SAMS Teach Yourself

- 全球销量逾百万册的系列图书
- 连续十余年打造的经典品牌
- 直观、循序渐进的学习教程
- 掌握关键知识的最佳起点
- “Read Less, Do More”（精读多练）的教学理念
- 以示例引导读者完成最常见的任务

每章内容针对初学者精心设计，1小时轻松阅读学习，
24小时彻底掌握关键知识

每章案例与练习题助你轻松完成常见任务，
通过实践提高应用技能，巩固所学知识

JavaScript 入门经典（第5版）

[美] Phil Ballard
Michael Moncur 著
王军 译

人民邮电出版社
POSTS & TELECOM PRESS

JavaScript 入门经典（第5版）

[美] Phil Ballard
Michael Moncur 著
王军 译

人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

JavaScript入门经典 : 第5版 / (美) 巴拉德
(Ballard,P.) , (美) 蒙库尔 (Moncur,M.) 著 ; 王军译
. -- 北京 : 人民邮电出版社, 2013.9
ISBN 978-7-115-31779-7

Ⅰ. ①J… Ⅱ. ①巴… ②蒙… ③王… Ⅲ. ①
JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2013)第095994号

版权声明

◆ 著　[美] Phil Ballard　Michael Moncur
译　王　军
责任编辑　陈冀康
责任印制　程彦红　杨林杰

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京中新伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：20
字数：491 千字　2013 年 9 月第 1 版
印数：1 – 3 000 册　2013 年 9 月北京第 1 次印刷

著作权合同登记号　图字：01-2012-5643 号

定价：45.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

本书是学习 JavaScript 编程的经典教程。全新的第 5 版涵盖了 JavaScript 1.8 及其以上版本、Ajax 和 jQuery 等内容。本书着力介绍 JavaScript 当今主要特性的基本技巧，从基本概念开始，逐步地介绍按照当今 Web 标准编写 JavaScript 代码的最佳方式。

全书分为七个部分，共 24 章。第一部分“JavaScript 基础”，包括第 1 章到第 5 章，介绍了如何使用常用函数编写简单的脚本，第二部分“JavaScript 进阶”，包括第 6 章到第 10 章，介绍了更复杂的编程范例，比如循环控制、事件处理、面向对象编程、JSON 标记、cookie。第三部分“文档对象模型（DOM）”，包括第 11 章到第 15 章，介绍了如何使用 CSS 遍历和编辑 DOM（文档对象模型）树，对页面元素进行样式代和动画。第四部分“Ajax”，包括第 16 到第 18 章，介绍如何利用 XMLHTTPRequest 对象向服务器进行后台调用，并且处理服务器的响应；建立简单的 Ajax 库，调试 Ajax 应用。第五部分“使用 JavaScript 库”，包括第 19 章到第 21 章，介绍如何使用第三方库，比如 Prototype 和 jQuery，简化跨浏览器的开发工作。第六部分“JavaScript 与其他 Web 技术的配合”，包括第 22 章到第 24 章，通过范例介绍如何使用 JavaScript 控制多媒体、展示 HTML5 的功能、编写浏览器插件。第七部分“附录”介绍了 JavaScript 编程常用工具，并给出了 JavaScript 快速参考。

本书内容循序渐进，逐步深入，概念和知识点讲解清楚，而且每章最后配有练习，供读者检查和巩固所学知识。本书适合对 Web 应用开发感兴趣的初中级中户阅读和自学，也可作为大中专院校相关专业的教材。

作者简介

Phil Ballard 的另一本著作是《Sams Teach Yourself Ajax in 10 Minutes》。他于 1980 年毕业于英国的利兹大学，以优异成绩获得了电子专业的荣誉学位。毕业后，他先是一家跨国公司的研究人员，接着担任了高科技部门的管理职务，然后成为了专职的软件工程顾问。

近些年来，通过“The Mouse Whisperer”项目（www.mousewhisperer.co.uk），Ballard 为全球的客户提供了关于网站、互联网设计与开发的服务。

Michael Moncur 是自由职业者。他作为 Web 站点管理员和作家，管理着多个网站，其中包括最古老的著名报价网站（该网站在 1994 年上线运行）。他曾编写了《Sams Teach Yourself DHTML in 24 Hours》，还著有数本网络、认证和数据库等方面的畅销书。他和妻子现居住在犹他州盐湖城。

我们期待读者的反馈

读者是我们最重要的批评家和评论家。我们非常重视读者的意见，想知道我们哪些方面做的不错，哪些方面还需要改善；想知道读者期望什么领域的图书，以及其他任何有益的建议。

我们欢迎读者反馈信息，读者可以通过电子邮件或书信来表达自己对于本书的看法和建议。

需要提醒的是，我们不能帮助读者解决实际的技术问题。

在向我们反馈信息时，请读者写明图书的名称、作者，以及自己的姓名和电子邮件地址。我们将仔细地查看这些建议，并且将它们转达给作者和编辑。

电子邮件地址：feedback@samspublishing.com

通信地址：Sams Publishing
ATTN:Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

读者服务

读者可以访问 www.informit.com/register，以本书信息进行注册，就可以方便地获得与本书相关的更新、下载和勘误。

前　言

目标读者

对于想学习 JavaScript 的读者来说，很可能已经掌握了 HTML 和 Web 页面设计的基本知识，希望为网页添加一些更好的互动性；或者，目前是在使用其他语言进行编程，想了解一下 JavaScript 能够提供哪些更多的功能。

如果对 HTML 没有任何了解，或是没有任何计算机编程经验，我们建议读者先了解一些 HTML 基本知识。HTML 是非常易于理解的，读者不必成为 HTML 专家就足以了解本书的 JavaScript 范例了。

JavaScript 很适合作为学习编程技术的出发点，在调试过程所掌握的基本概念大多可以用于其他的编程语言，比如 C、Java 或 PHP。

本书的目标

JavaScript 最初的用途是相当有限的，它只具备基本的功能，对于浏览器的支持也很不稳定，所以只被看作花哨的小技巧。现在，随着浏览器对 W3C 标准的支持越来越好，对 JavaScript 的实现不断改善，JavaScript 已经成为一种很正规的编程语言。

其他高级编程语言里的编程规则能够方便地应用于 JavaScript，比如面向对象编程方法有助于编写稳定、易读、易维护和易重用的代码。

“低调”的编程技术和 DOM 脚本都致力于为 Web 页面增加更好的互动，同时保持 HTML 简单易读，并且能够轻松地与代码分离。

本书着力介绍 JavaScript 当今主要特性的基本技巧，从基本概念开始，逐步地介绍按照当今 Web 标准编写 JavaScript 代码的最佳方式。

本书约定

本书全部代码范例都是符合 HTML5 标准的，但考虑到目前并不是所有 Web 浏览器都支持 HTML5，这些范例一般会避免使用 HTML5 特有的语法。无论读者使用什么样的计算机或

操作系统，这些代码都能够在常见浏览器上正常运行。

除了每个课程里的正文之外，书中还有一些标记为“说明”、“提示”或“注意”的方框。

NOTE **说明**：这里的内容提供了额外的解释，帮助读者理解正文和范例。

TIP **提示**：这些方框里的内容提供额外的技巧、提示，帮助读者更轻松地进行编程。

CAUTION **注意**：了解这些内容以避免常见的错误。

实践练习

每章都包括至少一个部分的内容指导读者尝试自己完成脚本，帮助读者建立编写 JavaScript 脚本的信心。

▲

问答、测验和练习

每章的最后都有这三部分内容：

“问答”主要是解答课程中最常遇到的问题；

“测验”用于测试读者对课程内容的掌握情况；

“练习”根据课程的内容提供一些让读者进一步深入学习的建议。

本书结构

本书正文分为六个部分，内容的难度逐步提高。

- 第一部分：JavaScript 基础

第一部分是 JavaScript 语言的基本知识，介绍了如何使用常用函数编写简单的脚本。这部分的内容主要针对缺少或没有编程知识及 JavaScript 知识的读者。

- 第二部分：JavaScript 进阶

这部分介绍更复杂的编程范例，比如循环控制、事件处理、面向对象编程、JSON 标记、cookie。

- 第三部分：文档对象模型（DOM）

这部分内容着重介绍如何使用 CSS 遍历和编辑 DOM（文档对象模型）树，对页面元素进行样式化和动画。其中强调了使用好的编码方式，比如低调 JavaScript。

