请访问稀酷客(www.ckook.com)

第32章 event 对象

本章包含哪些内容？

- event 对象的生存周期
- 不同浏览器版本所支持的事件
- 从事件中获取信息

在版本 4 浏览器之前，用户和系统动作(也就是事件)主要通过事件处理程序捕获，这些事件处理程序在 HTML 标记中定义为特性。例如，用户单击按钮时，click 事件会触发标记中的 onclick 事件处理程序，该处理程序调用一个函数或者执行一些内联 JavaScript 脚本。即使如此，事件本身也是“哑的”——要么发生，要么不发生。事件发生的区域(鼠标单击时指针所在的屏幕坐标)和其他的相关事件信息(例如是否同时按下键盘上的修改键)并非事件的一部分，直到版本 4 才是。

尽管第 4 版浏览器使用完全向后兼容的旧事件处理机制，但它有了第一个事件模型，可将事件转化为一级对象，在事件发生时，其属性自动传送许多相关信息。这些属性可以由脚本使用，使页面更加智能地响应用户对页面和其上元素的操作。

第 4 版浏览器的事件模型的另一个新特点是“事件传播”。只要多个对象需要共用一个事件处理程序，就可以由层次结构中当前元素的上层对象来处理事件。这样处理的事件就会带有目标信息和其他有用的信息，使事件处理函数智能地处理事件，而不必调用事件处理程序，来传递与目标相关的所有信息。

不过，这种新功能由于对象模型的不兼容而有所减弱。事件对象模型分为两种：

- IE4+模型
- 在 NN6+/Moz/和基于 Gecko、WebKit 和 Presto 的浏览器中实现的、由 W3C DOM Level 2 采用的模型。

第 26 章在概述对象模型时，介绍了模型之间的区别。本章将阐述包含各种信息的真正 event 对象，并在可能的地方介绍了跨浏览器的相关问题。

32.1 事件

图形用户界面比以前的命令行界面更难编程。在命令行或菜单驱动的系统中，用户能执行

更多资源请访问稀酷客(www.ckook.com)

的动作类型是有限的。世界被模式化了，主要为了便于程序员在严格的程序结构中引导用户。

这一切在图形用户界面中完全改变了，如 Windows、MacOS、XWindow 系统和从 Xerox Star 系统的早期版本派生出的其他图形系统。程序员的挑战是在这个领域中，好的用户界面允许用户完成各种不同的动作，如移动鼠标、单击按钮、按键、选择文本和下拉菜单等。为了实现这些功能，程序(或操作系统)就必须识别可能来自所有输入端口(鼠标、键盘或网络连接)的动作。

在操作系统级别，常见的处理方法是查询各类事件，确定它是来自用户的动作还是计算机产生的行为。然后操作系统和程序就会决定如何处理这类事件。另外，这些事件必须有许多标记，以便程序确定事件的类型及其在屏幕上的位置。

32.1.1　事件的内容和事件发生时间

尽管在三种对象模型中，event 对象的引用方式不同，但有一点是相同的：一旦发生事件，就会建立 event 对象。例如，如果单击按钮，就在浏览器的内存中创建相应的 event 对象。对象建立后，浏览器就为其属性赋值，这些属性反映了该事件的大量特征。对于 click 事件，该信息包括单击处的坐标和触发事件的鼠标键。为便于处理，浏览器会执行一些快速计算，来确定 click 事件的坐标是否匹配屏幕上按钮元素的矩形区域。所以 event 对象的一个属性用于引用被单击的“屏幕元素”。

大多数 event 对象属性是只读的(在有些事件模型中，所有属性都是只读的)，因为 event 对象是事件动作的快照。如果允许事件模型更改事件属性，就可能执行有用的动作，也可能执行不友好的动作。例如，用户在文本框中输入数据时，如果按下正确的键，而文本框显示的却是完全不同的字符，会令人非常沮丧。另一方面，在有些情况下，确保文本框中输入的所有内容都转换为大写形式是非常有用的。每种事件模型对于这种情形都有自己的处理方法，例如 IE4+ 事件模型允许脚本改变键盘字符事件，而 W3C DOM 事件模型不允许。

最重要的是，只要脚本处理事件，event 对象就存在。事件可以触发事件处理程序，而事件处理程序通常是一个函数，当然它也可以调用其他函数。只要响应事件处理程序的语句仍在执行，event 对象及其所有属性就仍然存在，并可用于脚本。但最后的脚本语句运行完毕后，event 对象就变为空对象。

event 对象的生存期如此之短，原因是每次仅能有一个 event 对象，换言之，不管事件触发得多快，不管事件处理函数多么复杂，这些函数都是连续执行的(对富有经验的程序员来说，只有一个执行线程)。操作系统将事件放入缓冲区，不让它们混成一团。缓冲区很少变满或不记录事件，事件处理程序按照事件发生的顺序执行。

32.1.2　静态 event 对象

下面讨论 event 对象(小写字母 e)，它是事件的实例，其属性指定了特定的事件动作。在 W3C DOM 事件模型中，还有一个静态 Event 对象(大写字母 E)，它包含额外的子类别。本章后面还会介绍这些子类别，这里主要比较 event 和 Event 对象，前者是一个生存期很短的对象，包含指定事件动作的细节；后者主要用于存储脚本可用的、与事件相关的常量值。在任何窗口或框架中，静态 Event 对象都可用于脚本。在 NN6+/Moz/和基于 Gecko、WebKit 和 Presto 的浏览器中，如果想查看 Event 对象属性列表，可在 The Evaluator(第 4 章)底部的文本框中输入

更多资源请访问稀酷客(www.ckook.com)

Event(也可以查看 NN6+/Moz 中的 KeyEvent 对象)。

静态 Event 对象还可以复制 event 对象，事件动作创建的事件对象可以使用(继承)静态 Event 对象的一些属性和方法。这些关系在 W3C DOM 事件模型中非常重要，W3C DOM 事件模型以 DOM 面向对象模型为基础，实现了事件模型。

32.2　事件传播

在第 4 版浏览器之前的版本中，事件在对象上激活。如果为事件和对象定义了事件处理程序，就执行它；如果没有事件处理程序，事件就会消失。但新浏览器可使事件在整个文档对象模型中传播。现有两种传播模型，分别应用于目前使用的一种事件模型：IE4+和在 NN6+/Moz/和基于 Gecko、WebKit 和 Presto 的浏览器中实现的 W3C DOM。NN4 还有一种专用的事件模型，它是 NN4 让位于现代浏览器之前的第三种模型。NN4 事件模型有历史关联性，因为它有助于理解后两种事件模型。NN4 和 IE4+传播模型在概念上完全相反：NN4 事件往内向目标传播，而 IE 事件从目标开始向外传播。但 W3C DOM 模型实现了这两种模型，但使用了全新的语法，以免与旧模型冲突。

所有三种模型都认为，每个事件都有一个目标。对于用户动作而言，这是非常明显的。如果单击按钮或在文本框中输入内容，按钮就是鼠标相关事件的目标；文本框是键盘事件的目标。系统生成的事件则不那么明显，例如页面载入完毕后触发的 onload 事件。在所有事件模型中，这个事件激活 window 对象。事件传播模型的区别在于事件如何到达目标，以及与目标相关的事件处理程序执行完毕后的变化。

32.2.1　仅用于 NN4 的事件传播

尽管 NN4 已让位于现代浏览器，但其传播模型的一些概念仍用于 W3C DOM 事件传播模型，NN4 模型的名称是事件捕获。

在 NN4 中，所有事件都从文档对象层次结构的顶端(从 window 对象开始)向下传播到目标对象，例如单击表单中的一个按钮时，click 事件要先经过 window 和 document(有时是 layer)对象，最后到达按钮。传播是即时的，所以不会产生额外的性能开销。

经过 window、document 和 layer 对象的事件是一个完整的 event 对象，它带有与事件动作相关的所有属性。所以，如果在窗口上处理事件，event 对象的一个属性就是目标对象的一个引用，这样窗口上的事件处理程序脚本可以找到有关信息，如按钮的名称和其父窗体的引用。

默认情况下关闭事件捕获功能。为使 window、document 和 layer 对象处理所传送的单击对象，需要启用 window、document 和/或 layer 对象的事件捕获功能。

1. 启用 NN4 事件捕获功能

上述三个对象 window、document 和 layer 都有 captureEvents()方法。使用这个方法可以激活任何对象上的事件捕获功能，这个方法需要一个或多个参数，其类型是对象应捕获的事件类型(由 Event 对象常量提供)，但不处理其他类型的参数。例如，如果需要 window 对象捕获所有 keyPress 事件，就应在载入页面时执行的脚本中包括如下语句：

更多资源请访问稀酷客(www.ckook.com)

```
window.captureEvents(Event.KEYPRESS);
```

也可在预期目标中定义事件处理程序，即使它们为空(如 onkeypress=" ")，也可以帮助 NN4 首先生成事件。如果要让窗口捕获多种类型的事件，可将事件类型常量串在一起，中间用管道符号隔开：

```
window.captureEvents(Event.KEYPRESS | Event.CLICK);
```

现在就必须在窗口上为每种事件类型指定一个操作，更确切地讲，是为事件定义要执行的函数。设置 window 对象的事件处理属性，就可以为事件处理程序指定函数引用：

```
window.onkeypress = processKeyEvent;
window.onclick = processClickEvent;
```

以后，用户再单击按钮或在窗口的一个域上输入内容时，事件就会由窗口上的事件处理函数来处理。

2. 关闭事件捕获功能

只要为文档中的某个事件类型激活了事件捕获功能，这个捕获功能就一直有效，直到卸载页面或者禁用捕获功能为止。使用 window、document 或 layer 的 releaseEvents()方法可为每个事件关闭事件捕获功能，releaseEvents()方法的参数与 captureEvents()方法相同——Event 对象类型常量。

释放某类事件，仅意味着即使定义了更高层对象的事件处理程序，事件也直接传递到其预设目标，而不是停留在其他地方进行处理。因为可根据为 releaseEvents()方法设置的参数释放单个事件类型，所以不会影响捕获的其他事件。

3. 向目标传递事件

如果在 NN4 中捕获了特定的事件类型，在事件到达指定目标之前，脚本会对事件执行有限的操作。例如，如果用户在按下 Shift 键的同时单击某一元素，就执行特别的操作。此时，文档中处理事件的函数会检查事件的 modifiers 属性，来确定事件发生时是否按下了 Shift 键。如果没有按下 Shift 键，事件就可以到达用户单击的元素。

为使事件通过对象层次结构传递到目标，需要使用 routeEvent()方法，将当前函数处理的 event 对象作为参数传递给它。routeEvent()方法不能保证事件到达既定的目的地，因为中间可能会有别的对象启用这个事件类型的事件捕获功能，并截取事件。这个对象也可以用自己的 routeEvent()方法来传递事件。

某些情况下，脚本需要知道 routeEvent()方法传递的事件是否激活了有返回值的函数。如果事件必须返回 true 或 false 值，以告诉对象是否应该继续其默认行为(例如，在事件处理程序返回 true 或 false 后，链接是激活其 href 特性 URL，还是取消)，则知道这一点尤为重要。routeEvent()方法的动作调用一个函数时，目标函数的返回值会传递回 routeEvent()方法，接着，该值就可以返回给最初捕获事件的对象。

更多资源请访问稀酷客(www.ckook.com)

4. 事件传递

最后一种情形是，高层对象会捕获一个事件，并将事件引导到层次结构中其他地方的某个对象。例如，如果 click 事件的 modifiers 属性表明按下了 Alt 键，文档级别的事件处理函数就把这些事件传递给 Help 按钮对象，其 onclick 事件处理程序会显示一个帮助面板(也可能显示另一个隐藏的层)。

还可以使用 handleEvent()方法将事件重定向到任何对象中。这个方法与本章中的其他方法不同，因为这个方法的对象引用是处理事件的对象引用(将事件对象作为参数传递给其他方法)。只要目标对象为事件定义了事件处理程序，它处理事件的方式就类似于直接从系统获得事件(即使事件对象的目标属性可能是另一个对象)。

32.2.2　IE4+事件传播

IE4+的事件传播模型称为事件冒泡，因为事件从目标对象开始，沿着 HTML 元素包含层次结构向上“冒泡”。区别以前的文档对象层次(在 NN4 事件捕获模型中采用)和更现代的 HTML 元素包含结构(在 W3C DOM 中采用)是非常重要的。

演示事件冒泡的最佳方式(事件冒泡默认情况下启用)是在一个简单的文档中填充大量事件处理程序，观察激活了哪个事件，以及事件激活的顺序。程序清单 32-1 为表单中的按钮、表单本身以及窗口层次结构中所有其他的元素和对象定义了 onclick 事件处理程序。

程序清单 32-1　事件冒泡示例

```
<!DOCTYPE html>
<html onclick="alert('Event is now at the HTML element. ')">
  <head>
    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <title>Event Bubbles</title>
    <script type="text/javascript">
    function init()
    {
      window.onclick = winEvent
      document.onclick = docEvent;
      document.body.onclick = docBodEvent;
    }
    function winEvent()
    {
      alert("Event is now at the window object level.");
    }
    function docEvent()
    {
      alert("Event is now at the document object level.");
    }
    function docBodEvent()
    {
      alert("Event is now at the BODY element.");
    }
```


更多资源请访问稀酷客(www.ckook.com)

```
    </script>
  </head>
  <body onload="init()">
    <h1>Event Bubbles</h1>
    <hr />
    <form onclick="alert('Event is now at the FORM element. ')">
      <input type="button" value="Button 'main1'" name="main1"
             onclick="alert('Event started at Button: ' + this.name)" />
    </form>
  </body>
</html>
```

该程序清单可在 IE4+、NN6+/Moz、Safari、Opera 或 Chrome 中运行，因为 W3C DOM 浏览器可以观察事件冒泡。WinIE4+、MacIE4+和 W3C DOM 浏览器在事件传播方面是有区别的。单击程序清单 32-1 中的按钮后，事件先在其目标(按钮)上激活；接着事件沿着 HTML 包含层次结构向上冒泡，先在包含按钮的 form 元素上激活，然后在 body 元素上激活。但在 body 元素之后，这些浏览器的区别就显现出来了。程序清单 32-1 中的事件处理程序是为表 32-1 中的对象定义的。表 32-1 中还显示了 3 种浏览器中 click 事件冒泡的目标对象。

表 32-1　程序清单 32-1 中的事件冒泡

事件处理程序位置	WinIE4+/Opera	MacIE4+	NN6+/Moz/Safari/Chrome
button	是	是	是
form	是	是	是
body	是	是	是
HTML	是	否	是
document	是	是	是
window	否	否	是

尽管存在表 32-1 所显示的差异，但事件都沿着类似于 HTML 包含层次结构向上冒泡。在该表中所示的所有浏览器类型中，document 对象拥有最大的全局作用域。

1. 禁止 IE 事件冒泡

冒泡默认为打开，但有时也需要禁止事件沿着层次结构冒泡。例如，如果 document 层有一个事件处理程序，用来处理一系列相关按钮的 click 事件，则接收 click 事件的任何其他对象允许这些事件向上冒泡到 document 层，除非取消冒泡。此时，允许事件向上冒泡就会与文档层的事件处理程序冲突。

IE 中的每个 event 对象都有 cancelBubble 属性，其默认值是 false，表示事件向下一个具有该类事件处理程序的外层容器冒泡。但如果在事件处理程序的执行过程中，该属性值设置为 true，则事件处理程序处理完毕后，不向高层冒泡。因此，要防止事件从当前事件处理程序向上冒泡，应在处理函数中包含如下语句：

```
event.cancelBubble = true;
```

为证明这一点，可修改程序清单 32-1，取消任意层的冒泡功能。例如，改变 form 元素中

更多资源请访问稀酷客(www.ckook.com)

的事件处理程序，添加取消冒泡的语句，IE 事件就仅在 form 中处理(尽管 WebKit 引擎和 Mizilla/Gecko 引擎的最新版本支持一些 IE 语法，但其语法与 NN6+/Moz 浏览器不同，后面会加以讨论)：

```
<form onclick="alert('Event is now at the form element. ');
   event.cancelBubble=true">
```

2. 禁止 IE 事件执行默认动作

以前，事件几乎总是利用标记特性绑定到元素上，阻止事件执行默认动作的技术是确保事件处理程序返回 false。例如，如果客户端表单验证失败，form 元素的 onsubmit 事件处理程序就可以阻止表单执行提交操作。

为了强化这种能力，特别是通过其他手段(例如对象元素属性)禁止事件执行默认操作，IE 的 event 对象包括一个 returnValue 属性。在事件处理函数中，给该属性赋予 false 将会阻止元素执行事件的默认动作：

```
event.returnValue = false;
```

在 IE 中，这种阻止执行默认动作的方式通常比以前的 return false 技术更有效。

3. 重定向事件

从 IE5.5 开始，可以将事件重定向到另一个元素，但有一些限制。可以使用所有 HTML 元素对象都有的 fireEvent()方法(见第 26 章)来实现这种机制。这个方法重定向事件时并非激活一个新事件，但可将一个旧 event 对象的引用作为 fireEvent()方法的第二个可选参数，通过新事件传递原始 event 对象的许多属性。

但这个技术的最大缺陷是对目标元素的引用会在传递给新事件的过程中丢失，因为 fireEvent()调用的目标对象的引用会覆盖旧事件的 srcElement 属性。例如，下面是 form 元素中按钮的 onclick 事件处理程序：

```
function buttonEvent() {
   event.cancelBubble = true;
   document.body.fireEvent("onclick", event);
}
```

取消事件冒泡，事件就不会向上传播到包含它的 form 元素；相反，事件会明确地重定向到 body 元素，并将当前的 event 对象传递为第二个参数。body 元素的事件处理函数运行时，其 event 对象就包含了原始事件的信息，如单击使用的鼠标键和坐标，但 event.srcElement 属性指向 document.body 对象。由于事件从 body 元素向上冒泡，因此 srcElement 属性继续指向 document.body 对象，IE5.5+的使用方式请参见程序清单 32-2。

程序清单 32-2　在 IE5.5+中取消和重定向事件

```
<!DOCTYPE html>
<html onclick="revealEvent('HTML', event)">
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Event Cancelling & Redirecting</title>
```


更多资源请访问稀酷客(www.ckook.com)

```
    <script type="text/javascript">
       // display alert with event object info
       function revealEvent(elem, evt)
       {
          var msg = "Event (from " + evt.srcElement.tagName + " at ";
          msg += event.clientX + "," + event.clientY + ") is now at the ";
          msg += elem + " element.";
          alert(msg);
       }
       function init()
       {
          document.onclick = docEvent;
          document.body.onclick = docBodEvent;
       }
       function docEvent()
       {
          revealEvent("document", event);
       }
       function docBodEvent()
       {
          revealEvent("BODY", event);
       }
       function buttonEvent(form)
       {
          revealEvent("BUTTON", event);
          // cancel if checked (IE4+)
          event.cancelBubble = form.bubbleCancelState.checked;
          // redirect if checked (IE5.5+)
             if (form.redirect.checked)
             {
                document.body.fireEvent("onclick", event);
             }
       }
    </script>
 </head>
 <body onload="init()">
    <h1>Event Cancelling &amp; Redirecting</h1>
    <hr />
    <form onclick="revealEvent('FORM', event)">
       <p><button name="main1" onclick="buttonEvent(this.form)">Button
          'main1'</button>
       </p>
       <p><input type="checkbox" name="bubbleCancelState"
                 onclick="event.cancelBubble=true" />Cancel Bubbling at BUTTON
          <br />
          <input type="checkbox" name="redirect"
                 onclick="event.cancelBubble=true" />Redirect Event to BODY
       </p>
    </form>
 </body>
```


更多资源请访问稀酷客(www.ckook.com)

```
</html>
```

程序清单 32-2 是程序清单 32-1 的修改版本，主要增加了各个层的增强事件处理程序，以显示作为事件 srcElement 属性的事件标记名和 click 事件的坐标。两个复选框都未选中时，事件从按钮向上冒泡，button 元素是整个冒泡层次结构中的原始目标。如果选中 Cancel Bubbling 复选框，事件就只能到达 button 元素，因为 button 元素关闭了事件冒泡功能。如果选中 Redirect Event to body 复选框，原始事件就在 button 层上取消，但在 body 元素中又激活一个新事件。但注意，将旧 event 对象传递为第二个参数，旧事件的单击位置属性就会应用于 body 中的新事件，该事件会继续从 body 向上冒泡。

另外，如果取消选中 Cancel Bubbling 复选框，而保留选中 Redirect Event 复选框，在 button 事件处理程序的末尾就可以看到重定向操作，此时还会出现一些特殊情况。在重定向的事件向上冒泡时，浏览器会保存原始事件。在事件处理分支结束后，原始冒泡传播过程在 form 层继续进行。注意，event 对象的目标仍是 button 元素，其他属性也保持不变。这意味着一段时间内，在浏览器的内存中有两个 event 对象，但一次只有一个是激活的，在重定向事件的传播过程中，window.event 仅表示激活的 event 对象。

4. 应用事件捕获

WinIE5+还提供了一种事件捕获功能，它重写了所有其他事件传播功能，主要用于临时捕获鼠标事件，它不是通过事件对象，而是通过所有 HTML 元素对象都有的 setCapture()和 releaseCapture()方法(参见第 26 章)来控制。

在处于捕获模式时，所有鼠标事件都定向到调用 setCapture()方法的元素对象上，无论事件的实际目标是什么都是如此。这就便于执行某些活动，如拖动元素，若鼠标事件在预期目标外触发(例如，鼠标指针拖动得过快，以至于动画无法跟踪)，事件就可以到达目标。不再需要拖动模式时，就调用 releaseCapture()方法，让鼠标事件正常传播。

32.2.3 W3C 事件传播

为了既支持 NN4 的事件捕获，又支持 IE5 的事件冒泡，W3C DOM 工作组建立了一个利用这两种传播系统的事件模型。W3C DOM 传播模型使用了新的语法，以免与较旧浏览器冲突，其捕获工作方式像 NN4，冒泡工作方式像 IE4+。换句话说，事件默认为向上冒泡，但如果需要，也可以启用事件捕获功能。这样，事件首先顺着元素包含层次结构滴流到目标，然后沿着相反的路径向上冒泡。

与 IE4+一样，默认情况下启用事件冒泡功能。要启用捕获功能，必须在某个层次更高的容器上将 W3C DOM 事件监听器应用于对象。任何可见的 HTML 元素或节点都使用 addEventListener()方法(参见第 26 章)。addEventListener()方法的一个参数决定，事件在冒泡或捕获时，是否应触发事件监听器函数。

程序清单 32-3 是一个用于 NN6+/Moz/W3C 的简化示例，它演示了用于按钮的 click 事件既可以捕获，又可以冒泡。大多数事件处理函数都在 init()函数中指定。利用程序清单 32-1 中的代码，把事件处理程序分配给 window、document 和 body 对象，作为属性。这些事件处理程序自动用作冒泡类型的事件监听器。接着给两个对象(document 和 form)提供捕获类型的 click 事

