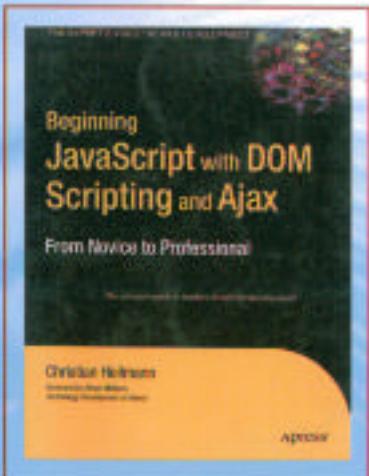


Beginning JavaScript
with DOM Scripting and Ajax

深入浅出 JavaScript

[德] Christian Heilmann 著
牛海彬 等译

- 世界级 JavaScript 程序员力作
- 全面、实用、丰富的经典示例
- 深入揭示现代 JavaScript 编程理念



人民邮电出版社
POSTS & TELECOM PRESS

站在巨人的肩上
Standing on Shoulders of Giants



www.turingbook.com

TURING 图灵程序员设计丛书 Web开发系列

深入浅出JavaScript

Beginning JavaScript
with DOM Scripting and Ajax

[德] Christian Heilmann 著

牛海彬 等译

人民邮电出版社
北京

图书在版编目(CIP)数据

深入浅出 JavaScript / (德) 海尔曼 (Heilmann, C.) 著;
牛海彬等译. —北京: 人民邮电出版社, 2008.4
(图灵程序设计丛书)
ISBN 978-7-115-17168-9

I. 深… II. ①海…②牛… III. Java 语言—程序设计
IV. TP312

中国版本图书馆 CIP 数据核字 (2007) 第 174306 号

内 容 提 要

本书是一部优秀的、注重实践的 JavaScript 教程。作者首先概览了 JavaScript，包括它的语法、良好的编码习惯、DOM 编程原则等；然后构建了 JavaScript 工具包，包括动态操作标记、使用 CSS 和 DOM 修改页面风格、验证表单、处理图像等；接着通过一个完整的案例研究阐明了如何使用多种 JavaScript 技术协同工作；最后单独设计一章来讲述第三方示例，演示了 YUI 和 jQuery JavaScript 库的使用。

本书适合初级和中级水平的 JavaScript 开发人员阅读，可作为高等院校计算机专业的 JavaScript 课程教材。

图灵程序设计丛书

深入浅出 JavaScript

-
- ◆ 著 [德] Christian Heilmann
 - 译 牛海彬 等
 - 责任编辑 傅志红 张继发
 - ◆ 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 电子函件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
 - 三河市海波印务有限公司印刷
 - 新华书店总店北京发行所经销
 - ◆ 开本: 800×1000 1/16
 - 印张: 25.25
 - 字数: 597 千字 2008 年 4 月第 1 版
 - 印数: 1~5 000 册 2008 年 4 月河北第 1 次印刷

著作权合同登记号 图字: 01-2006-5095 号

ISBN 978-7-115-17168-9/TP

定价: 55.00 元

读者服务热线: (010) 88593802 印装质量热线: (010) 67129223
反盗版热线: (010) 67171154

版 权 声 明

Original English language edition, entitled *Beginning JavaScript with DOM Scripting and Ajax* by Christian Heilmann, published by Apress L.P., 2560 Ninth Street, Suite 219, Berkeley, CA 94710 USA.

Copyright © 2006 by Christian Heilmann. Simplified Chinese-language edition copyright © 2008 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由 Apress L.P. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译者序

随着Ajax技术的兴起，JavaScript重新回到了大众的视野。Ajax是Asynchronous JavaScript and XML（异步JavaScript和XML）的缩写，使用它，不刷新整个页面也可以和Web服务器交换数据，这样提高了传输效率，增强了用户体验，搜狐和新浪的博客系统就是比较典型的成功案例。

JavaScript是一种解释性的脚本语言，它是网景公司于1996年发明的。当时把它嵌入到Netscape Navigator (NN) 2.0浏览器中，需要解释器来读取和执行添加到.html页面的JavaScript代码。随着因特网的发展，现在几乎所有的浏览器都支持JavaScript，也就是说有上亿台电脑安装了JavaScript的运行环境，这说明它有着广泛的用户群。

JavaScript刚开始是作为一种脚本语言推出的，因此人们往往错误地认为这种技术很简单，其实它是一种功能强大而且比较复杂的语言，在企业级开发和Web开发中起着非常重要的作用。我做了多年的企业级开发和Web开发，发现真正会用JavaScript的人并不多，大部分人也只是复制和粘贴，很少有人能写出自己的JavaScript应用，我想这与大家对它的不重视可能有很大关系。

译完本书，我感觉作者真的很棒，他没有介绍一些华而不实的例子，而是从基础开始教你如何编写高质量的JavaScript代码。除此以外，本书还包含了许多最新的JavaScript技术，包括在Ajax、RSS以及其他Web 2.0技术中的应用。它不仅适合JavaScript的初学者，而且适合希望对JavaScript技术有深入研究的人。本书还可作为计算机专业开设的JavaScript课程的教材。

本书第3章和第9章由屈晓光翻译，第7章由易继开翻译，感谢他们与我合作完成了本书。

感谢人民邮电出版社图灵公司的傅志红编辑和谢工给予我的帮助和指导。

感谢我的父母和许多朋友给予我的帮助和支持。

由于时间比较仓促，加上译者学识有限，译文中错误疏漏在所难免，欢迎大家指正。我的电子邮件和博客地址分别是pleasechess@126.com和<http://newhappy2008.blog.sohu.com>，欢迎大家来信或留言，把问题反馈给我。

牛海彬
2007年年底

序

也许从来没有什么时候比现在学习JavaScript更令人兴奋的了。经历了多年的浏览器之争后，JavaScript终于成为Web开发人员必备的工具。不再只是广告和恶作剧的手段，现在它已是下一代Web应用程序非常有价值的组成部分。

是什么原因导致它突然又备受关注呢？第一个原因完全是实践的结果：浏览器的改进最终使得编写跨浏览器平台的JavaScript可行。第二个原因具有更多的革命性：Ajax作为一种使用新名称的老技术，使得客户端代码可以直接连接到服务器上而不用重新刷新整个页面。这种简单的功能使Web应用程序开发更开阔，它启用了创新的界面，并且戏剧性地改变着用户对网页界面行为表现的期望。

越来越多的人认识到，JavaScript不是一种玩具语言，这进一步促进了人们对它的接受。尽管它还有许多不足之处，但是在其非常简单的外表之下有着许多现代语言中都没有的强大功能：闭包、原型继承以及对函数式编程风格的广泛支持。这样一种灵活的语言现在被安装到了数以亿计的计算机上，这是值得庆祝的。

“只是因为你可以做”并不意味着就应该那样去做。并不是所有的浏览器都生而平等，可访问性（人和各种其他设备都可以访问）仍然是Web开发非常重要的一个方面。理解这些问题和有关渐进增强的技术都是JavaScript学习非常重要的一部分。

JavaScript开发所面临的挑战是非常巨大的。

浏览器经常会背离现有的标准规范，而且伪标准非常常见且经常无法避免。

不断开发出来的功能强大的新应用程序可能会暴露许多隐藏多年的浏览器bug。这些应用程序本身就很复杂，维护大量的代码又会引起新的问题。

幸运的是，应对这个挑战的全球JavaScript社区已经出现了。大量的代码和资源都在等待着无畏的开发人员使用，但是这些财富隐藏的价值只有通过对潜在平台的一致理解才能发掘出来。本书将会为你提供这方面的知识。

作为这个社区中一名长期的导师和领导者，Christian是通向这个复杂世界的理想引路人。本书还包含了许多只有通过多年的经验积累才能获得的智慧。

Christian会教你以一种高雅、可靠且优美的方式应用JavaScript，这会使你的用户非常满意并且会给你的同事留下深刻的印象。

Simon Willison
著名Web开发人员
Django框架开发者之一

前　　言

如果你想从零开始学习JavaScript——它的含义、功能，以及如何综合使用它和其他技术（如CSS和HTML），那么你就选对书了。如果你已经对JavaScript有了一定的经验，但还想了解一些在最新的知识，那么你也选对了书——在最近几年里，JavaScript开发已经改变了许多。

在20世纪90年代中后期，JavaScript第一次开始应用到Web开发的时候（在1996年首次被Netscape 2支持），它很快就遭遇了许多批评和抱怨。这有很多原因——即使最好的时候，浏览器的支持也很一般，而最糟的时候，针对不同的浏览器（浏览器之争的主角是Netscape 4和IE 4）需要以不同的方式实现各种不同的功能。这就导致开发人员如果想获得跨浏览器平台的支持，就不得不编写完全不同的网站版本，或者纵容杂乱的代码分叉。

此外，还有很多人为因素。JavaScript的坏名声，开发人员和浏览器生产商的过错几乎要对半开。那时开发人员喜欢用JavaScript实现看上去很酷的效果，但是引起了许多可用性和可访问性的问题（这就是所谓DHTML，那时的一个流行词指的是使用JavaScript、CSS和HTML应用程序来产生动态效果）。如果由于某种原因JavaScript不可用或者用户要使用屏幕阅读器，那么网页就会完全失效。而且许多Web开发人员在不懂实际工作机理的情况下，就把一些脚本复制并粘贴到自己的网站中，导致了更多说不清楚的可用性和代码维护的问题。

正如我在前面所说，现在事情已经发生了改变。浏览器的支持性现在已处于可控级别，现代的浏览器大都使用DOM（文档对象模型）和其他构造的相同实现，而且现代技术也很大程度上都考虑了可访问性等。学完本书后，你会发现像DOM脚本这样的现代技术也都是围绕着下面的假设来建立的：既要将结构（在标记文件中）和表现（在CSS文件中）分离，也要分离JavaScript文件中的行为（而不是遍布在标记文档中）。JavaScript并不是只能做恶，它可以用来对网站编码，使用户体验更丰富，而在JavaScript不可用的时候也不会完全失效。这叫做分离式JavaScript（unobtrusive JavaScript）^①——JavaScript增强应该被看作是那些能够使用它们的用户的一种额外好处，而不是运行一个网站所必需的特性。

如果以前使用过JavaScript，那么阅读本书时你要准备好接受一种新思想。如果你完全是刚刚学JavaScript，那么可以松一口气，很幸运你没有刚说过的JavaScript早期开发的经历！

^① unobtrusive JavaScript是现代JavaScript编程的核心思想，主要通过将JavaScript与表现层和结构层分离，从而实现平稳退化不影响禁用了某些功能的用户的目的。也译为“不唐突的（或非干扰的）JavaScript”。——编者注

本书内容

JavaScript可能是在Web开发中被低估且被滥用最严重的语言，但是如果正确使用，它会是一种非常有价值的工具。在接下来的章节中，我们会介绍JavaScript的基础和现代的JavaScript技术，包括用于动态行为和样式控制以及事件处理的DOM编程。接着，我们会学习一些JavaScript最基础的应用，包括数据验证、图片和窗口操作，以及表单和导航菜单的动态增强。

接下来，我们把注意力转移到与JavaScript相关的、可能是目前最热的一个词语——Ajax。Ajax代表异步JavaScript与XML（Asynchronous JavaScript and XML），这有点儿名不符实，因为这种技术不必包括XML，而且可能经常和HTML一起使用。但是不要只关注这一点，它主要指在网页上创建运行的动态功能，因为不用刷新整个页面，网页的各小部分就可以更新，例如，在线的邮件应用程序中的联系人信息（Gmail是我们最容易想到的例子）。现在实现Ajax最常用的方式就是使用XMLHttpRequest（XHR）对象。它非常流行，因为它允许创建拥有丰富功能且其外观和运行方式都和桌面应用程序类似的Web应用程序。但是Ajax确实也带来了它自己的一系列问题，这个也会在书中涉及。

接下来是一个综合的案例研究，它展示了一个成熟的含有现代JavaScript增强的Web应用程序。

最后，第11章重点讲解现代JavaScript开发的另一个重要的方面——使用第三方JavaScript解决方案。当你开发JavaScript应用程序的时候，不需要每次都对所有的东西从头编码。和创建自己可复用的对象和函数一样（这个会在本书前面几章中讲到），在Web上还有许多第三方资源，你可以下载并在自己的应用程序中使用。从函数库到成熟的API（应用编程接口），许多资源都可以供你使用。其中，我们重点介绍了jQuery、Google Maps API、Yahoo API等。

社区和支持

JavaScript是什么？你用它来做什么？询问拥有不同技术背景或面向设计的开发人员的时候，你很可能会得到完全不同的答案。本书会教你如何成为一个可以和所有这些开发人员一起工作的JavaScript开发人员，而且通过使用JavaScript来增强网站、构造Web应用程序，甚至不用强迫用户改变他的使用方式或者进行硬件设置就能扩展软件，使他们对你刮目相看。

本书所提供的所有代码例子都可以在网站<http://www.beginningjavascript.com>下载和测试，在那里你还会发现更多的信息、勘误以及其他例子（Apress出版社在<http://www.apress.com>上也提供了勘误表和可下载的代码）^①。

但是遇到问题时该怎么办呢？你有3种选择。第一，可以通过网站（<http://wait-till-i.com>）通知我或者把问题通过邮件发送到Apress（详细的联系地址在<http://www.apress.com>上）。

第二，在因特网的一些JavaScript论坛里求助，其中比较好的有下列几个。

- evolt的thelist论坛：<http://lists.evolt.org/mailman/listinfo/thelist>。
- Mozilla JavaScript论坛：<http://developer.mozilla.org/en/docs/JavaScript>。

^① 图灵网站为本书中文版提供了配套网页，有代码下载和勘误表以及其他资源。——编者注

- Webdeveloper.com JavaScript论坛: <http://www.webdeveloper.com/forum/forumdisplay.php?f=3>。
- comp.lang.javascript FAQ: <http://jibbering.com/faq/>。

这些论坛经常有许多像你这样寻找问题答案的人光顾，还有许多非常有经验的JavaScript高手，他们乐于帮助社区里的人解决问题。要确保你的问题经过了思考，不要只粘贴你的代码，然后就问“这里什么地方有错误？”。也可以看一下论坛里过去的帖子，或许你的问题其他的人已经问过并得到解答了。

最后一个，阅读博客！许多天才的JavaScript高手喜欢通过博客和大家分享他们的思想、创新以及经验，其中包括我。这是获取新思想非常好的一种方式。我推荐你阅读下面的博客。

- Jeremy Keith^①: <http://www.adactio.com>。
- Simon Willison: <http://simon.incipio.com/>。
- WaSP DOM脚本编程任务组: <http://www.webstandards.org/action/dstf/>。
- Stuart Langridge: <http://kryogenix.org/days/>。
- Robert Nyman: <http://robertnyman.com/>。
- Jon Snook: <http://www.snook.ca/jonathan/>。

你现在是一个非常活跃的社区的一份子了。除了可以学习许多有用的东西，你还会遇到各种各样的有趣的人，而且这种学习方式会很快乐！让我们继续快乐地学习吧——不断地阅读……

致谢

感谢在完成本书的过程中所有帮助过我的人——Chris、Beth、Ami、Katie以及Apress出版社的Charles和Jon Stephens。我学到了许多，尤其是认识到了写书要做的工作比我想象的要多得多。

还要感谢那些通过解决问题并询问更多功能而由此帮助我的人——WaSP DOM脚本编程任务组的成员Stuart Colville、Matt Warden、Jens Grochtdreis、Ingo Chao、Volkan Ozcelik以及许多在evolt列表、CSS讨论区和我的博客上留言的朋友。

感谢以前在Agilisys的同事和现在在Yahoo的同事所做的测试和支持，还要感谢Good for Food、Spence、Pizzadelique以及Belle Epoque饭店为我提供了食物，让我可以保持健康的身体（还要感谢这些地方的邻居所提供的无线接入点）。

最后，我还要感谢你，因为购买本书说明有人真地想学习JavaScript，而不只是复制和粘贴脚本。

^① 畅销书《JavaScript DOM 编程艺术》、《Bulletproof Ajax 中文版》（人民邮电出版社）的作者。——编者注

目 录

第1章 JavaScript入门	1
1.1 JavaScript产生的原因	3
1.2 JavaScript是什么	3
1.3 JavaScript的问题和价值	4
1.4 JavaScript不可靠为什么还要用	5
1.5 网页中的JavaScript和基本语法	6
1.5.1 JavaScript语法	7
1.5.2 执行代码	8
1.5.3 函数	9
1.6 对象	10
1.7 简单的JavaScript示例	11
1.8 小结	13
第2章 数据和判定	15
2.1 数据、数据类型和数据运算符	15
2.1.1 字符串数据类型	16
2.1.2 运算符	18
2.1.3 JavaScript变量	20
2.1.4 不同数据类型的转换	22
2.2 复合数据类型：数组和对象	25
2.2.1 JavaScript提供的对象：String、Date和Math	26
2.2.2 数组	33
2.3 在JavaScript中进行判定	39
2.3.1 逻辑运算符和比较运算符	39
2.3.2 条件语句	41
2.3.3 测试多个值：switch语句	44
2.3.4 重复事件：循环	45
2.4 小结	50
第3章 从DHTML到DOM编程	51
3.1 作为“行为层”的JavaScript	53

3.1.1 对象检测与浏览器依赖性的比较	55
3.1.2 渐进增强	57
3.2 JavaScript和可访问性	58
3.3 良好的编码实践	59
3.3.1 命名习惯	59
3.3.2 代码布局	60
3.3.3 注释	62
3.3.4 函数	64
3.3.5 使用三元运算符简化代码	66
3.3.6 函数的分类和复用	67
3.3.7 变量和函数作用域	67
3.3.8 使用对象字面量保证脚本安全	68
3.4 小结	70
第4章 HTML与JavaScript	71
4.1 HTML文档剖析	71
4.2 在网页中使用JavaScript提供反馈信息：老的方式	75
4.3 通过DOM访问文档	80
4.4 元素的子节点、父节点、兄弟节点和值	83
4.4.1 从父节点到子节点	84
4.4.2 从子节点到父节点	85
4.4.3 兄弟节点之间	86
4.5 修改元素属性	90
4.6 创建、移除和替换元素	91
4.6.1 避免NOSCRIPT	94
4.6.2 通过innerHTML简化脚本	96
4.6.3 DOM小结：你的备忘单	97
4.6.4 DOMhelp：我们自己的辅助函数库	98
4.7 小结	102