- 第四部分：Ajax

这部分介绍如何利用 XMLHttpRequest 对象向服务器进行后台调用，并且处理服务器的响应；建立简单的 Ajax 库，调试 Ajax 应用。

- 第五部分：使用 JavaScript 库

这部分介绍如何使用第三方库，比如 Prototype 和 jQuery，简化跨浏览器的开发工作。

- 第六部分：JavaScript 与其他 Web 技术的配合

最后这部分的范例介绍如何使用 JavaScript 控制多媒体、展示 HTML 5 的功能、编写浏览器插件等。

必要工具

编写 JavaScript 并不需要昂贵和复杂的工具，比如集成开发环境（IDE）、编译器或调试器。

本书的范例代码都可以利用像 Windows 记事本这样的文本编辑软件生成。每个操作系统都会提供至少一款这样的软件，而且互联网上还有大量免费或廉价的类似软件。

说明：

附录 A “JavaScript 开发工具”列出的 JavaScript 开发工具和资源都可以方便地获得。

为了查看代码的运行情况，我们需要一个 Web 浏览器，比如 IE、Firefox、Opera、Safari 或 Chrome。建议使用浏览器的最新稳定版本。

本书绝大多数范例代码在运行时并不需要互联网连接，只要把源代码保存到计算机上，然后用浏览器打开它们就可以了。例外的情况是关于 cookie 和 Ajax 的，这些代码需要一个 Web 连接（或者是局域网上的一个 Web 服务连接）和一些 Web 空间来上传代码。对于尝试过 HTML 编码的读者来说，都应该具备上述配置了；即使没有这些配置，使用业余级别的 Web 主机账户就可以满足要求，而这些都是很便宜的。（如果想测试 Ajax 代码，请确认 Web 主机允许运行 PHP 语言脚本，而基本上所有主机都是允许的。）

目　录

第一部分　JavaScript 基础

第七部分 附录

第一部分

JavaScript 基础

第 1 章

了解 JavaScript

本章主要内容包括：

- 关于服务器端和客户端编程
- JavaScript 如何改善 Web 页面
- JavaScript 历史
- “文档对象模型”（DOM）基础知识
- window 和 document 对象
- 如何使用 JavaScript 给 Web 页面添加内容
- 如何利用对话框提示用户

与其只有文本内容的祖先相比，现代的 Web 几乎是完全不同的，它包含了声音、视频、动画、交互导航等很多元素，而 JavaScript 对于实现这些功能扮演了非常重要的角色。

在第 1 章中，我们将简要介绍 JavaScript，回顾它的发展历史，展示它如何能够改善 Web 页面，读者还会直接开始编写一些实用的 JavaScript 代码。

1.1 Web 脚本编程基础

阅读本书的读者很可能已经熟练使用万维网，而且对于使用某种 HTML 编写 Web 页面有一些基本的理解。

HTML 不是编程语言（如其名所示），而是一款标签语言，用于标签页面的各个部分在浏览器里以何种方式展现，比如加粗或斜体字，或是作为标题，或是一系列选项，或是数据表格，或是其他修饰方式。

一旦编写完成，这些页面的本质就决定了它们是*静态*的。它们不能对用户操作做出响应，

不能进行判断，不能调整页面元素显示。无论用户何时访问这些页面，其中的标签都会被以相同的方式进行解析和显示。

根据使用万维网的经验，我们知道网站可以做的事情要多的多。我们时常访问的页面基本上都不是静态的，它们能够包含“活”的数据，比如分享商品价格或航班到达时间，字体和颜色的动画显示，或是点击相册或数据列表这样的交互操作。

这些灵活的功能是通过程序（通常称为“脚本”）来实现的，它们在后台运行，操纵着浏览器显示的内容。

NOTE **说明**：“脚本”这个术语显然来自于话剧和电视领域，那里所用的脚本决定了演员或主持人要做的事情。对于 Web 页面来说，主角是页面上的元素，而脚本是由某种脚本语言（比如 JavaScript）生成的。对于本书描述的内容来说，“程序”与“脚本”两个术语基本上是可以通用的。

1.2 服务器端与客户端编程

给静态页面添加脚本有两种最基本的方式。

- 让 Web 服务器在把页面发送给用户之前执行脚本。这样的脚本可以确定把哪些内容发送给浏览器以显示给用户，比如从在线商店的数据库获取产品价格，在用户登录到站点的私有区域之前核对用户身份，或是从邮箱获取邮件内容。这些脚本通常运行在 Web 服务器上，而且是在生成页面并提供给用户之前运行的。
- 另外一种方式并不是在服务器运行脚本，而是把脚本与页面内容一起发送给用户的浏览器。然后浏览器运行这些脚本，操作已经发送给浏览器的页面内容。这些脚本的主要功能包括动画页面的部分内容，重新安排页面布局，允许用户在页面内拖放元素，验证用户在表单里输入的内容，把用户重定向到其他页面，等等。自然而然，这些脚本称为“客户端脚本”。

NOTE **说明**：有一种很酷的方法可以把来自于服务器端脚本的输出组合到客户端脚本，在本书第四部分介绍 Ajax 技术时将有所涉及。

本书介绍的 JavaScript 是互联网上最广泛应用的客户端脚本。

1.3 JavaScript 简介

用 JavaScript 编写的程序能够访问 Web 页面的元素和运行它的浏览器，对这些元素执行操作，还可以创建新元素。JavaScript 常见的功能包括：

- 以指定尺寸、位置和样式（比如是否具有边框、菜单、工具栏等）打开新窗口；
- 提供给用户友好的导航帮助，比如下拉菜单；
- 检验 Web 表单输入的数据，在向 Web 服务器提交表单之前确保数据格式正确；

- 在特定事件发生时，改变页面元素的外观与行为，比如当鼠标指针经过元素时；
- 检测和发现特定浏览器支持的功能，比如第三方插件，或是对新技术的支持。

由于 JavaScript 代码只在用户浏览器内部运行，页面会对 JavaScript 指令做出快速响应，从而改善用户的体验，让 Web 应用更像在用户本地计算机运行的程序而不是一个页面。另外，JavaScript 能够检测和响应特定的用户操作，比如鼠标点击和键盘操作。

几乎每种 Web 浏览器都支持 JavaScript。

说明：虽然 JavaScript 与 Java 的名称有些相同部分，但两者几乎没有什么联系。虽然它们有一些相同的语法，但这些共同之处并不比与其他语言的共同之处多。 **NOTE**

1.4 JavaScript 起源

JavaScript 的祖先可以追溯到 20 世纪 90 年代中期，首先是 Netscape Navigator 2 引入了 1.0 版本。

随后，“欧洲计算机制造商协会”（ECMA）开始介入，制定了 ECMAScript 规范，奠定了 JavaScript 迅猛发展的基础。与此同时，微软开发了自己版本的 JavaScript：jScript，在 IE 浏览器上使用。

说明：JavaScript 不是仅有的客户端脚本语言，微软的浏览器还支持自己的 Visual Basic 面向脚本的版本：VBScript。 **NOTE**

但是，JavaScript 得到了更好的浏览器支持，现代浏览器几乎都支持它。

浏览器战争

在 20 世纪 90 年代后期，Netscape Navigator 4 和 IE 4 都宣布对 JavaScript 提供更好的支持，比以前版本的浏览器大有改善。

但不幸的是，这两组开发人员走上了不同的道路，分别给 JavaScript 语言本身及如何与 Web 页面交互定义了自己的规范。

这种荒唐的情况导致开发人员总是要编写两个版本的脚本，利用一些复杂的、经常可能导致错误的程序来判断用户在使用什么浏览器，然后再切换到适当版本的脚本。

好在“网际网络联盟”（W3C）非常努力地通过 DOM 规范各个浏览器制作商生成和操作页面的方式。1 级 DOM 于 1998 年完成，2 级版本完成于 2000 年年末。

关于 DOM 是什么或它能做什么，本书的相应章节会有所介绍。

说明：“网际网络联盟”（W3C）是一个国际组织，致力于制定开放标准来支撑互联网的长期发展。其站点 http://www.w3.org/包含了大量与 Web 标准相关的信息与工具。 **NOTE**

1.5 <script>标签

当用户访问一个页面时，页面中包含的 JavaScript 代码会与其他页面内容一起传递给浏览器。

在 HTML 里使用<script>和</script>标签，可以在 HTML 代码里直接包含 JavaScript 语句。

```
<script>
    …JavaScript 语句…
</script>
```

本书的代码都是符合 HTML5 规范的，也就是说，<script>元素没有任何必须设置的属性（在 HTML5 里，type 属性是可选的，本书的范例里都没有使用这个属性）。但如果是在 HTML4.x 或 XHTML 页面里添加 JavaScript，就需要使用 type 属性了：

```
<script type="text/javascript">
    …JavaScript 语句…
</script>
```

编译语言和解释语言
解释语言不需要编译

NOTE

说明：JavaScript 是一种解释型语言，不是 C++或 Java 那样的编译语言。JavaScript 指令以普通文本形式传递给浏览器，然后依次解释执行。它们不必首先“编译”成只有计算机处理器能够理解的机器码，这让 JavaScript 程序很便于阅读，能够迅速地进行编辑，然后在浏览器里重新加载页面就可以进行测试。