更多资源请访问稀酷客(www.ckook.com)

件监听器，document 对象的事件监听器调用与冒泡类型事件处理程序相同的函数(警告文本包括一些星号，说明这是在事件的捕获和冒泡阶段中显示的同一警告)。不过，对于 form 对象，捕获类型的事件监听器指向一个函数，而其冒泡类型的监听器指向另一个函数。换句话说，事件滴流到目标时，form 对象调用一个函数，而事件开始向上冒泡时，它调用另一个函数。许多事件处理函数实时读取 event 对象的 eventPhase 属性，以确定在调用事件处理程序时，事件传播的哪个阶段有效(不过在 document 对象的事件捕获过程中，一个明显的程序错误报告了不正确的阶段)。

程序清单 32-3　W3C 事件的捕获和冒泡

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>W3C DOM Event Propagation</title>
      <script type="text/javascript">
         function init()
         {
            // using old syntax to assign bubble-type event handlers
            window.onclick = winEvent;
            document.onclick = docEvent;
            document.body.onclick = docBodEvent;
            // turn on click event capture for two objects
            document.addEventListener("click", docEvent, true);
            document.forms[0].addEventListener("click", formCaptureEvent, true);
            // set event listener for bubble
            document.forms[0].addEventListener("click", formBubbleEvent, false);
         }
         function winEvent(evt)
         {
            alert("Event is now at the window object level
                  (" + getPhase(evt) + ").");
         }
         function docEvent(evt)
         {
            alert("Event is now at the **document** object level
                  (" + getPhase(evt) + ").");
         }
         function docBodEvent(evt)
         {
            alert("Event is now at the BODY level (" + getPhase(evt) + ").");
         }
         function formCaptureEvent(evt)
         {
            alert("This alert triggered by FORM only on CAPTURE.");
         }
         function formBubbleEvent(evt)
         {
            alert("This alert triggered by FORM only on BUBBLE.");
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
            // reveal event phase of current event object
            function getPhase(evt)
            {
               switch (evt.eventPhase)
               {
                  case 1:
                     return "CAPTURING";
                     break;
                  case 2:
                     return "AT TARGET";
                     break;
                  case 3:
                     return "BUBBLING";
                     break;
                  default:
                     return "";
               }
            }
         </script>
      </head>
      <body onload="init()">
         <h1>W3C DOM Event Propagation</h1>
         <hr />
         <form>
            <input type="button" value="Button 'main1'" name="main1"
                   onclick="alert('Event is now at the button object level ('
                                + getPhase(event)
                                + '). ')" />
         </form>
      </body>
   </html>
```

如果启用事件捕获功能后又想要禁用它，可在最初添加事件监听器的对象上使用 removeEventListener()方法(参见第 26 章)。因为可将多个事件监听器关联到同一对象，所以 removeEventListener()方法的三个参数应与 addEventListener()方法完全相同。

1. 禁止 W3C 事件的冒泡或捕获

在 W3C DOM 中的一个事件对象方法对应于 IE4+中 event 对象的 cancelBubble 属性。阻止任何阶段中传播事件的方法是 stopPropagation()。此方法可在事件监听器函数中的任意位置调用，调用它后，当前函数会执行完毕，但事件不进一步传播。

程序清单 32-4 扩展了程序清单 32-3 中的示例，它包括两个复选框，允许在捕获或冒泡阶段，在 form 元素上停止事件的传播。

程序清单 32-4　阻止冒泡和捕获

```
<!DOCTYPE html>
<html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<head>
   <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>W3C DOM Event Propagation</title>
   <script type="text/javascript">
      function init()
      {
         // using old syntax to assign bubble-type event handlers
         window.onclick = winEvent;
         document.onclick = docEvent;
         document.body.onclick = docBodEvent;
         // turn on click event capture for two objects
         document.addEventListener("click", docEvent, true);
         document.forms[0].addEventListener("click", formCaptureEvent, true);
         // set event listener for bubble
         document.forms[0].addEventListener("click", formBubbleEvent, false);
      }
      function winEvent(evt)
      {
         if (evt.target.type == "button")
         {
            alert("Event is now at the window object level ↵
                  (" + getPhase(evt) + ").");
         }
      }
      function docEvent(evt)
      {
         if (evt.target.type == "button")
         {
            alert("Event is now at the **document** object level ↵
                  (" + getPhase(evt) + ").");
         }
      }
      function docBodEvent(evt)
      {
         if (evt.target.type == "button")
         {
            alert("Event is now at the BODY level (" + getPhase(evt) + ").");
         }
      }
      function formCaptureEvent(evt)
      {
         if (evt.target.type == "button")
         {
            alert("This alert triggered by FORM only on CAPTURE.");
            if (document.forms[0].stopAllProp.checked)
            {
               evt.stopPropagation();
            }
         }
      }
```


更多资源请访问稀酷客(www.ckook.com)

```
         function formBubbleEvent(evt)
         {
            if (evt.target.type == "button")
            {
               alert("This alert triggered by FORM only on BUBBLE.");
               if (document.forms[0].stopDuringBubble.checked)
               {
                  evt.stopPropagation();
               }
            }
         }
         // reveal event phase of current event object
         function getPhase(evt)
         {
            switch (evt.eventPhase)
            {
               case 1:
                  return "CAPTURING";
                  break;
               case 2:
                  return "AT TARGET";
                  break;
               case 3:
                  return "BUBBLING";
                  break;
               default:
                  return "";
            }
         }
      </script>
   </head>
   <body onload="init()">
      <h1>W3C DOM Event Propagation</h1>
      <hr />
      <form>
         <input type="checkbox" name="stopAllProp" />Stop all propagation at FORM
         <br />
         <input type="checkbox" name="stopDuringBubble" />Prevent bubbling past FORM
         <hr />
         <input type="button" value="Button 'main1'" name="main1"
                onclick="alert('Event is now at the button object level ('
                        + getPhase(event)
                        + '). ')" />
      </form>
   </body>
</html>
```

除了 W3C DOM 的 stopPropagation ()方法之外，为便于使用语法，NN6+、Moz、Safari、Opera 和 Chrome 也支持 IE 的 cancelBubble 属性。

更多资源请访问稀酷客(www.ckook.com)

2. 阻止 W3C 事件的默认动作

在 W3C DOM 中，与 IE 的 returnValue 属性对应的是事件对象的 preventDefault()方法。在希望阻止元素执行事件的默认动作时，可在事件处理函数中调用此方法：

```
evt.preventDefault();
```

3. 重定向 W3C DOM 事件

在 W3C 中，向正常传播模式之外的对象发送事件的机制与 IE4+类似，但使用不同的语法，而且有一项重要的要求。NN6+/Moz 和基于 Gecko、WebKit 和 Presto 的浏览器使用 W3C DOM 的 dispatchEvent()方法来代替 IE4+的 fireEvent()方法。此方法唯一的参数是事件对象，但它不能是已经在沿着元素层次结构传播的事件对象，而必须通过 W3C DOM 事件对象构造函数(本章稍后介绍)创建一个新的事件对象。程序清单 32-5 与用于 IE4+的程序清单 32-2 相同，但做了少量修改，以便在 W3C 事件模型中运行。注意，dispatchEvent()方法把一个新创建的事件对象作为其唯一的参数。

程序清单 32-5　在 W3C DOM 中取消和重定向事件

```
<!DOCTYPE html>
<html onclick="revealEvent('HTML', event)">
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Event Cancelling & Redirecting</title>
      <script type="text/javascript">
         // display alert with event object info
         function revealEvent(elem, evt)
         {
            var msg = "Event (from " + evt.target.tagName + " at ";
            msg += evt.clientX + "," + evt.clientY + ") is now at the ";
            msg += elem + " element.";
            alert(msg);
         }
         function init()
         {
            document.onclick = docEvent;
            document.body.onclick = docBodEvent;
         }
         function docEvent(evt)
         {
            revealEvent("document", evt);
         }
         function docBodEvent(evt)
         {
            revealEvent("BODY", evt);
         }
         function buttonEvent(form, evt)
         {
            revealEvent("BUTTON", evt);
```


更多资源请访问稀酷客(www.ckook.com)

```
            // redirect if checked
               if (form.redirect.checked)
               {
                  document.body.dispatchEvent(evt);
               }
            // cancel if checked
               if (form.bubbleCancelState.checked)
               {
                  evt.stopPropagation();
               }
         }
      </script>
   </head>
   <body onload="init()">
      <h1>Event Cancelling & Redirecting</h1>
      <hr />
      <form onclick="revealEvent('FORM', event)">
         <p><button name="main1" onclick="buttonEvent(this.form, event)">Button
              'main1'</button>
         </p>
         <p><input type="checkbox" name="bubbleCancelState"
                  onclick="event.stopPropagation()" />Cancel Bubbling at BUTTON
            <br />
            <input type="checkbox" name="redirect"
                  onclick="event.stopPropagation()" /> Redirect Event to BODY
         </p>
      </form>
   </body>
</html>
```

32.3 引用事件对象

当今的浏览器有两种不同的事件对象模型，脚本访问那些对象的方式也分为两类：IE 方式和 W3C 方式。首先介绍较简单的 IE 方式。

在 IE4+中，event 对象可以作为 window 对象的属性来访问：

```
window.event
```

但注意，引用中的 window 部分是可选的，因此脚本可将 event 对象作为全局引用：

```
event.propertyName
```

因此，事件处理函数中的任何语句都可以访问 event 对象，而不需要任何准备或初始化。

在 W3C 事件模型中，情况要复杂一些。在某些情况下，必须明确地将事件对象作为参数传递给事件处理函数；而在其他情况下，事件对象会自动传递为参数。区别取决于事件处理函数与对象的绑定方式。

更多资源请访问稀酷客(www.ckook.com)

32.4 绑定事件

脚本中事件处理的最重要方面是将事件绑定到页面上的元素。可以采用几种不同的方式来实现事件绑定，但它们在不同的浏览器上并不完全兼容。另外，某些技术过时了，因为更现代的方式改进了它们。下面是将事件绑定到元素上的 4 种主要技术：

- 用标记特性来指定。
- 用对象属性来指定。
- IE 中的附加功能。
- NN/Moz/W3C 中的事件监听器。

下面将详细介绍这些事件绑定选项，重点展示如何利用后两种技术创建现代的、跨浏览器的事件绑定函数。

32.4.1 使用标记特性绑定事件

在某些最早支持 JavaScript 的浏览器中，将事件处理程序绑定到对象上的最初方式是使用元素标记中的特性。要按这种方式绑定事件，只需在元素特性中指定内联 JavaScript 代码，如下所示：

```
<input type="button" value="Click Me" onclick="handleClick();" />
```

特性名称是要处理的事件名称，其值是事件触发时执行的内联 JavaScript 代码。在事件特性中可以包含多条语句，如下所示：

```
<input type="button" value="Click Me"
onclick="doSomething(this); doSomethingElse(this.form);" />
```

在支持 W3C 事件模型的现代浏览器(NN6+/Moz、基于 Gecko、WebKit 和 Presto 的浏览器)中，如果想在事件处理函数中检查事件的属性，必须通过传递 event 参数，将 event 对象指定为参数，如下：

```
<input type="button" value="Click Me" onclick="handleClick(event);" />
```

W3C 模型仅在这里明确引用 event(小写 e)对象，就像它是一个全局引用。此引用在任何其他上下文中都无效，只能作为事件处理函数的参数。如果有多个参数，event 引用可放在任何位置，但最好将其放在最后：

```
<input type="button" value="Click Me" onclick="doSomething(this, event);" />
```

因而，绑定到元素上的函数定义应该有一个参数变量，来接收事件对象参数：

```
function doSomething(widget, evt) {...}
```

此参数变量的命名没有限制，在本书的一些示例中，该变量命名为 event，更常用的名称是 evt。在编写跨浏览器的脚本时，不应该使用 event 作为参数变量名，以免与 Internet Explorer 的 window.event 属性冲突。

更多资源请访问稀酷客(www.ckook.com)

通过事件标记特性来绑定事件在所有浏览器中都有效，但这与 Web 设计的主流趋势相背：将 HTML 内容与使页面互动的代码相分离。换句话说，Web 开发人员尽力使 JavaScript 代码独立于 HTML 代码。

这个概念与样式表将内容与显示分开的思想密切相关。这样，网页就有三种不同的组件：HTML 内容、CSS 和 JavaScript 代码。尽可能分离这三种组件，会得到更清晰、更便于管理的代码。

注意：

本书在演示各种对象、属性和方法时，有许多事件标记特性绑定的示例。这是有意为之的，因为事件紧密绑定到元素上时，通常更容易理解所讨论的概念。

为在 JavaScript 事件绑定和 HTML 代码之间有明确的界限，应只在脚本代码而不是 HTML 元素的特性中绑定事件，前面提到的后三种绑定方式都提供了这种界限。

32.4.2　使用对象特性绑定事件

在 NN3 和 IE4 中，元素对象的事件属性可通过赋值来绑定事件。对于元素能接收和响应的每个事件，都有一个适当命名的属性，属性名全部小写。例如，button 元素对象的 onclick 属性就对应 onclick 事件。将一个函数引用赋给 onclick 属性，就可以把事件处理程序绑定到 button 元素上：

```
document.forms[0].myButton.onclick = handleClick;
```

注意：

事件属性名应采用全小写形式(onclick)，但某些浏览器也识别混合大小写的事件名称(onClick)。

通过对象属性来绑定事件的一个问题是：乍看起来似乎不能给所调用的事件处理函数传递自己的参数。W3C 浏览器把事件对象作为事件处理函数的唯一参数，不允许传递自己的参数。这意味着函数将在一个参数变量中接收传递过来的事件对象：

```
function doSomething(evt) {...}
```

event 对象包含对事件目标对象的引用，可以通过该引用访问该对象的任何属性，如包含表单控件对象的 form 对象。

事实上完全可以传递自己的参数，只需一个中间匿名函数来完成中间人的工作。例如，下面的代码演示了如何传递标准的 event 对象和一个自定义参数：

```
document.forms[0].myButton.onclick =
   function(evt) {doSomething("Cornelius", evt);};
```

在这个示例中，名称字符串是事件处理程序的第一个参数，而 event 对象(自动传递为匿名函数的唯一参数，并赋予参数变量 evt)是第二个参数。实际的事件处理程序代码如下：

```
function doSomething(firstName, evt) {...}
```

对于函数内的语句，doSomething()事件处理函数中的 evt 参数变量用作 event 对象的引用。如果需要从这里调用其他函数，可根据需要进一步传递 event 对象引用。只要事件动作触发的

更多资源请访问稀酷客(www.ckook.com)

执行链持续不断，event 对象就保留其属性。

32.4.3 使用 IE 附加功能绑定事件

在 IE5 中，Microsoft 建立了一种通过附加功能将事件绑定到元素上的新方式，该方式最初用于 IE 操作(参见配书光盘上的第 51 章)。最终，事件绑定的附加方式超出了 IE 操作的范围，而成为事件绑定的 IE 事实标准。如下一节所述，由于 IE(到版本 8 为止)仍然不支持 W3C 的事件绑定方式，在可预见的未来，应该将附加功能作为 IE 中处理事件的首选方式。

IE 事件附加功能通过 attachEvent()和 detachEvent()方法来管理，所有能够接收事件的元素对象都支持这两个方法。使用这两个方法可以在整个应用程序中，根据需要绑定事件和解除事件绑定。

attachEvent()方法的形式如下：

```
elementReference.attachEvent("event", functionReference);
```

为了正确使用此形式，下面列举一个在 IE 中使用 attachEvent()方法绑定事件的示例：

```
document.getElementById("myButton").attachEvent("onclick", doSomething);
```

attachEvent()的第一个参数是事件的字符串名称，包括 on 前缀，如 onclick；第二个参数是对事件处理函数的引用。

IE 事件附加方式提供的一个新功能是将同一事件多次附加到同一元素上(可能指向不同的事件处理函数)。注意，如果选择将同一类型的多个事件绑定到同一元素上，它们的处理顺序就与绑定顺序相反，这意味着添加的第一个事件会最后处理。

由于 IE 事件模型是基于 event 对象(它是 window 对象的一个属性)的，因此不把 event 对象传递到事件处理函数中。要访问事件属性，只需使用 window.event 或 event 来访问窗口的 event 对象。后一种方式可以奏效，是因为在客户端脚本编程中总是假定使用了 window 对象。32.5 节说明了如何协调 IE 的 window.event 属性和 W3C 中 event 事件处理程序的参数。

IE 事件绑定方式还提供了解除事件绑定的功能，解除事件绑定后，目标元素将不再接收事件通知。在元素上调用 detachEvent()方法，就可以解除 IE 事件的绑定，如：

```
document.getElementById("myButton").detachEvent("onclick", doSomething);
```

通过此示例可以了解到，detachEvent()方法使用与 attachEvent()完全相同的语法。

32.4.4 通过 W3C 监听器绑定事件

W3C 绑定事件的方式在逻辑上与 IE 事件附加方式类似，它也使用两个方法：addEventListener()和 removeEventListener()，这两个方法允许元素监听事件，然后做出响应。与 IE 的 attachEvent()和 detachEvent()方法类似，addEventListener()和 removeEventListener()也成对使用，分别用于添加和删除事件监听器。

addEventListener()方法的形式如下：

```
elementReference.addEventListener("eventType", functionReference,
   captureSwitch);
```


更多资源请访问稀酷客(www.ckook.com)

下面的示例说明了该方法的实际用法：

```
document.getElementById("myButton").addEventListener("click", doSomething,
   false);
```

注意，指定的事件名称没有 on 前缀，这与 IE 事件附加方式中使用的名称不同。W3C 事件监听器与 IE 事件附加方式的另一个明显区别在于，addEventListener()方法的第三个参数 captureSwitch 决定在事件传播的捕获阶段，元素是否应该监听事件。本章稍后将介绍事件传播，以及如何使用此参数调整事件的传播，目前只需要知道该参数通常设为 false。

与 IE 事件附加方式类似，同一事件监听器可多次添加到同一元素上。不过，与 IE 方式不同的是，按这种方式添加的 W3C 事件按照其添加的顺序处理，这意味着第一个添加的事件最先处理。

W3C 事件模型与 IE 事件处理的另一个类似之处是能够从元素上解除事件的绑定。W3C 使用 removeEventListener()方法解除事件的绑定，如下所示：

```
document.getElementById("myButton").removeEventListener("click", doSomething,
   false);
```

此示例说明，removeEventListener ()方法的参数与 addEventListener()相同。

32.4.5 跨浏览器的事件绑定解决方案

综合一下前面讲述的现代事件处理技术，显然一定可以协调 IE 和 W3C 的事件绑定方式。事实上，并不需要太多额外的代码，就可以使用最新的事件绑定技术来绑定事件，且仍能给老浏览器使用旧技术(对象属性)。

下面的跨浏览器函数可为元素添加事件绑定：

```
function addEvent(elem, evtType, func)
{
   if (elem.addEventListener)
   {
      elem.addEventListener(evtType, func, false);
   }
   else if (elem.attachEvent)
   {
      elem.attachEvent("on" + evtType, func);
   }
   else
   {
      elem["on" + evtType] = func;
   }
}
```

该函数的参数是元素的引用、事件类型字符串(即没有 on 前缀的版本)，以及在元素上触发事件时要调用的函数的引用。addEvent()函数首先尝试在所提供的元素上使用 addEventListener()方法，这满足了现代 W3C 浏览器(NN6+/Mozilla/Safari/Opera/Chrome)的要求；如果操作失败，就使用 attachEvent()，这兼容了现代 IE 浏览器(IE5+)；如果这也行不通，函数就直接将事件处

更多资源请访问稀酷客(www.ckook.com)

理函数赋予事件对象属性，这在绝大多数的浏览器上都有效。

注意：

通过添加第 4 个参数，并传递给 addEventListener()而不是传递 false，可以方便地扩展 addEvent()函数，以便允许 addEventListener()方法使用 captureSwitch 参数。

当然，只有调用 addEvent()函数，才能为页面绑定事件。onload 事件提供了绑定事件的极好机会，但最好不要在 HTML 特性 onload 中仅调用 addEvent()函数，这会违反前述的所有规则。应先为 onload 事件添加一个匿名事件处理程序，然后在该函数中绑定其他事件。下面是一个示例：

```
addEvent(window, "load", function()
{
   addEvent(document.getElementById("myButton"), "click", handleClick);
   addEvent(document.body, "mouseup",
      function(evt) {handleClick(evt);});
});
```

如果需要解除事件的绑定，可使用下面的跨浏览器函数：

```
function removeEvent(elem, evtType, func)
{
   if (elem.removeEventListener)
   {
      elem.removeEventListener(evtType, func, false);
   }
   else if (elem.detachEvent)
   {
      elem.detachEvent("on" + evtType, func);
   }
   else
   {
      elem["on" + evtType] = null;
   }
}
```

32.5 事件对象兼容性

虽然 W3C COM 和 IE 中 event 对象到达事件处理函数的方式不兼容，但很容易将该对象赋给一个各种浏览器类型都可用的变量。例如，下面的函数段不仅接收 W3C DOM 的 event 对象，还接收 IE 的 event 对象：

```
function doSomething(evt)
{
   evt = (evt) ? evt : ((window.event) ? window.event : null);
   if (evt)
   {
      // browser has an event to process
      ...
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
    }
```

接收到 event 对象参数后，该对象还可以继续用作 evt。如果不是这样，函数就保证 window.event 对象可用，并将其赋给 evt 变量。最后，如果浏览器不能处理 event 对象，evt 变量就是 null，只有 evt 包含事件对象，才继续处理。

这是简单的部分，比较难的是细节：属性名称在两种事件对象模型中区别很大，读取事件对象的属性非常困难。本章后面将介绍两种事件对象模型中每个方法和属性的细节，现在仅对比一下属性的术语。表 32-2 列出了事件对象的共有信息和动作，以及在两种事件对象模型中使用的方法或属性名称。

表 32-2 event 对象的共有属性和方法

属性/动作	IE4+	W3C DOM
目标元素	srcElement	target
事件类型	type	type
元素中的 X 坐标	offsetX	n/a+
元素中的 Y 坐标	offsetY	n/a+
页面上的 X 坐标	n/a+	pageX++
页面上的 Y 坐标	n/a+	pageY++
窗口中的 X 坐标	clientX	clientX
窗口中的 Y 坐标	clientY	clientY
屏幕上的 X 坐标	screenX	screenX
屏幕上的 Y 坐标	screenY	screenY
鼠标按钮	button	button
键盘按键	keyCode	keyCode
按下 Shift 键	shiftKey	shiftKey
按下 Alt 键	altKey	altKey
按下 Ctrl 键	ctrlKey	ctrlKey
前一个元素	fromElement	relatedTarget
后一个元素	toElement	relatedTarget
取消冒泡	cancelBubble	preventBubble()
阻止默认动作	returnValue	preventDefault()

+用其他属性进行计算，就可以获得其值。

++不是正式的 W3C DOM 属性，但 Mozilla、Safari、Opera 和 Chrome 支持它。

如表 32-2 所示，IE4+和 W3C 事件对象的属性有许多相同之处。最重要的不兼容之处是如何引用作为事件目标的元素。在代码中加入分支结构，就可以得到引用元素的公用变量。例如把一个语句嵌入前面的函数段，如下所示：