第5章 表现与行为（CSS与事件处理）	103		
5.1 通过JavaScript改变表现层	103	7.2.4 定制表单元素	245
5.2 通过事件处理改变文档的行为	129	7.2.5 表单与JavaScript小结	246
5.2.1 W3C标准兼容的事件	131	7.3 小结	246
5.2.2 修正事件以适应W3C 不兼容的浏览器	139		
5.2.3 永不停止优化	144	第8章 与Ajax后端交互	247
5.2.4 页面加载问题及其解决方案	145	8.1 Ajax到底是什么	248
5.2.5 读取和过滤键盘输入	146	8.2 高速缓存竟带来了麻烦	254
5.2.6 事件处理的危险	150	8.3 把X放回到Ajax里面	255
5.3 小结	151	8.3.1 使用JSON代替XML	259
第6章 JavaScript的常用对象：		8.3.2 使用服务器端脚本来访问 第三方内容	261
图片和窗口	152	8.3.3 关于缓慢链接的XHR问题	264
6.1 图片与JavaScript	152	8.3.4 一个更大的Ajax示例： 关联选择框	266
6.1.1 图片编程基础	153	8.3.5 可选的动态Ajax菜单	273
6.1.2 预载图片	154	8.4 小结	280
6.1.3 翻转效果	155		
6.1.4 幻灯片显示	163	第9章 数据验证技术	282
6.1.5 图片与JavaScript小结	176	9.1 客户端JavaScript验证的优点和缺点	282
6.2 窗口与JavaScript	177	9.2 使用JavaScript保护文件内容	283
6.2.1 窗口属性	178	9.3 全能验证的神话	284
6.2.2 窗口方法	179	9.4 使用字符串和数字方法的基本 JavaScript验证	284
6.2.3 窗口与JavaScript小结	198	9.4.1 字符串验证方法	284
6.3 小结	199	9.4.2 数字验证方法	290
第7章 JavaScript与用户的交互：		9.5 正则表达式	293
导航与表单	200	9.5.1 语法和属性	294
7.1 导航与JavaScript	200	9.5.2 通配符搜索、约束范围 以及其替换	295
7.1.1 重新加载网页的恐惧	200	9.5.3 使用量词约束字符的数量	295
7.1.2 JavaScript导航基础	201	9.5.4 词界、空白字符以及其他 快捷符号	296
7.1.3 浏览器导航	203	9.5.5 使用正则表达式的方法	297
7.1.4 页内导航	204	9.5.6 圆括号分组的功能	297
7.1.5 网站导航	212	9.5.7 正则表达式资源	298
7.1.6 分页	219	9.6 验证方法小结	299
7.1.7 使用JavaScript进行导航小结	226	9.7 表单验证技术	299
7.2 表单与JavaScript	226	9.7.1 指定强制字段	299
7.2.1 JavaScript表单基础	227	9.7.2 隐藏字段方法	300
7.2.2 表单元素	228	9.7.3 指示元素方法	301
7.2.3 交互式表单：隐藏或显示 独立元素	241		

9.7.4 CSS 类方法.....	301
9.7.5 自定义属性方法.....	302
9.7.6 这些方法的缺点.....	302
9.7.7 共用验证规则.....	302
9.8 为用户反馈验证信息.....	304
9.8.1 显示错误字段的列表.....	304
9.8.2 使用可单击的错误消息 代替主表单.....	308
9.8.3 单独地突出显示错误的字段.....	310
9.8.4 即时验证反馈.....	313
9.9 其他的动态验证方法.....	314
9.10 小结.....	317
第 10 章 现代的 JavaScript 案例研究： 动态图库.....	319
10.1 缩略图图库基础.....	319
10.2 缩略图图库是什么以及它应该 做什么.....	319
10.3 静态缩略图图库.....	320
10.4 使用 JavaScript 模拟动态图库.....	320
10.5 显示标题.....	326
10.6 动态的缩略图图库.....	330
10.7 从文件夹中创建图片徽章.....	333
10.8 小结.....	340
第 11 章 使用第三方 JavaScript.....	341
11.1 网络为你提供了什么.....	341
11.2 代码片段、RSS 提要、各种 API 以及函数库.....	342
11.2.1 RSS 提要和 REST API.....	342
11.2.2 REST API 示例.....	344
11.3 使用简短精练的函数库：jQuery.....	344
11.4 使用 API：用 Google Maps 为你的 网站添加地图.....	351
11.5 完整的服务：雅虎开发人员网络 以及 YUI.....	360
11.5.1 使用 YUI 的弹性标题.....	361
11.5.2 使用 YUI 的连接管理器和 容器组件代替弹出窗口.....	366
11.5.3 YUI 小结.....	370
11.6 小结.....	371
附录 A 调试 JavaScript.....	372

JavaScript入门

本书主要讲述一种名为JavaScript的脚本语言以及如何在实际开发中使用它。在读完本书之后，你就能够：

- 理解JavaScript的语法和结构。
- 创建容易理解和维护的脚本。
- 编写不与其他JavaScript冲突的脚本。
- 编写脚本，使网站更加容易使用，且不排斥未启用JavaScript的用户。
- 编写与浏览器或用户代理无关的脚本——也就是在未来许多年中它们仍然是有用的，不会依赖于过时的技术。
- 使用JavaScript增强网站，允许没有任何编程知识的开发人员改变网站的界面外观。
- 使用JavaScript增强网站文档，允许HTML开发人员通过给某个元素简单地添加一个CSS类来使用你开发的功能。
- 在用户代理允许的时候，可以使用渐进增强来使Web文档变得更好。
- 使用Ajax架设客户端和服务器端之间的桥梁，创建更易维护，而且用户体验更加平滑流畅的网站。
- 将JavaScript作为Web方法的一部分，使你能够独立地维护它而不与其他的开发流程冲突。在这里你不会发现：
 - 如何创建华丽的但对访问者没有任何价值的特效的指导。
 - 特定于某种浏览器的JavaScript应用。
 - 只是为了证明它可以被使用但不能增强访问者体验的JavaScript代码。
 - 强制“推销”垃圾内容的JavaScript脚本，如弹出窗口或其他展示动画效果的华而不实的技术。

我坚信JavaScript在现代的Web开发中占有重要的地位，但我们无法保证访问者能够使用或体验用JavaScript所能达到的所有特效和功能。JavaScript允许通过添加、移除、显示或隐藏元素来完全地改变网页效果。我们可以提供丰富的界面，如拖放式的应用程序或是多层次下拉菜单。但是，一些访问者并不能使用拖放式的界面，因为他们只能使用键盘或是依赖语音识别来访问我们的网站，还有一些用户可能是通过听而不是看（通过屏幕阅读器）来访问的，所以不能够察觉由

JavaScript带来的变化。不仅如此，有的用户不能启用JavaScript功能，如在银行等安全级别高的环境中。因此，必须通过一些服务器端解决方案为我们用JavaScript实现的许多功能提供备用支持。

遗憾的是，JavaScript也有一段被用于弹出强制信息的历史，这些信息往往并不是用户所请求的（例如弹出窗口）。对于这种行为，我和许多专业Web设计人员感觉都很头疼。希望你不要使用从本书中获得的知识去做这样的事情。

注解 网页设计经过这些年的发展已经成熟——我们已经不再使用FONT标签，而且一些可视化属性例如bgcolor也被废弃了。我们推荐把所有的格式化和界面表现属性放到CSS文件中。

发生在JavaScript上的演化过程同样是Web开发的一部分。我们已把内容、结构和表现分开了，现在是把网站的行为从其他层分离出来的时候了。Web开发现在主要考虑的是业务需求和用户体验，而不是把一些程序放到那里，然后痴心妄想它们能在大多数平台环境下都可以使用。

现在是把JavaScript看作整个开发技术一部分的时候了，这意味着我们开发JavaScript程序与其他的像HTML、CSS这样的技术不仅不冲突，而且要和它们配合使用或弥补它们的不足。现在，我们看到出现了一种新的技术（或者至少是现有技术的一种新的使用方式），它叫Ajax。第8章将会讨论它。

在20世纪90年代，Web开发得到了飞速发展，现在创建静态或者大小固定的网站已经意义不大了。任何现代的网页设计都必须考虑未来的扩展需求。同样，它对每个人来说都应该是可访问的（这并不意味每个人都会得到相同的显示效果：例如，一个好的多列布局。可能在高分辨率的显示器上显示得很好却很难在手机或者PDA上使用），并且是可以国际化的。如今，我们已经不可能一劳永逸，做一个产品，永远都好用。因为Web是与内容相关的，且需要不断地变化，所以如果不经常升级Web产品并允许其他数据源为其添加新数据或从中获取数据，那么它很快就会过时。

简介已经足够了——你拿这本书是来学习JavaScript的。在深入讲解它之前，让我们先来快速介绍一下JavaScript的历史和渊源。

本章主要介绍：

- JavaScript是什么以及它的功用。
- JavaScript的优缺点。
- 如何把JavaScript添加到一个Web文档中，JavaScript的基本语法。
- 面向对象编程（OOP）与JavaScript的关系。
- 如何编写并运行一个简单的JavaScript程序。

很有可能你已接触过JavaScript，并且对JavaScript的含义及功用有了自己的想法，所以我们先快速浏览一下这种语言的一些基础和用途。如果你对JavaScript已经有了一定的了解，想简单了解一下更多新的特性和概念，可以直接跳到第3章。不用在这里占用太多的时间——不过如果你已经遗忘了一些知识点，复习一下也不会有坏处。

1.1 JavaScript 产生的原因

在Web发展的初期，主要有HTML和公共网关接口（CGI）。HTML定义了大部分的文本文档并指示用户代理（通常是网页浏览器）如何来显示。举个例子，标签

</p>

之间的文字就变成一个段落，在这个段落中可以使用标签

</h1>

来定义最主要的页面标题。注意大多数开始标签，都会有对应的以</开头的结束标签。

HTML有个缺点，即它的状态是固定不变的。如果想改变一些东西或者使用用户输入的数据，就需要向服务器做一个往返的请求。使用动态技术（如ColdFusion、ASP、ASP.NET、PHP或JSP）可以从表单或者参数中把信息发送到服务器，服务器然后完成计算、测试、数据库查找等。和这些技术相关联的应用程序服务器会写一个HTML文档来显示结果，然后把处理的结果以HTML文档的形式返回到浏览器来供用户查看。

这样做的问题在于任何时候只要有变化，以上整个过程都需要再重复一遍（并且重新加载网页）。这样显得比较笨重缓慢，没有网络这个新媒介向我们承诺的那么美好。现在，人们已经普遍拥有了快速的因特网连接。但是显示一个页面仍然意味着重新加载，这是一个经常失败的缓慢过程（遇到过Error 404没有？）。

我们需要更加灵活的东西——要允许Web开发人员快速对用户给予反馈并且不用从服务器重新加载页面来改变HTML。可以想象一个表单，只要有一个字段中产生了错误，它都需要重新加载，如果能够不用重新从服务器加载页面，就能快速地获得错误提示，岂不是更方便实用？这正是JavaScript的用武之地。

一些信息（比如表单上的一些计算和验证信息）并不需要依靠服务器。JavaScript可以由访问者电脑上的用户代理（通常是一个浏览器）执行。我们把这叫作客户端代码（client-side code）。这样可以减少与服务器的交互成本并且使网站运行更快。

1.2 JavaScript 是什么

JavaScript的前身是LiveScript，但是网景公司后来把名字改成了JavaScript，很可能是由于Java的火爆。这个名字经常会令人感到迷惑，因为尽管Java与JavaScript有些语法比较相近，但它们之间并没有必然的联系。

Java之于JavaScript就好比Car（汽车）之于Carpet（地毯）。

——来自Usenet^①上的JavaScript讨论组

网景公司在1996年创造了JavaScript语言，它包含在Netscape Navigator（NN）2.0浏览器中，用解释器来读取和执行添加到.html页面的JavaScript代码。从此，这种语言稳步发展壮大并越来越普及，现在大多数的浏览器都支持它。

^① 一个世界性的新闻组网络系统。——译者注

这意味着JavaScript可以用于网页中，被所有现代的浏览器所理解。但是，不同的浏览器在实现JavaScript的方式上是不同的，尽管核心的JavaScript语言是一样的。不过，JavaScript可以被用户关闭掉，并且一些公司和机构从安全角度考虑要求他们的用户这样做。这个我们稍后（贯穿本书）会进一步讨论。

关于JavaScript最大的特点就是，一旦学会了如何在浏览器编程中使用它，你就可以把它应用到其他领域中。微软的服务器（IIS）使用JavaScript去做服务器端网页编程（ASP），PDF文件现在也使用JavaScript，甚至Windows的任务管理也可以使用JavaScript代码来自动运行。许多应用程序，如Dreamweaver和Photoshop，都可以使用JavaScript来编写脚本。操作系统上的许多插件，如苹果公司的Dashboard或者Linux和Windows平台上的Konfabulator^①，甚至允许使用JavaScript编写小的帮助程序。

最近许多大公司也提供了可用在网页中的JavaScript对象和方法组成的API（应用编程接口），Google Maps就是其中的一种。只需要使用几行代码就可以在你的网站中提供可缩放和可滚动的地图。

另一个更好的特点就是，JavaScript比高级编程语言和服务器端编程语言更容易开发。它不需要像Java和C++那样需要编译，也不需要像Perl、PHP或Ruby语言那样运行在服务器上或需要在命令行执行。编写、执行、调试和应用JavaScript脚本所需的就是文本编辑器和浏览器，而这两者在所有的操作系统中都提供。当然，也有工具可以使你更加方便，例如Mozilla Venkman、Microsoft Script Debugger和kjscmd这样的JavaScript调试器。

1.3 JavaScript 的问题和价值

正如我在本章的开始提到的，JavaScript在过去的几年里已经成为Web开发的一个完整部分，但它也经常被错误地使用。结果，它就落了一个不好的名声。导致这个结果的原因是某些严重影响用户的JavaScript特效，如移动的页面元素和弹出窗口。这种情况你第一次看到印象会很深刻，但很快就变成是“有了也不错”，在有些情况下，甚至变成“没有更好”。许多类似的效果都来自DHTML时代（详见第3章）。

术语用户代理（user agent）和对其含义的缺乏理解同样也是个问题。通常，用户代理是指一个浏览器，如微软的IE、Netscape、Mozilla（Moz）、Firefox（Fx）、Opera或Safari。但是浏览器不是Web上唯一的用户代理，其他用户代理还包括：

- 辅助技术，用来帮助用户克服它们的缺陷——如语音合成（text-to-speech）软件或者盲文显示器。
- 纯文本代理，如Lynx。
- 支持Web的应用程序。
- 游戏控制台。
- 手机。

^① Konfabulator已被Yahoo!收购，并被更名为Yahoo! Widget Engine。——编者注

- 个人数字助理 (PDA)。
- 交互式的电视机顶盒。
- 搜索引擎和其他索引程序。
- 其他。

这么多类用户代理（以及一些没有更新的旧的用户代理），它们使用了不同的技术手段，对于JavaScript也是一个非常大的危险。

并不是网站所有的访问者都能体验你所应用的JavaScript增强，许多人都会出于安全的考虑关闭JavaScript。JavaScript既可以用来做好事，也可以用来做坏事。如果操作系统（如没有打补丁的Windows）允许，可以通过JavaScript来在某台计算机上安装病毒或木马，或者读取用户的信息并把它发送到另一个服务器上。

注解 没有办法知道访问者在使用什么工具，他计算机的功能如何。再者，你永远不知道访问者的经验和能力如何。这也是网络很好的一个方面——每个人都可以参与。然而，这样会给JavaScript程序员带来意外的后果。

在大多情况下，你可能想要一种服务器端的替代方案。它可以测试用户代理是否支持所需要的功能，如果不支持，服务器会使用其替代方案。

脚本语言的独立性对于网站是法律方面的要求，在英国的数字歧视法（Digital Discrimination Act）、美国法律的508条款和世界上其他地区的法律中都有定义。这意味着如果你开发的站点没有JavaScript就不能使用，或你的JavaScript增强需要用户或用户代理在不高效运行的情况下具有一定能力，那么你的客户就可以控告你歧视。

然而，JavaScript既不是邪恶的也不是没用的，它是一个优秀的工具，可以帮助在你平滑流畅的网站上冲浪的用户花费更少的时间。

1.4 JavaScript 不可靠为什么还要用

正如我提到的，JavaScript可能不是一直可用的，但这并不意味着它根本不能用。只不过它不应该是用户交互的唯一方式。

使用JavaScript有以下优点。

- 服务器交互较少：可以在提交页面到服务器前对用户输入的内容进行验证。这样减少了服务器的通信量，就意味着节约了金钱。
- 对访问者快速反馈：他们不用等待页面重新加载才可以看到是否忘了输入某些东西。
- 自动修正小错误：举个例子，如果你有一个数据库系统，预期的日期格式是`dd-mm-yyyy`，用户输入的格式是`dd/mm/yyyy`，一个智能的JavaScript脚本可以在提交表单前纠正这个错误。如果那是访问者唯一的错误，你可以给他一条错误信息，这样网站会显得好用得多。

- 通过允许访问者不用重新加载页面就可以改变用户界面或与用户界面进行交互，增强易用性；例如，使用JavaScript收缩或展开页面的某部分或为访问者提供额外的选择。这里有个典型的例子，就是选择复选框来快速过滤，例如只显示某个机场的可到达目的地，不用你重新加载页面，然后等待结果。
- 增强交互性：可以创建界面，在用户鼠标滑过的时候或使用键盘激活它们的时候做出反应。这一点使用CSS和HTML也可以做到，但是JavaScript为你提供了更多的支持和更宽的选择范围。
- 界面更丰富：如果用户允许，可以使用JavaScript包含一些拖放式的模块和进度条——一些原来只能出现在用户必须另外安装的胖客户程序（thick client application）中，例如Java applet或者像Flash这样的浏览器插件。
- 使环境轻量级：不用像Java applet或者Flash影片那样需要下载一个大文件，脚本的文件大小是比较小的并且一旦被加载就会被缓存起来（保存在内存中）。JavaScript还使用浏览器控件而不是它自己的用户界面（像Flash或Java applet）来操作其功能。这样使用户操作起来更容易，因为他们知道这些控件并且知道如何使用它们。流行的Flash和Macromedia Flex应用程序的确有流媒体的选项并且（因为基于矢量）在视觉上是可调节的，JavaScript和HTML的控件是不可以这样的。但是，它们需要安装插件。

1.5 网页中的JavaScript 和基本语法

把JavaScript应用到网页中是非常容易的，只要使用这个脚本标签：

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

对于老版本的浏览器，或如果想使用严格型XHTML（HTML的最新版本）代替过渡型 XHTML，你需要把这段代码加上注释，确保用户代理不在网页中显示它或试着把它当作HTML 标签来呈现。给代码加注释有2种不同的语法。对于HTML文档和过渡型XHTML，使用标准的 HTML注释：

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

在严格型XHTML中，需要使用CDATA注释语法来注释代码。然而，最好不要在严格型 XHTML文档中添加任何JavaScript，而要把它放到自己的文档中。详见第3章。

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

从技术上讲，在HTML文档的任何地方加入JavaScript都是可行的，并且浏览器都会解释它。可是，在流行的脚本中，有许多理由说明为什么这样做非常糟糕。但是现在我们权且先在文档的主体中加入JavaScript示例，这样可以立即看到第一个脚本在做什么。这会比第3章中更为现代和高级的技术更容易帮助你熟悉JavaScript。

注解 也有一个与script“相反”的标签——noscript，它用于添加一些只有在JavaScript不可用的情况下才会显示的内容。然而，noscript在XHTML和严格型HTML中已被废弃。如果创建的JavaScript是分离式的，就没有必要使用它。

1.5.1 JavaScript 语法

在进一步探讨之前，我们先来讨论一下JavaScript基本语法：

- //表示当前行的其他部分是注释而不是要执行的代码，所以解释器不会试图运行它。注释是一个把说明文字放到代码中的一个便捷方式，它可以提示我们这段代码打算做什么，或者帮助其他阅读这段代码的人理解代码是做什么的。
- /*表示一个多行注释的开始。
- */表示一个多行注释的结束。如果想阻止执行某段代码但是又不想永久地删除它，就可以使用多行注释。如果你对一个代码块有疑问，例如，你不能确定哪些行有问题，那么你可以每次注释掉它的一部分，从而分离出问题来。
- 大括号({和})用来表示一个代码块。它们确保大括号中间的所有行都被当作一个代码块来看待。当我们在讨论像if或for这样的结构语句和函数时，你会看到有关大括号更多的应用。
- 分号或换行定义了语句的结束，语句就是一个单行命令。在实际应用中，分号是可选的。但是在语句结束的时候，可以使用它们来使代码更清晰、更有条理，因为这样做可以使代码更容易阅读和调试（尽管你可以把许多语句放到一行中，但是最好还是把它们分别放到独立的行中以使代码更容易阅读）。在大括号后面不需要使用分号。