偶尔还会看到<script>元素使用属性 language="JavaScript"，这种方式已经被抛弃很久了。除非是需要支持很古老的浏览器，比如 Navigator 或 Mosaic，否则完全不必使用这种方式。

NOTE

说明：“抛弃”这个词对于软件功能或编码方式来说意味着最好避免使用，因为它们已经被新功能或新方式取代了。

虽然为了实现向下兼容而仍然使用这类的功能，但“被抛弃”这个状态通常暗示这样的功能会在不久之后被清除。

本书的范例把 JavaScript 代码放置到文档的 body 部分，但实际上 JavaScript 代码也能出现在其他位置。我们还可以利用<script>元素加载保存在外部文件里的 JavaScript 代码，关于这方面的详细介绍请见第 2 章。

1.6 DOM 简介

DOM浏览器可见内容模型
显示列表，列表中的文
本图片节点等

“文档对象模型”（DOM）是对文档及其内容的抽象表示。

每次浏览器要加载和显示页面时，都需要解释（更专业的术语是“解析”）构成页面的 HTML 源代码。在解析过程中，浏览器建立一个内部模型来代表文档里的内容，这个模型就是 DOM。在浏览器渲染页面的可见内容时，就会引用这个模型。我们利用 JavaScript 可以访问和编辑这个 DOM 模型的各个部分，从而改变页面的显示内容和用户交互的方式。

在早期，JavaScript 只能对 Web 页面的某些部分进行最基本的访问，比如访问页面里的图像和表单，可以选择“页面上第二个表单”，或是“名称为 registration 的表单”。

Web 开发人员有时把这种情形称为 0 级 DOM，以便与 W3C 的 1 级 DOM 向下兼容。0 级 DOM 有时也被称为 BOM（浏览器对象模型）。从 0 级 DOM 开始，W3C 逐渐扩展和改善了 DOM 规范。它更大的野心是不仅让 DOM 能够用于 Web 页面与 JavaScript，也能用于任何编程语言和 XML。

说明：本书使用 1 级和 2 级 DOM 定义。如果想了解各种级别 DOM 的详细内容，可以访问 https://developer.mozilla.org/en/DOM_Levels。

NOTE

1.6.1 W3C 和标准兼容

浏览器制作商在最近的版本中对 DOM 的支持都有了很大的改善。在编写本书时，IE 最新版本是 9，Netscape Navigator 以 Mozilla Firefox 重新出世（当前版本是 9），其他竞争对手还包括 Opera、Konqueror、苹果公司的 Safari、谷歌的 Chrome 和 Chromium，都对 DOM 提供了出色的支持。

Web 开发人员的处境有了很大改善。除了极特殊的一些情况，只要我们遵循 DOM 标准，在编程时基本上可以不考虑为某个浏览器编写特殊代码了。

说明：早期浏览器，比如 Netscape Navigator（任何版本）和 IE 5.5 以前版本，现在基本上已经没有人使用了。本书只关注与 1 级或更高级别 DOM 兼容的现代浏览器，比如 IE 7+、Firefox、Google Chrome、Apple Safari、Opera 和 Konqueror。我们建议读者把自己使用的浏览器升级到最新版本。

NOTE

1.6.2 window 和 document 对象

浏览器每次加载和显示页面时，都在内存里创建页面及其全部元素的一个内部表示体系，也就是 DOM。在 DOM 里，页面的元素具有一个逻辑化、层级化的结构，就像一个由父对象和子对象组成的树形结构。这些对象及其相互关系构成了 Web 页面及显示页面的浏览器的抽象模型。每个对象都有“属性”列表来描述它，而利用 JavaScript 可以使用一些方法来操作这些属性。

这个层级树的最顶端是浏览器 window 对象，它是 DOM 树里一切对象的根。

window 对象具有一些子对象，如图 1.1 所示。图 1.1 中第一个子对象是 document，这也是本书最经常使用的对象。浏览器加载的任何 HTML 页面都会创建一个 document 对象，包含全部 HTML 内容及其他构成页面显示的资源。利用 JavaScript 以父子对象的形式就可以访问这些信息。这些对象都具有自己的属性和方法。

图 1.1 中 window 对象的其他子对象是 location（包含着当前页面 URL 的全部信息）、history（包含浏览器以前访问的页面地址）和 navigator（包含浏览器类型、版本和兼容的信息 。第 4 章将会更详细地介绍这些对象，其他章节也会使用它们，但目前我们着重于 document 对象。

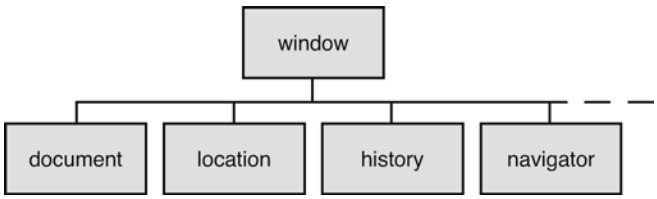

图 1.1 window 对象及其一些子对象

1.6.3 对象标签法

我们用句点方式表示树形结构里的对象：

```
parent.child
```

如图 1.1 所示，document 对象是 window 对象的子对象，所以在 DOM 里就像这样表示它：

```
window.document
```

HTML 页面的 body 部分在 DOM 里是 document 对象的一个子对象，所以表示为：

```
window.document.body
```

这种表示法的最后一个部分除了可以是对象外，还可以是属性或方法：

```
object1.object2.property

object1.object2.method
```

举例来说，如果想访问当前文档的 title 属性，也就是 HTML 标签<title>和</title>，我们可以这样表示：

```
window.document.title
```

TIP **提示**：window 对象永远包含当前浏览器窗口，所以使用 window.document 就可以访问当前文档。作为一种简化表示，使用 document 也能访问当前文档。

如果是打开了多个窗口，或是使用框架集，那么每个窗口或框架都有单独的 window 和 document 对象，为了访问其中的某一个文档，需要使用相应的窗口名称和文档名称。

1.7 与用户交互

现在来介绍 window 和 document 对象的一些方法。首先介绍的这两个方法都能提供与用户交互的手段。

1.7.1 window.alert()

即使不知道 window.alert()，我们实际上在很多场合已经看到过它了。window 对象位于 DOM 层级的最顶端，代表了显示页面的浏览器窗口。当我们调用 alert()方法时，浏览器会弹出一个对话框显示设置的信息，还有一个“确定”按钮。范例如下：

```
<script>window.alert("Here is my message");</script>
```

这是第一个使用句点表示法的范例，其中调用了 window 对象的 alert()方法，所以按照 object.method 表示方法就写为 window.alert。

提示：在实际编码过程中，可以不明确书写 window 对象名称。因为它是 DOM 层级结构的最顶层（有时也被称为“全局对象”），任何没有明确指明对象的方法调用都会被指向 window，所以<script>alert("Here is my message");</script>也能实现同样功能。 ***TIP***

请注意要显示的文本位于引号之中。引号可以是双引号，也可以是单引号，但必须有引号，否则会产生错误。

这行代码在浏览器执行时，产生的弹出对话框如图 1.2 所示。

图 1.2 window.alert()对话框

提示：图 1.2 显示的弹出对话框由运行在 Windows 7 旗舰版环境下的 Chrome 浏览器产生。不同操作系统、不同浏览器、不同显示设置都会影响这个对话框的最终显示情况，但它总是会包含要显示的信息和一个“确定”按钮。 ***TIP***

提示：在用户点击“确定”按钮之前，页面上是不能进行其他任何操作的。具有这种行为模式的对话框被称为“模态”对话框。 ***TIP***

1.7.2 document.write()

从这个方法名称就可以猜到它的功能。显然它不是弹出对话框，而是直接向 HTML 文档写入字符，如图 1.3 所示。

```
<script>document.write("Here is another message");</script>
```

图 1.3 使用 document.write()

NOTE

说明：实际上，无论从功能来说，还是从编码风格与可维护性来说，document.write 都是一种向页面输出内容的笨拙方式。大多数“严肃”的 JavaScript 程序员都不会使用这种方式，更好的方式是使用 JavaScript 和 DOM。但在本书第一部分介绍 JavaScript 语言的基本知识时，我们还会使用这个方法。

实践

JavaScript 编写的“Hello World!”

在介绍一种编程语言时，如果不使用传统的“Hello World!”范例似乎说不过去。这个简单的 HTML 文档如程序清单 1.1 所示。

程序清单 1.1　使用 alert()对话框实现“Hello World!”

