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

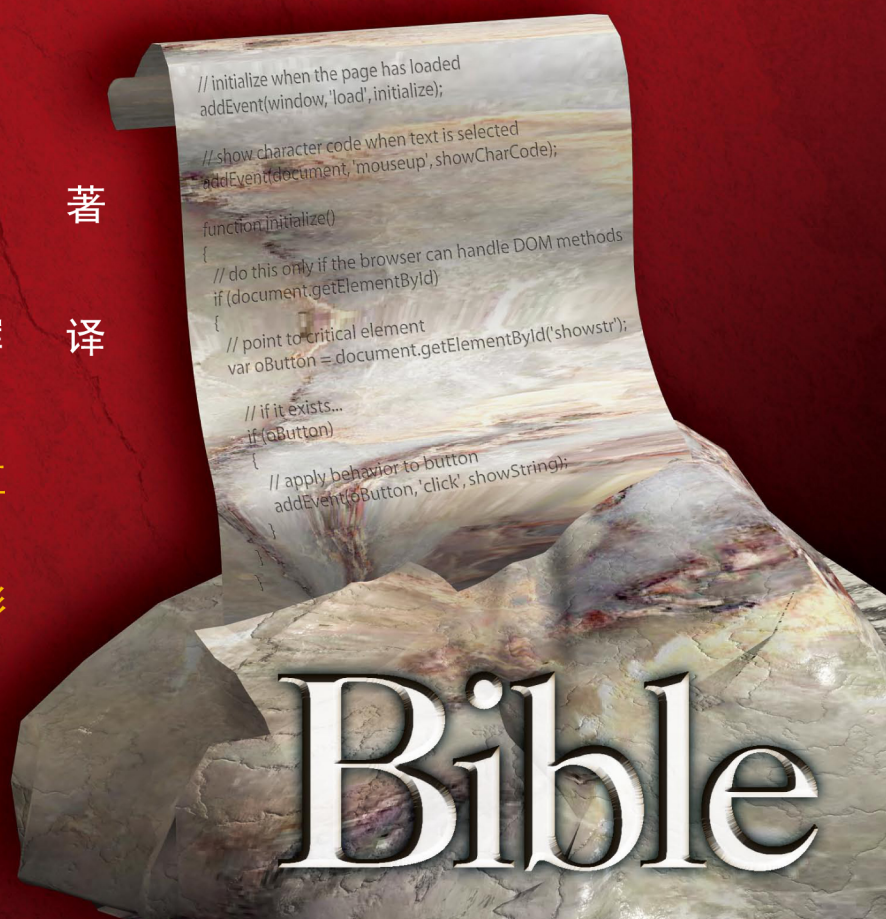
JavaScript Bible, Seventh Edition

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

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

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：
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　　总　　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
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：