我们通过实际代码中来思考一下上述语法：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
<body>
<script type="text/JavaScript">
  // 说明代码作用的一行注释

/*
  多行注释，用于较长的注释
  也可用于测试时注释掉代码
*/
```

```

/*
代码由此开始，声明一个变量MyName
将用户通过提示框输入的值（参见第2章）赋给它
每条语句最后应用分号作为结束
*/
var myName = prompt ("Enter your name","");
// 如果用户输入的名字是Chris Heilmann
if (myName == "Chris Heilmann")
{
    // 弹出一个新窗口，显示"Hello Me"
    alert("Hello Me");
}

// 如果用户输入的名字不是Chris Heilmann
else
{
    // 显示另一串文字
    alert("hello someone else");
}
</script>
</body>
</html>

```

如果读者以前没有什么JavaScript经验，可能无法完全理解上面的代码。先不要管它，现在只需要弄清楚注释是怎么使用的、代码段是什么和为什么一些语句的结尾有个分号。如果你喜欢，可以运行这段代码：只要把它复制到HTML网页中，使用.html扩展名保存文档，然后在浏览器中打开它。

尽管一些语句（像if和else）跨越不止一行而且包含其他语句，但它们被看成是单条语句，后面不需要加分号。JavaScript解释器知道在if语句后面的那几行语句应该被当作一个代码块来处理，因为有大括号{}。虽不是强制的，但是把大括号内部的代码进行缩进处理是个很不错的注意，这样可以使阅读和调试更容易。我们会在第2章中讨论变量和条件语句（if和else）。

1.5.2 执行代码

浏览器从上到下来读取页面，所以代码执行的顺序取决于脚本块的顺序。一个脚本块（script block）是指<script>和</script>标签之间的代码。（同样需要注意到，不只是浏览器可以阅读代码，网站的用户也可以查看代码，所以不要把一些私密信息或者敏感内容放到里面。）下面的例子中有3个脚本块。

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
<script type="text/javascript">

```

```

alert( 'First script Block ');
alert( 'First script Block - Second Line ' );
</script>
</head>
<body>
<h1>Test Page</h1>
<script type="text/JavaScript">
  alert( 'Second script Block' );
</script>
<p>Some more HTML</p>
<script type="text/JavaScript">
  alert( 'Third script Block' );
  function doSomething() {
    alert( 'Function in Third script Block' );
  }
</script>
</body>
</html>

```

运行它，你会首先看到第一个脚本块中的alert()对话框，显示着下面的消息：

First script Block

紧跟着的是第二行中的下一个alert()对话框，显示着下面的消息：

First script Block - Second Line.

解释器会继续往下执行来到第二个脚本块，在这里alert函数会显示这样一个对话框：

Second script Block

接着其后包括alert()语句的第三个脚本块会显示下面的消息：

Third script Block

尽管最下面几行的函数中存在另一个alert语句，但是它不会执行并显示消息。这是因为它在函数定义(function doSomething())的内部，函数内部的代码只有在函数被调用的时候才会执行。

1.5.3 函数

第3章将对函数进行深入讨论，这里之所以介绍它们，是因为不了解函数就很难对JavaScript做进一步的了解。函数(function)就是一个已命名的、可复用的代码块，它被成对的大括号括起来，用来完成一个特定的任务。JavaScript包含许多内置函数，可以直接使用它们来完成一些任务，如给用户显示消息的alert()。合理地使用函数可以使程序员避免编写重复的代码。

也可以自己创建函数，在前面的代码段中我们就已经创建了一个。假定我们创建了一些代码，它们可以在某个元素里向网页中写出一条消息。我们可能需要在不同的情况下重复地使用它。当

然我们可以在需要使用它们的地方通过复制和粘贴代码段来实现，但是这种方法使代码过于冗长；如果同样的代码段在一个页面中出现三四次，那么它同样很难理解和调试。我们可以把这段代码封装到一个函数中，然后使用参数（parameter）来给函数传递运行所需要的信息。函数也可以为调用它的代码返回一个值。

要调用函数，写上它的名字，后跟一对括号()就行。（可以使用括号来传递参数，但是在没有参数的时候，仍然必须使用括号。）可以想到，调用函数必须在函数被创建之后，我们可以在第3个代码块中调用它：

```
<script type="text/JavaScript">
  alert( 'Third script Block ' );
  function doSomething(){
    alert( 'Function in Third script Block ' );
  }
  // Call the function doSomething
  doSomething();
</script>
</body>
</html>
```

本章到目前为止已经讨论了JavaScript语言的优缺点，了解了一些语法规则，学习了这种语言的一些主要组成部分（虽然只是简要地），并运行了一些JavaScript代码。你已经接触了相当多的内容。在下一章继续对JavaScript语言进行更详细的讲解前，让我们先来讨论一下成功的JavaScript开发的关键：对象。

1.6 对象

对象（object）是使用JavaScript的核心。JavaScript中的对象在许多方面和现实世界中的对象是相似的（它的确存在）。在现实世界中，对象就是“事物”（许多关于面向对象编程的书中把对象比作名词）：汽车、桌子、椅子，还有我敲的键盘。对象拥有：

- 属性（property，可比作形容词）。这个汽车是红色的。
- 方法（method，像一个句子里的动词）。启动汽车的方法可能是转动车钥匙。
- 事件（event）。转动车钥匙导致了汽车启动事件。

面向对象编程（OOP）试图通过对现实世界中的对象建模来使编程更加容易。让我们来创建一个汽车模拟程序。首先，我们会创建一个汽车对象，给它一些属性，如颜色和当前速度。接着我们需要创建方法：可能一个start方法来启动汽车，一个break方法来给汽车减速，在其中我们需要传递刹车力度信息来决定减速的效果。最后，我们需要一些事件，例如，汽油过低事件提醒我们给汽车加油。

面向对象编程就要使用这些概念。这种设计软件的方式现在在许多编程领域都非常通用和流行，但对我们最重要的就是，它是JavaScript和Web浏览器编程的核心。

我们使用的一些对象是JavaScript这种语言规范提供的一部分，例如String对象、Date对象及Math对象。同样的JavaScript对象会在一个PDF文件或者一个Web服务器中提供。这些对象提供了

许多有用的功能，可以节省大量的编程时间。举例来说，Date对象允许你从客户端（如一个用户的PC）获取当前的日期和时间。它保存日期，还提供了许多与日期相关的有用的函数，例如，把一个时区的日期/时间转换为另一个时区的。这些对象通常称作核心对象（core object），因为它们是独立于实现的。浏览器通过一些获取它的相关信息和改变应用程序界面外观的对象，使我们能够对它编程。举个例子，浏览器提供了Document对象，该对象代表对JavaScript可用的网页。可以在JavaScript中使用这个对象来为浏览器用户查看的页面添加新的HTML元素。如果你曾在不同的主机中使用JavaScript，假设使用一个Windows服务器，你会发现存放JavaScript的服务器提供了一组非常不同的主机对象，它们的功能与你想在Web服务器上做的事情是相关的。

你会在第3章中发现，JavaScript允许我们创建自己的对象。这是一个非常强大的特性，它允许我们使用JavaScript为现实世界的问题进行建模。要创建一个新的对象，我们需要使用一个叫做类（class）的模板来指定它的属性和方法。类与建筑师的图纸有点类似，它指定了什么东西应该到什么地方去做什么事情，但是它实际上并没有创建对象。

注解 JavaScript是基于对象的语言还是面向对象的语言，这个问题存在一些争论。不同之处在于基于对象的语言使用对象来编程但是不允许程序员在他们的代码设计中使用面向对象编程。面向对象的语言不只使用对象，也允许使用面向对象的设计方法来简化开发和代码设计。JavaScript允许我们创建自己的对象，但是与Java、C#等基于类的语言实现方式不同。无论如何，我们在这里不要太计较面向对象是或不是什么，而应该关注本书中对象是如何实际应用的，我们会在需要的地方关注一些基本的面向对象编程^①。

通读完本书后，你会对下列对象有更深的认识：JavaScript语言的核心对象、浏览器提供的使用JavaScript进行访问和操作的对象以及自己创建的对象。到目前为止，所有你需要知道的就是JavaScript中的对象就是可以用来给网页中添加功能的“实体”，它们可以拥有属性和方法。例如，Math对象有一个表示π值的属性，以及一个可以产生随机数的方法。

1.7 简单的 JavaScript 示例

我们用一个简单的脚本来结束本章。它首先会判断访问者的屏幕宽度，然后应用一种合适的样式表（通过在页面中添加一个额外的LINK元素）。我们会使用Screen对象，它是用户屏幕的一种表示。这个对象有一个availWidth属性，可以用它来决定该加载哪种样式。

下面是这段代码：

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

^① 基于对象语言只是部分支持面向对象编程，不支持的特性有：无隐式继承；无多态；只有部分值是对象。此外，基于对象语言往往是基于原型（一种无类的对象）的。——编者注

```

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>CSS Resolution Demo</title>
<!-- Basic style with all settings -->
<link rel="StyleSheet" href="basic.css" type="text/css" />
<!--
Extra style (applied via JavaScript) to override default settings
according to the screen resolution
-->
<script type="text/javascript">
    // Define a variable called cssName and a message
    // called resolutionInfo
    var cssName;
    var resolutionInfo;
    // If the width of the screen is less than 650 pixels
    if( screen.availWidth < 650 ) {
        // define the style Variable as the low-resolution style
        cssName = 'lowres.css';
        resolutionInfo = 'low resolution';
    // Or if the width of the screen is less than 1000 pixels
    } else {
        if( screen.availWidth > 1000 ) {
            // define the style Variable as the high-resolution style
            cssName = 'highres.css';
            resolutionInfo = 'high resolution';
        // Otherwise
        } else {
            // define the style Variable as the mid-resolution style
            cssName = 'lowres.css';
            resolutionInfo = 'medium resolution';
        }
    }
    document.write( '<link rel="StyleSheet" href="' +
    cssName + '" type="text/css" />' );
</script>
</head>
<body>
    <script type="text/javascript">
        document.write( '<p>Applied Style:' +
        resolutionInfo + '</p>' );
    </script>
</body>
</html>

```

尽管第2章会详细讲解if语句和循环，但你可能已经能够大概明白它的工作原理。第一行的if语句判断screen.availWidth是否小于650：

```
if ( screen.availWidth < 650 )
```

如果用户的屏幕是640×480，那么它的宽度就小于650，大括号里的代码就会执行，低解析样式和消息会得到定义。

```

if ( screen.availWidth < 650 ) {
// define the style Variable as the low-resolution style
cssName = 'lowres.css';
resolutionInfo = 'low resolution';
}

```

这段代码使用else语句继续检查屏幕的大小。最后的else语句只有在其他判断语句都不执行的时候才会发生，所以我们假定屏幕的大小是800×600，并相应地定义一个媒介样式和消息。

```

else {
// define the style Variable as the mid-resolution style
cssName = 'lowres.css';
resolutionInfo = 'medium resolution';
}

```

还值得注意的是，我们在这里测量的是用户的屏幕大小，而用户可能拥有一个800×600的屏幕，但是他们的浏览器窗口不一定是最化的。我们可能在应用一种并不是很合适的样式。

我们使用了另一个对象，即document对象，用它来往网页（HTML文档）中写数据。document对象的write()方法允许我们在页面中插入HTML。注意，document.write()实际上并没有改变源HTML页面的内容，只是改变了用户在他自己电脑上看到的网页。

注解 实际上，看完本书的前几章后，你就会发现document.write()非常有用。它非常适合展示脚本工作原理的小例子；适合与用户进行交互；甚至适合对一个你不太确定的程序段进行调试，以明白它是否会按你预期那样运行。它还可以在所有支持JavaScript的浏览器上运行。许多现代的浏览器为调试提供了更好的工具和方法，我们会在第3章中做更多介绍。

使用document.write()来在文件头中写一个使用了我们已定义好的样式的link元素：

```
document.write( '<link rel="StyleSheet" href="' +  
cssName + '" type="text/css" />' );
```

而且，在文档的主体部分，我们会写出消息来说明应用了哪种解析样式：

```
<script type="text/javascript">  
document.write( '<p>Applied Style: ' + resolutionInfo + '</p>' );  
</script>
```

稍后，我们会接触更复杂的示例，这些示例使用JavaScript来测试用户代理和用户界面的类型。现在，我希望这个简单示例让你对可以使用JavaScript为网页中加入灵活性有一个大致的了解。

1.8 小结

本章讲述了什么是JavaScript，它是如何工作的，以及它的优缺点。它最大的缺点是我们不能确信获得的信息完全可靠，不过，我们也提到，使用JavaScript可以使网站获得更好、更流畅的体验。

你已经运行了一些JavaScript代码，看到了如何在代码中添加注释，如何使用分号把JavaScript语句分开。你也明白了使用大括号可以让JavaScript把一组多行的代码看作一个独立的块，例如在if语句中。你已学习了JavaScript执行通常是自上而下，从第一个脚本块到最后一个脚本块，函数是一个例外，函数只有在你调用它们时才会执行。

我们还学习了对象，它是编写JavaScript的核心。不仅JavaScript本身依赖对象，浏览器也使用对象和方法来使它自己和文档可编程，最后，我们看了一个简单的示例，它读取用户的屏幕分辨率，并应用一种适合的样式表。

下一章将讲解JavaScript语言的基础。你会看到JavaScript如何存储和操纵数据，以及如何在计算中使用数据。我们还会看到使用判定语句来创建“智能的”JavaScript程序。这种语句允许我们对数据进行求值、用数据进行计算，并选择一个合适的动作分支。在熟练掌握了第2章的内容后，你就拥有了大部分基础知识，可以继续学习更多令人兴奋和真正有用的Web编程了。

数据和判定

数据和判定是所有“智能”程序的基础。在这章的开始，我们先看一下JavaScript是如何理解或表示数据的。这很重要，因为JavaScript包含许多数据类型（data type），而且根据数据类型来操作数据。错误地使用不同的数据类型会导致意想不到的结果。我们来看一些常见的数据类型问题，你会明白如何把一种数据类型转化成其他的。

我们也会学习条件语句（conditional statement）和循环（loop），2种最有价值的判定工具。为了在计算机语言中进行判定，我们需要让程序知道为了响应特定的情况应该发生什么，这就需要条件语句来解决。另一方面，循环允许简单地重复某一动作直到特定的条件被满足。例如，你可能需要循环表单里的每个输入并检查它包含的信息是否有效。

本章将从下面的许多方面阐述JavaScript。

- 在JavaScript中对信息进行分类和操作：数据类型和数据运算符；
- 变量；
- 数据类型转换；
- 数据对象简介：String、Date和Math对象；
- 数组：保存有顺序的数据集合，例如一个购物车中的商品；
- 使用条件语句、循环和数据评估进行判定。

注解 这一章中的例子会尽可能地保持简单，因此我们使用`document.write()`作为一种反馈机制，可以让你方便地看到结果。在以后的章节中，你会学到其他更流行和通用的方法。

2.1 数据、数据类型和数据运算符

数据是用来保存信息的。为了更有效地保存数据，JavaScript需要为每个数据分配一个类型（type）。这个类型规定了使用这个数据可以干什么或不能干什么。举个例子，JavaScript中有一种数据类型是数字（number），它允许对它拥有的数据进行一些计算。

在JavaScript中用来保存数据的基本类型有以下3种。

- 字符串（string）：一连串的字符，如“some characters”。

- 数字 (number): 一个数, 包括浮点数。
- 布尔值 (boolean): 包含一个真值 (true) 或假值 (false)。

有时候它们也被称作基本 (primitive) 数据类型, 因为它们只保存单个值。还有2个不同的基本数据类型: 它们不保存信息, 用来对特定的情况给出我们警告。

- 空值 (null): 表示没有数据。
- 未定义 (undefined): 表示没有定义也没有赋值。当你使用变量的时候, 这种类型非常重要的。

在这一章中我们会大量地使用这些数据类型。

2.1.1 字符串数据类型

JavaScript解释器要求字符串数据被单引号或双引号 (通称为定界符, delimiter) 括起来。例如, 下面这个脚本会往页面中写入some characters:

```
<html>
  <body>
    <script type="text/javascript">
      document.write( "some characters" );
    </script>
  </body>
</html>
```

引号不会被写到页面上因为它们不是字符串的一部分; 它们只是简单地告诉JavaScript字符串的开始和结束的地方。使用单引号也可以轻松地做到这一点:

```
<html>
  <body>
    <script type="text/javascript">
      document.write( 'some characters' );
    </script>
  </body>
</html>
```

只要你使用与开始该字符串对应的引号结束这个字符串, 2种方法都可以。不要试图以以下方式使用它们:

```
document.write( 'some characters' );
document.write( "some characters" );
```

当然, 你可能需要在字符串本身中使用单引号或者双引号, 这种情况下需要使用一个不同的定界符。如果你使用双引号来标记, 这个指令会按你意愿进行解释:

```
document.write( "Paul's characters" );
```

但如果你使用单引号来进行标记, 它们将无法正确解释:

```
document.write( 'Paul's characters' );
```

这样做会给你一个语法错误, 因为JavaScript解释器以为这个字符串在Paul中的l后结束了,

它不明白后面要发生什么。

注解 JavaScript语法和英语的语法类似，是使语言“可理解”的规则集合。就像英语中的语法错误会使一个句子显得没有意义一样，JavaScript中的语法错误也会使这个指令没有意义。

要避免产生JavaScript语法错误，可以使用单引号标记来分隔包含双引号的字符串或者相反：

```
document.write("Paul's numbers are 123");
document.write('some "characters"' );
```

另一方面，如果你想在字符串中既包含单引号，也包含双引号，你需要使用转义序列(escape sequence)。实际上，使用转义序列来代替我们目前使用的引号是一个好的编程习惯。因为它们使你的代码更加易懂。

转义序列

转义序列也可用于需要使用键盘不能直接输入的字符时（比如普通键盘中没有的人民币符号¥）。表2-1列出了一些最常用的转义序列。

表2-1 常用的转义序列

转义序列	字符表示的意思
\b	回退
\f	换页
\n	换行
\r	回车符
\t	制表符
\'	单引号
\"	双引号
\\\	反斜杠
\xNN	NN是一个十六进制的数，表示拉丁字符集中的一个字符（拉丁字符集是说英语国家的规范）
\uDDDD	DDDD是一个十六进制的数，表示一个Unicode字符

我们来改进一下这个会引起语法错误的字符串：

```
document.write('Paul's characters');
```

使用这个转义序列(\')就可以被正确的解释了：

```
document.write('Paul\'s characters');
```

这个转义序列告诉JavaScript解释器，这个单引号属于字符串本身，而不是一个定界符。

ASCII是一个字符编码方法，它使用0~254的值。另外，可以使用十六进制\xNN转义序列表示的ASCII值来指定一个字符。这个字符C在十进制中是67，在十六进制中是43，因此我们可以像这样使用转义序列来往页面中写入它：

```
document.write( "\x43" );
```