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello from JavaScript!</title>
</head>
<body>
    <script>
        alert("Hello World!");
    </script>
</body>
</html>
```

在文本编辑器里创建一个文档，命名为 hello.html，输入上述代码，保存到计算机，然后在浏览器里打开它。

CAUTION

注意：有些文本编辑器会尝试给我们指定的文件名添加.txt 后缀，因此在保存文件时要确保使用了.html 后缀，否则浏览器可能不会正常打开它。

几乎全部操作系统都允许我们用鼠标右键单击 HTML 文件图标，从弹出菜单里选择“打开方式…”（或类似的字眼）。另外一种打开方式是先运行喜欢的浏览器，然后从菜单栏里选择“文件”>“打开”，找到相应的文件，加载到浏览器。

CAUTION

注意：有些浏览器的默认安全设置会在打开本地内容（比如本地计算机上的文件）时显示警告内容，如果看到这样的提示，只要选择允许继续操作即可。

这时会看到如图 1.2 所示的对话框，但其中的内容是“Hello World!”。如果计算机里安装了多个浏览器，可以尝试用它们打开这个文件，比较得到的结果。对话框外观可能有细微差别，但信息和“确定”按钮都是一样的。

▲

1.7.3　读取 document 对象的属性

正如前文所述，DOM 树包含着方法和属性。前面的范例展示了如何使用 document 对象

的 write 方法向页面输出文本，现在我们来读取 document 对象的属性。以 document.title 属性为例，它包含了 HTML 文档里<title>标签的内容。

在文本编辑器里修改 hello.html，修改对 window.alert()方法的调用：

```
alert(document.title);
```

注意到 document.title 并没有包含在引号里，这时如果使用引号，JavaScript 会认为我们要输出文本“document.title”。在不使用引号的情况下，JavaScript 会把 document.title 属性的值传递给 alert()方法，得到的结果如图 1.4 所示。

图 1.4 显示 document 对象的属性

1.8 小结

本章简要介绍了服务器端脚本和客户端脚本的概念，还简述了 JavaScript 和 DOM 的历史演变，大概展示了 JavaScript 能够实现什么功能来改善页面和优化用户体验。

本章还简单介绍了 DOM 的基本结构，展示了如何使用 JavaScript 访问特定对象及其属性，并且使用这些对象。

后面的章节将基于这些基本概念逐渐展开更高级的脚本编程项目。

1.9 问答

问：如果使用服务器端脚本（比如 PHP 或 ASP），还能在客户端使用 JavaScript 进行编程吗？

答：当然可以。事实上，这种组合方式能够形成一个有力的平台，实现功能强大的应用。Google Mail 就是个很好的范例。

问：应该对多少种不同的浏览器进行测试呢？

答：方便的情况下越多越好。编写与标准兼容的避免使用浏览器专用功能的代码，从而让程序在各个浏览器上都能顺畅运行，这不是简单的事情。浏览器在特定功能的实现上有一两处细微差别，总是难免的。

问：包含 JavaScript 代码会不会增加页面加载的时间？

答：是的，但通常这种影响很小，可以忽略不计。如果 JavaScript 代码的内容比较多，就应该在用户可能使用的最慢连接上进行测试。除了一些极其特殊的情况，这一般不会成为什么问题。

1.10 作业

请先回答问题，再参考后面的答案。

1.10.1　测验

1．JavaScript 是解释型语言还是编译型语言？

a．编译型语言

b．解释型语言

c．都不是

d．都是

2．若要添加 JavaScript 语句，必须在 HTML 页面里使用什么标签？

a．<script>和</script>

b．<type="text/javascript">

c．<!--和-->

3．DOM 层级结构的最顶层是：

a．document 属性

b．document 方法

c．document 对象

d．window 对象

1.10.2　答案

1．选 b。JavaScript 是一种解释型语言，它以纯文本方式编写代码，一次读取并执行一条语句。

2．选 a。JavaScript 语句添加在<script>和</script>之间。

3．选 d。window 对象位于 DOM 树的顶端，document 对象是它的一个子对象。

1.11　练习

在本章的“实践”环节中，我们使用了这样一行代码：

```
alert(document.title);
```

它可以输出 document 对象的 title 属性。请尝试修改这段脚本，输出 document.lastModified 属性，它包含的是 Web 页面最后一次修改的日期和时间。（提示：属性名称是区分大小写的，注意这个属性里大写的 M。）还可以尝试用 document.write()代替 alert()方法向页面直接输出信息。

在不同的浏览器里运行本章的范例代码，观察页面显示情况有什么区别。

第 2 章

创建简单的脚本

本章主要内容包括：

- 在 Web 页面里添加 JavaScript 的各种方式
- JavaScript 语句的基本语法
- 声明和使用变量
- 使用算术操作符
- 代码的注释
- 捕获鼠标事件

第 1 章介绍了 JavaScript 是一种能够让 Web 页面更具有交互性的脚本语言。

本章将介绍如何向 Web 页面添加 JavaScript，以及它的一些基本语法，比如语句、变量、操作符和注释。同时，本章将涉及更加实用的脚本范例。

2.1 在 Web 页面里添加 JavaScript

正如上一章所介绍的，JavaScript 代码是和页面内容一起发送给浏览器的，这是如何做到的呢？有两种方法可以把 JavaScript 代码集成到 HTML 页面，它们都要使用第 1 章介绍的<script>和</script>标签。

第一种方法是把 JavaScript 语句直接包含在 HTML 文件里，就像上一章所介绍的一样。

```
<script>
    …JavaScript 语句…
</script>
```

第二种方法，也是更好的方法，是把 JavaScript 代码保存到单独的文件，然后利用<script>标签的 src（源）属性把这个文件包含到页面里。

```
<script src='mycode.js'></script>
```

前例包含了一个名为mycode.js的文件，其中有我们编写的JavaScript语句。如果JavaScript文件与调用脚本不在同一个文件夹，就需要使用相对或绝对路径：

```
<script src='/path/to/mycode.js'></script>
```

或

```
<script src='http://www.example.com/path/to/mycode.js'></script>
```

NOTE **说明：**按照惯例，JavaScript 代码文件的名称后缀是.js。但从实际情况来看，代码文件的名称可以使用任何后缀，浏览器都会把其中的内容当作 JavaScript 来解释。

把 JavaScript 代码保存到单独的文件有不少好处：

- 当 JavaScript 代码有更新时，这些更新可以立即作用于使用这个 JavaScript 文件的页面。这对于 JavaScript 库是尤为重要的（本书稍后会有介绍）。
- HTML 页面的代码可以保持简洁，从而提高易读性和可维护性。
- 可以稍微改善一点性能。浏览器会把包含文件进行缓存，当前页面或其他页面再次需要使用这个文件时，就可以直接从内存读取了。

CAUTION **注意：**外部文件里不能使用<script>和</script>标签，也不能使用任何 HTML 标签，只能是纯粹的 JavaScript 代码。

程序清单 2.1 是第 1 章里 Web 页面的代码，修改为在<body>区域里包含了一个 JavaScript 代码文件。JavaScript 可以放置到 HTML 页面的<head>或<body>区域里，但一般情况下，我们把 JavaScript 代码放到页面的<head>区域，从而让文档的其他部分能够调用其中的函数。第 3 章将介绍函数的有关内容。就目前而言，我们把范例代码暂时放到文档的<body>区域。

程序清单 2.1 包含了 JavaScript 文件的 HTML 文档

```
<!DOCTYPE html>
<html>
<head>
    <title>A Simple Page</title>
</head>
<body>
    <p>Some content ...</p>
    <script src='mycode.js'></script>
</body>
</html>
```

当 JavaScript 代码位于文档的 body 区域时，在页面被呈现时，遇到这些代码就会解释和执行，然后继续完成页面的其他内容。

说明：有时在<script>标签里可以看到 HTML 风格的注释标签<!--和-->，比如：

NOTE

```
<script>
    <!--
    …JavaScript 语句…
    -->
</script>
```

这是为了兼容不能识别<script>标签的老版本浏览器。这种“注释”方式可以防止老版本浏览器把 JavaScript 源代码当作页面内容显示出来。除非我们有特别明确的需求要支持老版本的浏览器，否则是不需要使用这种技术的。

2.2 JavaScript 语句

JavaScript 程序是由一些单独的指令组成的，这些指令被称为“语句”。为了能够正确地解释语句，浏览器对语句的书写方式有所要求。第一种方式是把每个语句一行：

```
语句 1
语句 2
```

另一种方式是在同一行里书写多个语句，每个语句以分号表示结束。

```
语句 1; 语句 2;
```

为了提高代码的可读性，也为了减少无意中造成的语法错误，最好是结合上述两种方式的优点，也就是一行书写一个语句，并且用分号表示语句结束：

```
语句 1;
语句 2;
```

代码注释

有些语句的作用并不是为了让浏览器执行，而且为了方便需要阅读代码的人。我们把这些语句称为“注释”，它有一些特定的规则。

长度在一行之内的注释可以在行首以双斜线表示：