```
var elem = (evt.target) ? evt.target : ((evt.srcElement) ?
```


更多资源请访问稀酷客(www.ckook.com)

```
    evt.srcElement : null);
```

每种事件模型都有其他模型没有的属性，本章其余部分将加以详细说明。

32.6 事件模型详析

虽然事件对象模型处理属性的方式有很大的区别，但使各种浏览器在操作事件方面相互兼容却不大困难。本节用两个脚本介绍重要的事件属性。第一个脚本指出，在触发事件时是否按下了修改键；第二个脚本为鼠标键和键盘键提取代码。在支持事件对象的所有现代浏览器上，都可以运行这两个脚本。

32.6.1 以跨平台方式检查修改键

程序清单 32-6 演示了检查触发事件时是否按下修改键的分支技术，modifiers 和 altKey 等事件对象的属性详见本章后面的讨论。为了查看页面的行为，单击一个链接，在文本框中输入内容，然后在按住任意组合修改键的同时单击按钮。底部的 4 个复选框代表 4 个修改键。当单击或输入时，就选中相应的修改键复选框。

程序清单 32-6　修改键的复选事件

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Event Modifiers</title>
      <script type="text/javascript">
         function checkMods(evt)
         {
            evt = (evt) ? evt : ((window.event) ? window.event : null);
                  if (evt)
                  {
                     var elem = (evt.target) ? evt.target : evt.srcElement;
                     var form = document.output;
                     form.modifier[0].checked = evt.altKey;
                     form.modifier[1].checked = evt.ctrlKey;
                     form.modifier[2].checked = evt.shiftKey;
                     form.modifier[3].checked = false;
                  }
            return false;
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }

         addEvent(window, "load", function()
         {
            addEvent(document.getElementById("link"), "mousedown",
               function(evt) {return checkMods(evt);});
            addEvent(document.getElementById("text"), "keyup",
               function(evt) {checkMods(evt);});
            addEvent(document.getElementById("button"), "click",
               function(evt) {checkMods(evt);});
         });
      </script>
   </head>
   <body>
      <h1>Event Modifiers</h1>
      <hr />
      <p>Hold one or more modifier keys and click on
         <a id="link" href="javascript:void(0)">this link</a>
         while looking at the checkboxes to see which keys you are holding.
      </p>
      <form name="output">
         <p>Enter some text with uppercase and lowercase letters
            (while looking at the checkboxes):
            <input id="text" type="text" size="40" />
         </p>
         <p>
            <input id="button" type="button"
            value="Click Here With Modifier Keys" />
         </p>
         <p>
            <input type="checkbox" name="modifier" />Alt
            <input type="checkbox" name="modifier" />Control
            <input type="checkbox" name="modifier" />Shift
            <input type="checkbox" name="modifier" />Meta
         </p>
      </form>
   </body>
</html>
```

这个脚本检查每个修改键的事件对象属性，以确定是否按下了修改键。

更多资源请访问稀酷客(www.ckook.com)

32.6.2　以跨平台方式捕获按键

为了演示两个事件捕获模型中的键盘事件，程序清单 32-7 捕获了输入文本框的键盘字符以及用于单击按钮的鼠标键。

程序清单 32-7　键盘和鼠标按键的复选框事件

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>Button and Key Properties</title>
      <script type="text/javascript">
         function checkWhich(evt)
         {
            evt = (evt) ? evt : ((event) ? event : null);
               if (evt)
               {
                  var thingPressed = "";
                  var elem = (evt.target) ? evt.target : evt.srcElement;
                     if (elem.type == "textarea")
                     {
                        thingPressed = (evt.charCode) ? evt.charCode : evt.keyCode;
                     }
                     else if (elem.type == "button")
                     {
                        thingPressed = (typeof evt.button !=
                        "undefined") ? evt.button : "n/a";
                     }
                  alert(thingPressed);
               }
            return false;
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
```


更多资源请访问稀酷客(www.ckook.com)

```
            addEvent(window, "load", function()
            {
               addEvent(document.getElementById("button"), "mousedown",
                        function(evt) {checkWhich(evt);});
               addEvent(document.getElementById("text"), "keypress",
                        function(evt) {checkWhich(evt);});
            });
         </script>
      </head>
      <body>
         <h1>Button and Key Properties</h1>
         <hr />
         <form>
            <p>Mouse down atop this
               <input id="button" type="button" value="Button" />
               with either mouse button (if you have more than one).
            </p>
            <p>Enter some text with uppercase and lowercase letters:
               <textarea id="text" cols="40" rows="4" wrap="virtual"></textarea>
            </p>
         </form>
      </body>
   </html>
```

为每个键盘事件显示的代码与字符键的 ASCII 值相同。如果需要其他键的代码，onkeydown 和 onkeyup 事件处理程序为按下的每个键提供了 Unicode 值。请参阅本章后面事件对象的 charCode 和 keyCode 属性，以便了解更多的细节。

32.7 事件类型

虽然版本 4 之前的浏览器没有可访问的事件对象，但最好总结一下现代浏览器中 type 属性的发展历程。type 属性表示生成事件对象的事件的类型(事件处理程序名称去掉 on)。IE4+和 NN6+/W3C 对象模型为每个 HTML 元素提供了事件处理程序，因此不仅能为可单击按钮定义 onclick 事件处理程序，也可以为 p 甚至 span 元素定义 onclick 事件处理程序。

旧版浏览器

早期的浏览器版本限制了特定元素的事件处理程序数量，它们只能处理对该元素有意义的事件。即便如此，很多脚本编写者仍希望更多的对象有更多的事件处理程序。在 IE4+/NN6+/W3C 中实现之前，脚本编写者需要了解对象模型的限制。表 32-4 列出了三种早期浏览器的对象可用的事件处理程序。每一列都表示引入事件类型的版本。例如，window 对象刚开始有 4 种事件类型，在 NN4 发布后，又增加了三种。与此形成对照的是，area 对象在 NN3 中第一次引入，所以它的第一个事件类型出现在 NN3 那一列。

除了 NN4 中的 layer 对象之外，新的浏览器支持表 32-3 中的所有对象，所以可以放心地使用这些事件处理程序。另外，在表 32-4 列出的浏览器中，只有 NN4 有可用于脚本的 event 对象。

更多资源请访问稀酷客(www.ckook.com)

表 32-3　旧版浏览器的事件类型

对象	NN2/IE3	NN3	NN4
window	blur		dragdrop
	focus		move
	load		resize
	unload		
layer			blur
			focus
			load
			mouseout
			mouseover
			mouseup
link	click	mouseout	dblclick
	mouseover		mousedown
			onmouseup
area		mouseout	click
		mouseover	
image		abort	
		error	
		load	
form	submit	reset	
text,textarea,password			
	blur		keydown
	change		keypress
	focus		keyup
	select		
all buttons	click		mousedown
			mouseup
select	blur		
	change		
	focus		
fileupload		blur	
		focus	
		select	

32.7.1　IE4+和 NN6+/W3C 中的事件类型

第 26 章定义了常用元素的事件处理程序。事件类型很多，大多数事件类型仅用于 IE4、IE5

更多资源请访问稀酷客(www.ckook.com)

和 IE5.5+，有的仅在 Windows 版本中才有。

但如果同时为 IE4+和 NN6+/W3C 编写页面，就需要知道这些浏览器系列和版本的常用事件类型。在 NN6+/Moz/和基于 Gecko、WebKit 和 Presto 的浏览器中，事件类型主要基于 W3C DOM Level 2 规范。虽然其中也包括键盘事件，但 DOM Level 3 仍在开发其正式标准。表 32-4 列出了现代浏览器的公共事件类型以及支持它们的对象。尽管 IE 事件列表不长，但表 32-4 的事件类型是所有浏览器都使用的事件类型的基本集合。

表 32-4 IE4+和 W3C DOM 中的共用事件类型

事件类型	可用元素
abort	object
blur	Window, button, text, password, label, select, textarea
change	text, password, textarea, select
click	所有元素
error	Window, frameset, object
focus	window, button, text, password, label, select, textarea
keydown	text, password,textarea
keypress	text, password, textarea
keyup	text, password, textarea
load	window, frameset,object
mousedown	所有元素
mousemove	所有元素
mouseout	所有元素
mouseover	所有元素
mouseup	所有元素
reset	form
resize	window
scroll	window
select	Text, password, textarea
submit	form
unload	Window, frameset
IE4+事件对象	
altKey	
altLeft	
behaviorCookie	
behaviorPart	
bookmarks	
boundElements	
button	

更多资源请访问稀酷客(www.ckook.com)

(续表)

IE4+事件对象
cancelBubble
clientX
clientY
contentOverflow
ctrlKey
ctrlLeft
dataFld
dataTransfer
fromElement
keyCode
nextPage
offsetX
offsetY
propertyName
qualifier
reason
recordset
repeat
returnValue
saveType
screenX
screenY
shiftKey
shiftLeft
srcElement
srcFilter
srcUrn
toElement
type
wheelData
X
Y

32.7.2　语法

访问 IE4+ event 对象的属性：

更多资源请访问稀酷客(www.ckook.com)

```
[window.]event.property
```

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari-，Opera-，Chrome-

32.7.3　关于 event 对象

IE4+的 event 对象是 window 对象的一个属性，本章前面介绍了其基本操作。

利用 The Evaluator(参见第 4 章)可以了解 event 对象。如果在底部文本框中输入 event，就可以检查 event 对象的属性，因为这时触发了显示 event 对象属性的函数；如果在文本框中按 Enter 键，就会运行内部脚本，显示 keypress 事件的属性。单击 List Properties 按钮，可以观察按钮激活的 click 事件的属性。在单击时按下修改键，可以查看它对一些属性的影响。

查看 event 对象的属性时，要特别注意每个属性的兼容性。这个对象的属性列表随 IE4+事件对象模型的发展而增加；并且，这里列出的大多数属性在 IE4 中是只读的。但在 IE5+中，如果利用 IE5.5+的 document.createEventobject()等方法人工建立事件，上述属性就是可读/写的。由用户或系统动作创建的事件对象中，可动态更改的属性很少，以防止脚本改变用户的动作。还要注意，一些属性与 W3C DOM 事件对象的属性相同，如兼容性描述所示。

32.7.4　属性

altKey，ctrlKey，shiftKey

值：Boolean，　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

对于为响应用户或系统动作而创建的 event 对象，这三个属性根据单击鼠标的同时是否按下相应的键来设置，如 Shift+单击。若按下相应的键，属性值就是 true，否则值为 false。

多数情况下，把包含这些属性的表达式用作 if 结构的条件语句。因为它们是 Boolean 值，可在一个条件中组合使用多个属性。例如，只有同时按下 Shift 和 Control 键，才执行某函数分支，语句如下：

```
if (event.shiftKey && event.ctrlKey)
{
   // statements to execute
}
```

相反，如果用户按下三个修改键中的任一键，可以采用更友好的方式进行特殊处理：

```
if (event.shiftKey || event.ctrlKey || event.altKey)
{
   // statements to execute
}
```

这种方式的作用是为用户提供快捷操作，而不是要求他们记住具体的组合键。

示例

参见程序清单 32-6，这些属性值给各种事件类型的相应复选框设置 checked 属性。

相关主题：altLeft，ctrlLeft，shiftLeft 属性。

更多资源请访问稀酷客(www.ckook.com)

altLeft，ctrlLeft，shiftLeft

值：Boolean，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

一些 Windows 版本在使用 IE5.5+时，只有按下左边的 Alt、Ctrl 和 Shift 键，才能更改事件。对于 event 对象记录的修改键，焦点必须在文档(body)中，而不在其他表单控件中。如果这些属性的左键版本的值是 false，而一般版本的值是 true，则脚本知道事件触发时按下了右边的键。

相关主题：altKey，ctrlKey，shiftKey 属性。

behaviorCookie，behaviorPart

值：整型，　　只读

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

这两个属性与 Microsoft 称为显示行为的 Windows 技术有关。与第 26 章讨论的 addBehavior()方法不同，显示行为用 C++编写，为用户提供了 Web 页的定制显示服务。要了解更多细节，请查看 http://msdn.microsoft.com/en-us/library/aa753628%28VS.85%29.aspx 中的 implementing Rendering Behaviors 文档。

bookmarks，boundElements，dataFld，qualifier，reason，recordset

值：见正文，　　只读

兼容性：WinIE6+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

这组 event 对象属性与在 IE4+的 Windows 版本中使用的数据绑定相关，本书不讨论数据绑定的细节，但表 32-5 汇总了 event 对象的许多数据绑定属性(大多数是数据绑定中的术语，但不会影响其他脚本)。要了解更多细节，可在 http://msdn.microsoft.com/en-us/library/default.aspx 中搜索 ActiveX Data Objects(ADO)。

表 32-5　ADO 相关的事件对象属性

属　性	值	首次实现	说　明
bookmarks	数组	IE4	在与接收事件的对象相关联的记录集中，记录了 ADO 书签(保存的位置)数组
boundElements	数组	IE5	所有元素的引用数组，绑定到当前事件所涉及的同一数据集上
dataFld	字符串	IE5	数据源列的名称，绑定到接收 cellchange 事件的表格单元格上
qualifier	字符串	IE5	与接收数据相关事件的数据源相关联的数据成员名称。只有数据源对象(DSO)允许包含多名称数据成员，或通过被绑定元素的 datasrc 特性明确设置了限定符，该属性才可用。在 IE5+中为读/写
Reason	整型	IE4	仅在 onDataSetComplete 事件中设置，提供数据集载入的结果代码(0-成功；1-传输中止；2-其他错误)
Recordset	对象	IE4	数据源对象的当前记录集的引用

更多资源请访问稀酷客(www.ckook.com)

注意：

尽管 IE 仍支持 Microsoft 原有的 ADO 技术，但它已被 ADO.NET 替代，ADO.NET 与 Microsoft 的.NET 架构集成得更加紧密。要了解这两种技术的区别，请访问：

```
http://msdn.microsoft.com/library/en-us/dndotnet/html/adonetprogmsdn.asp
```

button

值：整型，　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

button 属性指出按下哪些键会激活鼠标事件。如果在 IE 中，没有按下鼠标键就触发了事件，该属性值为 0。整数 1～7 代表鼠标的单键和多键组合，其中也包括操作系统可识别的三键鼠标。在 IE 中，对应按键的整型值如表 32-6 所示。

表 32-6　对应按键的整型值

数　值	说　明	数　值	说　明
0	无键	4	中键
1	左键(主键)	5	左键和中键
2	右键	6	右键和中键
3	左键和右键一起	7	左键、中键和右键

除主键外，大多数其他按键组合很容易触发 mousedown 或 mouseup(不是 onclick)事件。注意，用户按下多个键时，每次按键都会激活一个 mousedown 事件。因此，如果用户先按下左键，则激活 mousedown 事件，并给 event.button 属性赋予 1；此时按下右键，会再次触发 mousedown 事件，但 event.button 值为 3。如果同时按下两个键，脚本就会执行特定的动作，忽略或不执行按下单个键的动作，因为在处理过程中很可能激活单键事件，干扰目标操作。

在 IE4+和 NN6+/Moz/W3C 中使用 event.button 属性编程时要非常小心，W3C DOM 事件模型为鼠标键定义了不同的按键值(0、1 和 2 分别代表左、右和中间键)，而没有为组合键定义值。

示例

在程序清单 32-7 中，event.button 属性显示在状态栏中。例如，在屏幕按钮上单击鼠标的各个键，然后试验多键组合，在状态栏中观察结果。

相关主题：无。

cancelBubble

值：Boolean，　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

cancelBubble 看起来像一个方法名，但它决定当前的 event 对象是否冒泡到文档的元素包含层次结构的更高层。其值默认为 false，也就是说若事件允许冒泡，就自动冒泡。

为禁止当前的事件冒泡，可在事件处理函数中将该属性设置为 true。另外，还可在元素的事件处理特性中直接取消冒泡：

更多资源请访问稀酷客(www.ckook.com)

```
onclick="doButtonClick(this); event.cancelBubble = true"
```

取消事件冒泡只对当前事件起作用，激活下一个事件将启用冒泡(假设事件允许冒泡)。

示例

参见程序清单 32-2，来了解 cancelBubble 属性。虽然该程序清单的一些特性用于 IE5.5+，但这个禁止冒泡的例子在 IE4 也有效。

相关主题：returnValue 属性。

clientX，clientY，offsetX，offsetY，screenX，screenY，x，y

值：整型， 读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

IE 的 event 对象在 4 个坐标空间中为事件提供坐标，这 4 个坐标空间是元素本身、事件目标的父元素、浏览器窗口的可视区域以及整个视频屏幕。但如本节所述，一些属性可能会返回相应坐标空间的错误值。注意，如果用户滚动了窗口，这些属性都不会提供事件相对于整个页面的准确位置。

最里面的坐标空间是事件目标的元素，offsetX 和 offsetY 属性提供了事件在目标元素中的像素坐标，这样就可以确定图像中的单击点，而不管图像是包含在 body 中还是在指定 div 的周围。在 Windows 直到 IE8 的版本中，大多数情况下会生成正确的值，但对于 body 元素的一些子元素，纵坐标(y)值是相对于可视窗口的，而不是相对于元素本身。在程序清单 32-8 中，单击页面顶部的 hl 或 p 元素，就可以看到这方面的一个例子。这个问题不会影响 MacIE，但 MacIE 存在另一个问题：如果页面从通常的初始位置滚动，滚动值需要从 clientX 和 clientY 值中减掉，这是一个不兼容的错误。如果需要获得滚动页上元素的单击坐标，就应考虑这个错误。由于在 Windows 中没有这个错误，所以只需更正 Mac 中的错误。

把坐标空间扩展到事件目标的父偏移元素，WinIE5+的 x 和 y 属性应返回事件相对于目标父偏移元素(该元素可以通过 offsetParent 属性得到)的坐标。对于大多数非定位元素，这些值与 clientX 和 clientY 属性相同，因为如后面所述，这些父偏移元素与其父元素 body 的偏移值为 0。注意在 WinIE4 和 MacIE5 中，除了 body 之外，x 和 y 属性没有考虑任何父位置的偏移。甚至在 WinIE5+中，属性也常常出错，最好不用。

另一对坐标 clientX 和 clientY 相对于浏览器窗口的可视文档区域。文档滚至顶部(或文档不滚动)时，这些坐标与整个页面的坐标相同。但由于页面可以向下滚动可视窗口，因此若滚动页面，该页的坐标就会改变。另外，在 Windows 版的 IE 中，可注册在 body 元素之外至多两个像素处触发的鼠标事件。所以，在 WinIE 中，如果单击 body 的背景，就会激活 body 元素上的事件，但 clientX/clientY 值比 offsetX/offsetY 大两个像素(它们在 MadE 中相等)。尽管存在轻微的差异，如果想得到事件在定位元素中的坐标，应使用 clientX 和 clientY 属性，但这些坐标应相对于整个可见窗口，而不仅仅是定位上下文。

确定事件的坐标时，通常必须考虑页面滚动。毕竟，除非生成一个大小固定的窗口，否则就不知道浏览器窗口的方向。若在页面的某个区域寻找单击点，就必须考虑页面的滚动。滚动因子可以从 document.body.scrollLeft 和 document.body.scrollTop 属性中得到。在读取 clientX 和 clientY 属性时，只有加上相应的滚动属性，才能得到事件在页面上的位置：

更多资源请访问稀酷客(www.ckook.com)

```
var coordX = event.clientX + document.body.scrollLeft;
var coordY = event.clientY + document.body.scrollTop;
```

这样做就不会出错了。

最后，screenX 和 screenY 属性返回事件在整个显示器屏幕上的像素坐标。如果 IE 提供更多的窗口尺寸属性，这两个值就会更有用。无论如何，只有光标在浏览器窗口的内容区域内，才激活鼠标事件，事件的屏幕坐标决不会在该区域外。

这些描述可能难以理解，如果出现了错误，就显得非常混乱。想一想如何在脚本中使用事件坐标，一般情况下，需要知道两类鼠标事件坐标：元素内部坐标和页面内部坐标。前者可使用 offsetX/ offsetY 属性来获得，后者可使用 clientX/clientY(加上滚动属性值)来获得。

坐标属性主要用于鼠标事件。如何确定用户在调整窗口的大小时，是按下标题栏中的最大化按钮(在 Mac 中称做缩放框)，还是使用了屏幕右下方的调整大小按钮？鼠标事件坐标在 event 对象的 resize 事件中记录。如果用户按下了最大化按钮，clientY 坐标就变成负值(在客户空间之上)，clientX 坐标在上一个窗口宽度(document.body.clientWidth)的 45 像素之内。当然，这在重新调整窗口大小之后改变，所以这无法阻止窗口调整其大小。

示例

程序清单 32-8 以交互方式提供了所有事件坐标属性的值。onmousedown 事件处理程序会触发所有事件处理，所以可在页面任意位置单击鼠标，看看会发生什么。查看鼠标事件的目标元素中的标记，可以直观地了解某些坐标属性是如何确定的。图像封装在定位 div 元素中，以帮助查看当事件目标在定位元素内时，某些属性会有何变化。

程序清单 32-8　IE4+事件坐标属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>X and Y Event Properties (IE4+ Syntax)</title>
      <script type="text/javascript">
         function checkCoords(evt)
         {
            evt = (evt) ? evt : ((window.event) ? window.event : null);
               if (evt)
               {
                  var elem = (evt.target) ? evt.target : evt.srcElement;
                  var form = document.forms[0];
                  form.srcElemTag.value = "<" + elem.tagName + ">";
                  form.clientCoords.value = evt.clientX + "," + evt.clientY;
                     if (typeof document.body.scrollLeft != "undefined")
                     {
                        form.pageCoords.value = (evt.clientX +
                        document.body.scrollLeft) +
                           "," + (evt.clientY + document.body.scrollTop);
                     }
                  form.offsetCoords.value = evt.offsetX + "," + evt.offsetY;
```


更多资源请访问稀酷客(www.ckook.com)

```
                form.screenCoords.value = evt.screenX + "," + evt.screenY;
                form.xyCoords.value = evt.x + "," + evt.y;
                   if (elem.offsetParent)
                   {
                      form.parElem.value = "<" + elem.offsetParent.tagName + ">";
                   }
                return false;
             }
       }
       function handleSize(evt)
       {
          evt = (evt) ? evt : ((window.event) ? window.event : null);
             if (evt)
             {
                document.forms[0].resizeCoords.value =
                    evt.clientX + "," + evt.clientY;
             }
       }

       // bind the event handlers
       function addEvent(elem, evtType, func)
       {
          if (elem.addEventListener)
          {
             elem.addEventListener(evtType, func, false);
          }
          else if (elem.attachEvent)
          {
             elem.attachEvent("on" + evtType, func);
          }
          else
          {
             elem["on" + evtType] = func;
          }
       }
       addEvent(window, "load", function()
       {
          addEvent(document.body, "mousedown",
                   function(evt) {checkCoords(evt);});
          addEvent(document.body, "resize",
                   function(evt) {handleSize(evt);});
       });
    </script>
</head>
<body>
    <h1>X and Y Event Properties (IE4+ Syntax)</h1>
    <hr />
    <p>Click on any element to see the coordinate values
       for the event object.
    </p>
    <form name="output">
```