转义序列\uDDDD的工作原理大致和以上相同，但它用的是拥有65 535个字符的Unicode字符编码方法。因为在前面的几百个ASCII编码和Unicode编码是一致的，可以使用如下的方式往页面中写入字符C：

```
document.write( '\u0043' );
```

要可以获得ASCII和Unicode的详细信息，最好查找网络。对于Unicode，可以试一下<http://www.unicode.org>。

2.1.2 运算符

JavaScript有许多运算符，你可以在程序中用来操作数据；你或许会从数学的角度认识它们。表2-2介绍了一些最常用的运算符。

表2-2 JavaScript运算符

运算符	功 能
+	对两个数进行加法运算或者连接两个字符串
-	从第一个数中减去第二个数
*	对两个数进行乘法运算
/	第一个数除以第二个数
%	获得除法的余数：例如 $98 \% 10 = 8$
--	数字递减1：只有在变量中有效，我们稍后会提到
++	数字递增1：只有在变量中有效，我们稍后会提到

下面是它们的应用：

```
<html>
<body>
<script type="text/javascript">
    document.write( 1 - 1 );
    document.write( "<br />" );
    document.write( 1 + 1 );
    document.write( "<br />" );
    document.write( 2 * 2 );
    document.write( "<br />" );
    document.write( 12 / 2 );
    document.write( "<br />" );
    document.write( 1 + 2 * 3 );
    document.write( "<br />" );
    document.write( 98 % 10 );
</script>
</body>
</html>
```

输出如下：

```
0  
2  
4  
6  
7  
8
```

跟数学一样，JavaScript也定义了一些运算优先级。乘法比加法的优先级要高，所以这个计算 $1 + 2 \times 3$ 会按下面的步骤执行：

$$2 * 3 = 6$$

$$6 + 1 = 7$$

所有的运算符都有一个优先级顺序。乘法、除法和求余的优先级相同，因此当它们出现在同一个等式中的时候，会按照从左到右的顺序执行。试着计算一下：

$$2 * 10 / 5 \% 3$$

计算结果是1，因为计算的顺序是简单的从左到右：

$$2 * 10 = 20$$

$$20 / 5 = 4$$

$$4 \% 3 = 1$$

加法和减法同样也拥有相同的优先级。

可以使用括号来提高某部分计算的优先级。例如，可以按照下面的方式，把1和1相加，然后乘以5：

$$(1 + 1) * 5$$

计算的结果会是10，但没有括号的话结果就会是6。实际上，使用括号是个非常好的主意，即使它们不必要，因为它们可以帮助使执行的顺序变得清晰。

如果使用不止一对括号，JavaScript会简单地按照从左到右的顺序执行；如果你使用了内部括号，则按照从内到外的顺序执行：

```
document.write( ( 1 + 1 ) * 5 * ( 2 + 3 ) );
```

下面是这个计算的优先执行顺序：

$$(1 + 1) = 2$$

$$(2 + 3) = 5$$

$$2 * 5 = 10$$

$$10 * 5 = 50$$

正如我们看到的，JavaScript的加法运算符把这些值加到一起。它如何处理这两个值取决于你

使用的数据类型。例如，如果你处理的是两个存为数字数据类型的数，这个+运算符会把它们加到一起。然而，如果你处理的数据其中有一个数据类型是字符串（使用定界符来表示的），这两个值会被拼接在一起。例如：

```
<html>
<body>
<script type="text/javascript">
    document.write( 'Java' + 'Script' );
    document.write( 1 + 1 );
    document.write( 1 + '1' );
</script>
</body>
</html>
```

能够对字符串使用加法运算是非常方便的[这种情况下也叫拼接运算（concatenation operate）]，但如果处理的其中一个值正好与你期望的数据类型不同，它也会产生意想不到的结果。我们稍后会看一些这样的简单示例，并且分解它们。

如果处理的文本（literal）值都如我们目前所使用的那样，那么它不会有大问题。可是，你在程序中需要处理的很多数据都是用户输入或者脚本产生的，因此没有办法预先准确地知道将要处理的是什么值。这就要用到变量（variable）。变量是脚本中的数据占位符，它们对JavaScript来说非常重要。

2.1.3 JavaScript 变量

对变量来说，JavaScript可能是最宽松的语言了。在使用它之前不必定义一个变量是什么，并且在脚本中可以随时改变一个变量的类型。然后，为了容易维护并保持一个严格的编码语法，在脚本的开头明确地声明变量或者在函数中明确定义局部变量是一个非常好的习惯。

我们通过一个唯一的名字和使用var关键字来定义一个变量。

变量名必须以字母表中的一个字符或者一个下划线开始，而名字的其他部分可以由数字、字母、美元符和下划线构成。不要用任何其他的字符。

注解 跟JavaScript中的大多数词语一样，变量名是区分大小写的：thisVariable和ThisVariable是不同的变量。给变量命名要非常小心；如果给它们命名不一致，那么你可能会碰到任何类型的错误。为了避免那样的结果，大多数程序员使用骆驼命名法（camel notation）：变量名以一个小写字母开始，而后面的单词则首字母大写并且没有空格。如变量名thisVariable。

始终给你的变量起有意义的名字。在下一个例子中，我们将写一个汇率转换程序，因此我们会使用像euroToDollarRate和dollarToPound这样的变量名。对变量进行描述性命名有2个优点：当你隔一段时间回过头再来看的时候，更容易记得代码是做什么的；刚接触代码的其他人可以更容易明白程序要做什么。代码的可读性和布局对网页的开发是非常重要的。它可以更快速、更容易地发现错误并调试它们，然后按照你想要的修改代码。

注解 变量声明以关键字var开始，尽管就技术而言并不是必需的。不使用它可能有其他问题，你会在下一步中看到^①。

讲完了声明变量的原则，下面开始声明变量。可以声明一个变量而不对它进行初始化（给它赋一个值）：

```
var myVariable;
```

然后它就准备着等待被赋值。用于保存用户输入的变量就经常如此。

也可以同时对变量进行声明和初始化：

```
var myVariable = "A String";
var anotherVariable = 123;
```

或者可以通过把prompt()函数的返回值或一个计算的总和赋给该变量来声明并初始化它：

```
var eurosToConvert = prompt( "How many Euros do you wish to convert", "" );
var dollars = eurosToConvert * euroToDollarRate;
```

prompt()函数是JavaScript的一个函数，它用来让用户输入一个值然后把它返回到代码中。这里我们把输入的值赋给变量eurosToConvert。

最好对变量初始化，尤其是在你可以给它们赋一个对程序有用的默认值的时候。哪怕初始化一个变量为空字符串也好，因为你可以对它进行核对而不会引起因为变量没有值而弹出的错误信息。

让我们看一下变量是如何提高你的代码的可读性和实用性的。下面是一个没有任何变量的代码块：

```
<html>
  <body>
    <script type="text/javascript">
      document.write( 0.872 * prompt( "How many Euros do you wish to convert", "" ) );
    </script>
  </body>
</html>
```

很难看出，这段代码是把欧元(euro)转化成美元(dollar)，因为没有任何东西告诉我们0.872就是汇率。可是这段代码工作得很好；如果你用数字10试验一下，你就会得到如下的结果：

8.72

我们在这个例子中使用了window对象的prompt()方法来获得用户的输入（这个实例中window对象是可选的，为了保持代码简短你可以忽略它）。这个方法有2个参数：一个是显示在用户输入框上面的标签，另一个是输入框的初始值。在第4章中，你会学到更多关于prompt()及其使用的知识。假设我们需要结果显示更多的信息，如：

^① 函数内用Var声明并初始化的变量只在函数内有效。参阅3.3.7节。——编者注

10 Euros is 8.72 Dollars

不使用变量，能实现的唯一方式就是让用户对他们想转换的欧元数输入两次，那样确实对用户显得不很友好。可是使用变量，我们可以临时把数据存起来，然后在我们需要的时候多次调用它：

```
<html>
<body>
<script type="text/javascript">
    // Declare a variable holding the conversion rate
    var euroToDollarRate = 0.872;

    // Declare a new variable and use it to store the
    // number of euros
    var eurosToConvert = prompt( "How many Euros do you wish to convert", "" );
    // Declare a variable to hold the result of the euros
    // multiplied by the conversion
    var dollars = eurosToConvert * euroToDollarRate;
    // Write the result to the page
    document.write( eurosToConvert + " euros is " + dollars +
        " dollars" );
</script>
</body>
</html>
```

我们使用了3个变量：一个用来存储欧元到美元的汇率，另一个存储需要转化的欧元数，最后一个保存着转换成美元的结果。然后使用两个变量把结果写出来。这个脚本不只更具实用性，而且也更加容易读懂。

2.1.4 不同数据类型的转换

大多数时候，JavaScript解释器能够识别出我们想使用的数据类型。举个例子，在下面的代码中，解释器明白数字1和2是数字类型，并会对它们做相应处理：

```
<html>
<body>
<script type="text/javascript">
    var myCalc = 1 + 2;
    document.write( "The calculated number is " + myCalc );
</script>
</body>
</html>
```

结果显示为：

The calculated number is 3

可是，如果重写代码，使用prompt()函数允许用户输入自己的数字，我们会得到一个完全不同的结果：

```
<html>
<body>
<script type="text/javascript">
var userEnteredNumber = prompt( "Please enter a number", "" );
var myCalc = 1 + userEnteredNumber;
var myResponse = "The number you entered + 1 = " + myCalc;
document.write( myResponse );
</script>
</body>
</html>
```

如果你在提示框中输入2，然后你就会被告知：

The number you entered + 1 = 12

JavaScript解释器不是把这两个数加在一起，而是把它们进行了连接操作。这是因为prompt()函数实际上把用户输入的值返回为字符串数据类型，尽管这个字符串中包含数字字符。连接运算发生在这一行：

```
var myCalc = 1 + userEnteredNumber;
```

从效果上看，如同我们写成了：

```
var myCalc = 1 + "2";
```

可是，如果我们使用减法运算符：

```
var myCalc = 1 - userEnteredNumber;
```

userEnteredNumber会从1中减去。减法运算符不适用于字符串数据，所以JavaScript能计算出我们想把该数据当作数字来对待，接着把这个字符串转化成一个数，然后计算。这个同样适用于*和/运算符。typeof()运算符返回传递给它的参数的数据类型，因此我们可以使用它来看JavaScript解释器正在处理哪种数据类型：

```
<html>
<body>
<script type="text/javascript">
var userEnteredNumber = prompt( "Please enter a number", "" );
document.write( typeof( userEnteredNumber ) );
</script>
</body>
</html>
```

这段代码会把字符串写到页面。要确保解释器正在使用我们想要的数字数据类型，方法就是显式地声明这个数据就是一个数。有3个函数可以用来实现这一目的。

- Number(): 设法把括号里的变量值转换成一个数。

- parseFloat(): 设法把括号里的值转换成一个浮点数。它从左到右地逐个字符地解析字符

串，直到遇到一个字符不能用在数字里。然后它会在那个点停止并把这个字符串转换成数字。如果第一个字符就不能在数字里使用，返回的结果是NaN（它代表非数字，Not a Number）。

- parseInt()：把括号里的值转换成一个整数，它不用四舍五入，而是把小数部分直接去掉。任何传递到这个函数中的非数字参数都会被丢弃掉。如果第一个字符不是+、-或一个阿拉伯数字，返回的结果是NaN。

让我们来看一下这些函数在实际上是如何工作的：

```
<html>
<body>
<script type="text/javascript">
var userEnteredNumber = prompt( "Please enter a number", "");
document.write( typeof( userEnteredNumber ) );
document.write( "<br />" );
document.write( parseFloat( userEnteredNumber ) );
document.write( "<br />" );
document.write( parseInt( userEnteredNumber ) );
userEnteredNumber = Number( userEnteredNumber )
document.write( "<br />" );
document.write( userEnteredNumber );
document.write( "<br />" );
document.write( typeof( userEnteredNumber ) );
</script>
</body>
</html>
```

如果输入23.50，你会得到这样的输出：

```
string
23.5
23
23.5
number
```

在第一行中输入的数据读成一个字符串。接着parseFloat()函数把23.50从一个字符串转换成一个浮点数，在下一行中parseInt()函数把小数部分给直接去掉（不四舍五入）。然后使用Number()函数把这个变量转换成一个数字，并使用变量userEnteredNumber来保存结果（覆盖它所保存的字符串），在最后的一行可以看到userEnteredNumber的数据类型确实是个数字。

如果在用户提示框中输入23.50abc，你会得到这样的输出：

```
string
23.5
23
NaN
number
```

两个结果是非常相似的，但是这次Number()函数返回了NaN。parseFloat()和parseInt()函数仍然返回一个数字，因为它们是从左到右尽可能多地把字符串转换成数字，当遇到一个非数字时才停止。Number()函数会拒绝任何包含非数字字符的字符串（阿拉伯数字、一个有效的小数位、+和-符号是允许的，其他的字符则不行）。

如果输入abc，你就会得到：

```
string
NaN
NaN
NaN
number
```

没有任何一个函数可以发现一个有效的数字，因此它们都返回NaN，可以把它看作一个数字数据类型，但不是一个有效的数字。这是一个检查用户输入有效性的很好方法，稍后我们就会使用它来进行验证。

回到开始时提出的问题：使用prompt()来重新得到一个数字。所有我们要做的就是使用讨论prompt()函数时用到的其中一个函数，告诉解释器用户输入的数据应该转化成数字数据类型：

```
<html>
<body>
<script type="text/javascript">
    var userEnteredNumber = Number( prompt( "Please enter a number", "" ) );
    var myCalc = 1 + userEnteredNumber;
    var myResponse = "The number you entered + 1 = " + myCalc;
    document.write( myResponse );
</script>
</body>
</html>
```

这样不会抛出任何错误，但是它对用户也没有多大帮助，因为NaN的含义并不是一个常识。稍后我们会处理条件语句，你会了解如何阻止对于不了解JavaScript的用户没有意义的输出。

到目前为止，我们已讲解了所有你需要了解的基本数据类型和变量。你已经明白，基本数据类型可以简单地保存一个值。可是，JavaScript也可以处理复杂的数据，它使用复合(composite)数据类型来实现这个功能。

2.2 复合数据类型：数组和对象

复合数据类型和简单数据类型是不同的，因为它们可以保存不止一个值。复合数据类型有两种。

- 对象(object)：包含任一对象（包括浏览器提供的对象）的一个引用。
- 数组(array)：包含一个或多个其他的数据类型。

先看一下对象数据类型。回想一下第1章中的讨论，对象的作用是模拟现实世界中的实体。这些对象可以保存数据并提供一些属性和方法。

2.2.1 JavaScript 提供的对象：String、Date 和 Math

这3个对象做着3种不同的事情。

- String对象：存储一个字符串，并提供处理字符串需要的属性和方法。
- Date对象：存储一个日期，并提供处理它的方法。
- Math对象：不存储数据，但提供操纵数学上数据的属性和方法。

我们开始学习String对象吧。

1. String对象

前面我们通过给基本类型的字符串一些需要保存的字符串来创建它们，像下面这样：

```
var myPrimitiveString = "ABC123";
```

String对象略有不同，它不止允许我们存储字符，还提供了操作和改变这些字符的方法。可以显式或隐式地创建字符串对象。

- 创建String对象

先看一下隐式的方法：首先声明一个新的变量，并给它赋一个基本类型的新字符串来初始化它。现在使用typeof()来确认变量myStringPrimitive中的数据就是一个基本字符串：

```
<html>
<body>
<script type="text/javascript">
var myStringPrimitive= "abc";
document.write( typeof( myStringPrimitive ) );
</script>
</body>
</html>
```

可是我们仍然可以对它使用String对象的方法。JavaScript会把这个字符串从基本类型简单地转换成一个临时的String对象，对它使用方法，然后再把数据类型转换成字符串。可以使用String对象的length属性来试验一下：

```
<html>
<body>
<script type="text/javascript">
var myStringPrimitive= "abc";
document.write( typeof( myStringPrimitive ) );
document.write( "<br>" );
document.write( myStringPrimitive.length );
document.write( "<br>" );
document.write( typeof( myStringPrimitive ) );
</script>
</body>
</html>
```

下面是你在浏览器窗口中看到的结果：

```
string
3
string
```

因此myStringPrimitive在临时转换后仍然保存着一个基本类型的字符串。我们也可以显式地创建String对象：使用new关键字和构造函数（constructor）String()：

```
<html>
<body>
<script type="text/javascript">
    var myStringObject = new String( "abc" );
    document.write( typeof( myStringObject ) );
    document.write( "<br />" );
    document.write( myStringObject.length );
    document.write( "<br />" );
    document.write( typeof( myStringObject ) );
</script>
</body>
</html>
```

加载这个页面会显示下面的结果：

```
object
3
object
```

这个脚本和前一个脚本的唯一差别就在于第一行，我们创建新对象并为这个String对象提供一些字符存储：

```
var myStringObject = new String( "abc" );
```

检查length属性的结果是一样的，不管我们是显式还是隐式地创建String对象。显式与隐式地创建String对象的唯一真正的区别是，如果你要重复地使用同样的字符串，显式地创建字符串有更高的效率。显式地创建String对象还有助于防止JavaScript解释器混淆数字和字符串。

- 使用String对象的方法

String对象有许多方法，这里只讨论其中的2个：indexOf()和substring()方法。

你已经看到，JavaScript字符串是由字符组成的。这些字符中的每一个都有一个索引。这个索引是从0开始的，所以第一个字符的位置的索引是0，第二个是1，以此类推。方法indexOf()查找并返回子字符串起始的索引位置（lastIndexOf()方法则返回子字符串结束的位置）。举个例子，如果想让用户输入Email地址，我们能检查他们的输入中是否包含@符号。（虽然这不能确保邮件地址是合法的，但至少可以在很大程度上确保其有效，稍候我们会在本书中接触更复杂的数据校验。）

接下来，使用prompt()方法获取用户的Email地址，然后检查输入中是否包含@符号，并使用indexOf()返回@符号的索引：

```
<html>
  <body>
    <script type="text/javascript">
      var userEmail= prompt( "Please enter your email" );
      address ", "" );
      document.write( userEmail.indexOf( "@" ) );
    </script>
  </body>
</html>
```

如果没有发现@，在页面中会输出-1。只要在这个字符出现在输入字符串中，不管具体在任何位置，都会返回索引中它的位置，换句话说某个大于-1的数。

`substring()`方法使用子字符串的起始和结束位置的索引作为参数，从另一个字符串中截取一个字符串。可以不使用第2个参数来返回从第一个索引到字符串结束的所有字符串。因此要截取从第3个字符（索引2）到第6个字符（索引5）的所有字符，我们这样写：

```
<html>
  <body>
    <script type="text/javascript">
      var myOldString = "Hello World";
      var myNewString = myOldString.substring( 2, 5 );
      document.write( myNewString );
    </script>
  </body>
</html>
```

在浏览器中会看到页面输出了llo。注意`substring()`方法复制了它返回的子字符串；但它没有改变原来的字符串。

处理未知的值时，`substring()`方法真的会用到它自己。下面是另一个例子，它同时使用了`indexOf()`和`substring()`方法：

```
<html>
  <body>
    <script type="text/javascript">
      var characterName = "my name is Simpson, Homer";
      var firstNameIndex = characterName.indexOf( "Simpson," );
      " ) + 9;
      var firstName = characterName.substring( firstNameIndex );
      document.write( firstName );
    </script>
  </body>
</html>
```