```
//注释内容
```

如果需要用这种方式添加多行注释，需要在每一行的行首都使用这个前缀：

```
//注释内容
//注释内容
```

实现多行注释的更简单方法是使用/*标签注释的开始，使用*/标签注释的结束。其中的注释内容可以跨越多行。

```
/* 这里的注释
   内容可以跨越
   多行  */
```

说明：JavaScript 还可以使用 HTML 注释语法来实现单行注释：

NOTE

```
<-- 注释内容 -->
```

但我们一般不在 JavaScript 中使用这种方式。

在代码里添加注释是一种非常好的习惯，特别是在编写较大、较复杂的 JavaScript 程序时。注释不仅可以作为我们自己的提示，还可以为以后阅读代码的其他人提供指示和说明。

NOTE **说明**：注释的确会略微增加 JavaScript 源文件的大小，从而对页面加载时间产生不好的影响。一般来说，这种影响小到可以忽略不计，但如果的确需要消除这种影响，我们可以清除 JavaScript 文件里的全部注释，形成所谓的“运行”版本，用于实际的站点。

2.3 变量

变量可以看作一种被命名的分类容器，用于保存特定的数据。数据可以具有多种形式：整数或小数、字符串或其他数据类型（本章稍后有所介绍）。变量可以用任何方式进行命名，但我们一般只使用字母、数字、美元符号（$）和下划线。

CAUTION **注意**：JavaScript 是区分大小写的，变量 mypetcat 和 Mypetcat 或 MYPETCAT 是不一样的。

JavaScript 程序员和其他很多程序员习惯使用一种名为“驼峰大小写”（或被称为“混合大小写”等）的方法，也就是把各个单词或短语连写在一起，没有空格，每个单词的首字母大写，但名称的第一个字母可以是大写或小写。按照这种方式，前面提到的变量就应该命名为 MyPetCat 或 myPetCat。

假设有个变量的名称是 netPrice。通过一个简单的语句就可以设置保存在 netPrice 里的数值：

```
netPrice = 8.99;
```

这个操作被称为给变量“赋值”。有些编程语言在赋值之前必须进行变量声明，JavaScript 不必如此。但变量声明是个很好的编程习惯，在 JavaScript 里可以这样做：

```
var netPrice;
netPrice = 8.99;
```

还可以把上述两个语句结合成一个语句，更加简洁和易读。

```
var netPrice = 8.99;
```

如果要把“字符串”赋值给一个变量，需要把字符串放到一对单引号或双引号之中：

```
var productName = "Leather wallet";
```

然后就可以传递这个变量所保存的值，比如传递给 window.alert 方法：

```
alert(productName);
```

生成的对话框会计算变量的值，然后显示出来，如图 2.1 所示。

图 2.1 显示变量 productName 的值

TIP **提示**：尽量使用含义明确的名称，比如 productName 和 netPrice。虽然像 var123 或 myothervar49 这样的名称也是合法的，但前者显然具有更好的易读性和可维护性。

2.4 操作符

如果不能通过计算操作变量里保存的值，那么这些值的作用就是十分有限的。

2.4.1 算术操作符

首先，JavaScript 可以使用标准的算术操作符进行加、减、乘、除。

```
var theSum = 4 + 3;
```

显然，前面这个语句执行之后，变量 theSum 的值是 7。在运算中，我们还可以使用变量名称：

```
var productCount = 2;
var subtotal = 14.98;
var shipping = 2.75;
var total = subtotal + shipping;
```

JavaScript 的减法（–）、乘法（*）和除法（/）也是类似的：

```
subtotal = total - shipping;
var salesTax = total * 0.15;
var productPrice = subtotal / productCount;
```

如果想计算除法的余数，可以使用 JavaScript 的“模”运算符，也就是“%”：

```
var itemsPerBox = 12;
var itemsToBeBoxed = 40;
var itemsInLastBox = itemsToBeBoxed % itemsPerBox;
```

上述语句运行之后，变量 itemsInLastBox 的值是 4。

JavaScript 对变量值的增加和减少有快捷操作符，分别是（++）和（—）：

```
productCount++;
```

上述语句相当于：

```
productCount = productCount + 1;
```

类似地，

```
items--;
```

与下面的语句作用相同：

```
items = items - 1;
```

关于 JavaScript 算术操作符的更详细介绍请见附录 B。

TIP

提示：如果变量值的增加或减少不是 1，而是其他数值，JavaScript 还允许把算术操作符与等号结合使用，比如+=和–=。

如下面两行代码的效果是相同的：

```
total = total + 5;
total += 5;
```

下面两行也是一样：

```
counter = counter - step;
counter -= step;
```

乘法和除法算术操作符也可以这样使用：

```
price = price * uplift;
price *= uplift;
```

2.4.2 操作符优先级

在一个计算中使用多个操作符时，JavaScript 根据“优先级规则”来确定计算的顺序。比如下面这条语句：

```
var average = a + b + c / 3;
```

根据变量的名称，这应该是在计算平均数，但这个语句不会得到我们想要的结果。在与 *a* 和 *b* 相加之前，*c* 会先进行除法运算。为了正确地计算平均数，需要添加一对括号，像下面这样：

```
var average = (a + b + c) / 3;
```

如果对于运算优先级不是十分确定，我们建议使用括号。这样做并不需要什么额外的代价，不仅能够让代码更易读（无论是编写者本人还是需要查看代码的其他人），还能避免优先级影响运算过程。

NOTE **说明**：对于有 PHP 或 Java 编程经验的读者来说，可以发现 JavaScript 的操作符优先级规则与它们基本是一样的。关于 JavaScript 操作符优先级的详细说明请见：http://msdn.microsoft.com/en-us/library/z3ks45k7(v=vs.94).aspx。

2.4.3 对字符串使用操作符“+”

当变量保存的是字符串而不是数值时，算术操作符基本上就没有什么意义了，唯一可用的是操作符“+”。JavaScript 把它用于两个或多个字符串的连接（按照顺序组合）：

```
var firstname="John";
var surname="Doe";
var fullname=firstname+" "+surname;
//变量 fullname 里的值是"John Doe"
```

如果把操作符“+”用于一个字符串变量和一个数值变量，JavaScript 会把数值转换为字符串，再把两个字符串连接起来。

```
var name = "David";
var age = 45;
alert(name + age);
```

图 2.2 所示是一个字符串变量和一个数值变量使用操作符“+”的结果。

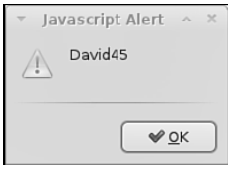

图 2.2 连接一个字符串和一个数值

本书的第 5 章将会更详细地讨论 JavaScript 的数据类型和字符串操作。

实践

把摄氏度转换为华氏度

把摄氏度转换为华氏度的方法是把数值乘 9，除以 5，然后加 32。用 JavaScript 可以这样做：

```
var cTemp=100;  //摄氏度
// 在表达式里充分使用括号
```

```
var hTemp=((cTemp*9)/5)+32;
```

实际上，我们可以省略代码里的括号，结果也是正确的：

```
var hTemp = cTemp*9/5 + 32;
```

然而使用括号可以让代码更易懂，而且有助于避免操作符优先级可能导致的问题。

让我们在页面里测试上述代码。

程序清单 2.2　摄氏度转换为华氏度

```
<!DOCTYPE html>
<html>
<head>
    <title>Fahrenheit From Celsius</title>
</head>
<body>
    <script>
        var cTemp = 100;  // temperature in Celsius
        // Let's be generous with parentheses
        var hTemp = ((cTemp * 9) /5 ) + 32;
        document.write("Temperature in Celsius: " + cTemp + "
degrees<br/>");
        document.write("Temperature in Fahrenheit: " + hTemp + "
degrees");
    </script>
</body>
</html>
```

把这段代码保存到文件 temperature.html，加载到浏览器，应该能够看到如图 2.3 所示的结果。

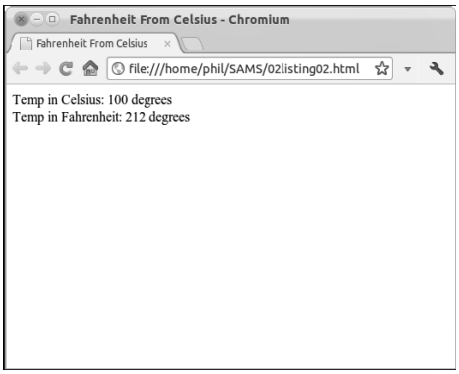

图 2.3　程序清单 2.2 的输出结果

编辑代码文件，给 cTemp 设置不同的值，每次都应该能够得到正确的结果。

▲

2.5　捕获鼠标事件

为页面增加与用户的交互是 JavaScript 的基本功能之一。为此，我们需要一些机制来检测用户和程序在特定时间在做什么，比如鼠标在浏览器的什么位置，用户是否点击了鼠标或按了按键，页面是否完整加载到浏览器，等等。

这些发生的事情，我们称之为“事件”，JavaScript 提供了多种工具来操作它们。第 9 章将详

细介绍事件和处理事件的高级技术，现在先来看看利用 JavaScript 检测用户鼠标动作的一些方法。

JavaScript 使用“事件处理器”来处理事件，本章介绍其中的 3 个：onClick、onMouseOver 和 onMouseOut。

2.5.1 onClick 事件处理器

onClick 事件处理器几乎可以用于页面上任何可见的 HTML 元素。使用它的方式之一是给 HTML 元素添加一个属性：