更多资源请访问稀酷客(www.ckook.com)

```
<table>
   <tr>
      <td colspan="2">IE Mouse Event Coordinates:</td>
   </tr>
   <tr>
      <td align="right">srcElement:</td>
      <td>
         <input type="text" name="srcElemTag" size="10" />
      </td>
   </tr>
   <tr>
      <td align="right">clientX, clientY:</td>
      <td>
         <input type="text" name="clientCoords" size="10" />
      </td>
      <td align="right">...With scrolling:</td>
      <td>
         <input type="text" name="pageCoords" size="10" />
      </td>
   </tr>
   <tr>
      <td align="right">offsetX, offsetY:</td>
      <td>
         <input type="text" name="offsetCoords" size="10" />
      </td>
   </tr>
   <tr>
      <td align="right">screenX, screenY:</td>
      <td>
         <input type="text" name="screenCoords" size="10" />
      </td>
   </tr>
   <tr>
      <td align="right">x, y:</td>
      <td>
         <input type="text" name="xyCoords" size="10" />
      </td>
      <td align="right">...Relative to:</td>
      <td>
         <input type="text" name="parElem" size="10" />
      </td>
   </tr>
   <tr>
      <td align="right">
         <input type="button" value="Click Here" />
      </td>
   </tr>
   <tr>
      <td colspan="2"><hr /></td>
   </tr>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <tr>
               <td colspan="2">Window Resize Coordinates:</td>
            </tr>
            <tr>
               <td align="right">clientX, clientY:</td>
               <td>
                  <input type="text" name="resizeCoords" size="10" />
               </td>
            </tr>
         </table>
      </form>
         <div id="display" style="position:relative; left:100">
            <img alt="image" src="nile.gif" width="320" height="240" border="0" />
         </div>
      </body>
   </html>
```

下面是 IE 在程序清单 32-8 载入的页面中执行的任务，以帮助你理解各种坐标属性对之间的关系：

(1) 单击 Click Here 按钮标签中 i 上的点。目标元素是按钮(input)元素，其 offsetParent 是表格单元格元素。offsetY 值非常小，因为单击位置靠近元素自身坐标空间的顶部；不过客户端坐标(和 x,y)是相对于窗口中可视区域的。在 Windows 中，如果最大化浏览器窗口，screenX 和 clientX 值就是相同的；screenY 和 clientY 之差是内容区域上窗框的总高。在窗口没有滚动时，无论是否考虑窗口的滚动，客户端坐标都是相同的。

(2) 记下各种坐标值，然后稍微向下滚动页面(单击滚动条，触发一个事件)，并再次单击按钮上的小点。clientY 值会变小，因为页面相对于可视区域上移了，使区域顶部相对于按钮的高度变小。Windows 仍返回相对于元素自身坐标空间的值，显示了正确的 offset 属性。但 Mac 从 offset 属性中减去了滚动量。

(3) 单击大图像，Windows 和 Mac 上的 client 属性都正确，screen 属性也正确。Windows 上的 x 和 y 属性正确地返回了相对于 img 元素的 offsetParent 的事件坐标，此 offsetParent 是包含图像的 div 元素。不过要注意，浏览器将 div“看成”从图像左侧 10 个像素的位置开始。在 WinIE5.5+ 中，在图像左侧的这 10 个透明像素内单击，就可以单击 div 元素。此填充会自动插入，并影响 x 和 y 属性的坐标。对图像内事件的更可靠度量是 offset 属性。只要页面未滚动，在 Macintosh 版本中情况也是如此，如果页面滚动，就会影响以上值，如步骤(2)所示。

(4) 单击标题下面的第一个 hr 元素。可能需要多次尝试才能单击到该元素(hr 元素显示在 srcElement 框中时，就表示单击到了)。这增强了 client 属性提供元素内坐标的方式(同样，在 Mac 版本中，要除去滚动页面的情形)。单击水平线的最左端，最终会找到(0, 0)坐标。

最后，Windows 用户可以尝试下面两个示例，看看坐标属性有什么不可预期的行为。

(1) 若页面未滚动，就单击页面右侧的任意位置，不要单击文本，这样 body 元素就是 srcElement。由于 body 元素在理论上会填满浏览器窗口的整个内容区域，所以除了 screen 坐标外，所有坐标对都应相同。但 offset 属性比其他属性小两个像素，一般情况下，这一差异在脚本中并不重要，但如果需要精确定位，就应考虑这个差异。由于无法解释的原因，offset 属性从窗口左侧和顶部向内两个像素的空间中度量。在 Macintosh 版本中情况并非如此，从 body 的

更多资源请访问稀酷客(www.ckook.com)

角度看，所有坐标对都相同。

(2) 单击 h1 或 p 元素的文本(在页面顶部长水平线的上方和下方)。理论上，offset 属性应该相对于这些元素所在的矩形区域(毕竟，它们是块元素)，但它们在与 client 属性相同的空间中度量(加上两个像素)。这一意外行为与文本光标无关，因为如果在任何文本框元素内单击，其 offset 属性都相对于其自身的矩形区域。Macintosh 版本没有这个问题。

这些属性大多在 W3C DOM 中，因此 W3C DOM 浏览器也支持它们。在这些浏览器中运行程序清单 32-8 时，浏览器不支持的属性值显示为 undefined。

在配书光盘上的第 43 章讨论在 IE 页面上拖动元素时，有一些示例使用了重要的事件坐标属性。

相关主题：fromElement，toElement 属性。

contentOverflow

值：Boolean，　　　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

其值表示页面内容是否溢出了当前布局的矩形。这个属性主要用于打印，作为将内容从一个页面溢出，放在另一个页面上的基础。

dataTransfer

值：对象，　　　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

dataTransfer 属性是 dataTransfer 对象的引用。在拖放操作(也就是与拖放操作相关的事件)中使用这个对象，不仅可以控制数据从源到目标的传送，还控制该过程中光标的外观。

表 32-7 列出了 dataTransfer 对象的属性和方法。

表 32-7　dataTransfer 对象的属性和方法

属性/方法	返回值	说　明
dropEffect	字符串	这个元素是拖放动作的潜在接收者，它可以使用 ondragenter、ondragover 或 ondrop 事件处理程序，来设置鼠标位于元素上时要显示的光标样式。在此之前，源元素的 ondragstart 事件处理程序必须给 event.effectAllowed 属性赋值。这两个属性可能的字符串值是 copy、link、move 或 none。这些属性对应于用户通常执行文件操作和其他文档操作时显示的 Windows 系统光标。还必须取消所有这些拖放元素事件处理程序(ondragenter、ondragover 和 ondrop)的默认动作(表示将 event.returnValue 设置为 false)
effectAllowed	字符串	设置此属性，是为了响应源元素的 ondragstart 事件，该属性确定要执行哪种拖放动作，其可能的字符串值是 copy、link、move 或 none。此属性值必须匹配目标元素的事件对象的 dropEffect 属性值，另外，还要在 ondragstart 事件处理程序中取消默认动作(表示将 event. returnValue 设置为 false)
clearData([format])	无	删除剪贴板中的数据。如果没有提供格式参数，就清除所有数据。数据格式可以是以下一个或多个字符串：Text、URL、File、HTML、image

更多资源请访问稀酷客(www.ckook.com)

(续表)

属性/方法	返回值	描　　述
getData(format)	字符串	从剪贴板中获取指定格式的数据，格式是以下字符串之一：Text、URL、File、HTML、Image。用户获取数据后，不清空剪贴板，这样就可以在多个顺序操作中获取该数据
setData(format, data)	Boolean	将字符串数据存储到剪贴板中，格式是以下字符串之一：Text、URL、File、HTML、Image。对于非文本数据格式，数据必须是指定内容的路径或 URL 的字符串。如果给剪贴板成功传输了数据，则返回 true

dataTransfer 对象用作数据的通道和控制器，脚本需要将这些数据从一个元素传给另一元素，来响应用户的拖放动作。我们需要跟踪事件处理程序触发的一系列明确定义的操作，也就是在拖放的过程中激活不同的事件时，要在不同的 event 对象实例中调用 dataTransfer 对象。

这一系列操作从源元素开始，ondragstart 事件处理程序通常给 dropEffect 属性赋值，使用 getData()方法明确地捕获传递到事件目标的源对象数据。例如，如果拖动一幅图像，就只传送图像的 URL，该数据可以从该事件的 event.srcElement.src 属性(即图像的 src 属性)中得到。

在目标元素中，必须定义三个事件处理程序：ondragenter、ondragover 和 ondrop。一般前两个事件处理程序只是为 dropEffect 标记元素(它必须匹配拖动操作开始时源元素的 effectAllowed)，并将 event.returnValue 设置为 false，以显示所需的光标。这些动作也在 ondrop 事件处理程序上执行，此事件处理程序还处理目标元素中的目标行为。这时调用 dataTransfer 对象的 getData()方法，以获取拖动操作开始时由 getData()存储的数据。如果想确保另一个事件不会无意中获得该数据，可调用 clearData()方法，从内存中清除数据。

注意，在这里讨论的拖动操作方式中，源元素并不在屏幕上移动(但可以编程实现)，而是像在 Windows Explorer 窗口或者桌面上那样处理拖放操作。对用户来说，可拖放的组件封装在光标中，所以 dataTransfer 对象的属性控制光标在放置点上的外形，以指定在当前的放置操作中要执行的操作类型。Apple 实现与 Safari 2 相同的行为。

示例

在用于 ondrag 事件处理程序的程序清单 26-37 中包含了 dataTransfer 属性的扩展例子。

相关主题：ondragend，ondragenter，ondragleave，ondragover，ondragstart，ondrop 事件处理程序。

fromElement，toElement

值：元素对象，　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari+，Opera+，Chrome+

fromElement 和 toElement 属性允许元素显示光标移入和移出的位置，这些属性将其范围扩展到当前元素之外(一般是相邻的元素)，增强了 onmouseover 和 onmouseout 事件处理程序的功能。

在元素上激活 onmouseover 事件时，光标应在另一个元素上。fromElement 属性保存对该元素的引用。相反，当触发 onmouseout 事件时，光标也在另一个元素上。toElement 属性保存对该元素的引用。

更多资源请访问稀酷客(www.ckook.com)

示例

程序清单 32-9 提供的示例说明了在鼠标指针滚入元素前后，fromElement 和 toElement 属性如何显示其操作。光标滚动到中间的方框中时(一个表格单元格)，其 onmouseover 事件处理程序会显示相关表格单元格的文本，鼠标指针是通过该表格单元格到达中间方框的。

程序清单 32-9　使用 toElement 和 fromElement 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>fromElement and toElement Properties</title>
      <style type="text/css">
         .direction {background-color:#00FFFF; width:100; height:50;
                    text-align:center;}
         #main {background-color:#FF6666; text-align:center;}
      </style>
      <script type="text/javascript">
         function showArrival(evt)
         {
            evt = (evt) ? evt : ((event) ? event : null);
            var direction = (event.fromElement.innerText) ?
               event.fromElement.innerText :
               "parts unknown";
            status = "Arrived from: " + direction;
         }
         function showDeparture(evt)
         {
            evt = (evt) ? evt : ((event) ? event : null);
            var direction = (event.toElement.innerText) ?
               event.toElement.innerText :
               "parts unknown";
            status = "Departed to: " + direction;
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
            addEvent(window, "load", function()
            {
               addEvent(document.getElementById("main"), "mouseover",
                        function(evt) {showArrival(evt);});
               addEvent(document.getElementById("main"), "mouseout",
                        function(evt) {showDeparture(evt);});
            });
         </script>
      </head>
      <body>
         <h1>fromElement and toElement Properties</h1>
         <hr />
         <p>Roll the mouse to the center box and look for arrival information in
            the status bar. Roll the mouse away from the center box and look for
            departure information in the status bar.</p>
         <table cellspacing="0" cellpadding="5">
            <tr>
               <td></td>
               <td class="direction">North</td>
               <td></td>
            </tr>
            <tr>
               <td class="direction">West</td>
               <td id="main" onmouseover="showArrival()"
                   onmouseout="showDeparture()">Roll</td>
               <td class="direction">East</td>
            </tr>
            <tr>
               <td></td>
               <td class="direction">South</td>
               <td></td>
            </tr>
         </table>
      </body>
   </html>
```

这个例子可在浏览器中试验，因为它还暴露了一个潜在的局限性。注册为 toElement 或 fromElement 的元素只有触发一个鼠标事件，才能在浏览器中注册自己。否则，这些属性会使用在序列中注册的下一个元素。例如，如果鼠标指针滚入中间的方框，然后非常快地滚动到页面底部，就可能完全忽略 South 框。显示在状态栏上的文本实际上是 body 元素的内部文本，body 元素捕获了第一个鼠标事件，将自己注册为中间表单元格的 toElement。

相关主题：srcElement 属性。

keyCode

值：整型，　　　　　　　　　　　　　　　　　　　　　　　　　　　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

更多资源请访问稀酷客(www.ckook.com)

对于键盘事件，keyCode 属性返回一个整数，它对应于字符(onkeypress 事件)或键盘字符键(onkeydown 和 onkeyup 事件)的 Unicode 值。这些编码系统存在很大的区别。

如果要得到用户所按键的 Unicode 值(与拉丁字符集中的 ASCII 值相同)，可以在 onkeypress 事件处理程序中使用 keyCode 属性。例如，小写字母 a 返回 97，而大写字母 A 返回 65。对于非字符键，如箭头、页面导航和功能键，onkeypress 事件的 keyCode 属性返回 null。也就是说，onkeypress 事件的 keyCode 属性更像一个字符代码，而不是键代码。

要捕获用户按下的准确键盘键，可使用 onkeydown 或 onkeyup 事件处理程序。对于这些事件，event 对象会捕获与键盘上的特定键对应的数字编码。对于字符键，由于所使用的系统语言不同，数字编码有很大的区别。重要的是字符键不分大小写：键盘上的拉丁字符 A 返回值 65，而不管是否按下了 Shift 键。同时，按下 Shift 键会激活它自己的 onkeydown 和 onkeyup 事件，将 keyCode 值设为 16。其他非字符键，如箭头、页面导航和功能键等，也有自己的编码值。这包含许多细节，包括数字键的特殊代码，它们与字母数字键第一行的数字完全不同。

交叉引用：

参见第 26 章有关键盘事件的部分，了解如何使用 keyCode 属性。

示例

程序清单 32-10 提供了额外的练习场，用户在 textarea 中输入内容时，可以查看三个键盘事件的 keyCode 属性，此页面以后可以用作编写工具，来获取不熟悉的键盘键的准确代码。

程序清单 32-10　显示 keyCode 属性值

```
<html>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>keyCode Property</title>
      <style type="text/css">
         td {text-align:center}
      </style>
      <script type="text/javascript">
         function showCode(which, evt)
         {
            evt = (evt) ? evt : ((event) ? event : null);
               if (evt)
               {
                  document.forms[0].elements[which].value = evt.keyCode;
               }
         }

         function clearEm()
         {
            for (var i = 1; i < document.forms[0].elements.length; i++)
            {
               document.forms[0].elements[i].value = "";
```


更多资源请访问稀酷客(www.ckook.com)

```
            }
        }

        // bind the event handlers
        function addEvent(elem, evtType, func)
        {
            if (elem.addEventListener)
            {
                elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
                elem.attachEvent("on" + evtType, func);
            }
            else
            {
                elem["on" + evtType] = func;
            }
        }
        addEvent(window, "load", function()
        {
            addEvent(document.getElementById("scratchpad"), "keydown",
                    function(evt) {clearEm(); showCode("down", evt);});
            addEvent(document.getElementById("scratchpad"), "keypress",
                    function(evt) {showCode("press", evt);});
            addEvent(document.getElementById("scratchpad"), "keyup",
                    function(evt) {showCode("up", evt);});
        });
    </script>
</head>
<body>
    <h1>keyCode Property</h1>
    <hr />
    <form>
        <p>
            <textarea id="scratchpad" name="scratchpad"
            cols="40" rows="5" wrap="hard"></textarea>
        </p>
        <table cellpadding="5">
            <tr>
                <th>Event</th>
                <th>event.keyCode</th>
            </tr>
            <tr>
                <td>onKeyDown:</td>
                <td><input type="text" name="down" size="3" /></td>
            </tr>
            <tr>
                <td>onKeyPress:</td>
                <td><input type="text" name="press" size="3" /></td>
            </tr>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <tr>
               <td>onKeyUp:</td>
               <td><input type="text" name="up" size="3" /></td>
            </tr>
         </table>
      </form>
   </body>
</html>
```

以下的任务尝试用此页面查看键代码(如果浏览器没有设置为使用英语和基于拉丁语的键盘，结果就可能不同)：

(1) 输入小写字母 a。注意 onkeypress 事件处理程序会显示代码 97，这是拉丁字母表中第一个小写字母的 Unicode(和 ASCII)值，但其他两个事件把键的代码记录为 65。

(2) 通过 Shift 键输入大写字母 A。如果仔细观察，可以看到 Shift 键为 onkeydown 和 onkeyup 事件生成了代码 16，但字符键随后为所有三个事件显示值 65，因为大写字母的 ASCII 值是该字母的键盘键代码。

(3) 按下并释放向下箭头键(确保光标仍然在 textarea 中闪烁，因为键盘事件在 textarea 中监控)。非字符键不触发 onkeypress 事件，但会触发其他事件，并把 40 指定为此键的代码。

(4) 尝试其他非字符键。有些非字符键可能打开对话框或菜单，但也会记录它们的键代码。注意，在 Macintosh 键盘上，并非所有的键都在 MacIE 上注册。

还要注意，在基于 Mozilla 的浏览器中，keyCode 属性不能用于 onkeypress 事件，因为 Mozilla 为 onkeypress 事件使用 charCode 属性，而不是 keyCode。在 showCode()函数中进行以下修改，可使程序清单中的代码能用于所有的现代浏览器：·

```
if (evt)
{
   var charCode = (evt.charCode) ? evt.charCode : evt.keyCode;
   document.forms[0].elements[which].value = charCode;
}
```

相关主题：onkeydown，onkeypress，onkeyup 事件处理程序。

nextPage

值：字符串，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

nextPage 属性只有在 WinIE5.5+页面执行 TemplatePrinter 操作时才可用，其值是下面的字符串中的一个：left、right 或空字符串。

propertyName

值：字符串，　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

只有激活了 onpropertychange 事件，才给 propertyName 属性赋值。

如果脚本更改了属性，就激活 onpropertychange 事件处理程序，属性的字符串名称写入 event.

更多资源请访问稀酷客(www.ckook.com)

propertyName 属性。如果该属性正好是与元素有关的 style 对象属性，propertyName 就是完整的属性引用，例如 style.backgroundColor。

示例

在介绍 onpropertychange 事件处理程序的小节中，查看程序清单 26-45 中这个属性返回的值。

相关主题：onpropertychange 事件处理程序(第 26 章)。

repeat

值：Boolean，　　　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

onkeydown 事件的 repeat 属性表示当前键是否处于重复模式(这在系统的键盘控制面板中设置)。使用这些信息，可防止自动触发重复模式，让浏览器识别多个字符。如果用户长时间按下键盘上的一个键，这个属性将非常有用。在文本框或 textarea 的 onkeydown 事件处理程序中，即使系统进入重复模式，下面的脚本段也防止在文本框或 textarea 中出现多个字符：

```
if (event.repeat)
{
   event.returnValue = false;
}
```

禁用重复模式的默认行为，就不能在文本框中输入重复的字符，直到取消重复模式(即按下另一个键)为止。

相关主题：onkeydown 事件处理程序。

returnValue

值：Boolean，　　　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari1.2+，Opera+，Chrome+

IE4+仍使用原来的方法，来禁止执行事件处理程序的默认行为(即事件处理程序的最后一个语句返回 false)，而 IE4+事件模型提供了一个属性，可在事件处理程序调用的函数中取消默认动作。默认情况下，event 对象的 returnValue 属性值是 true，这表示脚本处理程序完成自己的工作后，由元素处理事件，就好像没有脚本一样。例如，一般的处理是显示输入的字符，单击后导航到链接的 href URL，或单击 Submit 按钮后，提交表单。

但默认行为并不总是执行。例如，对于只允许输入数字的文本框，onkeypress 事件处理程序调用一个函数，来检查每个输入的字符，若它不是数字，就不显示在文本框中。下面的验证函数就由文本框的 onkeypress 事件处理程序调用：

```
function checkIt()
{
   var charCode = event.keyCode;
      if (charCode < 48 || charCode > 57)
      {
         alert("Please make sure entries are numerals only.");
         event.returnValue = false;
```


更多资源请访问稀酷客(www.ckook.com)

```
        }
}
```

使用这个事件处理程序，错误的字符就不显示在文本框中。

注意，这个属性并不替代函数的 return 语句，如果希望给调用语句返回一个值，则除了设置 event.returnValue 属性外，还可以使用一个 return 语句。

示例

第 1 章和第 26 章列举了多个 returnValue 属性的示例，具体而言，是程序清单 26-30、程序清单 26-33、程序清单 26-36、程序清单 26-37、程序清单 26-38 和程序清单 26-44。此外，如果只在 IE4+上运行脚本，第 26 章中的许多其他示例可以取代用 returnValue 属性来取消默认动作的方式。

相关主题：return 语句(第 23 章)。

saveType

值：字符串，　　只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

只有 oncontentsave 事件绑定到 WinIE DHTML 行为文件上，才给 saveType 属性赋值。更多信息请参看 http://msdn.microsoft.com/en-us/library/ms531079%28VS.85%29.aspx。

相关主题：addBehavior()方法。

srcElement

值：元素对象索引　　只读

兼容性：WinIE4+，MacIE4+，NN-，Moz-，Safari1.2+，Opera+，Chrome+

srcElement 属性是事件初始目标的 HTML 元素对象引用。因为事件沿着元素容器层次结构冒泡，并可以在任一层次上处理，所以可以用一个属性指向触发事件的元素。有了元素的引用，就可以读写该元素的属性，或调用它的方法。

示例

程序清单 32-11 简单地演示了 srcElement 属性的功能，它只为 body 元素定义了两个事件处理程序，每个处理程序调用一个函数。onmousedown 和 onmouseup 事件将从其目标向上冒泡，事件处理函数将确定哪个元素是目标，并修改该元素的颜色样式。

脚本还添加了一个额外功能：每个函数还检查目标元素的 className 属性。如果 className 是 bold(段落中三个 span 元素共享的一个类名)，就修改该类的样式表规则，使所有项都有相同的颜色。脚本可以进一步筛选传递给函数的对象，以便在特定的对象或对象组上执行指定的操作。

注意，脚本不必知道页面上对象的任何信息，就能访问每个被单击的对象，这是因为 srcElement 属性提供了操作目标元素需要的全部细节。

程序清单 32-11　使用 srcElement 属性