我们在变量`characterName`的字符串中提取Homer，使用`indexOf()`查找姓的起始位置，然后给它加上9得到名的起始位置的索引（因为“Simpson.”是9个字符的长度），并把它存储在`firstNameIndex`中。它在`substring()`方法中用来提取从名的起始位置开始的所有字符——我们没有指定最后一个索引，因此这个字符串中的其余字符都会被返回。

现在来看一下`Date`对象。它允许存储日期并提供一些有用的日期/时间相关功能。

2. Date对象

JavaScript没有基本的日期数据类型，所以只能显式地创建Date对象。创建新的Date对象和创建String对象的方式是一样的，使用关键字new和Date构造函数。下面这行创建了一个包含当前日期和时间的Date对象：

```
var todaysDate = new Date();
```

为了创建一个存储了特定日期或时间的Date对象，可以简单地把日期或时间放到括号的里面：

```
var newMillennium = new Date( "1 Jan 2000 10:24:00" );
```

不同的国家使用不同的顺序来描述日期。举个例子，美国指定的日期格式是MM/DD/YY，而欧洲的日期格式是DD/MM/YY，在中国它们的格式是YY/MM/DD。如果使用简写的名字指定月份，那么可以以任意的顺序使用：

```
var someDate = new Date( "10 Jan 2002" );
var someDate = new Date( "Jan 10 2002" );
var someDate = new Date( "2002 10 Jan" );
```

实际上，Date对象可以拥有许多参数：

```
var someDate = new Date( aYear, aMonth, aDate,➥
anHour, aMinute, aSecond, aMillisecond )
```

要使用这些参数，首先需要指定年份和月份，然后使用所需要的参数，可是你不得不按顺序逐个使用且不能在它们之间选择。举例来说，可以指定年份、月份、日期和小时：

```
var someDate = new Date( 2003, 9, 22, 17 );
```

可是你不能指定年份、月份，然后指定小时：

```
var someDate = new Date( 2003, 9, , 17 );
```

注解 尽管我们通常认为9月（September）是第9个月份，但是JavaScript从0（1月）开始计算月份，因此月份8表示9月。

- 使用Date对象

Date对象拥有许多方法，可以用来获取或设置一个日期或时间。可以使用本地时间（你的计算机所在时区的时间）或者UTC（协调世界时间，也叫格林尼治标准时间）。虽然这个可能非常有用，但你要意识到在处理Date时，许多人并没有正确地设置他们的时区。下面的示例展示了其中的一些方法：

```
<html>
<body>
<script type="text/javascript">
// Create a new date object
var someDate = new Date( "31 Jan 2003 11:59" );
```

```

// Retrieve the first four values using the
// appropriate get methods
document.write( "Minutes = " + someDate.getMinutes() + "<br>" );
document.write( "Year = " + someDate.getFullYear() + "<br>" );
document.write( "Month = " + someDate.getMonth() + "<br>" );
document.write( "Date = " + someDate.getDate() + "<br>" );
// Set the minutes to 34
someDate.setMinutes( 34 );
document.write( "Minutes = " + someDate.getMinutes() + "<br>" );
// Reset the date
someDate.setDate( 32 );
document.write( "Date = " + someDate.getDate() + "<br>" );
document.write( "Month = " + someDate.getMonth() + "<br>" );
</script>
</body>
</html>

```

下面是你会得到的结果:

```

Minutes = 59
Year = 2003
Month = 0
Date = 31
Minutes = 34
Date = 1
Month = 1

```

代码的这一行第一眼看起来可能有点违反常规:

```
someDate.setDate( 32 );
```

JavaScript知道在一月份中没有32天，因此解释器没有把日期设置为1月32日，而是从1月1日起数了32天，所以给我们返回了2月1日。

如果需要在一个日期上增加一些天数，这是个非常方便的特性。通常如果想在一个日期上增加许多天，我们不得不考虑在不同月份中的天数，还有是不是闰年，但是如果我们使用JavaScript的日期来处理则非常地容易：

```

<html>
<body>
<script type="text/javascript">
// Ask the user to enter a date string
var originalDate = prompt( Enter a date (Day, Name of the Month, Year), "31 Dec 2003" );
// Overwrite the originalDate variable with a new Date object
var originalDate = new Date( originalDate );
// Ask the user to enter the number of days to be added, and convert to number
var addDays = Number( prompt( "Enter number of days to be added", "1" ) )

```

```

// Set a new value for originalDate of originalDate
// plus the days to be added
originalDate.setDate( originalDate.getDate( ) + addDays )
// Write out the date held by the originalDate
// object using the toString( ) method
document.write( originalDate.toString( ) )
</script>
</body>
</html>

```

如果在提示时输入31 Dec 2003（2003年12月31日），还有1作为需要增加的天数，那么得到的结果将会是Thu Jan 1 00:00:00 UTC+0800 2004（星期四，2004年1月1日，0点0分0秒）。

注解 注意到在这个脚本的第3行中，我们使用了Math对象的Number()方法。如果不使用程序也能正常运行，但是结果将是不一样的。如果你不希望使用这个方法来转换不同的数据类型，那么有一个小技巧：使用parseInt()、parseFloat()或者Number()函数从一个可以转换成数字的字符串中减去0，那么你就会把它转换成一个数，如果给一个数上加一个空字符串''，那么就会把它转换成一个字符串，这个功能通常需要使用toString()来处理。

在代码的第4行中，我们设置日期为originalDate.getDate()返回的值加上需要增加的天数而计算出的当前日期，最后一行使用toString()方法以字符串的形式输出originalDate这个日期对象所包含的日期。如果正在使用IE 5.5以上版本或者基于Gecko的浏览器（Mozilla、Netscape 6以上版本），单独使用日期的toDateString()函数会产生一个格式化好的字符串。可以使用同样的方法获取和设置日期。如果你正在使用UTC时间，所有需要做的就是把UTC加到这个方法名上。因此getHours()变为getUTCHours()、setMonth()变为setUTCMonth()，以此类推。也可以使用getTimezoneOffset()方法返回计算机的本地时间和UTC时间以小时计的时差。（你必须确认用户已正确设置了他们的时区，并知道不同国家之间的夏令时差别。）

注解 对于严格的日期处理，JavaScript可能并不是最合适的技术，因为你无法信任客户端的计算机都正确地设定了。但可以使用服务器端的语言来填充JavaScript脚本中的初始日期，达到同样的目的。

3. Math对象

Math对象提供了许多与数学相关的功能，例如获得一个数的平方或者产生一个随机数。Math对象和Date与String对象有2点不同：

- 不能显式地创建一个Math对象，直接使用它就可以了。
- Math对象不存储数据，和String与Date对象不同。

可以按照下面的格式来调用Math对象的方法：

```
Math.methodOfMathObject( aNumber ):
alert( "The value of pi is " + Math.PI );
```

接下来我们会学习几个常用的方法（详细参考请在<http://www.mozilla.org/docs/web-developer/>上搜索）。在这里我们看一下舍入数字和产生随机数的方法。

● 舍入数字

在前面你曾看到parseInt()函数会通过消去小数点后面的一切来使一个小数变成整数（因此24.999变为24）。经常地，你会需要更多的算术方面的精确计算，举个例子，如果你正在处理财务计算，可以使用Math对象的3个舍入函数种的一个：round()、ceil()和floor()。下面是它们如何工作的。

- round(): 当小数是0.5或者大于0.5的时候向上入一位。
- ceil(): 始终向上舍入，因此23.75变为24，23.25也是如此。
- floor(): 始终向下舍入，因此23.75变为23，23.25也是如此。

这里通过一个简单的示例来看看它们是如何工作的：

```
<html>
<body>
<script type="text/javascript">
var numberToRound = prompt( "Please enter a number", "" );
document.write( "round( ) = " + Math.round( numberToRound ) );
document.write( "<br>" );
document.write( "floor( ) = " + Math.floor( numberToRound ) );
document.write( "<br>" );
document.write( "ceil( ) = " + Math.ceil( numberToRound ) );
</script>
</body>
</html>
```

尽管我们使用prompt()可以从用户那里获取一个值（前面看到它会返回一个字符串），但是返回的数仍然会被当作一个数来对待。这是因为只要字符串中包含能被转换为数字的东西，舍入函数就会进行转换。

如果输入23.75，就会得到如下的结果：

```
round() = 24
floor() = 23
ceil() = 24
```

如果输入-23.75，我们得到：

```
round() = -24
floor() = -24
ceil() = -23
```

● 生成一个随机数

可以使用Math对象的random()方法，生成一个大于等于0但小于1的随机小数。通常为了利用

它，你需要乘以某个数，然后再使用其中的一个舍入方法。

举个例子，为了模拟一次掷骰子事件，我们需要生成一个1~6之间的一个随机数，可以通过把随机小数乘以5，获得0~5之间的一个小数，接着使用round()方法对这个小数进行四舍五入得到一个整数。（我们不能乘以6，然后每次使用ceil()方法向上舍入，因为那样会偶然地得到0。）然后该整数位于0~5之间，所以通过加1，我们就能得到一个1~6之间的整数。这种方法不能给我们模拟一个完美的掷骰子事件，但是对于大多数的目的都足够了。下面是这段代码：

```
<html>
<body>
<script type="text/javascript">
    var diceThrow = Math.round( Math.random( ) * 5 ) + 1;
    document.write( "You threw a " + diceThrow );
</script>
</body>
</html>
```

2.2.2 数组

JavaScript允许我们使用一个数组（array）来存储和访问相关的数据。一个数组有点像一行单元格（元素），每个单元格包含一个独立的数据项。一个数组可以存储JavaScript支持的任何数据类型。因此，举例来说，可以使用一个数组来处理用户从中选择的项目列表，或者一组图像坐标，或者一组图片的引用。

Array对象，类似String和Date对象，需要使用new关键字和构造函数来创建。可以在创建一个Array对象时初始化它：

```
var preInitArray = new Array( "First item", "Second item", -->
    "Third Item" );
```

或者设置它来保存一定数目的数据项：

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

或者只创建一个空数组：

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

可以通过为这些元素赋值的方式来为数组添加新的数据项：

```
anArray[0] = "anItem"
anArray[1] = "anotherItem"
anArray[2] = "andAnother"
```

提示 可以不使用array()构造函数；相反使用简写方式是完全有效的：

```
var myArray = [1, 2, 3];
var yourArray = ["red", "blue", "green"]
```

只要构造了一个数组，就可以使用中括号，通过索引和位置（它也是基于0的）来访问它的元素：

```
<html>
<body>
<script type="text/javascript">
var preInitArray = new Array( "First Item", ↪
"Second Item", "Third Item" );
document.write( preInitArray[0] + "<br>" );
document.write( preInitArray[1] + "<br>" );
document.write( preInitArray[2] + "<br>" );
</script>
</body>
</html>
```

如果想循环整个数组，使用索引数来存储数据项是非常有用的，我们下面会看一下循环。可以不使用数字索引而使用关键字来访问数据元素，像下面这样：

```
<html>
<body>
<script type="text/javascript">
// Creating an array object and setting index
// position 0 to equal the string Fruit
var anArray = new Array( );
anArray[0] = "Fruit";
// Setting the index using the keyword
// 'CostOfApple' as the index.
anArray["CostOfApple"] = 0.75;
document.write( anArray[0] + "<br>" );
document.write( anArray["CostOfApple"] );
</script>
</body>
</html>
```

关键字在下面的一些情况下是非常有效的：在你可以给数据分配有用的标签，或者存储一些只有在上下文中才有效的实体的地方，如一个图像坐标列表。可是，如果它们已经使用关键字被设置了，那么你不能使用索引数来访问实体（在一些其他的语言中是可以的，如PHP）。也可以使用变量作为索引。我们使用变量（分别保存一个字符串和一个数）代替文本值，改写一下前面的示例：

```
<html>
<body>
<script type="text/javascript">
var anArray = new Array( );
var itemIndex = 0;
var itemKeyword = "CostOfApple";
anArray[itemIndex] = "Fruit";
anArray[itemKeyword] = 0.75;
document.write( anArray[itemIndex] + "<br>" );
```

```

document.write( anArray[itemKeyword] );
</script>
</body>
</html>

```

让我们把已讨论过的数组和Math对象的知识应用到一个示例中。编写一个简单的脚本，它随机地选取一个广告横幅以显示到网页的顶部。

使用一个Array对象来保存一些图片的源文件名：

```

var bannerImages = new Array();
bannerImages[0] = "Banner1.jpg";
bannerImages[1] = "Banner2.jpg";
bannerImages[2] = "Banner3.jpg";
bannerImages[3] = "Banner4.jpg";
bannerImages[4] = "Banner5.jpg";
bannerImages[5] = "Banner6.jpg";
bannerImages[6] = "Banner7.jpg";

```

接着我们需要6个具有对应名字的图片，放在和HTML网页相同的文件夹里。可以使用你自己的图片或者从网站<http://www.beginningjavascript.com>上下载我提供的。

接下来初始化一个新的变量randomImageIndex，使用它来产生一个随机数。在这里使用与前面曾经用过的产生一个随机掷骰子数相同的方法，但是不用在结果上加1，因为我们需要产生一个0~6之间的随机数：

```
var randomImageIndex = Math.round( Math.random() * 6 );
```

然后使用document.write()把随机选择的图片写到网页中。下面是它完整的代码：

```

<html>
<body>
<script type="text/javascript">
var bannerImages = new Array();
bannerImages[0] = "Banner1.jpg";
bannerImages[1] = "Banner2.jpg";
bannerImages[2] = "Banner3.jpg";
bannerImages[3] = "Banner4.jpg";
bannerImages[4] = "Banner5.jpg";
bannerImages[5] = "Banner6.jpg";
bannerImages[6] = "Banner7.jpg";
var randomImageIndex = Math.round( Math.random() * 6 );
document.write( "<img alt=\"\" src=\"" + bannerImages[randomImageIndex] + "\">" );
</script>
</body>
</html>

```

上面就是与之相关的所有东西了。使广告横幅不断地变化比每次访问网页的时候都显示同样的横幅广告更加容易引起访问者的注意，当然，它还会给访问者一个印象，即这个站点在经常地更新。

数组对象的方法和属性

数组对象最常用的一个属性就是length属性，它返回比数组中最后一个数组项索引大1的索引。举个例子，如果你正在处理一个数组，它拥有索引为0、1、2、3的元素，length属性就是4。如果想添加另外的元素，知道这一点是非常有用的。

Array对象提供了许多方法用来操作数组，包括从数组中截取一部分元素的方法，或者把两个数组连接到一起。下面我们来看连接、截取和排序的方法。

- 截取数组的一段

slice()方法对于一个Array对象就如同substring()方法对于一个String对象。只要简单地告诉这个方法你想要截取哪些元素。这是非常有用的，譬如说当你想要一个使用URL传递的信息片段的时候。

slice()方法拥有2个参数：片段中第一个元素的索引，它包含在这个片段中；最后一个元素的索引，片段中不会包含它。为了访问一个总共包含5个值的数组中的第2、第3和第4个值，我们使用索引1和4：

```
<html>
<body>
<script type="text/javascript">
    // Create and initialize the array
    var fullArray = new Array( "One", "Two", "Three", "Four", "Five" );
    // Slice from element 1 to element 4 and store
    // in new variable sliceOfArray
    var sliceOfArray = fullArray.slice( 1, 4 );
    // Write out new ( zero-based ) array of 3 elements
    document.write( sliceOfArray[0] + "<br>" );
    document.write( sliceOfArray[1] + "<br>" );
    document.write( sliceOfArray[2] + "<br>" );
</script>
</body>
</html>
```

这个新数组存储数字是从0开始的，因此片段的索引0、1和2会给我们下面的结果：

```
Two
Three
Four
```

最初的数组并未受影响，但是如果需要，可以通过设置它为slice()方法返回的结果，重写变量中的这个数组对象：

```
fullArray = fullArray.slice( 1, 4 );
```

- 连接2个数组

Array对象的concat()方法允许连接数组。可以使用这个方法把2个或多个数组连接到一起，每个新数组在前面数组结束的地方开始。这里，我们连接了3个数组：arrayOne、arrayTwo和

arrayThree:

```
<html>
<body>
<script type="text/javascript">
var arrayOne = new Array( "One", "Two", "Three", "Four", "Five" );
var arrayTwo = new Array( "ABC", "DEF", "GHI" );
var arrayThree = new Array( "John", "Paul", "George", "Ringo" );
var joinedArray = arrayOne.concat( arrayTwo, arrayThree );
document.write( "joinedArray has " + joinedArray.length + " elements<br>" );
document.write( joinedArray[0] + "<br>" )
document.write( joinedArray[11] + "<br>" )
</script>
</body>
</html>
```

这个新数组joinedArray拥有12个数据项。这个数组中的数据项和它们在以前的数组中是一样的；它们只是被简单地连接到一起。原来的数组仍然没有改变。

- 数组和字符串之间的相互转换

当你想循环元素或选取某些元素的时候，把数据放到数组里是非常便利的。可是，当你需要把数据传送到其他地方的时候，把数据转换成字符串可能是个非常好的主意。可以通过循环数组并把每个元素的值加到一个字符串里来实现。可是没有必要那么去做，因为Array对象有一个叫join()的方法可以用来完成这个功能。这个方法需要一个字符串作为参数。这个字符串会被添加到每个元素的中间。

```
<script type="text/javascript">
var arrayThree = new Array( "John", "Paul", "George", "Ringo" );
var lineUp=arrayThree.join( ', ' );
alert( lineUp );
</script>
```

作为结果的字符串lineUp的值是"John, Paul, George, Ringo"。与join()相反的操作是split()，这个方法可以把字符串转换为数组。

```
<script type="text/javascript">
var lineUp="John, Paul, George, Ringo";
var members=lineUp.split( ', ' );
alert( members.length );
</script>
```

- 数组排序

sort()方法允许把数组中的数据项按字母或者数字顺序进行排序：

```

<html>
  <body>
    <script type="text/javascript">
      var arrayToSort = new Array( "Cabbage", "Lemon", ↪
      "Apple", "Pear", "Banana" );
      var sortedArray = arrayToSort.sort( );
      document.write( sortedArray[0] + "<br>" );
      document.write( sortedArray[1] + "<br>" );
      document.write( sortedArray[2] + "<br>" );
      document.write( sortedArray[3] + "<br>" );
      document.write( sortedArray[4] + "<br>" );
    </script>
  </body>
</html>

```

数据项会按下面的方式排序：

```

Apple
Banana
Cabbage
Lemon
Pear

```

可是，如果把其中的一个字母小写，例如Apple中的A字母，那么你会得到一个完全不同的结果。这个排序是按照严格的数学顺序——ASCII码中的字符编号进行的，而不是像人那样会按照单词进行排序。

如果想改变已经排序好的元素的显示顺序，可以使用reverse()方法来把字母表中的最后的字母作为第一个元素来显示：

```

<script type="text/javascript">
  var arrayToSort = new Array( "Cabbage", "Lemon", ↪
  "Apple", "Pear", "Banana" );
  var sortedArray = arrayToSort.sort( );
  var reverseArray = sortedArray.reverse( );
  document.write( reverseArray[0] + "<br />" );
  document.write( reverseArray[1] + "<br />" );
  document.write( reverseArray[2] + "<br />" );
  document.write( reverseArray[3] + "<br />" );
  document.write( reverseArray[4] + "<br />" );
</script>