```
onclick="…一些 JavaScript 语句…"
```

NOTE **说明**：虽然给 HTML 元素直接添加事件处理器是完全可行的，但目前已经不认为这是个好的编程方式了。本书的第一部分仍然会使用这种方式，但后面的章节里会介绍更强大、更灵活的方式来使用事件处理器。

先来看一个范例，如程序清单 2.3 所示。

程序清单 2.3 使用 onClick 事件处理器

```
<!DOCTYPE html>
<html>
<head>
    <title>onClick Demo</title>
</head>
<body>
    <input type="button" onclick="alert('You clicked the button!')"
➥value="Click Me" />
</body>
</html>
```

上述 HTML 代码在页面的 body 区域添加一个按钮，并且设置了它的 onClick 属性，从而在它被点击时运行相应的 JavaScript 代码。当用户点击这个按钮时，onClick 事件被激活（通常称为“被触发”），然后属性中所包含的 JavaScript 语句被执行。

本例中只有一个语句：

```
alert('You clicked the button!')
```

图 2.4 是单击这个按钮得到的结果。

图 2.4 使用 onClick 事件处理器

说明：也许有人注意到了，我们称这个事件处理器为 onClick，而在 HTML 元素里添加它时却使用小写的 onclick。这是因为 HTML 是不区分大小写的，而 XHTML 是区分大小写的，并且要求全部的 HTML 元素及属性名称都使用小写字母。

NOTE

2.5.2 onMouseOver 和 onMouseOut 事件处理器

如果需要检测鼠标指针与特定页面元素的位置关系，可以使用 onMouseOver 和 onMouseOut 事件处理器。

当鼠标进入页面上某个元素占据的区域时，会触发 onMouseOver 事件。而 onMouserOut 事件，很显然是在鼠标离开这一区域时触发的。

程序清单 2.4 示范了一个简单的 onMouseOver 事件处理过程。

程序清单 2.4 使用 onMouseOver 事件处理器

```
<!DOCTYPE html>
<html>
<head>
    <title>onMouseOver Demo</title>
</head>
<body>
    <img src="image1.png" alt="image 1" onmouseover="alert('You entered
➥the image!')" />
</body>
</html>
```

图 2.5 展示了上述代码的执行结果。如果把程序清单 2.4 里的 onmouseover 替换为 onmouseout，就会在鼠标离开图像区域（而不是进入）时触发事件处理器，从而弹出警告对话框。

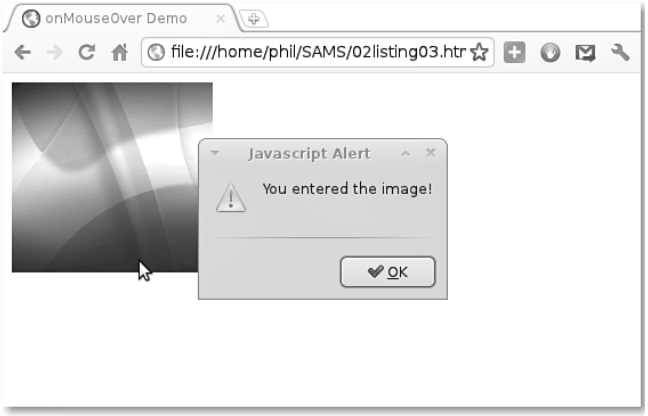

图 2.5 使用 onMouseOver 事件处理器

实践

实现图像变化

利用 onMouseOver 和 onMouseOut 事件处理器可以在鼠标位于图像上方时，改变图像的显示方式。为此，当鼠标进入图像区域时，可以利用 onMouseOver 改变<img>元素的 src 属

性；而当鼠标离开时，利用 onMouseOut 再把这个属性修改回来。代码如程序清单 2.5 所示。

程序清单 2.5 利用 onMouseOver 和 onMouseOut 实现图像变化

```
<!DOCTYPE html>
<html>
<head>
    <title>OnMouseOver Demo</title>
</head>
<body>
    <img src="tick.gif" alt="tick" onmouseover="this.src='tick2.gif';"
onmouseout="this.src='tick.gif';" />
</body>
</html>
```

上述代码中出现了一些新语法，在 onMouseOver 和 onMouseOut 的 JavaScript 语句中，使用了关键字 this。

当事件处理器是通过 HTML 元素的属性添加到页面时，其中的 this 是指 HTML 元素本身。本例中就是“当前图像”，this.src 就是指这个图像对象的 src 属性。

本例中使用了两个图像：tick.gif 和 tick2.gif。当然可以使用任何可用的图像，但为了达到最佳效果，两个图像最好具有相同尺寸，而且文件不要太大。

使用编辑软件创建一个 HTML 文件，包含程序清单 2.5 所示的代码。可以根据实际情况修改图像文件的名称，但要确保所使用的图像与 HTML 文件位于同一个目录里。保存 HTML 文件并且在浏览器里打开它。

我们应该可以看到鼠标指针进入时，图像改变；当指针离开时，图像恢复原样，如图 2.6 所示。

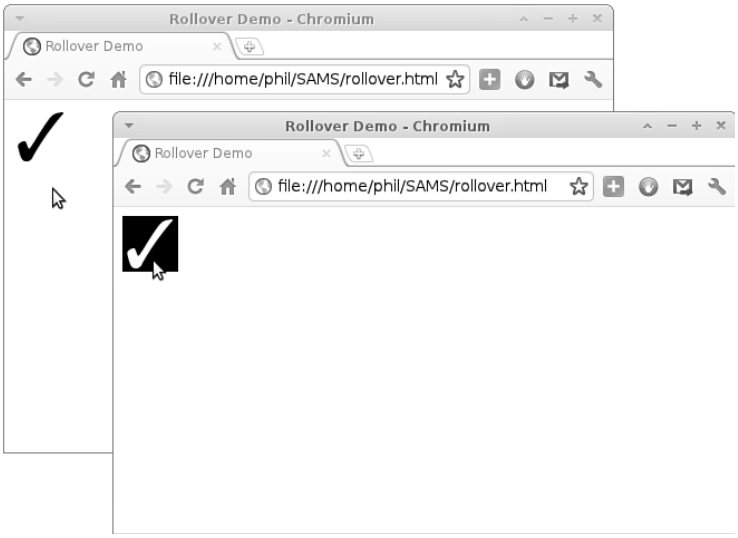

图 2.6 利用 onMouseOver 和 onMouseOut 实现的图像变化

NOTE **说明：**这曾经是图像变化的经典方式，现在已经被更高效的“层叠样式表”（CSS）取代了，但它仍不失为展示 onMouseOver 和 onMouseOut 事件处理器的简洁方式。

▲

2.6 小结

本章的内容相当丰富。

首先是在 HTML 页面里添加 JavaScript 代码的不同方式。

接着是在 JavaScript 里声明变量，给变量赋值以及利用算术操作符操作变量。

最后是 JavaScript 的一些事件处理器，展示如何检测用户鼠标的特定行为。

2.7 问答

问：在程序清单和片段里，有时把<script>开始和结束标签写在一行里，有时写在不同的行，这有什么区别吗？

答：空格、制表符和空行这类空白内容在 JavaScript 里是完全被忽略的。我们可以利用这些空白调整代码的布局，使它们更容易理解。

问：是否能使用同一个<script>元素来引用外部 JavaScript 文件，同时包含 JavaScript 语句？

答：不行。如果利用<script>元素的 src 属性包含了外部 JavaScript 文件，就不能在<script>和</script>之间包含 JavaScript 语句了，而是必须为空。

2.8 作业

请先回答问题，再参考后面的答案。

2.8.1 测验

1. 什么是 onClick 事件处理器？
   a. 检测鼠标在浏览器里位置的一个对象
   b. 响应用户点击鼠标动作时执行的脚本
   c. 用户能够点击的一个 HTML 元素
2. 页面里允许有几个<script>元素？
   a. 0
   b. 仅 1 个
   c. 任意数量
3. 关于变量，下列哪个说法是不正确的？
   a. 名称是区分大小写的
   b. 可以保存数值或非数值信息

c．名称里可以包含空格

2.8.2 答案

1．选 b。用户点击鼠标时，onClick 事件被触发。

2．选 c。可以根据需要使用多个<script>元素。

3．选 c。JavaScript 的变量名称不能包含空格。

2.9 练习

从程序清单 2.4 入手，删除<img>元素里的 onMouseOver 和 onMouseOut 事件处理器，添加 onClick 事件处理器，把图像的 title 属性设置为 My New Title。（提示：利用 this.title 可以访问图像的 title 属性。）

有什么办法可以方便地确定脚本正确地设置了新的图像标题？

第 3 章

使用函数

本章主要内容包括：

- 如何定义函数
- 如何调用（执行）函数
- 函数如何接收数据
- 从函数返回值
- 变量的作用域

很多情况下，程序在执行过程中会反复完成相同或类似的任务，为了避免多次重复编写相同的代码段，JavaScript 把部分代码包装为能够重复使用的模块，称为“函数”。函数可以在程序的其他部分使用，就像它是 JavaScript 语言的组成部分一样。

使用函数可以让代码更加易读和维护。举例来说，我们编写了一个计算货运成本的程序，当税率或公路运费改变时，就需要修改脚本，如果不使用函数，这可能涉及多达 50 处执行计算的代码。在这个修改过程中，就很可能漏掉某些部分，从而导致错误的发生。如果这些计算都被集中到几个函数中，然后在程序中使用，那么就只需要修改这几个函数，其结果会作用于整个程序。

函数是 JavaScript 的基本模块之一，几乎会出现在每个脚本中。本章将介绍如何创建和使用函数。

3.1 基本语法

创建函数就好像是创建一个新的 JavaScript 命令，能够在脚本的其他部分使用。

下面是创建函数的基本语法：