```
<!DOCTYPE html>
<html>
```


更多资源请访问稀酷客(www.ckook.com)

```
<head>
   <meta http-equiv="content-type" content="text/html;charset=utf-8">
   <title>srcElement Property</title>
   <style type="text/css">
      .bold {font-weight:bold}
      .ital {font-style:italic}
   </style>
   <script type="text/javascript">
      function highlight()
      {
         var elem = event.srcElement;
            if (elem.className == "bold")
            {
               document.styleSheets[0].rules[0].style.color = "red";
            }
            else
            {
               elem.style.color = "#FFCC00";
            }
      }
      function restore()
      {
         var elem = event.srcElement;
            if (elem.className == "bold")
            {
               document.styleSheets[0].rules[0].style.color = "";
            }
            else
            {
               elem.style.color = "";
            }
      }

      // bind the event handlers
      function addEvent(elem, evtType, func)
      {
         if (elem.addEventListener)
         {
            elem.addEventListener(evtType, func, false);
         }
         else if (elem.attachEvent)
         {
            elem.attachEvent("on" + evtType, func);
         }
         else
         {
            elem["on" + evtType] = func;
         }
      }
      addEvent(window, "load", function()
      {
```


更多资源请访问稀酷客(www.ckook.com)

```
            addEvent(document.body, "mousedown", highlight);
            addEvent(document.body, "mouseup", restore);
         });
      </script>
   </head>
   <body>
      <h1>srcElement Property</h1>
      <hr />
      <p>Click on the bolded text, then click on the unbolded text.
         What do you see happen?
      </p>
      <p>One event handler...</p>
      <ul>
         <li>Can</li>
         <li>Cover</li>
         <li>Many</li>
         <li>Objects</li>
      </ul>
      <p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
         <span class="bold">sed do</span>
         eiusmod tempor incididunt
         <span class="ital">ut labore et</span>
         dolore magna aliqua. Ut enim adminim veniam,
         <span class="bold">quis nostrud exercitation</span>
         ullamco laboris nisi ut aliquip ex ea
         <span class="bold">commodo consequat</span>.
      </p>
   </body>
</html>
```

相关主题：fromElement，toElement 属性。

srcFilter

值：字符串，　　只读

兼容性：WinIE4+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

根据 Microsoft 的定义，srcFilter 属性返回用来触发 onfilterchange 事件处理程序的筛选字符串名称。当 event 对象包含该属性时，其值总是 null，至少到 WinIE8 都是如此。

相关主题：onfilterchange 事件处理程序；style.filter 对象。

srcUrn

值：字符串，　　只读

兼容性：WinIE5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

如果在与元素关联的 WinIE 行为中激活事件，并为该行为定义了一个 URN 标识符，srcUrn 属性就返回 URN 标识符的字符串，更多信息请查看 http://msdn.microsoft.com/enus/library/ms531079%28VS.85%29.aspx。

相关主题：addBehavior()方法。

更多资源请访问稀酷客(www.ckook.com)

toElement

详见 fromElement。

type

值：字符串， 只读

兼容性：WinIE4+，MacIE4+，NN4+，Moz+，Safari+，Opera+，Chrome+

可以通过 type 属性来确定激活了哪种事件来创建当前的 event 对象。该属性值是事件名称的字符串版本(事件名称去掉 IE 加在其上的 on 前缀)。在指定一个事件处理函数，来处理不同类型的事件时，就可以使用这个属性。例如，对象的 onmousedown 和 onclick 事件处理程序可以调用一个函数，在该函数中编写了一个分支：对于不同类型的 mousedown 或 click 事件，有不同的处理方法。这并非共享事件处理函数，但此功能有助于脚本结构找到属性。

该属性及其值在 NN6+/Moz/W3C 事件模型中完全兼容。

示例

使用 The Evaluator(第 4 章)查看 type 属性的返回值。在底部文本框中输入以下对象名称，并按 Enter/Return 键：

```
event
```

如有必要，滚动页面，显示出 Results 框，以查看 type 属性，该属性应该显示为 keypress。现在单击 List roperties 按钮，该属性更改为 click。显示这些不同类型的原因在于，event 对象触发了一个函数来显示属性。在文本框中，onkeypress 事件处理程序触发该进程；在按钮中，onclick 事件处理程序执行该操作。

相关主题：所有事件处理程序(第 26 章)。

wheelData

值：整数， 只读

兼容性：WinIE5.5+，MacIE-，NN-，Moz-，Safari-，Opera-，Chrome-

wheelData 属性返回一个整数值，表明为 onmousewheel 事件向哪个方向滚动了鼠标滚轮，返回的值通常是 120 或-120，正值表示鼠标滚轮朝屏幕方向滚动，负值表示鼠标滚轮朝相反方向滚动。

32.8 NN6+/Moz 的 event 对象

属　性	方　法	事件处理程序
altKey	initEvent()	
bubbles	initKeyEvent()	
button	initMouseEvent()	
cancelable	initMutationEvent()	

更多资源请访问稀酷客(www.ckook.com)

(续表)

属　性	方　法	事件处理程序
cancelBubble	initUIEvent()	
charCode	preventDefault()	
clientX	stopPropagation()	
clientY		
ctrlKey		
currentTarget		
detail		
eventPhase		
isChar		
isTrusted		
keyCode		
layerX		
layerY		
metaKey		
originalTarget		
pageX		
pageY		
screenX		
screenY		
shiftKey		
target		
timeStamp		
type		
view		

32.8.1　语法

访问 NN6+/Moz 事件对象的属性和方法：

```
eventObject.property | method([parameters])
```

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

32.8.2　关于 event 对象

虽然 NN6+/Moz 的 event 对象大都基于 W3C DOM Level 2 定义的 event 对象，但它们仍然具备 NN4 event 对象的几个特征，有些属性主要用于向后兼容。但由于 Mozilla 的发展先于 NN4 DOM 和事件模型，因此可以忽略这些内容(如下所述)。应尽可能使用事件模型的 W3C DOM 特

更多资源请访问稀酷客(www.ckook.com)

征。例如，Safari、Opera 和 Chrome 实现了许多 W3C DOM 事件模型，但不包括所有旧的 NN4 属性。

NN6/Moz 事件模型提供了与 IE4+一样的冒泡事件传播模型，但事件模型之间的事件对象引用仍不兼容。在 W3C DOM(也包括 NN4)中，event 对象作为参数明确地传递给事件处理(或事件监听器)函数。在函数中将一个浏览器特定的事件对象赋予变量后，一些重要属性在 IE4+和 W3C DOM 事件模型中有相同的名称。若 Microsoft 在将来的 IE 版本中采用更多的 W3C DOM 事件模型，就会提高兼容性。

本节讨论的 event 对象是为了响应用户或系统事件行为而创建的事件实例。W3C DOM 包括另一个静态 event 对象，事件实例继承了静态 event 对象的许多属性，因此本节包括这些共享属性的详细信息，因为在编写脚本时会经常使用这些 event 对象。

在下面的许多代码中，一些引用以 evt 引用开头。它假设函数中的语句将 event 对象赋给 evt 参数变量：

```
function myFunction(evt) {...}
```

如本章前面所示，因为脚本使用相同的或类似的 event 对象属性，可以在实际运用时平等对待 W3C DOM 和 IE4+event 对象引用。最终的结果一般存储在 evt 变量中。

32.8.3　属性

altKey，ctrlKey，metaKey，shiftKey

值：Boolean，　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

为响应用户或系统行为而创建事件对象时，根据是否按下相应的键来设置这 4 个属性，例如单击时是否按下 Shift 键。如果按下了该键，则将属性赋值为 true，否则值为 false。metaKey 属性对应于 Macintosh 键盘上的 Command 键，但 Wintel 计算机没有注册 Windows 键。

多数情况下，包含该属性的表达式用作 if 结构的条件语句，因为这些属性都是 Boolean 值，所以可在一个条件中组合多个属性。例如，只有同时按下 Shift 和 Control 键，才执行函数的一个分支，其条件如下：

```
if (evt.shiftKey && evt.ctrlKey)
{
   // statements to execute
}
```

相反，如果用户按下 4 个修改键中的任何一个，可使用下面更友好的方式来执行特殊处理：

```
if (evt.shiftKey || evt.ctrlKey || evt.metaKey || evt.altKey)
{
   // statements to execute
}
```

这个方法的作用是为用户提供快捷操作，而不是让他们记住具体的组合键。

更多资源请访问稀酷客(www.ckook.com)

示例

在程序清单 32-6 中，这些属性值用来给许多事件类型设置相应复选框的 checked 属性。

相关主题：无。

bubbles

值：Boolean，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

并不是所有的事件都冒泡。例如，onsubmit 事件只传递到与事件相关的表单元素。不冒泡的事件将其事件对象的 bubbles 属性设置为 false；而冒泡的事件将该属性设置为 true。该属性很少用于通过一个事件处理函数来处理大量事件的场合。可冒泡的事件可能需要特殊处理，而其他事件不需要特殊处理，此时，可在 if 条件语句中使用这个属性：

```
if (evt.bubbles)
{
   // special processing for bubble-able events
}
```

不一定要使用分支结构，只取消冒泡即可。如果告诉非传播事件，它不能传播，它并不在意。

相关主题：cancelBubble 属性。

button

值：整型，　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera8+，Chrome+

button 属性表示激活鼠标事件的按键。在 W3C DOM 中，左键(主键)按钮返回值 0。对于三键鼠标，中键返回 1。多键鼠标中的右键都返回 2。

除主键外，其他鼠标按键很容易触发 mousedown 或 mouseup 事件(不是 onclick 事件)。如果用户按下多个按键，则只注册最近按下的按键。

在各种浏览器上对 button 属性编程时要注意，不同的事件模型为鼠标按键定义了不同的按键值。

示例

在程序清单 32-7 中，button 属性在状态栏上显示。试一试在屏幕按钮上按下鼠标键。

相关主题：无。

cancelable

值：Boolean　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

如果事件是可以取消的，就可以通过脚本阻止执行其默认动作。大多数事件都是可以取消的，但有些不能取消。cancelable 属性允许查询特定的事件对象，确定其事件类型是否为 cancelable。该属性值是 Boolean。可以取消的事件可能需要特殊处理，而其他事件不需要特殊处理，此时，可在 if 条件语句中使用此属性：

更多资源请访问稀酷客(www.ckook.com)

```
if (evt.cancelable)
{
   // special processing for cancelable events
}
```

不一定要使用分支结构，而可以阻止执行事件的默认动作。如果告诉不可取消的事件阻止其默认动作，该事件不会在意。

相关主题：preventDefault()方法。

cancelBubble

值：Boolean，　　读/写

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

cancelBubble 属性是一个 IE4+事件属性，它在 NN6+/Moz 和基于 Gecko、WebKit 和 Presto 的浏览器中也实现了，但未在 W3C DOM 中定义，这是很少见的。该属性的操作与 IE4+相同，也是决定当前事件对象是否在文档的元素包含层次结构中向更高层次冒泡。该属性的默认值为 false，即如果事件设置为可以冒泡，事件则会自动冒泡。

为了禁止当前事件冒泡，可在事件处理函数中将该属性设置为 true。只能对当前事件取消冒泡，下一个触发的事件(若可以冒泡)将自动启用冒泡。

为尽可能向 W3C DOM 移植代码，应使用 stopPropaaation()方法而不是 cancelBubble。而为了确保跨浏览器的兼容性，cancelBubble 是一种安全的选择。

示例

程序清单 32-2 显示了 IE 环境下的 cancelBubble 属性，虽然该程序清单有许多用于 IE5.5+的特性，这个取消冒泡的例子在 IE4 中也有效。

相关主题：stopPropagation()方法。

charCode，keyCode

值：整数，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera-，Chrome+

W3C DOM 事件对象模型明确区分了与键盘的字母数字键相关联的 Unicode 字符和与每个键盘键(不管其字符是什么)相关联的代码。要查看键字符，可使用 onkeypress 事件创建 event 对象，然后查看 event 对象的 charCode 属性，该属性为 a 返回 97、为 A 返回 65，因为它与按键动作关联的字符有关。对于 onkeydown 和 onkeyup 事件，此属性值为 0。

相反，按下字母数字键时，只有来自 onkeydown 和 onkeyup 事件的 keyCode 属性为非 0 值(onkeypress 将属性设置为 0)；对大多数其他非字符键，所有三个事件都把 keyCode 属性设置为非 0 值。通过此属性可以查找非字符键，如箭头、翻页键和功能键。对于字符键，大小写没有区别：无论是否按下 Shift 键，拉丁键盘上的 A 键都返回值 65。但同时，按下 Shift 键会触发其自身的 onkeydown 和 onkeyup 事件，从而将 keyCode 值设置为 16(但在 Safari 中，并不采用这种方式注册修改键)。其他非字符键(箭头、翻页键、功能键等)也有自己的代码。这包含许多细节，包括数字键的特殊代码，它们与字母数字键第一行的数字完全不同。

更多资源请访问稀酷客(www.ckook.com)

交叉引用：

要查看在应用程序中使用 keyCode 属性的示例，请参阅第 26 章中关于键盘事件的部分。

示例

程序清单 32-12 提供了额外的练习场，用户在 textarea 中输入内容时，可以查看三个键盘事件的 charCode 和 keyCode 属性，此页面以后可以用作编写工具，来获取不熟悉的键盘键的代码。

程序清单 32-12　显示 charCode 和 keyCode 属性值

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>charCode and keyCode Properties</title>
      <style type="text/css">
         td {text-align:center}
      </style>
      <script type="text/javascript">
         function showCode(which, evt)
         {
            document.forms[0].elements[which + "Char"].value = evt.charCode;
            document.forms[0].elements[which + "Key"].value = evt.keyCode;
         }
         function clearEm()
         {
            for (var i = 1; i < document.forms[0].elements.length; i++)
            {
               document.forms[0].elements[i].value = "";
            }
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            addEvent(document.getElementById("scratchpad"), "keydown",
```


更多资源请访问稀酷客(www.ckook.com)

```
                    function(evt) {clearEm(); showCode("down", evt);});
            addEvent(document.getElementById("scratchpad"), "keypress",
                    function(evt) {showCode("press", evt);});
            addEvent(document.getElementById("scratchpad"), "keyup",
                    function(evt) {showCode("up", evt);});
        });
    </script>
  </head>
  <body>
    <h1>charCode and keyCode Properties</h1>
    <hr />
    <form>
      <p>
        <textarea id="scratchpad" name="scratchpad"
         cols="40" rows="5" wrap="hard"></textarea>
      </p>
      <table cellpadding="5">
        <tr>
          <th>Event</th>
          <th>event.charCode</th>
          <th>event.keyCode</th>
        </tr>
        <tr>
          <td>onKeyDown:</td>
          <td><input type="text" name="downChar" size="3" /></td>
          <td><input type="text" name="downKey" size="3" /></td>
        </tr>
        <tr>
          <td>onKeyPress:</td>
          <td><input type="text" name="pressChar" size="3" /></td>
          <td><input type="text" name="pressKey" size="3" /></td>
        </tr>
        <tr>
          <td>onKeyUp:</td>
          <td><input type="text" name="upChar" size="3" /></td>
          <td><input type="text" name="upKey" size="3" /></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

在 NN6+/Moz 中，用此页面执行以下任务，来查看键代码(如果浏览器没有设置为使用英语和基于拉丁语的键盘，结果就可能不同)：

(1) 输入小写字母 a。注意 onkeypress 事件处理程序会显示代码 97，这是拉丁字母表中第一个小写字母的 Unicode(和 ASCII)值，但其他两个事件类型将键的代码记录为 65。

(2) 通过 Shift 键输入大写字母 A。如果仔细观察，可看到 Shift 键为 onkeydown 和 onkeyup 事件生成了代码 16。但字符键随后为所有三个事件显示值 65(除非释放了 Shift 键)，因为大写字母的 ASCII 值是该字母的键盘键代码。

更多资源请访问稀酷客(www.ckook.com)

(3) 按下并释放向下箭头键(确保光标仍然在 textarea 中闪烁，因为键盘事件在 textarea 中监控)。对于非字符键，所有三个事件都给 keyCode 属性赋值，但给 charCode 赋予 0，此键的 keyCode 值为 40。

(4) 尝试其他非字符键。有些非字符键可能打开对话框或菜单，但也会记录它们的键代码。

相关主题：onkeydown，onkeypress，onkeyup 事件处理程序。

clientX，clientY，layerX，layerY，pageX，pageY，screenX，screenY

值：整型， 只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C DOM event 对象借鉴了 NN4 和 IE4+事件模型中的鼠标坐标属性，如果在这些浏览器中使用过事件坐标，W3C DOM 兼容浏览器的鼠标坐标属性就没有新的内容了。

与 IE4+中的 event 对象一样，W3C DOM 中 event 对象的 clientX 和 clientY 属性是窗口可视内容区域的坐标，这些值相对于窗口坐标空间，而不是文档坐标空间。但与 IE4+不同的是，不需要在文档中计算坐标的位置，因为另一对 NN/Moz 属性 pageX 和 pageY 自动提供这些信息。如果页面不滚动，则用户坐标和页面坐标的值相同。但事件相对于文档(而不是窗口)的坐标通常更重要，所以 pageX 和 pageY 坐标最常用。

另一对 NN/Moz 属性 layerX 和 layerY 是从已不用的 NN4 层机制中借鉴的术语，但这些属性非常有用。这些坐标是相对于接收事件的元素的定位上下文来计算的。对于文档 body 部分中一般的非定位元素而言，这个定位上下文是 body 元素。这样，对于这些元素，页面和层的坐标就是相同的。但如果创建了定位元素，坐标空间是从该定位元素的左上方计算的。如果使用坐标来帮助编写定位元素的拖动操作，可以将拖动范围限制在定位元素中。

目标元素坐标系统对应于 IE4+的 offsetX 和 offsetY 属性，在 NN6/Moz 中没有该坐标系统，而在 Safari 中有该系统。在 NN6/Moz 中，从页面坐标属性中减去目标元素及其定位上下文的 offsetLeft 和 offsetTop 属性，就可以计算出这些坐标值。例如，要得到鼠标事件在图像中的坐标值，事件处理程序可以这样计算这些值：

```
var clickOffsetX = evt.pageX - evt.target.offsetLeft -
   document.body.offsetLeft;
var clickOffsetY = evt.pageY - evt.target.offsetTop -
   document.body.offsetTop;
```

最后一对坐标属性 screenX 和 screenY 提供了相对于整个视频显示器的值。在所有这些属性中，W3C DOM Level 2 标准只定义了 client 和 screen 坐标。

在多数 W3C DOM 兼容的浏览器中，事件目标包括元素中的文本节点。因为节点不必具备元素的所有属性(如它们没有 offset 属性表示它们在文档中的位置)，可以到目标节点的父节点中获取元素对象，它的 offset 属性提供必要的页面位置信息。当然，只有脚本考虑文本上的鼠标事件时，才需要它。

示例

在程序清单 32-13 的页面中可以看到坐标系统和相关的 NN6/Moz 属性，查看 4 种度量体系的坐标值以及某些计算值。该页面提供了两个可单击的对象，以演示层外对象与层内对象之间

更多资源请访问稀酷客(www.ckook.com)

的区别(不过用户看到的所有对象都是可以单击的，包括文本节点)。图 32-1 给出了在定位层内单击的结果。

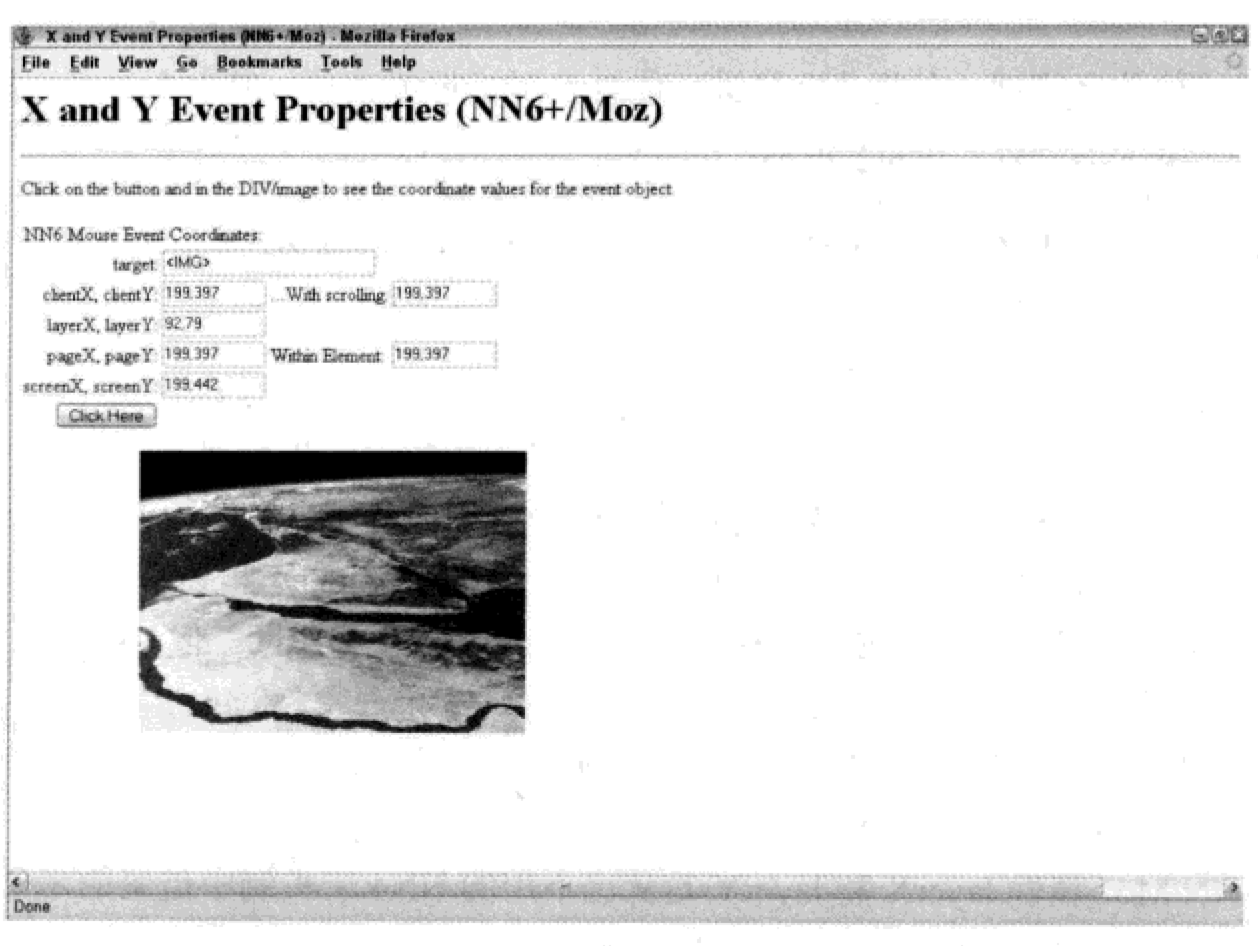


图 32-1　在定位元素内单击的 NN6+/Moz 事件坐标

一个计算域将窗口滚动值应用到 client 坐标上。但这些计算值与更方便的页面坐标相同。另一个计算域显示了相对于目标元素的矩形空间的坐标。注意在代码中，如果目标的 nodeType 表示文本节点，就使用该节点的父节点(元素)来计算。

程序清单 32-13　NN6+/Moz 事件坐标属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>X and Y Event Properties (NN6+/Moz)</title>
      <script type="text/javascript">
         function checkCoords(evt)
         {
            var form = document.forms["output"];
            var targText, targElem;
               if (evt.target.nodeType == 3)
               {
                  targText = "[textnode] inside <" +
                             evt.target.parentNode.tagName + ">";
                  targElem = evt.target.parentNode;
               }
               else
               {
                  targText = "<" + evt.target.tagName + ">";
```