```

结果序列现在是按原序列的反序进行排列的：

```

Pear
Lemon
Cabbage
Banana
Apple

```

2.3 在 JavaScript 中进行判定

判定 (decision making) 赋予了程序智能。不使用它你就不能写出一个好的程序，不管是创建游戏、校验密码、根据用户前面做出的选择给予用户一组选择，还有一些其他的情况。

判定是以条件语句为基础的，条件语句是一种值或真或假的简单语句。这就是基本数据类型中的布尔 (Boolean) 数据类型该应用的地方了。循环是进行判定的另一个重要工具，例如，可以从头到尾地循环用户的输入或循环一个数组，并作出对应的决策。

2.3.1 逻辑运算符和比较运算符

有2组主要的运算符，我们会学到下面的内容。

- **数据比较运算符 (data comparision operator)**: 比较操作数并返回布尔值。
- **逻辑运算符 (logical operator)**: 对多于一种条件的情况进行测试。

我们先看一下比较运算符。

1. 数据比较

表2-3列出了一些最常用的比较运算符。

表2-3 JavaScript中的比较运算符

运算符	描述	例子
<code>==</code>	检查左边和右边的操作数是否相等	123 == 234 返回假值 123 == 123 返回真值
<code>!=</code>	检查左边与右边的操作数是否不相等	123 != 123 返回假值 123 != 234 返回真值
<code>></code>	检查左边的操作数是否大于右边的操作数	123 > 234 返回假值 234 > 123 返回真值
<code>>=</code>	检查左边的操作数是否大于或等于右边的操作数	123 >= 234 返回假值 123 >= 123 返回真值
<code><</code>	检查左边的操作数是否小于右边的操作数	234 < 123 返回假值 123 < 234 返回真值
<code><=</code>	检查左边的操作数是否小于或等于右边的操作数	234 <= 123 返回假值 234 <= 234 返回真值

警告 小心这个`==`相等运算符：很容易在脚本中由于被误用为赋值运算符“`=`”而产生许多错误。

这些运算符常与字符串类型数据和数字类型一起使用，并且是区分大小写的：

```
<html>
<body>
<script type="text/javascript">
document.write( "Apple" == "Apple" )
document.write( "<br />" );
document.write( "Apple" < "Banana" )
```

```

document.write( "<br />" );
document.write( "apple" < "Banana" )
</script>
</body>
</html>

```

下面是得到的结果：

```

true
true
false

```

当计算一个字符串比较的表达式时，JavaScript解释器会依次比较两个字符串中每个字符的ASCII码——每个字符串的第一个字符，然后是第二个字符，依此类推。大写的A在ASCII中表示65，B表示66，C表示67，依此类推。计算这个表达式"Apple" < "Banana"，JavaScript解释器使用每个字符串中的第一个字符的ASCII码来进行比较：65 < 66，因此A排在前面，这个比较是真值。当检验这个表达式"apple" < "Banana"的时候，JavaScript解释器做了同样的事情：然而，小写字母a的ASCII码是97，因此这个表达式"a" < "B"转变为97 < 66，它的值是假。可以使用<、<=、>、>=运算符来进行字母顺序的比较。如果需要确保所有的字母都是同样的大写或小写，可以使用String对象的toUpperCase()和toLowerCase()方法。比较运算符和数字运算符是一样的，都可以使用变量。如果想按字母顺序比较apple和Banana，可以这样做：

```

<html>
<body>
<script type="text/javascript">
var string1 = "apple";
var string2 = "Banana";
string1 = string1.toLowerCase();
string2 = string2.toLowerCase();
document.write( string1 < string2 )
</script>
</body>
</html>

```

可是在使用等号运算符比较String对象的时候，有一些细节需要注意。试一下这个：

```

<html>
<body>
<script type="text/javascript">
var string1 = new String( "Apple" );
var string2 = new String( "Apple" );
document.write( string1 == string2 )
</script>
</body>
</html>

```

你会得到一个返回false值。事实上，我们在上面比较的是两个String对象，而不是两个基本类型的字符串包含的字符。返回的false结果表明，即使两个字符串对象持有相同的字母，它们也不是同一个对象。

如果你确实需要比较两个对象持有的字符串，可以使用`valueOf()`方法来对数据值进行比较：

```
<html>
<body>
<script type="text/javascript">
    var string1 = new String( "Apple" );
    var string2 = new String( "Apple" );
    document.write( string1.valueOf() == string2.valueOf() );
</script>
</body>
</html>
```

2. 逻辑运算符

有时候需要把多个比较放到一个条件组中。你也许想检查用户输入的信息是否有意义，或者根据用户前面的回答，限制他们可以选择的数据项。可以使用表2-4中的逻辑运算符来完成这样的功能。

表2-4 JavaScript中的逻辑运算符

符 号	运 算 符	描 述	例 子
<code>&&</code>	与	两个条件都必须为真	<code>123 == 234 && 123 < 20 (false)</code> <code>123 == 234 && 123 == 123 (false)</code> <code>123 == 123 && 234 < 900 (true)</code>
<code> </code>	或	其中一个或两个必须为真	<code>123 == 234 123 < 20 (false)</code> <code>123 == 234 123 == 123 (true)</code> <code>123 == 123 234 < 900 (true)</code>
<code>!</code>	非	原逻辑取反	<code>!(123 == 234) (true)</code> <code>!(123 == 123) (false)</code>

一旦求出了数据的值，我们需要能够根据这个输出结果进行判定选择。这正是条件语句和循环语句可以发挥作用的地方。你会发现本章中介绍的运算符会经常用到条件语句或循环语句中。

2.3.2 条件语句

`if...else`结构用来测试条件，格式如下：

```
if ( condition ) {
// Execute code in here if condition is true
} else {
// Execute code in here if condition is false
}
// After if/else code execution resumes here
```

如果这个条件测试为真，紧跟`if`后面的大括号中的代码就会被执行，但如果它不为真则不会执行。也可以使用`else`语句，创建在`if`条件不满足的情况下应该执行的代码块。

我们来改进一下在本章前面建立的货币转换程序，创建一个循环来处理用户的非数字输入：

```
<html>
<body>
<script type="text/javascript">
    var euroToDollarRate = 0.872;
```

```

    // Try to convert the input into a number
    var eurosToConvert = Number( prompt( "How many Euros do you wish to convert", "" ) );
    // If the user hasn't entered a number, then NaN will be returned
    if ( isNaN( eurosToConvert ) ) {
        // Ask the user to enter a value in numerals
        document.write( "Please enter the number in numerals" );
        // If NaN is not returned, then we can use the input
    } else {
        // and do the conversion as before
        var dollars = eurosToConvert * euroToDollarRate;
        document.write( eurosToConvert + " euros is " + dollars + " dollars" );
    }
</script>
</body>
</html>

```

这个if语句使用了isNaN()函数，如果变量eurosToConvert中的值不是一个数它会返回true。

注解 记住尽可能地使错误信息友好并且有用。良好的错误信息可以明晰地告知用户他们该做什么，这样可以使他们更有耐心地使用应用程序。

使用逻辑运算符和嵌套的if语句，可以创建更复杂的条件：

```

<html>
<body>
<script type="text/javascript">
    // Ask the user for a number and try to convert the
    // input into a number
    var userNumber = Number( prompt( "Enter a number between 1 and 10", "" ) );
    // If the value of userNumber is NaN, ask the user
    // to try again
    if ( isNaN( userNumber ) ) {
        document.write( "Please ensure a valid number is entered" );
    } else {
        if ( userNumber > 10 || userNumber < 1 ) {
            document.write( "The number you entered is not between 1 and 10" );
        } else {
            // Otherwise the number is between 1 and 10 so
            // write to the page
            document.write( "The number you entered was " + userNumber );
        }
    }
</script>
</body>
</html>

```

```

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

我们知道这个数只要是一个数字并且小于10就可以。

注解 注意这段代码的布局。我们对if与else语句以及代码块使用了缩进，所以非常容易阅读和理解代码块是从哪里开始和结束的。使你的代码尽可能简单明了是非常重要的。

试着阅读一下没有使用缩进和空格的这段代码：

```

<html>
<body>
<script type="text/javascript">
// Ask for a number using the prompt() function and try to make it a number
var userNumber = Number(prompt("Enter a number between 1 and 10", ""));
// If the value of userNumber is NaN, ask the user to try again
if (isNaN(userNumber)){
document.write("Please ensure a valid number is entered");
}
// If the value is a number but over 10, ask the user to try again
else {
if (userNumber > 10 || userNumber < 1) {
document.write("The number you entered is not between 1 and 10");
}
// Otherwise the number is between 1 and 10, so write to the screen
else{
document.write("The number you entered was " + userNumber);
}
}
</script>
</body>
</html>
```

它不是不能阅读，但是即使在这个简短的脚本中，也很难分辨出哪个代码块属于if和else语句。在更长的代码中，不一致或不合理的缩进使代码很难读懂，它反过来会留给你更多的需要修正的漏洞且使你的工作不必要地更困难。

也可以在另一个以if语句开始的else语句的地方使用else if语句，像下面这样：

```

<html>
<body>
<script type="text/javascript">
var userNumber = Number( prompt( "Enter a number between"-
1 and 10", "" ) );
if ( isNaN( userNumber ) ){
document.write( "Please ensure a valid number is"-
entered" );
```

```

} else if ( userNumber > 10 || userNumber < 1 ) {
    document.write( "The number you entered is not" +
    "between 1 and 10" );
} else {
    document.write( "The number you entered was " + ↪
userNumber );
}
</script>
</body>
</html>

```

这段代码和前面的代码段完成同样的事情，但是使用了一个else if语句代替一个嵌套的if语句，并且代码少了两行。

跳出一个分支或循环

在继续之前需要注意一件事情：可以使用break语句来中断一个条件语句或循环。它简单地中断了代码块的运行并且使进程跳到下一语句。我们会在下一个例子中用到它。

只要你喜欢，可以使用许多的if、else和else if语句，尽管过多使用会使代码十分复杂。如果在一个代码块中检查一个值有许多可能的条件，那么我们接下来会看到的switch语句将非常有用。

2.3.3 测试多个值：switch语句

switch语句允许根据一个变量或表达式的值在多个代码段之间进行“转换”。下面是一个switch语句的概要格式：

```

switch( expression ) {
    case someValue:
        // Code to execute if expression == someValue;
        break; // End execution
    case someOtherValue:
        // Code to execute if expression == someOtherValue;
        break; // End execution
    case yesAnotherValue:
        // Code to execute if expression == yetAnotherValue;
        break; // End execution
    default:
        // Code to execute if no values matched
}

```

JavaScript计算switch(expression)的值，然后把它与每个case进行比较。只要发现了一个匹配的项，代码就会在那个位置开始执行，并继续遍历执行所有的case语句直到发现了一个break语句。如果没有一个case语句匹配，最好包含一个默认的情况。以用于尽快找出错误发生的地方，例如，我们期望一种情况被满足，但是一个程序bug阻止了它发生。

case的值可以是任何数据类型，例如数字或字符串。可以根据自己的需要选择一个或多个case语句。下面来看一个简单的示例：

```

<html>
  <body>
    <script type="text/javascript">
      // Store user entered number between 1 and 4 in userNumber
      var userNumber = Number( prompt( "Enter a number between 1 and 4", "" ) );
      switch( userNumber ) {
        // If userNumber is 1, write out and carry on
        // executing after case statement
        case 1:
          document.write( "Number 1" );
          break;
        case 2:
          document.write( "Number 2" );
          break;
        case 3:
          document.write( "Number 3" );
          break;
        case 4:
          document.write( "Number 4" );
          break;
        default:
          document.write( "Please enter a numeric value between 1 and 4." );
          break;
      }
      // Code continues executing here
    </script>
  </body>
</html>

```

运行它。你只会得到自己刚输入的数字或者这个句子“Please enter a numeric value between 1 and 4.”。

这个示例也阐明了break语句的重要性。如果我们在每个case语句后不包括break，那么这个代码块中的代码会继续执行直到最后switch结束。试一下去掉所有的break然后输入2。匹配项后的所有代码都会执行，因此给你这样一个输出结果：

```
Number 2Number 3Number 4Please enter a numeric value between 1 and 4
```

可以在switch语句中使用任何有效的表达式，例如一个计算：

```
switch( userNumber * 100 + someOtherVariable )
```

也可以在case语句间使用一个或多个语句。

2.3.4 重复事件：循环

在本节中，我们会学习只要一组条件为真如何重复一个代码块。举个例子，我们可能需要循环一遍HTML表单中的每个输入或者循环遍历一个数组中的每个元素。

1. 重复一个集合多次：for循环

for循环用来循环一个代码块许多次，格式如下：

```
for( initial-condition; loop-condition; alter-condition ) {
    //
    // Code to be repeatedly executed
    //
}
// After loop completes, execution of code continues here
```

和条件语句类似，关键字for后紧跟着括号。这次，括号中包含由分号分割的三部分。

第一部分初始化一个变量，它用来当作保存循环次数的计数器。第二部分测试条件。只要这个条件为真，循环就会继续运行。最后一个部分在每遍循环之后增加或减少这个在第一部分中创建的计数（事实上你会发现这个之后在编程中很重要）。

例如，看一下这个循环，只要loopCounter小于等于10，它就会继续运行：

```
for( loopCounter = 1; loopCounter <= 10; loopCounter++ )
```

只要循环条件为真，循环就会继续执行——正如只要loopCounter小于或等于10。一旦它达到了11，这个循环就会终止并且代码的执行会从loop的结束括号后面的下一个语句开始。

让我们来看一个使用for循环来遍历一个数组的示例。我们会使用一个for循环来遍历一个叫做theBeatles的数组，它使用一个叫loopCounter的变量。当loopCounter的值小于数组的长度时，循环就会继续执行：

```
<html>
<body>
<script type="text/javascript">
    var theBeatles = new Array( "John", "Paul", "George", ▶
        "Ringo" );
    for ( var loopCounter = 0; loopCounter < ▶
        theBeatles.length; loopCounter++ ) {
        document.write( theBeatles[loopCounter] + "<br>" );
    }
</script>
</body>
</html>
```

这个例子可以运行因为我们使用了一个从零开始计数的数组，在它里面数据项按顺序被添加到索引上。如果使用了按如下方式来保存数据项的关键字，那么这个循环就不会执行了：

```
theBeatles["Drummer"] = "Ringo";
```

在前面讲述数组的时候，我强调了Array对象有一个知道数组长度的属性（有多少个元素）。当循环遍历数组的时候，如前面的例子，我们使用了数组名后跟一个点和length作为条件。它阻止了循环数到数组的长度以外去，这样会造成一个“越界”的错误。

JavaScript也支持for..in循环（尽管IE从IE 5才开始支持它，但它从NN 2就开始到处使用了）。

不再使用计数器，`for...in`循环使用访问数组的变量来遍历数组中的每个元素：

```
<html>
<body>
<script type="text/javascript">
    // Initialize theBeatles array and store in a variable
    var theBeatles = new Array( );
    // Set the values using keys rather than numbers
    theBeatles["Drummer"] = "Ringo";
    theBeatles["SingerRhythmGuitar"] = "John";
    theBeatles["SingerBassGuitar"] = "Paul";
    theBeatles["SingerLeadGuitar"] = "George";
    var indexKey;
    // Write out each indexKey and the value for that
    // indexKey from the array
    for ( indexKey in theBeatles ) {
        document.write( "indexKey is " + indexKey + "<br>" );
        document.write( "item value is " + theBeatles[indexKey] + "<br><br>" );
    }
</script>
</body>
</html>
```

在循环的每一遍中，关键字`indexKey`会与数组中按照同样顺序使用关键字取出的值一起输出：

```
indexKey is Drummer
item value is Ringo

indexKey is SingerRhythmGuitar
item value is John

indexKey is SingerBassGuitar
item value is Paul

indexKey is SingerLeadGuitar
item value is George
```

2. 依据一个条件判定重复执行动作脚本：`while`循环

目前我们所接触的循环都是从脚本自身的内部获取停止循环的指令。可能存在多次这样的情况，你想让用户决定循环应该何时停止，或者当一个用户决定的条件满足的时候停止循环。循环`while`和`do...while`正是用来解决这类问题的。

`while`循环最简单的形式如下所示：

```
while ( some condition true ) {
    // Loop code
}
```

括号里的条件可以是你在一个`if`语句当中可能会使用到的任何内容。按下面的方式使用代码，它允许用户输入数字并且在输入数字99后停止输入的程序。

```

<html>
  <body>
    <script type="text/javascript">
      var userNumbers = new Array( );
      var userInput = 0;
      var arrayIndex = 0;
      var message = '';
      var total = 0;
      // Loop for as long as the user doesn't input 99
      while ( userInput != 99 ) {
        userInput = prompt( "Enter a number, or 99 to exit", "99" );
        userNumbers[arrayIndex] = userInput;
        arrayIndex++;
      }
      message += 'You entered the following:\n';
      for ( var i = 0; i < arrayIndex-1; i++ ) {
        message += userNumbers[i] + '\n';
        total += Number( userNumbers[i] );
      }
      message += 'Total: ' + total + '\n';
      alert( message );
    </script>
  </body>
</html>

```

在这里while循环的条件是userInput不等于99，所以只要这个条件为真，循环就会继续。当用户输入99并且条件被检验的时候，它会计算得到假值并且循环会终止。注意用户输入99后，循环不会立即终止，只有当条件在循环的另一遍开始处被再次检验时，循环才会终止。

在while和do...while循环之间，有一个比较小但非常明显不同之处：while循环在代码执行前检验条件，并且只有在条件为真的情况下才会执行代码块。而do...while在检验条件之前执行代码块，如果条件为真会执行另一遍循环。简而言之，do...while循环在条件检查前循环体中的代码至少会执行一次。可以如下所示，使用一个do...while循环来写一个前面的例子：

```

<html>
  <body>
    <script type="text/javascript">
      var userNumbers = new Array( );
      var message = '';
      var total = 0;
      // Declare the userInput but don't initialize it.
      var userInput;
      var arrayIndex = 0;
      do {
        userInput = prompt( "Enter a number, or 99 to exit", "99" );
        userNumbers[arrayIndex] = userInput;
        arrayIndex++;
      } while ( userInput != 99 )
    </script>
  </body>
</html>

```

```

message += 'You entered the following:\n';
for ( var i = 0; i < arrayIndex-1; i++ ) {
    message += userNumbers[i] + '\n';
    total += Number( userNumbers[i] );
}
message += 'Total: ' + total + '\n';
alert( message );
</script>
</body>
</html>

```

不需要初始化userInput，因为循环中的代码在第一次检验它之前已经为它设置了一个值。

3. 继续循环

你已经看到了，break语句用于只要某个特定的事件发生就跳出任一类型的循环。这个continue关键字的工作原理和break类似，可以用来停止循环的执行。可是，continue语句不是直接地跳出循环，而是在下一次循环遍历中使代码重新开始执行。

首先改变一下前面的例子，使用break来达到这样的一种效果，如果用户没有输入一个数字而是输入了其他的，那么这个输入的值不会被记录并且循环会终止。

```

<html>
<body>
<script type="text/javascript">
var userNumbers = new Array( );
var userInput;
var arrayIndex = 0;
do {
    userInput = Number( prompt( "Enter a number, or 99 to exit", "99" ) );
    // Check that user input is a valid number,
    // and if not, break with error msg
    if ( isNaN( userInput ) ) {
        document.write( "Invalid data entered: please enter a number between 0 and 99 in numerals" );
        break;
    }
    // If break has been activated, code will continue from here
    userNumbers[arrayIndex] = userInput;
    arrayIndex++;
} while ( userInput != 99 )
// Next statement after loop
</script>
</body>
</html>

```

现在再把它改变一下，使用continue语句，可以不让它跳出循环，而是忽略用户的输入并保持循环。