```
function sayHello() {
```

```
    alert("Hello");
    //…其他语句…
}
```

首先是关键字是 function，接着是函数的名称，后面紧跟着一对圆括号，然后是一对花括号。花括号里面是构成函数的 JavaScript 语句。在前面这个例子里只有一行代码，用于弹出一个警告对话框。我们可以根据需要添加任意数量的代码来实现函数的功能。

CAUTION **注意**：关键字 function 必须是小写的，否则会产生错误。

为了让代码更整洁，可以在一个<script>元素里创建多个函数。

```
<script>
    function doThis() {
        alert("Doing This");
    }
    function doThat() {
        alert("Doing That");
    }
</script>
```

TIP **提示**：函数名称与变量名称一样，是区分大小写的，如函数 MyFunc()与 myFunc()是不同的。与变量名称一样，使用含义明确的函数名称可以提高代码的易读性。

3.2 调用函数

在页面加载时，包含在函数定义区域内的代码不会被执行，而是在被“调用”时执行。

调用函数只需要使用函数名称（以及一对括号），就可以在需要的地方执行函数的代码：

```
sayHello();
```

举例来说，可以在按钮的 onClick 事件处理器里调用函数 sayHello()：

```
<input type="button" value="Say Hello" onclick="sayHello()" />
```

TIP **提示**：本书前面的内容里展示了不少使用 JavaScript 对象方法的代码，比如 document.write()或 window.alert()。“方法”实际上就是属于特定类的函数。关于对象的更详细介绍请见第 4 章。

把 JavaScript 代码放置到页面的<head>区域

到目前为止，我们的范例都把 JavaScript 代码放置到 HTML 页面的<body>区域。为了更好地发挥函数的作用，我们要采取更适当的方式，也就是把 JavaScript 代码放置到页面的<head>区域。当函数位于页面<head>区域的<script>元素里，或是位于页面<head>区域的<script>元素的 src 属性所指向的外部文件时，它就可以从页面的任何位置被调用。把函数放到文档的 head 部分能够确保它们在被调用前已经被定义了。

程序清单 3.1 展示了一个范例。

程序清单 3.1 位于页面<head>区域的函数

```
<!DOCTYPE html>
<html>
<head>
    <title>Calling Functions</title>
    <script>
        function sayHello() {
            alert("Hello");
        }
    </script>
</head>
<body>
    <input type="button" value="Say Hello" onclick="sayHello()" />
</body>
</html>
```

在这段代码里，可以看到函数定义位于页面<head>区域的<script>元素里，而函数的调用则位于完全不同的位置，本例是页面<body>区域里按钮的 onClick 事件处理程序。

点击按钮后的结果如图 3.1 所示。

图 3.1 调用 JavaScript 函数

3.3 参数

如果函数只是像前面范例中那样在每次调用时只能实现完成相同的操作，那么其应用就会受到很大的局限。

好在我们可以通过向函数传递数据来扩展函数的功能，其实现方法是在调用函数时给它传递一个或多个“参数”：

```
functionName(arguments)
```

下面是一个简单的函数，可以计算一个数的立方并且显示结果：

```
function cube(x) {
    alert(x * x * x);
}
```

现在来调用这个函数，用一个数值来代替其中的 *x*。调用方式是这样的：

```
cube(3);
```

得到的对话框里会显示计算的结果，本例就是 27。

当然，我们还可以传递一个变量作为参数。下面的代码也会显示一个对话框，其中显示数值 27：

```
var length = 3;
cube(length);
```

多参数

函数不只能接收一个参数。在使用多个参数时，只需要使用逗号分隔它们就行了：

```
function times(a,b) {
    alert(a*b);
}
times(3,4);   //显示 12
```

根据需要可以使用任意多个参数。

CAUTION

注意：在调用函数时，要确保包含了与函数定义相匹配的参数数量。如果函数定义里的某个参数没有接收到值，JavaScript 可能会报告错误，或是函数执行结果不正确。如果调用函数时传递了过多的参数，JavaScript 会忽略多出来的参数。

需要明确的是，函数定义中参数的名称与传递给函数的变量名称没有任何关系。参数列表里的名称就是占位符，用于保存函数被调用时传递过来的实际值。这些参数的名称只会在函数定义内部使用，实现函数的功能。

本章稍后在讨论变量“作用域”时会有更详细的介绍。

实践

输出消息的函数

现在我们利用已经学到的知识来创建一个函数，当用户点击按钮时，向用户发送关于按钮的信息。这个函数放在页面的<head>区域，具有多个参数。

这个函数的代码如下：

```
function buttonReport(buttonId, buttonName, buttonValue) {
    //按钮 id 信息
    var userMessage1="Button id: "+ buttonId+"\n";
    //按钮名称
    var userMessage2="Button name: "+buttonName+"\n";
    //按钮值
    var userMessage3="Button value: "+buttonValue;
    //提醒用户
```

```
    alert(userMessage1+userMessage2+userMessage3);
}
```

函数 buttonReport 具有三个参数，分别是被点击按钮的 id、name 和 value。根据这些三个参数，函数组成简短的信息，然后把三段信息组合成一个字符串，传递给 alert()方法，从而在对话框里进行显示。

提示：从代码中可以看到，前两条消息的末尾添加了"\n"，这是表示“新行”的字符，能够让对话框里的文本另起一行，从左侧开始显示。在字符串里，像这样的特殊字符如果想要发挥正确的功能，必须以“\”作为前缀。这种具有前缀的字符被称为“转义序列”，更详细的介绍请见第 4 章。

TIP

为了调用这个函数，我们在 HTML 页面上放置一个按钮，并且定义它的 id、name 和 value 属性：

```
<input type="button" id="id1" name="Button 1" value="Something" />
```

接着添加一个 onClick 事件处理器，从中调用我们定义的函数。这里又要用到关键字 this：

```
onclick = "buttonReport(this.id, this.name, this.value)"
```

完整的代码如程序清单 3.2 所示。

程序清单 3.2　调用多个参数的函数

```
<!DOCTYPE html>
<html>
<head>
    <title>Calling Functions</title>
    <script>
        function buttonReport(buttonId, buttonName, buttonValue) {
            // information about the id of the button
            var userMessage1 = "Button id: " + buttonId + "\n";
            // then about the button name
            var userMessage2 = "Button name: " + buttonName + "\n";
            // and the button value
            var userMessage3 = "Button value: " + buttonValue;
            // alert the user
            alert(userMessage1 + userMessage2 + userMessage3);
        }
    </script>
</head>
<body>
    <input type="button" id="id1" name="Left Hand Button" value="Left"
➥onclick = "buttonReport(this.id, this.name, this.value)"/>
    <input type="button" id="id2" name="Center Button" value="Center"
➥onclick = "buttonReport(this.id, this.name, this.value)"/>
    <input type="button" id="id3" name="Right Hand Button" value="Right"
➥onclick = "buttonReport(this.id, this.name, this.value)"/>
</body>
</html>
```

利用编辑软件创建文件 button.html，输入上述代码。它的运行结果类似于图 3.2 所示，具体的输出内容取决于点击了哪个按钮。

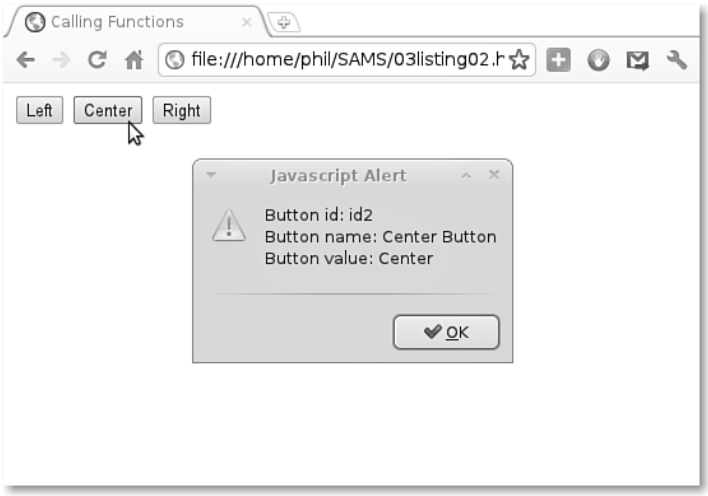

图 3.2 使用函数发送消息

3.4 从函数返回值

前面介绍了如何向函数传递参数，让函数对这些数据进行处理。那么，如何从函数获得数据呢？毕竟，我们不能只通过弹出对话框来获得函数的结果。

从函数调用获得数据的机制是“返回值”，其工作方式如下所示：