更多资源请访问稀酷客(www.ckook.com)

```
                targElem = evt.target;
            }
            form.srcElemTag.value = targText;
            form.clientCoords.value = evt.clientX + "," + evt.clientY;
            form.clientScrollCoords.value = (evt.clientX + window.scrollX) +
                                         "," + (evt.clientY + window.scrollY);
            form.layerCoords.value = evt.layerX + "," + evt.layerY;
            form.pageCoords.value = evt.pageX + "," + evt.pageY;
            form.inElemCoords.value =
               (evt.pageX - targElem.offsetLeft - document.body.offsetLeft) +
               "," + (evt.pageY - targElem.offsetTop - document.body.offsetTop);
            form.screenCoords.value = evt.screenX + "," + evt.screenY;
            return false;
         }

         // bind the event handler
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            addEvent(document.body, "mousedown",
                     function(evt) {checkCoords(evt);});
         });
      </script>
   </head>
   <body>
      <h1>X and Y Event Properties (NN6+/Moz)</h1>
      <hr />
      <p>Click on the button and in the DIV/image to see the coordinate values
         for the event object.
      </p>
      <form name="output">
         <table>
            <tr>
               <td colspan="2">NN6 Mouse Event Coordinates:</td>
            </tr>
            <tr>
               <td align="right">target:</td>
```


更多资源请访问稀酷客(www.ckook.com)

```
                <td colspan="3"><input type="text" name="srcElemTag"
                <td size="25" /></td>
             </tr>
             <tr>
                <td align="right">clientX, clientY:</td>
                <td><input type="text" name="clientCoords" size="10" /></td>
                <td align="right">...With scrolling:</td>
                <td><input type="text" name="clientScrollCoords" size="10" /></td>
             </tr>
             <tr>
                <td align="right">layerX, layerY:</td>
                <td><input type="text" name="layerCoords" size="10" /></td>
             </tr>
             <tr>
                <td align="right">pageX, pageY:</td>
                <td><input type="text" name="pageCoords" size="10" /></td>
                <td aligh="right">Within Element:</td>
                <td><input type="text" name="inElemCoords" size="10" /></td>
             </tr>
             <tr>
                <td align="right">screenX, screenY:</td>
                <td><input type="text" name="screenCoords" size="10" /></td>
             </tr>
             <tr>
                <td align="right"><input type="button" value="Click Here" /></td>
             </tr>
          </table>
       </form>
       <div id="display" style="position:relative; left:100">
          <img alt="image" src="nile.gif" width="320" height="240" border="0" />
       </div>
    </body>
</html>
```

相关主题：target 属性。

currentTarget

值：元素对象引用， 只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

事件沿着其传播路径传递时，事件监听器会处理该事件。尽管事件知道其目标是什么，但事件监听器函数知道哪个元素的事件监听器在处理事件也是有益的。currentTarget 属性提供了一个元素对象的引用，该元素对象的事件监听器正在处理事件。这就允许一个监听器函数处理不同层次上的事件，创建分支代码，以包含处理事件的不同元素层次。

比较有价值的事件信息是 eventPhase 属性，它有助于事件监听器函数决定事件是处于捕获模式还是冒泡模式，或者事件已到达目标，这个属性将在下一节加以介绍。

示例

程序清单 32-14 表明，currentTarget 属性可在事件传播过程中显示正在处理事件的元素。

更多资源请访问稀酷客(www.ckook.com)

此示例与程序清单 32-3 中的代码类似，但更简单，因为事件对象的属性完成了更多的工作，来显示处理事件的每个元素名称。为不同传播模式指定的事件监听器赋给文档中的各种节点，单击按钮后，传播链中的每个监听器按顺序触发，警告对话框显示哪个节点在处理事件。另外，与程序清单 32-3 一样，eventPhase 属性用于在每个节点处理事件时显示有效的传播模式。

程序清单 32-14　currentTarget 和 eventPhase 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>currentTarget and eventPhase Properties</title>
      <script type="text/javascript">
         function processEvent(evt)
         {
            var currTargTag, msg;
               if (evt.currentTarget.nodeType == 1)
               {
                  currTargTag = "<" + evt.currentTarget.tagName + ">";
               }
               else
               {
                  currTargTag = evt.currentTarget.nodeName;
               }
            msg = "Event is now at the " + currTargTag + " level ";
            msg += "(" + getPhase(evt) + ").";
            alert(msg);
         }
         // reveal event phase of current event object
         function getPhase(evt)
         {
            switch (evt.eventPhase)
            {
               case 1:
                  return "CAPTURING";
                  break;
               case 2:
                  return "AT TARGET";
                  break;
               case 3:
                  return "BUBBLING";
                  break;
               default:
                  return "";
            }
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
```


更多资源请访问稀酷客(www.ckook.com)

```
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            // using old syntax to assign bubble-type event handlers
            document.onclick = processEvent;
            document.body.onclick = processEvent;
            // turn on click event capture for document and form
            document.addEventListener("click", processEvent, true);
            document.forms[0].addEventListener("click", processEvent, true);
            // set bubble event listener for form
            document.forms[0].addEventListener("click", processEvent, false);
            // turn on event capture for the button
            document.getElementById("main1").addEventListener("click",
            processEvent, true);
         });
      </script>
   </head>
   <body>
      <h1>currentTarget and eventPhase Properties</h1>
      <hr />
      <form>
         <input type="button" value="A Button" id="main1" name="main1" />
      </form>
   </body>
</html>
```

也可以单击页面上的其他位置。例如，如果单击按钮的右侧，将单击 form 元素，并相应地调整事件的传播和处理。同理，如果单击标题文本，则只有文档和 body 级的事件监听器能看到事件。

相关主题： eventPhase 属性。

detail

值： 整型，　　只读

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari2+，Opera8+，Chrome+

Detail 是 W3C DOM 规范中的一个额外属性，其作用由浏览器厂商决定。基于 Mozilla、WebKit 和 Presto 的浏览器给对象 click 事件的快速实例增加了一个数字值。

更多资源请访问稀酷客(www.ckook.com)

相关主题：无。

eventPhase

值：整型，只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

事件可能在三个事件阶段中激活：事件捕获、到达目标和冒泡过程。因为同一个事件监听器函数可以在多个阶段处理事件，所以检查事件对象的 eventPhase 属性值，可以确定调用函数时事件处于哪个阶段。该属性的值为整型值 1(捕获)、2(目标)和 3(冒泡)。

示例

查看本章前面程序清单 32-14 中的示例，了解如何根据当前事件对象所处的阶段，使用 switch 结构进行分支处理。

相关主题：currentTarget 属性。

isChar

值：Boolean，只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

检测 isChar 属性，可确定在键盘事件中按下的是否是字符键。然而，该属性最常见的用法是利用捕获键盘动作的事件处理程序，来筛选字符键或非字符键：onkeypress 用于字符键；onkeydown 或 onkeyup 用于非字符键。注意，在 NN6 的第一版中，即使对于同一个键，isChar 属性返回的值也不一致。

相关主题：charCode，keyCode 属性。

isTrusted

值：Boolean，只读

兼容性：WinIE-，MacIE-，NN8+，Moz1.7.5+，Safari-，Opera-，Chrome-

由于事件可由用户动作或脚本触发，所以可以使用 isTrusted 属性来确定事件的触发方式。从安全的角度看，由用户动作触发的事件是可信任的。因此，此属性返回 true，表示事件是由用户活动(单击、按键等)触发的。

originalTarget

值：节点对象引用，只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari-，Opera-，Chrome-

originalTarget 属性提供了一个节点对象的引用，该节点对象用作事件的第一个目标。此信息通常与某些元素的内部构造相关，因此对脚本编程的帮助不大。另外，在许多情况下，onginalTarget 属性的值与 target 属性相同。

relatedTarget

值：元素对象，只读

更多资源请访问稀酷客(www.ckook.com)

兼容性： WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

relatedTarget 属性允许元素显示光标移入和移出的位置。该属性将 onmouseover 和 onmouseout 事件处理程序的范围扩展到当前元素之外(通常是相邻元素)，增强了它们的功能。W3C DOM 的这个属性与 IE4+中 event 对象的 fromElement 和 toElement 属性功能相同。

在元素上激活 onmouseover 事件时，光标应在另一个元素上，relatedtarget 属性保存这个元素的引用。相反，激活 onmouseout 事件时，光标已经在另一个元素上，relatedTarget 属性保存那个元素的引用。

示例

程序清单 32-15 中的示例说明了在鼠标指针滚入元素前后，relatedTarget 属性如何显示鼠标指针。鼠标滚动到中间的方框(一个表格单元格)中时，其 onmouseover 事件处理程序会显示相关表格单元格的文本，鼠标指针是通过该表格单元格到达中间方框的(表格单元格内文本节点的 nodeValue)。如果鼠标指针是从某个角进入该方框的(不容易做到)，将显示不同的消息。

报告结果的两个函数进行了一些筛选，以确保仅当事件发生在一个元素上，且 relatedTarget 元素不是中央单元格元素的嵌套文本节点时，它们才处理事件对象。由于在 W3C DOM 浏览器中，节点会响应事件，因此只要鼠标指针从中央的 td 元素移动到其嵌套文本节点上，此额外的筛选操作就禁止处理事件对象。

程序清单 32-15　使用 relatedTarget 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>relatedTarget Properties</title>
      <style type="text/css">
         .direction {background-color:#00FFFF; width:100; height:50;
                              text-align:center;}
         #main {background-color:#FF6666; text-align:center;}
      </style>
      <script type="text/javascript">
         function showArrival(evt)
         {
            if (evt.target.nodeType == 1)
            {
               if (evt.relatedTarget != evt.target.firstChild)
               {
                  var direction = (evt.relatedTarget.firstChild) ?
                     evt.relatedTarget.firstChild.nodeValue : "parts unknown";
                  window.status = "Arrived from: " + direction;
               }
            }
         }
         function showDeparture(evt)
         {
            if (evt.target.nodeType == 1)
```


更多资源请访问稀酷客(www.ckook.com)

```
            {
               if (evt.relatedTarget != evt.target.firstChild)
               {
                  var direction = (evt.relatedTarget.firstChild) ?
                     evt.relatedTarget.firstChild.nodeValue : "parts unknown";
                  window.status = "Departed to: " + direction;
               }
            }
         }

         // bind the event handlers
         function addEvent(elem, evtType, func)
         {
            if (elem.addEventListener)
            {
               elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
               elem.attachEvent("on" + evtType, func);
            }
            else
            {
               elem["on" + evtType] = func;
            }
         }
         addEvent(window, "load", function()
         {
            addEvent(document.getElementById("main"), "mouseover",
                     function(evt) {showArrival(evt);});
            addEvent(document.getElementById("main"), "mouseout",
                     function(evt) {showDeparture(evt);});
         });
      </script>
   </head>
   <body>
      <h1>relatedTarget Properties</h1>
      <hr />
      <p>Roll the mouse to the center box and look for arrival information in
         the status bar. Roll the mouse away from the center box and look for
         departure information in the status bar.
      </p>
      <table cellspacing="0" cellpadding="5">
         <tr>
            <td></td>
            <td class="direction">North</td>
            <td></td>
         </tr>
         <tr>
            <td class="direction">West</td>
            <td id="main">Roll</td>
```


更多资源请访问稀酷客(www.ckook.com)

```
            <td class="direction">East</td>
        </tr>
        <tr>
            <td></td>
            <td class="direction">South</td>
            <td></td>
        </tr>
    </table>
  </body>
</html>
```

相关主题：target 属性。

target

值：元素对象引用，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

target 属性是 HTML 元素对象的引用，HTML 元素对象是事件的初始目标。由于事件沿着元素包含层次结构滴流和冒泡，并在各层次上处理，所以需要一个属性指向触发事件的元素。有了这个元素的引用，就可以读写这个元素的属性，或调用它的方法。

示例

程序清单 32-16 简单地演示了 target 属性的功能，它只为 body 元素定义了两个事件处理程序，每个处理程序调用一个函数。onmousedown 和 onmouseup 事件将从其目标向上冒泡，事件处理函数将确定哪个元素是目标，并修改该元素的颜色样式。

在脚本中还添加了一个额外功能，即每个函数还检查目标元素的 className 属性。如果 className 是 bold(段落中三个 span 元素共享的一个类名)，就修改该类的样式表规则，以使所有项都有相同的颜色。脚本可以进一步筛选传递给函数的对象，以在特定的对象或对象组上执行特定的操作。

注意，脚本不必知道页面上对象的任何信息，就能单独访问每个被单击的对象，因为 target 属性提供了操作目标元素所需的全部细节。

程序清单 32-16　使用 target 属性

```
<!DOCTYPE html>
<html>
  <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>target Property</title>
      <style type="text/css">
         .bold {font-weight:bold;}
         .ital {font-style:italic;}
      </style>
      <script type="text/javascript">
         function highlight(evt)
         {
            var elem = (evt.target.nodeType == 3) ?
```


更多资源请访问稀酷客(www.ckook.com)

```
                evt.target.parentNode : evt.target;
            if (elem.className == "bold")
            {
                document.styleSheets[0].cssRules[0].style.color = "red";
            }
            else
            {
                elem.style.color = "#FFCC00";
            }
        }
        function restore(evt)
        {
            var elem = (evt.target.nodeType == 3) ?
                evt.target.parentNode : evt.target;
            if (elem.className == "bold")
            {
                document.styleSheets[0].cssRules[0].style.color = "black";
            }
            else
            {
                elem.style.color = "black";
            }
        }

        // bind the event handlers
        function addEvent(elem, evtType, func)
        {
            if (elem.addEventListener)
            {
                elem.addEventListener(evtType, func, false);
            }
            else if (elem.attachEvent)
            {
                elem.attachEvent("on" + evtType, func);
            }
            else
            {
                elem["on" + evtType] = func;
            }
        }
        addEvent(window, "load", function()
        {
            addEvent(document.body, "mousedown",
                     function(evt) {highlight(evt);});
            addEvent(document.body, "mouseup",
                     function(evt) {restore(evt);});
        });
    </script>
</head>
<body>
    <h1>target Property</h1>
```


更多资源请访问稀酷客(www.ckook.com)

```
        <hr />
        <p>One event handler...</p>
        <ul>
           <li>Can</li>
           <li>Cover</li>
           <li>Many</li>
           <li>Objects</li>
        </ul>
        <p>Lorem ipsum dolor sit amet, consectetaur adipisicing elit,
           <span class="bold">sed do</span>
           eiusmod tempor incididunt
           <span class="ital">ut labore et</span>
           dolore magna aliqua. Ut enim adminim veniam,
           <span class="bold">quis nostrud exercitation</span>
           ullamco laboris nisi ut aliquip ex ea
           <span class="bold">commodo consequat</span>.
        </p>
     </body>
  </html>
```

相关主题：relatedTarget 属性。

timeStamp

值：整型， 只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera-，Chrome+

每个事件都接收一个时间戳(以毫秒为单位)，这个时间戳基于与 Date 对象相同的基准日期(1970 年 1 月 1 日)。与 Date 对象一样，该时间戳的准确性完全依赖于用户计算机系统时钟的准确性。

虽然事件的精确时间只在一些情况下有价值，但事件之间的时间间隔对应用程序很有用，例如计时练习和动作游戏。可将最近事件的触发时间保存在全局变量中，并比较当前的时间戳和全局变量保存的值，来决定事件之间的时间间隔。

示例

程序清单 32-17 使用 timeStamp 属性来计算用户在 textarea 中输入数据时的瞬时输入速度。这个计算相当粗略，它只处理了内部击键次数，而没有进行复杂打字程序所执行的平均或平滑处理。计算出的值圆整为最接近的整数。

程序清单 32-17 使用 timeStamp 属性

```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="content-type" content="text/html;charset=utf-8">
      <title>timeStamp Property</title>
      <script type="text/javascript">
         var stamp;
         function calcSpeed(evt)
```


更多资源请访问稀酷客(www.ckook.com)

```
        {
           if (stamp)
           {
              var gross = evt.timeStamp - stamp;
              var wpm = Math.round(6000/gross);
              document.getElementById("wpm").firstChild.nodeValue = wpm + " wpm.";
           }
           stamp = evt.timeStamp;
        }

        // bind the event handlers
        function addEvent(elem, evtType, func)
        {
           if (elem.addEventListener)
           {
              elem.addEventListener(evtType, func, false);
           }
           else if (elem.attachEvent)
           {
              elem.attachEvent("on" + evtType, func);
           }
           else
           {
              elem["on" + evtType] = func;
           }
        }
        addEvent(window, "load", function()
        {
           addEvent(document.getElementById("scratchpad"), "keypress",
                    function(evt) {calcSpeed(evt);});
        });
     </script>
  </head>
  <body>
     <h1>timeStamp Property</h1>
     <hr />
     <p>Start typing, and watch your instantaneous typing speed below:</p>
     <p><textarea id="scratchpad" cols="60" rows="10" wrap="hard"></textarea></p>
     <p>Typing Speed: <span id="wpm"> </span></p>
  </body>
</html>
```

相关主题：Date 对象。

type

值：字符串，　　　　只读

兼容性：WinIE4+，MacIE4+，NN6+，Moz+，Safari+，Opera+，Chrome+

可以通过 type 属性确定激活了哪种事件来创建当前的 event 对象。该属性值是事件名称的字符串版本(事件名称去掉 NN6/Moz/W3C 通常为事件监听器名称加上的 on 前缀)。在指定一个

更多资源请访问稀酷客(www.ckook.com)

事件处理函数，来处理不同类型的事件时，就可以使用这个属性。例如，对象的 onmousedown 和 onclick 事件处理程序可以调用一个函数。在函数内部，用分支结构对不同类型的 mousedown 或 click 事件执行不同的处理方法。这并不是共享事件处理函数，但此功能有助于脚本结构找到属性。

这个属性及其值与 NN4 和 IE4+事件模型完全兼容。

警告：

Safari 早期版本中的键盘事件报告其类型为 khtml_keydown、khtml_keypress 和 khtml_keyup。这可能是为了避免使用未完成的 W3C DOM Level 3 键盘事件规范。Safari 的当前版本报告键盘事件时，没有 khtml 前缀。

相关主题：所有事件处理程序(第 26 章)。

view

值：Window 对象引用，　　只读

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

在 W3C DOM 2 规范中，与浏览器窗口最接近的对象是抽象视图对象(AbstractView 类)，该对象的唯一属性是它所含文档的引用，即根文档节点。用户事件总是在这些视图中发生，并反映在 event 对象的 view 属性中。这个属性包含对 window 对象(可以是一个框架)的引用，而事件在 window 对象中触发。这个引用允许将事件对象传递到其他框架的脚本中，之后，这些脚本就可以访问目标元素所在窗口的 document 对象。

相关主题：window 对象。

32.8.4　方法

initEvent("*eventType*", *bubblesFlag, cancelableFlag*),

initKeyEvent("*eventType*", *bubblesFlag, cancelableFlag, view, ctrlKeyFlag, altKeyFlag, shiftKeyFlag, metaKeyFlag, keyCode, charCode*),

initMouseEvent("*eventType*", *bubblesFlag, cancelable-Flag, view, detailVal, screenX, screenY, clientX, clientY, ctrlKeyFlag, altKeyFlag, shiftKeyFlag, metaKeyFlag, buttonCode, relatedTargetNodeRef*),

initMutationEvent("*eventType*", *bubblesFlag, cancelableFlag, relatedNodeRef, prevValue, newValue, attrName, attrChangeCode*),

initUIEvent("*eventType*", *bubblesFlag, cancelableFlag, view, detailVal*)

返回值：无

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

W3C DOM 的 event 对象初始化方法可以用与该事件相关的完整属性值集来初始化新建的事件。每个初始化方法的参数随着要初始化的事件类型而异。不过，所有初始化方法的前三个参数都相同：eventType、bubblesFlag 和 cancelableFlag。eventType 参数是事件类型的字符串标识符，如 mousedown 或 keypress；bubblesFlag 参数是一个 Boolean 值，它指定事件的默认传播

更多资源请访问稀酷客(www.ckook.com)

行为是(true)否(false)为冒泡；cancelableFlag 参数也是一个 Boolean 值，它指定能(true)否(false)调用 preventDefault()方法来阻止事件执行其默认动作。

一些方法还有 view 和 detailVal 参数，view 对应于事件发生的窗口或框架，detailVal 对应于与事件相关的详细数据的整数代码。一些方法还指定了额外的参数，这些参数是要初始化的事件所特有的。

如果需要简单事件，就不必使用更详细的方法。例如，如果需要一个简单的 mouseup 事件，可以用 initEvent()初始化一个一般事件，并将该事件发送给所需的元素，而不必使用 initMouseEvent()方法的坐标、按钮和其他参数。

```
var evt = document.createEvent("MouseEvents");
evt.initEvent("mouseup", true, true);
document.getElementById("myButton").dispatchEvent(evt);
```

有关 W3C DOM 事件类型，以及每个更复杂初始化方法所需值的详细信息，请参见 http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-eventgroupings。

相关主题：document.createEvent()方法。

preventDefault()

返回值：无。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

NN6+仍使用原来的方式来禁止事件处理程序执行其默认行为(让事件处理程序的最后一个语句返回 false)，而 W3C DOM 事件模型提供了一个方法，可在事件处理程序调用的函数中取消默认行为。例如，对于只允许输入数字的文本框，onkeypress 事件处理程序可以调用一个函数，来检查每个输入的字符。如果输入的不是数字，就不显示在文本框中。下面的验证函数可以由文本框的 onkeypress 事件处理程序调用：

```
function checkIt(evt)
{
   var charCode = evt.charCode;
      if (charCode < 48 || charCode > 57)
      {
         alert("Please make sure entries are numbers only.");
         evt.preventDefault();
      }
}
```