```

<html>
<body>

```

```

<script type="text/javascript">
    var userNumbers = new Array( );
    var userInput;
    var arrayIndex = 0;
    do {
        userInput = prompt( "Enter a number, or 99 to exit", "99" );
        if ( isNaN( userInput ) ) {
            document.write( "Invalid data entered: please enter a number between 0 and 99 in numerals" );
            continue;
        }
        userNumbers[arrayIndex] = userInput;
        arrayIndex++;
    } while ( userInput != 99 )
    // Next statement after loop
</script>
</body>
</html>

```

break语句已被continue替换了，因此在循环中的代码不会执行了，但是while语句中的条件会再次执行。如果条件为真，循环的另一个遍历就会被触发执行；否则这个循环就结束了。

使用下面的经验规则来决定使用哪一种循环结构：

- 如果想重复一个动作一定的次数，那么使用for循环。
- 当你想一个动作被重复执行直到满足某个条件的时候，使用while循环。
- 如果想保证这个动作至少被执行一次，那么使用do...while循环。

2.4 小结

本章涉及了许多方面，实际上，我们讨论了JavaScript语言的大部分要点。

你学习到了JavaScript如何处理数据，并明白了许多数据类型：字符串、数字、布尔值以及对象，还有一些特殊的类型NaN、null和undefined。你看到了JavaScript支持多种运算符，它们可以对数据进行操作，例如数学计算或把字符串拼接到一起。

我们接着学习了JavaScript允许如何利用变量存储值。变量可能在页面中持续其生命周期，而用户通过var关键字在自己创建的函数中声明的局部变量则持续其在该函数中的生命周期。我们也学会了如何把一种数据类型转换为另一种。

接下来我们接触了3种JavaScript内置的对象：String、Date和Math对象。这些对象提供了许多有用的功能用来操纵字符串、日期和数字。本章还说明了Array对象，它允许多个数据项存储在单个变量中。

本章最后讲述了判定，它提供了编程语言的逻辑和智能。我们使用了if和switch语句来进行判定，使用条件来检验数据的有效性并遵照这些结论行事。循环也使用了条件，并且它允许按特定次数重复一个代码块或者直到某种条件为真。

从DHTML到DOM编程

在本章中，你将学习DHTML是什么，现今它为什么被看作是一种不好的方式，以及应该使用什么样的现代技术和思想来替代它。你还可以学到函数是什么，以及如何使用函数。同时你可以了解变量、函数作用域和一些最新的最佳实践，它将指导你的脚本同其他脚本一起良好地运行。

如果对JavaScript有兴趣，并在网上搜索过一些脚本，那么你一定遇到过DHTML这个术语。在20世纪90年代末和新千年的初期，DHTML是IT和网络产业中最时髦的词语之一。你可能阅读过许多关于如何在过时的浏览器上产生一定视觉特效的教程，相比之下，却很少见到有资料解释这些特效为什么会产生以及这些脚本究竟做了什么。这些正是DHTML所涉及的方面。

注解 DHTML（动态HTML）从来不是一项真正的技术或者万维网联盟（W3C）的标准，仅仅是由市场及广告代理机构发明的一个名词而已。

DHTML就是与CSS（Cascading Style Sheet，层叠样式表）和Web文档（HTML格式）进行交互生成动态页面的JavaScript。网页的一部分可以飘来飘去、放大或缩小，对于页面上的每一个元素，在用户使用鼠标经过它的时候它都会有所反应，而且我们每周都会创造一种网页导航的新方式。

虽然这一切都非常有趣，且具有技术挑战性，但这并未给访问者帮多大的忙。这种“令人叫绝”的影响很快就失效了，特别是当浏览器不能再支持它的时候，它们也就走到了尽头。

DHTML听起来非常棒，它也确实带来了一种优越感：JavaScript开发人员在代码可维护性方面并没有遇到太多的麻烦，因为在任何情况下，任何并不理解如何使东西动起来的人都不会修改他们的代码。对于自由开发人员这也意味着稳定的收入，因为每次更改都要由他们来完成。

当大量的资金停止涌入的时候，大批这样的代码被放在网上供其他开发人员使用，大到JavaScript DHTML库，小到一些脚本集合网站上的小脚本。没有人为更新这些代码而操心，因此这些资源并不能作为现在专业环境中的可行选择。

一般的DHTML脚本有以下几个问题。

- **JavaScript的依赖性和缺少优雅的降级：**如果用户关闭了JavaScript（自己的选择或是出于公司安全设置的考虑）将无法使用其功能，当激活页面上的元素时，这些元素毫无反应，甚至网页根本无法进行导航。
- **浏览器和版本的依赖性：**通常用来测试脚本是否可执行的方法，是在导航器对象中读出浏览器的名字。大批脚本是在Netscape 4和IE 5比较流行时创建的，因此它们无法支持较新的浏览器——这些脚本根本没有考虑浏览器的新版本，只是测试Netscape 4或IE 5。
- **代码分叉：**由于不同的浏览器支持不同的DOM，一些代码需要被复制并且浏览器之间的差别是不可避免的，这就使得编写模块化的代码变得很困难。
- **维护代价大：**由于大多数脚本语言的界面外观是在脚本语言内部写定的，任何修改都意味着你至少必须懂得基本的JavaScript。由于JavaScript是为不同浏览器开发的，对于不同的浏览器必须将不同的修改写入到浏览器对应的JavaScript中。
- **标记依赖性：**与通过DOM来生成或访问HTML不同，许多脚本直接通过document.write指令写出内容并添加到每个文档的主体中，而不是将一切放入独立的缓存文档中。

上面所说的一切与我们现在必须满足的需求形成了鲜明的对比：

- **代码必须是易维护的，在不同的项目之间拥有复用的可能性。**
- **法律要求，比如英国的数字歧视法、美国的508条款都非常强烈地反对，有时候甚至是禁止Web产品依赖于脚本编程。**
- **更多的浏览器、手机这样的设备上的用户代理或帮助残疾用户参与到网络当中的辅助技术，使得脚本不可能依赖于浏览器识别。**
- **新的市场战略产生了这样一个需求：更快地、低代价地更改网站或Web应用程序的界面外观——甚至可能由内容管理系统完成。**

如果我们还是想要使用JavaScript并将其出售给客户，且能够迎合不断变化的市场所带来的挑战，我们就非常需要重新考虑一下使用这种Web技术的方式。

第一步是使JavaScript成为“有了更好”的项而不是一种需求任务。在JavaScript不可用的时候，不应该有太多的空白网页或是什么都不做的链接。术语分离式JavaScript（unobtrusive JavaScript）是由Stuart Langridge (<http://www.kryogenix.org>) 创造的。如果在Google中搜索一下，就可以找到我以前编写的关于这一主题的自学教程。

分离式JavaScript是指这样一种脚本，它不强迫用户使用它或遵循它的使用方式，它会检查自身是否可以被应用，如可以则发挥效用。分离式JavaScript就像一个舞台管理者，做她在后台所擅长的并且对整个舞台有益的事，而不是占据整个舞台充当女主角——每当事情出错或是不合其意时，就对着管弦乐团和她的同事大喊大叫。

稍后又出现了DOM脚本编程(DOM Scripting)这一术语。在2004年伦敦举行的@media大会上，WaSP DOM脚本编程任务组成立。这个任务组由许多的程序员、博客作者、设计人员组成，他们期望能使JavaScript更加成熟、更加以用户为中心——可以通过<http://domscripting.webstandards.org>了解进一步的信息。

因为在传统的Web开发方法中，JavaScript并没有一个固定的位置（以前都认为JavaScript只是

可以从网上下载，然后进行修改的东西，或者在需要的时候由编辑工具来生成的东西），因此一个叫做“行为层”的术语在各种Web出版物中开始出现。

3.1 作为“行为层”的JavaScript

Web开发可以被认为是由几个不同的层组成的，如图3-1所示。

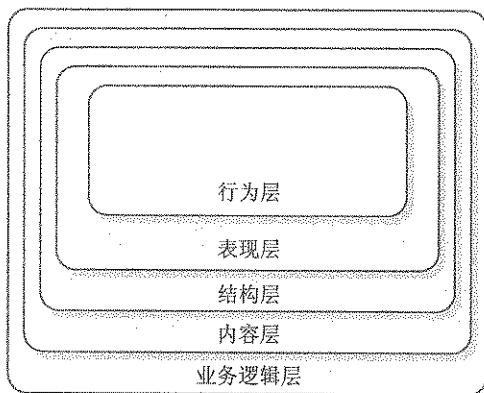


图3-1 Web开发的层次结构

- 行为层：在客户端被执行，定义了不同的元素在与客户端交互时的行为（JavaScript或用于Flash网站的ActionScript）。
- 表现层：在客户端显示的网页的外观（CSS）。
- 结构层：由用户代理转化或显示，用来定义某文本或媒体含义的标签（XHTML）。
- 内容层：存储在服务器端，由网站所需要的所有文本、图片、多媒体内容组成（XML、数据库和媒体文件）。
- 业务逻辑层或称后端层：在服务器端运行，决定着如何处理传输过来的数据以及将什么返回给用户。

注意，这只是简单定义了可用的层，并没有提及各层之间如何交互。比如，需要将内容转化为结构的部分（如XSLT），还有在业务逻辑层需要与其上的4层进行连接。

如果能实现这些层之间的彼此分离，但可以彼此通信，那么你已经成功地开发了一个可访问且易维护的网站。在实际的开发和业务中，这种情况很难达到。然而，越是接近这个目标，以后所要面对的恼人的问题也就越少。CSS很强大，它允许在一个将被用户代理高速缓存的文件中定义不计其数的网页界面外观。JavaScript也通过script标签的属性src和一个.js文件应用于不计其数的网页。

本书的前几章中，我们将JavaScript直接嵌入到HTML文档中（也许你会想起对于XHTML我们也是这样做的），从现在起我们不会再这样做了，相反我们将创建单独的JavaScript文件，并且将它们链接到文档的头部：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Demo</title>
  <style type="text/css">@import 'styles.css';</style>
  <script type="text/javascript" src="scripts.js"></script>
  <script type="text/javascript" src="morescripts.js"></script>
</head>
<body>
</body>
</html>

```

我们应该试着在文档当中不再嵌入任何的脚本块，主要是因为这样会混淆结构层和行为层，并导致因JavaScript出错而产生的用户代理停止显示页面的问题。这对维护也是一个噩梦，将所有的JavaScript添加到一个.js文件意味着我们可以在一个地方维护整个网站的脚本，而不是去遍历网站的所有文档。

注解 Firefox、Safari和Opera将.js文件视为文本，微软的IE却试图执行它。这意味着不要双击它，或是在微软的IE中去调试.js文件——在调试代码的时候这是一件非常烦的事。这样立即执行代码存在安全问题，这正是Windows XP2将所有本地文件中的JavaScript内容标记为安全问题的原因，如图3-2所示。不要被此愚弄——这是你的代码，除非你是一个不怀好意的编码者，否则这不是一个安全问题。

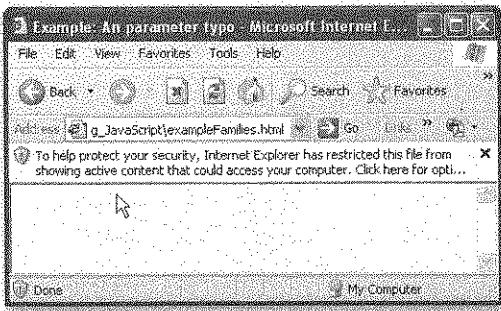


图3-2 在Windows XP2操作系统下微软的IE在执行本地的JavaScript代码时所显示的警告信息

将所有JavaScript放入单独的文件，这样开发一个JavaScript不可用时仍然可用的网站变得很简单；当网站的行为需要修改的时候只修改js文件也会更简单一些。

这就解决了DHTML的一个主要毛病，现在，HTML可以在没有JavaScript的情况下单独存在，而且不需要遍历许多文档去查明bug所在。DHTML的另一个毛病是其依赖于浏览器，现在我们可以使用对象检测来替换。

3.1.1 对象检测与浏览器依赖性的比较

确定所使用的是哪种浏览器的一种方法是测试navigator对象，通过这个对象的appName和appVersion属性可以揭示所使用浏览器的名称和版本。

例如，下面的脚本获取浏览器的名称和版本并将其写入到文档中：

```
<script type="text/javascript">
  document.write("You are running " + navigator.appName);
  document.write(" and its version is " + navigator.appVersion);
</script>
```

在我机器上的Macromedia HomeSite的预览模式下，这个脚本的报告如下（因为HomeSite使用微软IE引擎预览HTML）：

```
You are running Microsoft Internet Explorer
and its version is 4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; .NET CLR 1.1.4322)
```

如果我在相同的机器上使用Firefox 1.07运行相同的脚本，就会得到如下的结果：

```
You are running Netscape and its version is 5.0 (Windows; en-GB)
```

一些老一点的脚本使用这一信息来确定浏览器是否能支持它们的功能。

```
<script type="text/javascript">
if(browserName.indexOf('Internet Explorer')!=-1
  && browserVersion.indexOf('6')!=-1)
{
  document.write('<p>This is MSIE 6!</p>');
}
else
{
  document.write('<p>This isn\'t MSIE 6!</p>');
}
</script>
```

乍一看，这种方法非常聪明，但是像Firefox所显示的输出一样，这并不是一个确定所使用的浏览器类型的完全保险的办法。一些浏览器（如Opera）不会将自己暴露给脚本语言，但是微软的IE浏览器会。

提示 Opera默认的设置是告诉脚本语言它是微软的IE。这样做的原因是Opera公司不希望它的浏览器被那些专门为微软IE浏览器所开发的网站所阻塞。可悲的是，Opera公司只是不想失去客户或者向一些愤怒的邮件解释为什么他们的“糟糕”浏览器在访问XYZ网站时不好使。这也意味着在网站的浏览器统计表中Opera不会出现，并且对于那些只在乎访问量和访问者所使用的浏览器类型的网站，Opera仍然是不重要的（很多统计软件使用navigator对象）。如果你是Opera用户并且想关掉这项预先的设置，那么可以按F12然后选择“Identify as Opera”来替换“Identify as Internet Explorer”。

读取浏览器的名称和版本——通常称为浏览器嗅探（browser sniffing）是不可取的，不只是因为我们刚才所遭遇的矛盾，也因为这使得我们的脚本依赖于特定的浏览器，而不是支持这种脚本的任何用户代理。

解决这种问题的办法称为对象检测（object detection），它的基本想法是我们检查用户代理是否支持一定的对象，并使之成为关键的区分标准。在相当老的脚本中，比如较早的图片翻转脚本，你也许会见到下面的代码：

```
<script type="text/javascript">
// preloading images
if(document.images)
{
    // Images are supported
    var home=new Image();
    home.src='home.gif';
    var aboutus=new Image();
    aboutus.src='aboutus.gif';
}
</script>
```

在if的条件块里检查浏览器是否支持images对象，只有在支持的情况下才执行条件内的代码。很长一段时间里，这样的脚本代码是处理图片非常标准的方法。在较新一些的浏览器中，许多通过JavaScript产生效果的图片可以由CSS完成，实际上使这种脚本代码过时了。然而JavaScript可以以CSS做不到的方式操作图片，第6章中我们将回到这个话题。

每个浏览器都提供了显示并用来操作的文档，这种操作是通过文档对象模型（Document Object Model, DOM）实现的。老的浏览器支持它们自己的DOM，但存在一个标准的DOM，它由万维网联盟（W3C）定义，今天它已被绝大多数浏览器所支持。以前你也许遇到过这样的测试脚本：

```
<script type="text/javascript">
if(document.all)
{
    // MSIE
}
else if (document.layers)
{
    // Netscape Communicator
}
else if (document.getElementById)
{
    // Mozilla, Opera 6, Safari
}
</script>
```

DOM document.all是由微软发明的，并在IE 4和IE 5中被支持。而Netscape 4有它自己的DOM来支持document.layers。可以通过document.getElementById测试一下由W3C推荐的DOM，除非确实需要支持微软IE 5.5或Netscape 4之前的版本，否则这是今天你所要做的唯一一项测试。

有个问题是一些Opera的中间版本支持getElementById对象，但并不支持所有的W3C DOM。

确定是否这种情况可以测试createTextNode:

```
<script type="text/javascript">
  if (document.getElementById && document.createTextNode)
  {
    // Mozilla, Opera 6, Safari
  }
</script>
```

这种情况下，如果你直接将这段测试代码嵌入到脚本代码当中，那么你可以完全确定，只有当用户代理支持DOM时，你编写的代码才会被执行。也可以在两个条件都不满足时直接返回，通过这个简单的方法来节省代码缩进：

```
<script type="text/javascript">
  if (!document.getElementById || !document.createTextNode){return;}
  // Other code
</script>
```

同样的思想将来可应用于其他任何方法，因为你在检查的是W3C所定义的标准，所以极有可能用户代理会支持，这比支持其他任何厂商可能提出的标准的可能性要大得多。可以对不同用户代理提供不同级别的支持，但这将导致大量的代码重复和不够优化的代码。

与其迎合特定的用户代理，不如在应用你编写的功能之前测试一下用户代理的能力——这正是现代Web设计思想的一部分，人们称之为渐进增强。

3.1.2 渐进增强

渐进增强（progressive enhancement）是这样一种实践，它只面向那些能够看到并且使用的用户提供功能，从最低的公共特性开始，然后检测用户是否支持不断提升的特性。这样没有较高特性支持能力的用户仍然可以充分完美地使用网站。一个相似的生活实例是你早上穿衣服的过程：

- 早上你从赤身裸体开始，希望完全进入工作状态——至少和昨天一样，这对你来说没有什么特别的（为了使例子比较简单我们不考虑内衣）。
- 也许你有极好的身体，但在寒冷的天气里这是不够的，而且过于“袒诚”，对周围的其他人也不礼貌——你需要一些东西装饰一下。
- 如果有衣服可供选择，你一定挑选那些适合天气、心情和今天你所要见的人的衣服，以及是否搭配、干净、大小适合。
- 穿上这些衣服就可以面对这一天了。如果你需要，还可以装饰一下。但做这些的时候请一定要考虑到其他的人（在拥挤的火车上你抹太多的香水恐怕不是什么好事）。

在网页开发的时候，这相当于以下情形：

- 我们从一个有效的、语义正确的、包含所有内容的(X)HTML（包括具有alt属性的替换文本相关图片）及一个有意义的结构开始。
- 我们添加样式表来提高结构的外观、易读性、清晰性——甚至可以添加简单的滚动效果使其更具活力。
- 添加JavaScript。

- 通过window对象的onload事件处理函数，可以使JavaScript在文档被载入时启动。
- JavaScript检测当前的用户代理是否支持W3C DOM。
- 然后检测是否所有必要的元素都可用，并且应用预期的功能。

在将渐进增强思想应用于JavaScript之前，你需要学习在脚本语言中如何实现HTML和CSS之间的交互，本书将用两章来讲述这个问题即第4章和第5章。此刻我们应该意识到，前面学习的对象检测帮助我们实现了渐进增强——我们确保只有那些能够正确认识对象的浏览器才去访问它们。