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。

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

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

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

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

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。

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。

目　录

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

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。

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。

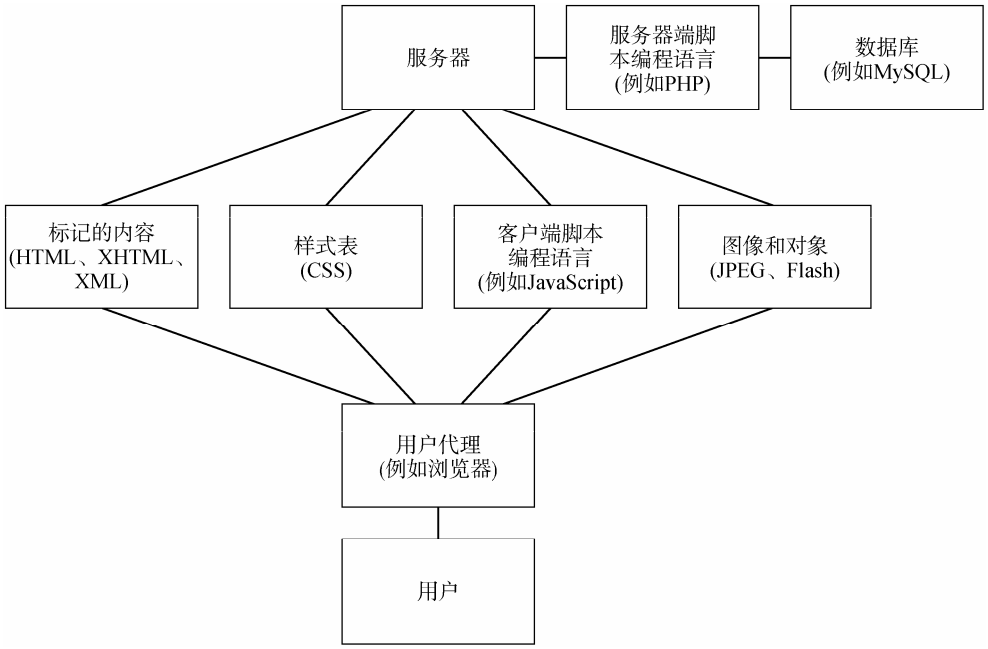


图 1-1　典型动态网站的部件

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

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

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

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

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。

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

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)。

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，以便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。

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

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>
```

```
        <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
```

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

- 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>
```

```
<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，

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

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

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

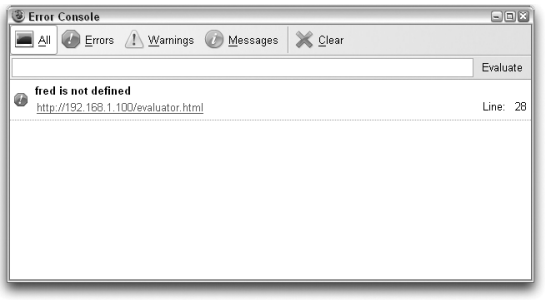


图 7-2　Mozilla 1.4 JavaScript 控制台窗口

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。

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>
```

第24章

全局函数和语句

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

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。

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()	

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。

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。

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+

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

(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

```
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);
```

```
            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];
```

```
            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。

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
}
```

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

```
// 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，

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

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。

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

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。

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

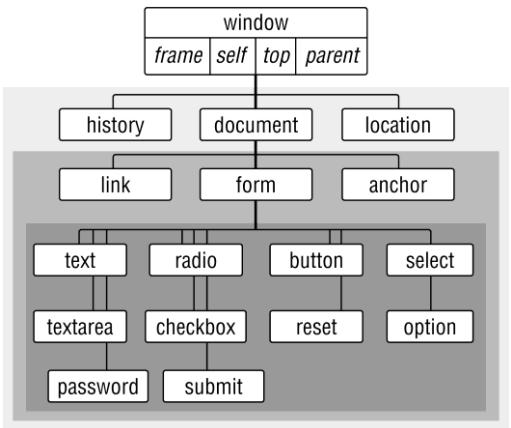


图 29-1　基本的文档对象模型层次结构

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。

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†

(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[]		

(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：

```
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。

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")
            {
```

```
      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()
{
```

```
            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.");
```

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>
```

```
        <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;
```

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+

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。

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

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

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。

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。

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 "";
}
```

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
```

```
// 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;
```

```
      }
   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.
```

```
//    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)
```

```
        {
           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。

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。

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。

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

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

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);
            }
```

```
            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。

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)
            {
```

```
                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>
```

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：

```
"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)。

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

(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

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

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"));
         }
```

```
      // 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">
```

```
                    </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。

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，

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>
```

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
```

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

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。

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

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)，

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

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。

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

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。

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
```

```
            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

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

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" />
```

```
      <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

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

(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

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);
```

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

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。

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。

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＜br＞，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

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；

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++)
            {
```

```
                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."
```

```
                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。

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;
         }
```

```
            // 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

(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。

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。

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。

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。

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;
         }
```

```
      </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)。

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

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，

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。

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()
```

```
{
   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。