这样，错误的字符就不会显示在文本框中。

在 NN6+/Moz 和基于 Gecko、WebKit 和 Presto 的浏览器中调用 proventDefault()方法，与 IE5+中给 event.returnValue 赋值 true 相同。

相关主题：cancelable 属性。

更多资源请访问稀酷客(www.ckook.com)

stopPropagation()

返回值：无。

兼容性：WinIE-，MacIE-，NN6+，Moz+，Safari+，Opera+，Chrome+

可以使用 stopPropagation()方法停止事件在元素包含层次结构中传递或冒泡。在事件监听器函数中，调用该方法的语句如下：

```
evt.stopPropagation();
```

还可以在元素的事件处理特性中直接取消冒泡：

```
onclick="doButtonClick(this); event.stopPropagation()"
```

如果编写跨浏览器的脚本，还可以选用 cancelBubble 属性，它与 IE4+兼容。

相关主题：bubbles，cancelBubble 属性。

更多资源请访问稀酷客(www.ckook.com)

附录 A

JavaScript 和浏览器对象快速参考

本指南包含核心 JavaScript 语言和浏览器对象模型(自 IE 5.5、Mozilla 和 Safari 以来)的快速参考。对象方框右上角的数字是详细介绍该对象的章节号。

所有的常见浏览器都支持每一项，但如果某项有一个上标，则只有该上标表示的浏览器品牌和版本支持该项：

E—Internet Explorer　M—Mozilla　S—Safari　O—Opera　C—Google Chrome

例如，M1.4 表示只有 Mozilla 1.4 或更新版本支持该项，E 表示只有 Internet Explorer 支持该项。

String	15
constructor length prototype	anchor("*anchorName*") big() blink() bold() charAt(*index*) charCodeAt([*i*]) concat(*string2*) fixed() fontcolor(*#rrggbb*) fontsize(*1to7*) fromCharCode(*n1*...)* indexOf("*str*"[,*i*]) italics() lastIndexOf("*str*"[,*i*]) link(*url*) localeCompare() match(*regexp*) replace(*regexp,str*) search(*regexp*) slice(*i,j*) small() split(*char*) strike() sub() substr(*start,length*) substring(*intA*, *intB*) sup() toLocaleLowerCase()

String	15
	toLocaleUpperCase() toLowerCase() toString() toUpperCase() valueOf()
*静态 String 对象的方法	

正则表达式	45
global ignoreCase input^ESC^ lastIndex multiline^EMSC^ lastMatch^EMSC^ lastParen^EMSC^ leftContext prototype rightContext source $1...$9	compile(*regexp*) exec("*string*")* test("*string*")
*返回带 index, input, [0],...[n]属性的数组	

更多资源请访问稀酷客(www.ckook.com)

数　组		18
Constructor	concat(*array2*)	
length	every(*func*[, *thisObj*])[M1.8S3O9.5C]	
prototype	filter(*func*[, *thisObj*])[M1.8S3O9.5C]	
	forEach(*func*[, *thisObj*])[M1.8S3O9.5C]	
	indexOf(*func*[, *thisObj*])[M1.8S3O9.5C]	
	join("*char*")	
	lastIndexOf(*func*[, *thisObj*])[M1.8S3O9.5C]	
	map(*func*[, *thisObj*])[M1.8S3O9.5C]	
	pop()	
	push()	
	reduce()[MS4]	
	reduceRight()[MS4]	
	reverse()	
	shift()	
	slice(*i*,[*j*])	
	some(*func*[, *thisObj*])M[1.8S3O9.5C]	
	sort(*compareFunc*)	
	splice(*i*,*j*[,*items*])	
	toLocaleString()	
	toString()	
	unshift()	

函　数		23
arguments	apply(*this*, *argsArray*)	
caller	call(*this*[,*arg*1[,...*argN*]])	
constructor	toString()	
length	valueOf()	
prototype		

Date		17
constructor	getFullYear()	
prototype	getYear()	
	getMonth()	
	getDate()	
	getDay()	
	getHours()	
	getMinutes()	
	getSeconds()	
	getTime()	
	getMilliseconds()	
	getUTCFullYear()	
	getUTCMonth()	
	getUTCDate()	
	getUTCDay()	
	getUTCHours()	
	getUTCMinutes()	
	getUTCSeconds()	
	getUTCMilliseconds()	
	setYear(*val*)	
	setFullYear(*val*)	
	setMonth(*val*)	
	setDate(*val*)	
	setDay(*val*)	
	setHours(*val*)	
	SetMinutes(*val*)	
	setSeconds(*val*)	
	setMilliseconds(*val*)	
	setTime(*val*)	
	setUTCFullYear(*val*)	
	setUTCMonth(*val*)	
	setUTCDate(*val*)	
	setUTCDay(*val*)	
	setUTCHours(*val*)	

Date		17
	setUTCMinutes(*val*)	
	setUTCSeconds(*val*)	
	setUTCMilliseconds(*val*)	
	getTimezoneOffset()	
	toDateString()	
	toGMTString()	
	toLocaleDateString()	
	toLocaleString()	
	ToLocaleTimeString()	
	toString()	
	toTimeString()	
	toUTCString()	
	parse("*dateString*")*	
	UTC(*dateValues*)*	
*静态 Date 对象的方法		

Math*		16
E	abs(*val*)	
LN2	acos(*val*)	
LN10	asin(*val*)	
LOG2E	atan(*val*)	
LOG10E	atan2(*val1*, *val2*)	
PI	ceil(*val*)	
SQRT1_2	cos(*val*)	
SQRT2	exp(*val*)	
	floor(*val*)	
	log(*val*)	
	max(*val1*, *val2*)	
	min(*val1*, *val2*)	
	pow(*val1*, *power*)	
	random()	
	round(*val*)	
	sin(*val*)	
	sqrt(*val*)	
	tan(*val*)	
*这些都是静态对象 Math 的属性和方法		

Error		21
prototype	toString()	
constructor		
description[E]		
fileName[M]		
lineNumber[M]		
message		
name		
number[E]		

条件语句　21

```
If (condition) {
  statementsIfTrue
}

if (condition) {
  statementsIfTrue
} else {
  statementsIfFalse
}

result = condition ? expr1 : expr2

for ([init expr]; [condition]; [update expr]) {
  statements
}
```


更多资源请访问稀酷客(www.ckook.com)

条件语句 21

```
for (var in object) {
  statements
}

for each ([var] varName in objectRef) {
  statements
}M1.8.1

with (objRef) {
  statements
}

do {
  statements
} while (condition)

while (condition) {
  statements
}

return [value]

switch (expression) {
  case labelN :
    statements
    [break]
  ...
  [default :
    statements]
}

label :
continue [label]
break [label]

try {
  statements to test
}
catch (errorInfo) {
  statements if exception occurs in try block
}
[finally {
  statements to run, exception or not
}]

throw value
```

Number 16

Constructor	toExponential(n)
MAX_VALUE	toFixed(n)
MIN_VALUE	toLocaleString()
NaN	toString([*radix*])
NEGATIVE_INFINITY	toPrecision(n)
POSITIVE_INFINITY	valueOf()
prototype	

Boolean 16

constructor	toString()
prototype	valueOf()

Globals 24

函数

atob()^{M}
btoa()^{M}
decodeURI("*encodedURI*")
decodeURIComponent("*encComp*")
encodeURI("*URIString*")
encodeURIComponent("*compString*")
escape("*string*" [,*1*])
eval("*string*")
isFinite(*number*)
isNaN(*expression*)
Number("*string*")
parseFloat("*string*")
parseInt("*string*" [,*radix*])
toString([*radix*])
unescape("*string*")
unwatch(*prop*)
watch(*prop*, *handler*)

语句

// /*...*/
const
var

操作符 22

操作符	说明
比较操作符	
==	等于
===	严格等于
!=	不等于
!==	严格不等于
>	大于
>=	大于等于
<	小于
<=	小于等于
算术/配对	
+	加(和字符串连接)
-	减
*	乘
/	除
%	取模
++	递增
--	递减
+val	求正
-val	求负
赋值	
=	等于
+=	加法赋值
-=	减法赋值
*=	乘法赋值
/=	除法赋值
%=	取模赋值
<<=	左移赋值
>>=	右移赋值
>>>=	填充 0 赋值
&=	按位与赋值
|=	按位或赋值
^=	按位异或赋值
[]=	解构赋值
布尔运算	
&&	与
||	或
!	非
按位运算	
&	按位与
|	按位或
^	按位异或
~	按位非
<<	左移
>>	右移
>>>	填充 0

更多资源请访问稀酷客(www.ckook.com)

操作符	22
杂项	
,	序列分隔符
delete	属性解构器
in	对象中的项
instanceof	某对象的实例
new	对象构建器
this	对象自引用
? :	条件操作符
typeof	值的类型
void	无返回值

Frameset	27
border[E]	(无)
borderColor[E]	
cols	
frameBorder[E]	
frameSpacing[E]	
rows	

iframe	27
align	
allowTransparency[E]	
contentDocument[E8MSOC]	
contentWindow	
frameBorder[E]	
frameSpacing[E]	
height	
hspace[E]	
longDesc	
marginHeight[E6MSOC]	
marginWidth[E6MSOC]	
name	
noResize	
scrolling	
src	
vspace[E]	
width	

frame	27
allowTransparency[E]	
borderColor[E]	
contentDocument[E8MSOC]	
contentWindow	
frameBorder	
height[ESC]	
longDesc	
marginHeight	
marginWidth	
name	
noResize	
scrolling	
src	
width[ESC]	

popup[E]		27
document	hide()	
isOpen	show()	

Location		28
hash	assign("*url*")	
host	reload([*unconditional*])	
hostname	replace("*url*")	
href		
pathname		
port		
protocol		
search		

history		28
current[M(signed)O]	back()	
Length	forward()	
next[M(signed)]	go(*int* | "*url*")	
previous[M(signed)]		

treeWalker[MSOC]		29
currentNode	firstChild()	
expandEntityReference	lastChild()	
filter	nextNode()	
root	nextSibling()	
whatToShow	parentNode()	
	previousNode()	
	previousSibling()	

更多资源请访问稀酷客(www.ckook.com)

Window		27
appCore[M]	addEventListener("evt", func,capt)[MS]	onabort[M]
clientInformation[ES1.2C]	alert("msg")	onafterprint[E]
clipboardData[E]	attachEvent("evt", func)[E]	onbeforeprint[E]
closed	back()[M]	onbeforeunload[EMSC]
Components[][M]	blur()	onblur
contentM	clearInterval(ID)	onclick
controllers[][M]	clearTimeout(ID)	onclose
cryptoM	close()	onerror[EM]
defaultStatus[EMSO]	confirm("msg")	onfocus
dialogArguments[EMSC]	createPopup()[E]	onhelp[E]
dialogHeight[E]	detachEvent("evt", func)[E]	onkeydown
dialogLeft[E]	dispatchEvent()[MS]	onkeypress
dialogTop[E]	dump("msg")[M1.4]	onkeyup
dialogWidth[E]	execScript("exprList" [, lang])[E]	onload
directories[M]	find(["str" [, case[, up]]])[MSC]	onmousedown
document	fireEvent("evt" [, evtObj])[E]	onmousemove
event/Event	focus()	onmouseout
external[EM]	forward()[M]	onmouseover
frameElementEMS1.2OC	geckoActiveXObject(ID)[M1.4]	onmouseup
frames[]	getComputedStyle(node, "")[MSOC]	onmove
fullScreen[M1.4]	getSelection()[MSOC]	Onreset
history	home()[MO]	Onresize
innerHeight[MSOC]	moveBy(Δx, Δy)[EMSO]	onscroll[EMS1.3OC]
innerWidth[MSOC]	moveTo(x, y)[EMSO]	onselect
length	navigate("url")[EO]	onsubmit
location	open("url", "name" [, specs])	onunload
locationbar[M]	openDialog("url", "name" [, specs])[M]	
menubar[M]	print()	
name	prompt("msg", "reply")	
navigator	removeEventListener("evt", func,capt)[MS]	
netscape[M]	resizeBy(Δx, Δy)[EMS1]	
offscreenBuffering[ES1.2C]	resizeTo(width, height)[EMS1]	
opener	scroll()	
outerHeight[MSOC]	scrollBy(Δx, Δy)	
outerWidth[MSOC]	scrollByLines(n)[M]	
pageXOffset[MSOC]	scrollByPages(n)[M]	
pageYOffset[MSOC]	scrollTo(x, y)	
parent	setInterval(func, msecs[, args])	
personalbar[M]	setTimeout(func, msecs[, args])	
pkcs11[M]	showHelp("url")[E]	
prompter[M]	showModalDialog("url"[,args][, features])[EMS2.01C]	
returnValue[EMS]	showModelessDialog("url" [, args][, features])[E]	
screen	sizeToContent()[M]	
screenLeft[ES1.2OC]	stop()[MSOC]	
screenTop[ES1.2OC]		
screenX[MS1.2OC]		
screenY[MS1.2OC]		
scrollbars[M]		
scrollMaxX[M1.4]		
scrollMaxY[M1.4]		
scrollX[MSOC]		
scrollY[MSOC]		
self		
sidebar[M]		
status[EMSO]		
statusbar[M]		
toolbar[M]		
top		
window		

更多资源请访问稀酷客(www.ckook.com)

	document	29
activeElement	clear()	onselectionchange[E]
alinkColor	close()	onstop[E]
anchors[]	createAttribute("*name*")[E6MSOC]	
applets[]	createCDATASection("*data*")[MSOC]	
baseURI[M]	createComment("*text*")[E6MSOC]	
bgColor	createDocumentFragment()[E6MSOC]	
body	createElement("*tagname*")	
charset[ESOC]	createElementNS("*uri*", "*tagname*")	
characterSet[MSOC]	createEvent("*evtType*")[MSOC]	
compatMode	createEventObject([*evtObj*])[E]	
contentType[M]	createNSResolver(*nodeResolver*)[MSOC]	
cookie	createRange()[MSOC]	
defaultCharset[ESC]	createStyleSheet(["*url*" [, *index*]])[E]	
defaultView[MSOC]	createTextNode("*text*")	
designMode	createTreeWalker(*root*, *what*, *filterfunc*, *exp*)[M1.4SOC]	
doctype	elementFromPoint(*x*, *y*)	
documentElement	evaluate("*expr*", *node*, *resolver*, *type*, *result*)[MSOC]	
documentURI[M1.7SOC]	execCommand("*cmd*" [, *UI*][, *param*])[EM1.3S1.3OC]	
domain	getElementById("*ID*")	
embeds[]	getElementsByName("*name*")	
expando[E]	hasFocus()[EMSC]	
fgColor	importNode(*node*, *deep*)[MSOC]	
fileCreatedDate[E]	open(["*mimetype*"][, "*replace*"])	
fileModifiedDate[E]	queryCommandEnabled("*commandName*")[EM1.3SOC]	
fileSize[E]	queryCommandIndterm("*commandName*")[EM1.3SC]	
forms[]	queryCommandState("*commandName*")[EM1.3SOC]	
frames[]	queryCommandSupported("*commandName*")[ESOC]	
height[MSC]	queryCommandText("*commandName*")[E]	
images[]	queryCommandValue("*commandName*")[EM1.3SOC]	
implementation[E6MSOC]	recalc([*all*])[E]	
inputEncoding[M1.8SC]	write("*string*")	
lastModified	Writeln("*string*")	
linkColor		
links[]		
location		
media[E]		
mimeType[E]		
nameProp[E6]		
namespaces[][E]		
parentWindow[EO]		
plugins[]		
protocol[E]		
referrer		
scripts[][ESOC]		
security[E]		
selection[EO]		
strictErrorChecking[M1.8]		
styleSheets[]		
title		
URL		
URLUnencoded		
vlinkColor		
width[MSC]		
xmlEncoding[M1.8SC]		
xmlStandalone[M1.8SC]		
xmlVersion[M1.8SC]		

更多资源请访问稀酷客(www.ckook.com)

所有 HTML 元素对象 26

- accessKey
- all[][EO]
- attributes[]
- baseURI[MSOC]
- behaviorUrns[][E]
- canHaveChildren[E]
- canHaveHTML[E]
- childNodes[]
- children[EMS1.2OC]
- cite[E6MSOC]
- className
- clientHeight
- clientLeft
- clientTop
- clientWidth
- contentEditable[EMS1.2OC]
- currentStyle[E]
- dateTime[E6MSOC]
- dataFld[E]
- dataFormatAs[E]
- dataSrc[E]
- dir
- disabled
- document[ES1.2O]
- filters[][E]
- firstChild
- height
- hideFocus[E]
- id
- innerHTML
- innerText[ESOC]
- isContentEditable[ES1.2OC]
- isDisabled[E]
- isMultiLine[E]
- isTextEdit[E]
- lang
- language[E]
- lastChild
- length
- localName[E8MSOC]
- namespaceURI[E8MSOC]
- nextSibling
- nodeName
- nodeType
- nodeValue
- offsetHeight
- offsetLeft
- offsetParent
- offsetTop
- offsetWidth
- outerHTML[ES1.3OC]
- outerText[ES1.3OC]
- ownerDocument
- parentElement[ES1.2OC]
- parentNode
- parentTextEdit[E]
- prefix[E8MSOC]
- previousSibling
- readyState[E]
- recordNumber[E]
- runtimeStyle[E]
- scopeName[E]
- scrollHeight
- scrollLeft
- scrollTop
- scrollWidth
- sourceIndex[ESOC]
- style
- tabIndex
- tagName
- tagUrn[E]
- textContent[M1.7SOC]
- title
- uniqueID[E]
- unselectable[EO]
- width

- addBehavior("*url*")[E]
- addEventListener("*evt*", *func*,*capt*)[MSOC]
- appendChild(*node*)
- applyElement(*elem*[, *type*])[E]
- attachEvent("*evt*", *func*)[EO]
- blur()
- clearAttributes()[E]
- click()
- cloneNode(*deep*)
- compareDocumentPosition(node)[M1.4SOC]
- componentFromPoint(*x*, *y*)[E]
- contains(*elem*)[ESOC]
- createControlRange()[E]
- detachEvent("*evt*", *func*)[EO]
- dispatchEvent(*evtObj*)[MSOC]
- doScroll("*action*")[E]
- dragDrop()[E]
- fireEvent("*evtType*" [, *evtObj*])[E]
- focus()
- getAdjacentText("*where*")[E]
- getAttribute("*name*" [, *case*])
- getAttributeNode("*name*")[E6MSOC]
- getAttributeNodeNS("*uri*", "*name*")[M]
- getAttributeNS("*uri*", "*name*")[MSOC]
- getBoundingClientRect()
- getClientRects()
- getElementsByTagName("*tagname*")
- getElementsByTagNameNS("*uri*", "*name*")[MSOC]
- getExpression("*attrName*")[E]
- getFeature("*feature*", "*version*")[M1.7.2O]
- getUserData("*key*")[M1.7.2SC]
- hasAttribute("*attrName*")
- hasAttributeNS("*uri*", "*name*")[MSOC]
- hasAttributes()
- hasChildNodes()
- insertAdjacentElement("*where*", *obj*)[ESOC]
- insertAdjacentHTML("*where*", "*HTML*")[ESOC]
- insertAdjacentText("*where*", "*text*")[ESOC]
- insertBefore(*newNode*, *refNode*)
- isDefaultNamespace("*uri*")[M1.7.2SOC]
- isEqualNode(*node*)[M1.7.2SOC]
- isSameNode(*node*)[M1.7.2SOC]
- isSupported("feature", "version")[MSOC]
- item(*index*)
- lookupNamespaceURI("*prefix*")[M1.7.2SOC]
- lookupPrefix("*uri*")[M1.7.2SOC]
- mergeAttributes(*srcObj*)[E]
- normalize()
- releaseCapture()[E]
- removeAttribute("*attrName*" [, *case*])
- removeAttributeNode(*attrNode*)[E6MSOC]
- removeAttributeNS("*uri*", "*name*")[MSOC]
- removeBehavior(*ID*)[E]
- removeChild(*node*)
- removeEventListener("*evt*", *func*,*capt*)[MSOC]
- removeExpression("*propName*")[E]
- removeNode(*childrenFlag*)[E]
- replaceAdjacentText("*where*", "*text*")[E]
- replaceChild(*newNode*, *oldNode*)
- replaceNode(*newNode*)[E]
- scrollIntoView(*topFlag*)[EMS2.02OC]
- setActive()[E]
- setAttribute("*name*", "*value*" [, *case*])
- setAttributeNode(*attrNode*)[E6MSOC]
- setAttributeNodeNS("*uri*", "*name*")[MSOC]
- setAttributeNS("*uri*", "*name*", "*value*")[MSOC]
- setCapture(*containerFlag*)[E]
- setExpression("*propName*", "*expr*")[E]
- setUserData("*key*", *data*, *handler*)[M1.7.2SC]
- swapNode(*nodeRef*)[E]
- tags("*tagName*")[ESOC]
- toString()
- urns("*behaviorURN*")[E]

- onactivate[E]
- onafterupdate[E]
- onbeforecopy[ES1.3]
- onbeforecut[ES1.3]
- onbeforedeactivate[E]
- onbeforeeditfocus[E]
- onbeforepaste[ES1.3C]
- onbeforeupdate[E]
- onblur
- oncellchange[E]
- onclick
- oncontextmenu[EMSC]
- oncontrolselect[E]
- oncopy[EMS1.3C]
- oncut[EMS1.3C]
- ondataavailable[E]
- ondatasetchanged[E]
- ondatasetcomplete[E]
- ondblclick
- ondeactivate[E]
- ondrag[EMS1.3C]
- ondragend[EMS1.3C]
- ondragenter[EMS1.3C]
- ondragleave[EMS1.3C]
- ondragover[EMS1.3C]
- ondragstart[EMS1.3C]
- ondrop[EMS1.3C]
- onerrorupdate[E]
- onfilterchange[E]
- onfocus
- onfocusin[E]
- onfocusout[E]
- onhelp[E]
- onkeydown
- onkeypress
- onkeyup
- onlayoutcomplete[E]
- onlosecapture[E]
- onmousedown
- onmouseenter[E]
- onmouseleave[E]
- onmousemove
- onmouseout
- onmouseover
- onmouseup
- onmousewheel
- onmove[E]
- onmoveend[E]
- onmovestart[E]
- onpaste[EMS1.3C]
- onpropertychange[E]
- onreadystatechange[EMS1.2OC]
- onresize
- onresizeend[E]
- onresizestart[E]
- onrowenter[E]
- onrowexit[E]
- onrowsdelete[E]
- onrowsinserted[E]
- onscroll
- onselectstart[ES1.3C]