```
function cube(x) {
    return x * x * x;
}
```

这个函数里没有使用 alert()对话框，而是在需要获取的结果前面使用了关键字 return。为了在函数外部得到这个值，只需要把函数返回的值赋予一个变量：

```
var answer = cube(3);
```

现在变量 answer 包含的数值是 27。

NOTE **说明**：函数返回的值不一定是数值，而是可以返回 JavaScript 支持的任何数据类型，详情请见第 5 章。

TIP **提示**：当函数返回一个值时，我们可以利用函数调用把返回的值直接传递给另一个语句，比如下面的代码：

```
var answer=cube(3);
alert(answer);
```

可以简单地写为：

```
alert(cube(3));
```

函数调用 cube(3)的返回值 27 直接成为传递给 alert()方法的参数。

3.5 变量作用域

前面已经介绍过如何使用关键字 var 声明变量。在函数里声明变量时，有一条最重要的原则需要了解：

“函数内部声明的变量只存在于函数内部。”

这种限制被称为变量的“作用域”。来看下面这个范例：

```
//定义函数 addTax()
function addTax(subtotal, taxRate) {
    var total=subtotal*(1+(taxRate/100));
    return total;
}
//调用这个函数
var invoiceValue=addTax(50,10);
alert(invoiceValue);  //正常工作
alert(total);  //不工作
```

运行上述代码，首先会看到一个 alert()对话框显示变量 invoiceValue 的值（应该是 55，但可能会看到类似 55.000 000 01 这样的数值，因为我们没有让 JavaScript 对结果四舍五入）。

之后，我们并不会看到 alert()对话框显示变量 total 的值。JavaScript 会生成一个错误，而我们是否能够看到这个错误提示取决于浏览器的设置（本书稍后会更详细地介绍有关错误处理的问题），但无论如何，JavaScript 都不能显示包含变量 total 值的 alert()对话框。

这是因为变量 total 的声明是在 addTax()函数内部进行的，在函数之外变量 total 就是不存在的（JavaScript 术语就是“未定义的”）。范例中利用关键字 return 返回的只是变量 total 里保存的值，然后这个值被保存到另一个变量 invoice Value 里。

我们把函数内部定义的变量称为“局部”变量，也就是属于函数这个“局部”。函数之外声明的变量称为“全局”变量。全局变量和局部变量可以使用相同的名称，但仍然是不同的变量！

变量能够使用的范围称为变量的“作用域”，因此可以称一个变量具有“局部作用域”或“全局作用域”。

实践

变量作用域示范

为了说明变量的作用域，来看下面这段代码：

```
var a=10;
var b=10;
function showVars() {
    var a=20;  //声明一个新的局部变量 a
    b=20;   //改变全局变量 b 的值
    return"Local variable'a' = "+a+"\nGlobal variable'b'="+b;
}
var message=showVars();
alert(message+"\nGlobal variable'a'="+a);
```

函数 showVars()操作了两个变量：*a* 和 *b*。变量 *a* 是在函数内部定义的，它是个局部变量，仅存在于函数内部，与脚本一开始定义的全局变量（名称也是 *a*）是完全不同的。

变量 *b* 不是在函数内部而是在外部定义的，它是个全局变量。

程序清单 3.3 是把上述代码放置于 HTML 页面的结果。

程序清单 3.3 全局和局部作用域

```
<!DOCTYPE html>
<html>
<head>
    <title>Variable Scope</title>
</head>
<body>
    <script>
        var a = 10;
        var b = 10;
        function showVars() {
            var a = 20; // declare a new local variable 'a'
            b = 20;     // change the value of global variable 'b'
            return "Local variable 'a' = " + a + "\nGlobal variable 'b' =
➥" + b;
        }
        var message = showVars();
        alert(message + "\nGlobal variable 'a' = " + a);
    </script>
</body>
</html>
```

当页面加载之后，showVars()函数返回一个消息字符串，其中包含了两个变量（*a* 和 *b*）在函数内部被更改之后的信息。这里的 *a* 具有局部作用域，*b* 具有全局作用域。

之后，全局变量 *a* 的当前值也附加到这个消息之后，完整地显示给用户。

把上述代码保存到 scope.html，用浏览器加载它，得到的结果如图 3.3 所示。

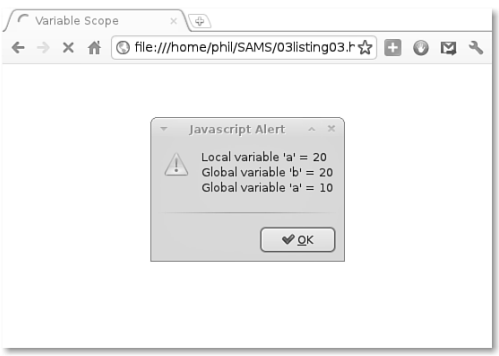

图 3.3 局部和全局作用域

▲

3.6 小结

本章介绍了什么是函数，如何创建函数，如何从代码中调用函数并以参数方式向其传递数据，以及如何从函数向调用语句返回数据。

最后，本章介绍了变量的局部作用域和全局作用域，以及变量的作用域如何影响函数对变量的操作。

3.7 问答

问：函数内部能够包含对其他函数的调用吗？

答：当然可以。我们可以根据需要进行多重的嵌套调用。

问：函数名称里可以具有哪些字符？

答：函数名称必须以字母或下划线开头，可以包含字母、数字和下划线，不能包含空格、标点符号和其他特殊字符。

3.8 作业

请先回答问题，再参考后面的答案。

3.8.1 测验

1. 调用函数时使用：
   a. 关键字 function
   b. 命令 call
   c. 函数名称及一对括号
2. 函数执行 return 语句的结果是什么？
   a. 生成一条错误信息
   b. 返回一个值，函数继续执行
   c. 返回一个值，函数停止执行
3. 在函数内部声明的变量称为：
   a. 局部变量
   b. 全局变量
   c. 参数

3.8.2 答案

1. 选 c。使用函数名称调用函数。
2. 选 c。在执行 return 语句之后，函数返回一个值，然后终止函数。
3. 选 a。函数内部定义的变量具有局部作用域。

3.9 练习

编写一个函数，接收摄氏度数值作为参数，返回相应的华氏度（参考第 2 章介绍的代码）。在 HTML 页面里测试这个函数。

JavaScript 入门经典（第5版）

SAMS

Teach Yourself

学习如何：

- 使用JavaScript创建动态的、交互性的网页；
- 调试脚本；
- 创建适合所有浏览器的脚本；
- 使用HTML5和CSS3；
- 使用流行的jQuery库；
- 使用简单的JavaScript来控制CSS；
- 向网页添加Ajax效果；
- 脚本动画和音乐。

24章阶梯教学

本书采用直观、循序渐进的方法，每章建立在前一章的基础之上，引导读者全面学习JavaScript的关键知识。

循序渐进的示例引导读者完成最常见的任务。

问与答、测验和练习帮助读者检验知识的掌握情况。

“注意”、“提示”和“警告”指出捷径和解决方案。

Phil Ballard是一位Web技术咨询师，他专注于Web站点的设计、搜索引擎的优化、服务器端脚本编程、客户端设计等等。他还是《Sams Teach Yourself Ajax in 10 Minutes》一书的作者。

Michael Moncur是Starling Technologies公司的所有人，这是一家专注于网络和互联网的咨询公司。他还是Web站点管理员和作家。他撰写过有关JavaScript、网络、MCSE认证方面的图书。

ISBN 978-7-115-31779-7

美术编辑 王建国

分类建议： 计算机/程序设计/JavaScript

人民邮电出版社网址：www.ptpress.com.cn