更多资源请访问稀酷客(www.ckook.com)

link		40
charset	(无)	onload[E]
disabled		
href		
hreflang[E6MSOC]		
media		
rel		
rev		
sheet[M]		
styleSheet[E]		
target		
type		

marquee		33
behavior	start()	onbounce[EM]
bgColor	stop()	onfinish[EM]
direction		onstart[EM]
height		
hspace		
loop[EMO]		
scrollAmount		
scrollDelay		
trueSpeed[EM]		
vspace		
width		

html	40
version[E6MSOC]	

ol	41
start	
type	

ul	41
type	

head	40
profile	

li	41
Type	
value	

meta	40
charset[E]	
content	
httpEquiv	
name	
url[E]	

canvas[M1.8S1.3OC]	31
fillStyle	arc(*x*, *y*, *radius*, *start*, *end*, *clockwise*)
globalAlpha	arcTo(*x1*, *y1*, *x2*, *y2*, *radius*)
globalComposite	bezierCurveTo(*cp1x*, *cp1y*, *cp2x*, *cp2y*, *x*, *y*)
Operation	beginPath()
lineCap	clearRect(*x*, *y*, *width*, *height*)
lineJoin	clip()
lineWidth	closePath()
miterLimit	createLinearGradient(*x1*, *y1*, *x2*, *y2*)
shadowBlur	createPattern(*img*, *repetition*)
shadowColor	createRadialGradient(*x1*, *y1*, *radius1*, *x2*, *y2*, *radius2*)
shadowOffsetX	drawImage(*img*, *x*, *y*)
shadowOffsetY	drawImage(*img*, *x*, *y*, *width*, *height*)
strokeStyle	fill()
target	fillRect(x, y, width, *height*)
	getContext(*contextID*)
	lineTo(*x*, *y*)
	moveTo(*x*, *y*)
	quadraticCurveTo(*cpx*, *cpy*, *x*, *y*)
	rect(*x*, *y*, *width*, *height*)
	restore()
	rotate(*angle*)
	save()
	scale(*x*, *y*)
	stroke()
	strokeRect(*x*, *y*, *width*, *height*)
	translate(*x*, *y*)

script	40
defer[E]	
event[E]	
htmlFor[E]	
src	
text	
type	

title	40
text	

base	40
href	
target	

boby		29
alink	createControlRange()[E]	onafterprint[E]
background	createTextRange()[E]	onbeforeprint[E]
bgColor	doScroll("*scrollAction*")[E]	onscroll
bgProperties[E]		
bottomMargin[E]		
leftMargin[E]		
link		
noWrap[E]		
rightMargin[E]		
scroll[E]		
scrollLeft		
scrollTop		
text		
topMargin[E]		
vLink		

Selection	33
anchorNode[MSOC]	addRange(*range*)[MSOC]
anchorOffset[MSOC]	clear()[E]
focusNode[MSOC]	collapse(*node*, *offset*)[MSOC]
focusOffset[MSOC]	collapseToEnd()[MSOC]
isCollapsed[MSOC]	collapseToStart()[MSOC]
rangeCount[MSOC]	containsNode(*node*, *entireFlag*)[MSOC]
type[E]	createRange()[EO]
typeDetail[E]	deleteFromDocument()[MSOC]
	empty()[EO]
	extend(*node*, *offset*)[MSOC]
	getRangeAt(*rangeIndex*)[MSOC]
	removeAllRanges()[MSOC]
	removeRange(*range*)[MSOC]
	selectAllChildren(*elementRef*)[MSOC]
	toString()[MSOC]

更多资源请访问稀酷客(www.ckook.com)

h1…h6	33
align	

br	33
clear	

blockquote, q	33
ciet[E6MSOC]	

font	33
color	
face	
size	

hr	33
align	
color	
noShade	
size	
width	

dl, dt, dd	41
compact	

img		31
align	(无)	onabort[E]
alt		onerror
border		onload
complete		
dynsrc[E]		
fileCreatedDate[E]		
fileModifiedDate[E]		
fileSizeE		
fileUpdatedDate[E]		
height		
href		
hspace		
isMap		
longDesc[E6MSOC]		
loop[E]		
lowsrc[E]		
mimeType[E6]		
name		
nameProp[E]		
naturalHeight[MSC]		
naturalWidth[MSC]		
protocol[E]		
src		
start[E]		
useMap		
vspace		
width		
x[MSC]		
y[MSC]		

TextNode, Text	33
data	appendData("*text*")
	deleteData(*offset*, *count*)
	insertData(*offset*, "*text*")
	replaceData(*offset*, *count*, "*text*")
	splitText(*offset*)
	substringData(*offset*, *count*)

TextRectangle[ES4O9.5C3]	33
bottom	
left	
right	
top	

Range	33
collapsed[MSOC]	cloneContents()[MSOC]
commonAncestorContainer[MSOC]	cloneRange()[MSOC]
endContainer[MSOC]	collapse([*start*])[MSOC]
endOffset[MSOC]	compareBoundaryPoints(*type*,*src*)[MSOC]
startContainer[MSOC]	compareNode(*node*)[MSC]
startOffset[MSOC]	comparePoint(*node*, *offset*)[MSOC]
	createContextualFragment("*text*")[MSOC]
	deleteContents()[MSOC]
	detach()[MSOC]
	extractContents()[MSOC]
	insertNode(*node*)[MSOC]
	intersectsNode(*node*)[MSOC]
	isPointInRange(*node*, *offoffsetset*)[MSC]
	selectNode(*node*)[MSOC]
	selectNodeContents(*node*)[MSOC]
	setEnd(*node*,*offset*)[MSOC]
	setEndAfter(*node*)[MSOC]
	setEndBefore(*node*)[MSOC]
	setStart(*node*,*offset*)[MSOC]
	setStartAfter(*node*)[MSOC]
	setStartBefore(*node*)[MSOC]
	surroundContents(*node*)[MSOC]
	toString()[MSOC]

TextRange[E]	33
boundingHeight	collapse([*start*])
boundingLeft	compareEndPoints("*type*",*range*)
boundingTop	duplicate()
boundingWidth	execCommand("*cmd*"[,*UI*[,*val*]])
htmlText	expand("*unit*")
text	findText("*str*"[,*scope*,*flags*])
	getBookmark()
	inRange(*range*)
	isEqual(*range*)
	move("*unit*"[,*count*])
	moveEnd("*unit*"[,*count*])
	moveStart("*unit*"[,*count*])
	moveToBookmark("*bookmark*")
	moveToElementText(*elem*)
	moveToPoint(*x*,*y*)
	parentElement()
	pasteHTML("*HTMLText*")
	queryCommandEnabled("*cmd*")
	queryCommandIndeterm("*cmd*")
	queryCommandState("*cmd*")
	queryCommandSupported("*cmd*")
	queryCommandText("*cmd*")
	queryCommandValue("*cmd*")
	select()
	setEndPoint("*type*",*range*)

map	31
areas[]	
name	

更多资源请访问稀酷客(www.ckook.com)

area	31
alt	
coords	
hash	
host	
hostname	
href	
noHref	
pathname	
port	
protocol	
search	
shape	
target	

textarea		36
cols	createTextRange()[E]	onafterupdate[E]
form	select()	onbeforeupdate[E]
name		onchange
readOnly		onerrorupdate[E]
rows		
type		
value		
wrap[E]		

select		37
form	add(*newOption*[, *index*])[E]	onchange
length	add(*newOption*, *optionRef*)[MS]	
multiple	item(*index*)[EMSO9]	
name	namedItem("*optionID*")[EMSO9]	
options[]	options[i].add(*elementRef*[, *index*])[ESOC]	
options[i].defaultSelected	options[i].remove()	
options[i].index	remove(*index*)	
options[i].selected		
options[i].text		
options[i].value		
selectedIndex		
size		
type		
value		

a	30
charset[E6MSOC]	
coords[E6MSOC]	
hash	
host	
hostname	
href	
hreflang[E6MSOC]	
methods[E]	
mimeType[E]	
name	
nameProp[E]	
pathname	
port	
protocol	
rel	
rev	
search	
shape[E6MSOC]	
target	
type[E6MSOC]	
urn[E]	

option	37
defaultSelected	
form	
label[E6MSOC]	
selected	
text	
value	

fieldset, legend	33/34
align	
form	

label	33/34
accessKey	
form	
htmlFor	

form		34
acceptCharset	reset()	onreset
action	submit()	onsubmit
autocomplete[E]		
elements[]		
encoding[E6MSOC]		
enctype[E6MSOC]		
length		
method		
name		
target		

optgroup[E6MSOC]	37
form	
label	

Caption	41
align	
vAlign	

input		35/36/37
checked[(†)]	blur()[(†††)]	onafterupdate[E(†††)]
complete[E(image)]	click()[(††)]	onbeforeupdate[E(†††)]
defaultChecked[(†)]	focus()[(†††)]	onblur[(†††)]
defaultValue[(††††)]	select()[(††††)]	onchange[(††††)]
form		onclick[(††)]
maxLength[(†††)]		onerrorupdate[E(†††)]
name		onfocus[(†††)]
readOnly[(††††)]		onmousedown[(button)]
size[(††††)]	† *checkbox, radio*	onmouseup[(button)]
src(image)	†† *button, checkbox, radio*	onselect[(†††)]
type	††† *text, password, hidden*	
value	†††† *text, password, hidden, file*	

screen	42
availHeight	
availLeft[MSC]	
availTop[MSC]	
availWidth	
bufferDepth[E]	
colorDepth	
fontSmoothingEnabled[E]	
height	
pixelDepth	
updateInterval[E]	
width	

更多资源请访问稀酷客(www.ckook.com)

table	41
align	createCaption()
background[E]	createTFoot()
bgColor	createTHead()
border	deleteCaption()
borderColor[E]	deleteRow(i)
borderColorDark[E]	deleteTFoot()
borderColorLight[E]	deleteTHead()
caption	firstPage()[E]
cellPadding	insertRow(i)
cells[E]	lastPage()[E]
cellSpacing	moveRow(*srcIndex, destIndex*)[E]
cols[E]	nextPage()[E]
datePageSize[E]	previousPage()[E]
frame	refresh()[E]
height	
rows	
rules	
summary[E6MSOC]	
tBodies	
tFoot	
tHead	
width	

tbody, tfoot, thead	41
align	deleteRow(i)
bgColor	insertRow(i)
ch[E6MSOC]	moveRow(*srcIndex, destIndex*)[E]
chOff[E6MSOC]	
rows	
vAlign	

tr	41
align	deleteCell(i)
bgColor	insertCell(i)
borderColor[E]	
borderColorDark[E]	
borderColorLight[E]	
cells	
ch[E6MSOC]	
chOff[E6MSOC]	
height[EO]	
rowIndex	
sectionRowIndex	
vAlign	

col,colgroup	41
align	
ch[E6MSOC]	
chOff[E6MSOC]	
span	
vAlign	
width	

td,th	41
abbr[E6MSOC]	
align	
axis[E6MSOC]	
background[E]	
bgColor	
borderColor[E]	
borderColorDark[E]	
borderColorLight[E]	
cellIndex	
ch[E6MSOC]	
chOff[E6MSOC]	
colSpan	
headers[E6MSOC]	
height	
noWrap	
rowSpan	
vAlign	
width	

navigator	42
appCodeName	javaEnabled()[MSOC]
appMinorVersion[EO]	preference(*name*[, *val*])[M(signed)]
appName	
appVersion	
browserLanguage[EO]	
cookieEnabled	
cpuClass[E]	
language[MSOC]	
mimeTypes[MSOC]	
onLine	
oscpu[M]	
platform	
plugins[MSOC]	
product[MSC]	
productSub[MSC]	
securityPolicy[M]	
systemLanguage[EO]	
userAgent	
userLanguage[EO]	
userProfile[E]	
vendor[MSC]	
vendorSub[MSC]	

event	32
altKey	initEvent()[MS]
altLeft[E]	initKeyEvent()[MS]
behaviorCookie[E6]	initMouseEvent()[MS]
behaviorPart[E6]	initMutationEvent()[MS]
bookmarks[E6]	initUIEvent()[MS]
boundElements[E6]	preventDefault()[MS]
bubbles[MS]	stopPropagation()[MS]
button	
cancelable[MSOC]	
cancelBubble	
charCode[MSC]	
clientX	
clientY	
contentOverflow[E5.5]	
ctrlKey	
ctrlLeft[E]	
currentTarget[MSOC]	
dataFld[E6]	
dataTransfer[E]	
detail[MS2O8C]	
eventPhase[MSOC]	
fromElement[ESOC]	
isChar[M]	
isTrusted[M1.7.5]	
keyCode[MSC]	
layerX[MSC]	
layerY[MSC]	
metaKey	
nextPage[E]	
offsetX[ESOC]	
offsetY[ESOC]	
originalTarget[M]	
pageX[MSOC]	
pageY[MSOC]	
propertyName[E]	
qualifier[E6]	
reason[E6]	
recordset[E6]	
relatedTarget[MSOC]	
repeat[E]	
returnValue[ES1.2OC]	

更多资源请访问稀酷客(www.ckook.com)

event	32
saveType[E]	
screenX	
screenY	
shiftKey	
shiftLeft[E]	
srcElement[ES1.2OC]	
srcFilter[E]	
srcUrn[E]	
target[MSOC]	
timeStamp[MSC]	
toElement[ESOC]	
type	
view[MSOC]	
wheelData[E]	
x[ESOC]	
y[ESOC]	

mimeType[MSOC]	42
description	
enabledPlugin	
type	
suffixes	

plugin[MSOC]		42
name	refresh()	
filename		
description		
length		

object(object)		23
constructor	hasOwnProperty("*propName*")	
prototype	isPrototypeOf(*objRef*)	
	propertyIsEnumerable("*propName*")	
	toSource()[M]	
	toString()	
	unwatch("*propName*")[M]	
	valueOf()	
	watch("*propName*")[M]	

XMLHttpRequest[EMS1.2OC]			39
readyState	abort()	onreadystatechange	
responseText	getAllResponseHeaders()		
responseXML	getResponseHeader("*headerName*")		
status	open("*method*", "*url*" [, *asyncFlag*])		
statusText	send(*data*)		
	setRequestHeader("*name*", "*value*")		

xml[E]	39
src	
XMLDocument	

applet		44
align	(Applet 方法)	
alt[E6MSOC]		
altHTML[E]		
archive[E6MSOC]		
code		
codeBase		
height		
hspace		
name		
object[E]		
vspace		
width		
(Applet 变量)		

embed			44
align[MSOC]	(Object 方法)	onload()	
height		onscroll()	
hidden[E]			
name			
pluginspage[E]			
src			
units[E]			
width			
(Object 变量)			

object (element)			44
align[ESOC]	(Object 方法)	(Object 变量)oncellchange	
alt[E6]		ondataavailable	
altHTML[E]		ondatasetchanged	
archive[E6MSOC]		ondatasetcomplete	
baseHref[E]		onload	
baseURI[MSOC]		onrowenter	
border[E6MSOC]		onrowexit	
classid[E]		onrowsdelete	
code		onrowsinserted	
codeBase		onscroll	
codeType			
contentDocument[MSOC]			
data			
declare[E6MSOC]			
form			
height			
hspace			
name			
object[E]			
standby[E6MSOC]			
type			
useMap[E6MSOC]			
vspace			
width			
(Object 变量)			

更多资源请访问稀酷客(www.ckook.com)

Style、currentStyle、runtimeStyle 38

文本和字体

color
font
fontFamily
fontSize
fontSizeAdjust[M]
fontStretch[M]
fontStyle
fontVariant[EMS1.3OC]
fontWeight
letterSpacing
lineBreak[E]
lineHeight
quotes[MO]
rubyAlign[E]
rubyOverhang[E]
rubyPosition[E]
textAlign
textAlignLast[E]
textAutospace[E]
textDecoration
textDecorationBlink[E]
textDecorationLineThrough[E]
textDecorationNone[E]
textDecorationOverline[E]
textDecorationUnderline[E]
textIndent
textJustify[E]
textJustifyTrim[E]
textKashidaSpace[E]
textOverflow[E6S1.3C]
textShadow[MS1.2OC]
textTransform
textUnderlinePosition[E]
unicodeBidi
whiteSpace
wordBreak[ESC]
wordSpacing[E6MSOC]
wordWrap[ES1.3C]
writingMode[E]

定位

bottom
height
left
pixelBottom[ESOC]
pixelHeight[ESOC]
pixelLeft[ESOC]
pixelRight[ESOC]
pixelTop[ESOC]
pixelWidth[ESOC]
posBottom[ESOC]
posHeight[ESOC]
posLeft[ESOC]
posRight[ESOC]
posTop[ESOC]
posWidth[ESOC]
position
right
top
width
zIndex

边框和边界

border
borderBottom
borderLeft
borderRight
borderTop
borderBottomColor
borderLeftColor
borderRightColor
borderTopColor
borderBottomStyle
borderLeftStyle
borderRightStyle
borderTopStyle
borderBottomWidth
borderLeftWidth
borderRightWidth
borderTopWidth
borderColor
borderStyle
borderWidth
margin
marginBottom
marginLeft
marginRight
marginTop
mozBorderRadius[M]
mozBorderRadiusBottomLeft[M]
mozBorderRadiusBottomRight[M]
mozBorderRadiusTopLeft[M]
mozBorderRadiusTopRight[M]
outline[M1.8.1S1.2OC]
outlineColor[M1.8.1S1.2OC]
outlineOffset[M1.8.1S1.2OC]
outlineStyle[M1.8.1S1.2OC]
outlineWidth[M1.8.1S1.2OC]
padding
paddingBottom
paddingLeft
paddingRight
paddingTop

表

borderCollapse[EMS1.3OC]
borderSpacing
captionSide[MSOC]
emptyCells[MS1.3OC]
tableLayout

列表

listStyle
listStyleImage
listStylePosition
listStyleType

背景

background
backgroundAttachment[EMS1.2OC]
backgroundColor
backgroundImage
gackgroundPosition
backgroundPositionX[ES1.3OC]
backgroundPositionY[ES1.3OC]
backgroundRepeat

内联显示和布局

clear
clip
clipBottom[E]
clipLeft[E]
clipRight[E]
clipTop[E]
content[MS1.3OC]
counterIncrement[M1.8SOC]
counterReset[M1.8SO9C]
cssFloat[MSOC]
cursor[EMS1.3OC]
direction
display
filter[E]
layoutGrid[E]
layoutGridChar[E]
layoutGridLine[E]
layoutGridMode[E]
layoutGridType[E]
markerOffset[M]
marks[M]
maxHeight[E7MSOC]
maxWidth[MSOC]
minHeight[MSOC]
minWidth[MSOX]
mozOpacity[M]
opacity[M1.7.2S1.2OC]
overflow
overflowX[EM1.8S1.2OC]
overflowY[EM1.8S1.2OC]
styleFloat[EO]
verticalAlign[EMS1.2OC]
visibility
width
zoomE

页面和打印

orphans[MO]
page[M]
pageBreakAfter[EMS1.3OC]
pageBreakBefore[EMS1.3OC]
pageBreakInside[E8MSOC]
size[MO]
widows[MO]

杂项

accelerator[E]
behavior[E]
cssText[EMS1.3OC]
imeMode[EM]

滚动条

scrollbar3dLightColor[EO]
scrollbarArrowColor[EO]
scrollbarBaseColor[EO]
scrollbarDarkShadowColor[EO]
scrollbarFaceColor[EO]
scrollbarHighlightColor[EO]
scrollbarShadowColor[EO]
scrollbarTrackColor[EO]

更多资源请访问稀酷客(www.ckook.com)

styleSheet		38
cssRules[MSO9C]	addimport("*url*" [, *index*])[E]	
cssText[E]	addRule("*selector*", "*spec*" [, *index*])[ESC]	
disabled	deleteRule(*index*)[MSOC]	
hrefEMSO9C	insertRule("*rule*", *index*)[MSOC]	
id[E]	removeRule(*index*)[ESC]	
imports[E]		
media		
ownerNode[MSOC]		
ownerRule[MSOC]		
owningElement[E]		
pages[E]		
parentStyleSheet[EMS]		
readOnly[E]		
rules[ESC]		
title		
type		

style (element)	38
media	
type	

cssRule, rule	38
cssText[MSOC]	
parentStyleSheet[MSOC]	
readOnly[E]	
selectorText	
style	
type[MSOC]	

更多资源请访问稀酷客(www.ckook.com)

附录 B

本书配套光盘内容

配书光盘包含本书中的大量 JavaScript 示例、附录 A 提到的快速参考的电子版以及 29 个随赠章节等。

B.1 系统需求

要使用例子程序，最好在计算机上安装基于 Mozilla 的浏览器(例如，Firefox 3.6+、-或 Camino 2+)、基于 WebKit 的浏览器(例如 Safari 4+或 Google Chrome 5+)或基于 Presto 的浏览器(例如 Opera 10+)。尽管多数脚本都能在这些浏览器上运行，但有些脚本演示的特征只能在有限的浏览器上使用。要编写脚本，可以使用简单的文本编辑器、字处理器或专门的 HTML 编辑器。

B.2 光盘内容

B.2.1 文本编辑器的 JavaScript 程序清单

从本书第III部分开始，几乎所有示例的程序清单都以完整的 HTML 文件格式存储在光盘中。将它们载入浏览器，就可以观察 JavaScript 项的操作。Listings 目录包含示例程序，其中的文件夹名是每章的名称，每个 HTML 文件名是本书中该例程的程序清单号，例如，程序清单 26-1 的文件名为 jsb26-01.html。注意，第 II 部分的教学部分没有程序清单文件，需要手工输入 HTML 或脚本代码。

为了方便操作，Listing 文件夹中的_index.html 文件提供了本书程序清单的 HTML 文件的完整目录，只要希望访问程序清单文件，就可以在浏览器上打开这个文件。若需要频繁访问该索引页，可以在浏览器中设置书签。在一些情况下，使用索引文件访问程序清单文件非常重要，因为必须在相关的框架集中打开几个文件，程序清单文件才能正常工作。通过_index.html 文件访问这些文件，能确保打开框架集。_index.html 文件还为每个程序清单显示了浏览器的兼容级别，这就节省了时间，不会在浏览器中打开不能运行的程序。

PDG

更多资源请访问稀酷客(www.ckook.com)

还可以直接从光盘中打开所有的示例文件，但若将其复制到硬盘上，会加快访问速度，更方便地修改文件。最好从光盘中把 Listings 文件夹复制到硬盘中。

B.2.2 附录 A 的可打印版本

若喜欢附录 A 的快速参考，可以将它打印出来。

B.2.3 本书随赠章节的 PDF 版本

配书光盘包括第Ⅵ部分、第Ⅶ部分、第Ⅷ部分以及第Ⅸ部分的共 29 个随赠章节的英文版内容。另上还包括 4 个附录。

B.3 疑难解答

如果安装或使用光盘程序遇到困难，可以尝试如下解决方案。

- 关闭正在运行的所有反病毒软件。安装程序有时会模拟病毒的行为，让计算机错误地以为遇到了病毒。要注意，安装完成后要打开反病毒软件。
- 关闭所有的运行程序。运行的程序越多，其他程序可用的内存就越少。通常情况下，安装程序会更新文件和程序，如果运行着其他程序，安装可能会不正常。

更多资源请访问稀酷客(www.ckook.com)