3.2 JavaScript 和可访问性

Web可访问性（accessibility）是指这样一种实践：使网站可以被所有人使用，无论他有什么样的残疾。比如，视觉有损伤的用户可能使用一种叫作屏幕阅读器（screen reader）的特殊软件来为他们读取网页的内容，还有一些行动困难的用户可能用一些工具来控制键盘进而导航网页，因为他们不能使用鼠标。残疾用户在网络用户中所占比例大小，所以那些不允许残疾用户使用其网站的公司也许会损失大量的业务。在一些国家，法律（如美国的508条款）规定任何提供公共服务的网站必须是可访问的。

在JavaScript中哪些方面与此有关呢？以前JavaScript技术的可访问性很差，因为可能会弄乱文档流畅性，比如，屏幕阅读器可能无法将读到的内容正确地返回给使用者（特别是当基本内容由JavaScript生成的时候——也可能是屏幕阅读器根本没有发现它！），这就迫使用户使用鼠标导航他们的网站（比如在复杂的DHTML导航菜单情况下）。这方面的问题很深，这里只是给你一个感性认识。

提示 如果你想阅读更多的关于网页可访问性的知识，可以参考*Web Accessibility: Web Standards and Regulatory Compliance*，由Jim Thatcher等著（friends of ED，2006）。

JavaScript和可访问性是一个争议很大的主题。不满意的开发人员和可访问性的拥护者经常在邮件列表、论坛、聊天室里争论不休。每一方都有他们自己的非常好的论据。

那些长期忍受着糟糕的浏览器和市场经理，不切实际的假设（“我在我表兄的网站上看见过它，那么你也一定能够在我们这个跨国公司的门户使用它”）的开发人员可不希望看到，多年来苦心钻研、不断摸索都付之东流，JavaScript不能再使用了。

而可访问性的拥护者则指出，JavaScript可以被关闭，W3C制定的可访问性指导原则似乎根本不允许这样做（该指导原则里关于这方面非常含混），但许多脚本语言都假设用户拥有并且能够使用精度很高的鼠标。

两种意见都是对的，但他们只说对了一半，事实上没有必要从一个可访问的网站里将所有的JavaScript完全移除。

必须改进的是有太多假设的JavaScript，可访问的JavaScript必须做到以下几点：

- 在拥有和没有JavaScript的情况下，网页文档应该具有相同的内容——访问者不应该必须关闭或必须打开JavaScript（因为访问者是否能够打开JavaScript经常不由自己决定）。

- 如果存在只有当JavaScript可用的时候才有意义的内容或者HTML元素，那么这些内容或者元素必须是由JavaScript创建的。没有什么比一个链接什么都不做或者一个文本所解释的功能对你不可用更让人沮丧的了。
 - 所有的JavaScript功能应该是独立于输入设备的。比如，用户可以使用拖放界面，也可以通过单击或是按键的方法来激活一个页面元素。
 - 网页中非交互性元素（实际上除了链接和表单元素外都不应该是）不应该变成交互性的——除非你提供了备用方案。例如，对于能使其后面的文本折叠或者展开的大字标题，虽然用JavaScript你可以轻易地使它变成可单击的，但这也意味着只能使用键盘的客户将永远无法使用这一功能。如果你在大字标题中创建了一个链接并使其可单击，那么用户应该能够通过“tab”移到这个链接并按下“回车”来激活这个效果。
 - 脚本不应该自动地将用户重定向到其他页面或是在未与用户交互的情况下提交表单。这是为了避免不完整的表单提交——因为一些辅助技术在处理onchange事件上存在问题。此外，病毒和间谍软件可以通过JavaScript将用户转到其他页面，因此这已经被一些软件所禁止。
- 使网站具有JavaScript可访问性的要点大概就是这些。当然可访问性HTML文档的优点还包括，允许元素使用大字号放大、为色盲的人和辩色正常的人提供足够的对比和颜色。

3.3 良好的编码实践

现在，你对向前兼容的、可访问的脚本编程应该已经有了一定的认识，让我们继续了解JavaScript的最佳编程实践。

3.3.1 命名习惯

JavaScript是区分大小写的，这意味着一个叫moveOption的变量或函数与名为moveoption或Moveoption的是不同的。任何名称，无论是函数、对象、变量或数组，必须只包含字母、数字、美元符或是下划线，并且不能以数字开头。

```
<script type="text/javascript">
  // 合法的例子
  var dynamicFunctionalityId = 'dynamic';
  var parent_element2='mainnav';
  var _base=10;
  var error_Message='You forgot to enter some fields: ';

  // 不合法的例子
  var dynamic ID='dynamic'; // 不允许有空格
  var 10base=10; // 以数字开头
  var while=10; // While是JavaScript的保留字
</script>
```

最后一个示例说明了另外一个问题：JavaScript有许多的保留字，基本上所有的JavaScript语句都用到了像while、if、continue、var或for这样的保留字。如果你不确定应该用什么作为变量

名，查一下JavaScript参考文档是一个不错的主意。为了避免这个问题，一些好的编辑器当你输入的时候会将保留字突出显示出来。

在JavaScript中对于名称的长度并没有限制；然而，为了避免大篇幅的脚本造成可读性差和调试困难的问题，可以使名称尽可能简单而具有描述性。尽量避免太一般化的名称：

- function1;
- variable2;
- doSomething()。

这些对于要调试或是学习这段代码的其他人（或是两个月之后你自己）没有什么实际意义。使用具有描述性的名称会更好一些，可以准确地告诉别人这个函数是做什么的，变量是什么：

- createTOC();
- calculateDifference();
- getCoordinates();
- setCoordinates();
- maximumWidth;
- address_data_file.

正如前几章我们提到的，可以使用下划线或“骆驼拼写法”（第一个单词小写，在它之后的每个单词首字母都大写）连接单词，然而骆驼拼写法更常用（DOM本身就使用它），而且习惯使用它会使你以后转向其他复杂的编程语言变得更加容易。使用骆驼拼写法的另一个好处是几乎在任何编辑器中你都可以通过一次双击来突出显示变量，然而对于用下划线分开的名字你需要使用鼠标选中完成突出显示。

警告 注意小写字母l和数字1！大多数编辑器使用一种像courier的字体，在这种情况下它们看起来是一样的，这可能带来一些混乱，并导致花费几个小时去寻找这个搞笑的bug。

3.3.2 代码布局

代码最重要的目的好像就被解释器转换，使计算机做一些事情，这其实是一个非常常见的误解。当代码正确的时候解释器会连“一个饱嗝都不打地”将所有代码“吞下”，然而编写好代码真正的挑战是当一个人想要编辑、调试、修改、扩展它的时候，不需要花上几个小时理解它的意图。对维护者来说，合乎逻辑而又简明的变量和函数名称是使代码维护变得简单的第一步——下一步则是合理的代码布局。

注解 如果真的无事可做，到一些程序员论坛上发表一下像“用空格比用tab键好”或“每一个大括号都应该换行”的观点，你可能收到上百封邮件来指出这些观点的优点和缺点。代码布局是一个非常热门的讨论话题。下面的示例我觉得很好，而且是一种常用的布局方式。在参与由多个开发人员组成的团队之前最好检查一下要遵循的标准是否有冲突，并使用下面所提到的方法。

简单地检查一下下面的代码示例；你现在或许还不明白它的作用（它实现这样一个小函数：在新窗口中打开具有smallpopup CSS类的每一个链接，并添加一个消息来显示将要发生什么），不要紧，现在我们要考虑的只是哪个调试和修改更容易一些。

没有缩进的效果：

```
function addPopUpLink(){
if(!document.getElementById||!document.createTextNode){return;}
var popupClass='smallpopup';
var popupMessage= '(opens in new window)';
var pop,t;
var as=document.getElementsByTagName('a');
for(var i=0;i<as.length;i++){
t=as[i].className;
if(t&&t.toString().indexOf(popupClass)!=-1){
as[i].appendChild(document.createTextNode(popupMessage));
as[i].onclick=function(){
pop=window.open(this.href,'popup','width=400,height=400');
return false;
}}}
window.onload=addPopUpLink;
```

使用缩进的效果：

```
function addPopUpLink(){
    if(!document.getElementById || !document.createTextNode){return;}
    var popupClass='smallpopup';
    var popupMessage= '(opens in new window)';
    var pop,t;
    var as=document.getElementsByTagName('a');
    for(var i=0;i<as.length;i++){
        t=as[i].className;
        if(t && t.toString().indexOf(popupClass)!=-1){
            as[i].appendChild(popupMessage);
            as[i].onclick=function(){
                pop=window.open(this.href,'popup','width=400,height=400');
                return false;
            }
        }
    }
    window.onload=addPopUpLink;
```

使用缩进且大括号换行的效果：

```
function addPopUpLink()
{
    if(!document.getElementById || !document.createTextNode){return;}
    var popupClass='smallpopup';
    var popupMessage= '(opens in new window)';
    var pop,t;
    var as=document.getElementsByTagName('a');
    for(var i=0;i<as.length;i++)
```

```

{
  t=as[i].className;
  if(t && t.toString().indexOf.popupClass)!=-1)
  {
    as[i].appendChild(document.createTextNode(popupMessage));
    as[i].onclick=function()
    {
      pop=window.open(this.href,'popup','width=400,height=400');
      return false;
    }
  }
}
window.onload=addPopUpLink;

```

我觉得已经非常明显了，缩进很不错，但通过tab键还是空格键生成缩进存在较大的争论。我个人喜欢用tab键，主要是因为这样删除容易并且键入的工作量较少。那些需要在非常基本的（或令人惊奇的，如果你懂得所有的快捷键）编辑器如vi和emacs上做许多工作的开发人员却不赞成，因为tab键经常会显示相当大的水平间隙。如果是这种情况，那么通过一个简单的正则表达式用两个空格代替一个tab并不是太大的问题。

大括号的开始是不是应该换行的问题应该由你自己来决定了。不换行的好处是容易删除错误的块，因为少了一行；换行的好处是代码看起来不那么密集。从个人角度来说，在JavaScript中我将大括号的开始与其他代码放在同一行，而在PHP中则换行——因为这似乎是两个开发人员社区各自的标准。

另一个问题是行的长度。今天的许多编辑器都有换行的选项，来保证你看代码的时候不需要水平滚动。然而，并不是所有的编辑器都能恰当地输出代码，并且以后的维护者可能没有那样理想的编辑器。所以使代码行短一些是个不错的选择——大概80个字符就好。

3.3.3 注释

只有人才能从注释中获益——尽管在一些高级的编程语言中，注释可以用来生成文档（PHP手册就是一个例子，正因为如此它有时才略显含糊）。对于代码运行来说，注释并不是必要的（如果使用了清晰的名称和缩进式的布局，代码应该是无需注释的）^①，但它可以大幅度地提高代码调试的速度。为上一个例子添加说明性的注释也许会使你更加容易理解：

```

/*
addPopUpLink
opens the linked document of all links with a certain
class in a pop-up window and adds a message to the
link text that there will be a new window
*/
function addPopUpLink(){
  // Check for DOM and leave if it is not supported

```

^① Martin Fowler在《重构》一书中将注释过多视为一种代码坏味，即代码存在需要重构的迹象。——编者注

```

if(!document.getElementById || !document.createTextNode){return;}
// Assets of the link - the class to find out which link should
// get the functionality and the message to add to the link text
var popupClass='smallpopup';
var popupMessage= '(opens in new window)';
// Temporary variables to use in a loop
var pop,t;
// Get all links in the document
var as=document.getElementsByTagName('a');
// Loop over all links
for(var i=0;i<as.length;i++)
{
    t=as[i].className;
    // Check if the link has a class and the class is the right one
    if(t && t.toString().indexOf(popupClass)!==-1)
    {
        // Add the message
        as[i].appendChild(document.createTextNode(popupMessage));
        // Assign a function when the user clicks the link
        as[i].onclick=function()
        {
            // Open a new window with
            pop=window.open(this.href,'popup','width=400,height=400');
            // Don't follow the link (otherwise the linked document
            // would be opened in the pop-up and the document).
            return false;
        }
    }
}
window.onload=addPopUpLink;

```

更好理解了，不是吗？但也有点夸张了。这样的例子可用于培训文档和自学教程，但若在最终的产品中出现就显得过了——在写注释的时候适度总是很关键的。在大多情况下，解释一下做什么以及将更改什么就足够了。

```

/*
addPopUpLink
opens the linked document of all links with a certain
class in a pop-up window and adds a message to the
link text that there will be a new window
*/
function addPopUpLink()
{
    if(!document.getElementById || !document.createTextNode){return;

    // Assets of the link - the class to find out which link should
    // get the functionality and the message to add to the link text
    var popupClass='smallpopup';
    var popupMessage=document.createTextNode(' (opens in new window)');
}

```

```

var pop,t;
var as=document.getElementsByTagName('a');
for(var i=0;i<as.length;i++)
{
    t=as[i].className;
    if(t && t.toString().indexOf.popupClass!=-1)
    {
        as[i].appendChild(popupMessage);
        as[i].onclick=function()
        {
            pop=window.open(this.href,'popup','width=400,height=400');
            return false;
        }
    }
}
window.onload=addPopUpLink;

```

这段注释使得理解整个函数的功能以及可以更改哪些设置变得容易了。这使得快速更改更加容易——无论如何更改函数的功能都需要维护人员更加仔细分析你的代码。

3.3.4 函数

函数(function)是可复用的代码块，因此它是今天大多数程序中不可或缺的一部分，包括使用JavaScript编写的程序。假设你必须一次又一次完成某个计算，或需要一遍又一遍地检查某个条件，可以在需要的地方复制和粘贴相同的代码行，但是，使用函数会更加高效。

函数可以以参数(有时叫自变量，argument)的方式获得值，并且可以在对获得的值完成测试和修改之后返回一个值。

要创建一个函数，可以使用关键字function后面跟着函数名称和在括号内由逗号分开的参数：

```

function createLink(linkTarget, LinkName)
{
    // Code
}

```

一个函数可以有多少个参数并没有限制，但不宜太多，因为这可能会变得非常混乱。如果看过一些DHTML代码，你会发现有带20多个参数的函数。在其他函数中调用的时候，记住这些顺序差不多会让你恨不得把所有的东西都重新写一遍。编写函数的时候，记住太多的参数意味着维护和调试工作要比它应有的难度大得多。

与PHP不同，在参数可用之前JavaScript并没有选项预先设置参数。可以通过用if条件语句检查参数是否为null(意思是，“空值，并不是0”)来解决这个问题：

```

function createLink(linkTarget, LinkName)
{
    if (linkTarget == null)
    {

```

```

linkTarget = '#';
}
if (linkName == null)
{
    linkName = 'dummy';
}
}
}

```

函数通过return关键字返回处理过的值。如果一个函数被事件处理程序调用之后返回布尔值false，那么正常触发的事件顺序将会被终止。当你想将函数应用于链接并且阻止浏览器导航到链接的href时，这是非常方便的。在3.1.1节我们也用到了这些。

其他在return语句后面的任何值都会被返回给调用的函数。让我们修改一下createLink函数，创建一个链接并在完成创建后将其返回：

```

function createLink(linkTarget,linkName)
{
    if (linkTarget == null) { linkTarget = '#'; }
    if (linkName == null) { linkName = 'dummy'; }

    var tempLink=document.createElement('a');
    tempLink.setAttribute('href',linkTarget);
    tempLink.appendChild(document.createTextNode(linkName));

    return tempLink;
}

```

另外一个函数使用生成的链接并将其添加在某个元素的后面。如果不存在已定义的元素ID，就将链接添加到文档主体。

```

function appendLink(sourceLink,elementId)
{
    var element=false;
    if (elementId==null || !document.getElementById(elementId))
    {
        element=document.body;
    }
    if(!element) {
        element=document.getElementById(elementId);
    }
    element.appendChild(sourceLink);
}

```

现在使用上面的两个函数，在另外一个函数中通过适当的参数调用它们。

```

function linksInit()
{
    if (!document.getElementById || !document.createTextNode) { return; }
    var openLink=createLink('#','open');
    appendLink(openLink);
}

```

```

var closeLink=createLink('closed.html','close');
appendLink(closeLink,'main');
}

```

函数linksInit()检查DOM是否可用（因为此函数是唯一一个调用其他函数的函数，在其他函数中不必再次检查）并且创建一个目标地址为#的链接，链接的文本为open。

然后调用appendLink函数并将新生成的链接作为参数传递给它，注意并没有传递目标元素，这意味着elementId是null，appendLink()函数将链接添加到文档的主体。

linksInit()第二次调用createLink()函数，传递给它目标链接地址closed.html和链接文本close，通过appendLink()函数将链接应用于HTML中ID为main的元素。如果存在ID为main的元素，appendLink()将链接添加到其后；不存在则将文档的主体作为第二选择。

如果你感到有些乱，别着急，以后会看到更多的示例。现在最重要的是理解什么是函数，还有它们的作用。

- 函数的作用是完成可能会反复执行的某个任务——将这样任务都编写为一个函数，不要创建一次做几件事情的巨大函数^①。
- 函数可以有任意多个参数，每个参数可以是任意类型——字符串、对象、数字、变量或数组。
- 在函数自己的定义中，不能够预先设定参数的值，但你可以检查它们是否被定义过，或是通过if条件语句设置默认值。可以通过三元运算符非常简洁地完成这样的操作，下一节将讲到三元运算符。
- 函数应该有一个合乎逻辑的名字来描述自己的功能。尽量使名字贴近任务的主题，因为像init()这样名字太一般的函数，可能在其他被包含的JavaScript文件中也会出现，从而导致函数覆盖。对象字面量能够避免这个问题，本章稍后将讲解这一点。

3.3.5 使用三元运算符简化代码

看一下上面的appendLink()函数，你可能预感到大量的if条件和switch语句可能导致冗长而复杂的代码。避免这种膨胀代码的一个小技巧是使用三元运算符（ternary operator）。三元运算符的语法如下：

```
var variable = condition ? trueValue:falseValue;
```

这对于布尔条件和较短的值是非常方便的。比如，可以用一行代码代替下面冗长的if条件语句。

```

// Normal syntax
var direction;
If(x<200)
{
  direction=1;
}
else
{

```

^① 这种函数也是一种代码典型错误，重构中称之为“过长方法”。——编者注

```

direction=-1
}
// Ternary operator
var direction = x < 200 ? 1 : -1;

```

其他例子：

```

t.className = t.className == 'hide' ? '' : 'hide';
var el = document.getElementById('nav')
    ? document.getElementById('nav')
    : document.body;

```

也可以嵌套三元运算符，但这样可读性相当差：

```

y = x < 20 ? (x > 10 ? 1 : 2) : 3;
// equals
if(x<20)
{
    if(x>10)
    {
        y=1;
    }
    else
    {
        y=2;
    }
}
else
{
    y=3
}

```

3.3.6 函数的分类和复用

如果有大量的JavaScript函数，那么最好把它们放入一个js文件中，只有在需要的时候才去应用它们。命名js文件的时候要根据其内函数的功能，比如，formvalidation.js或dynamicmenu.js。

某种程度上很多函数已经是现成的了，因为有了大量打好包的JavaScript库（函数和方法的集合）帮助我们实现各种特定的功能。第11章将介绍一些JavaScript库，下面几章中我们依然创建自己的函数。

3.3.7 变量和函数作用域

在函数内通过关键字var创建的变量只在函数内部有效。这看起来像是一个缺点，但这恰恰意味着你的脚本之间互不干涉——在使用JavaScript库和自己函数集合的时候这可能是至关重要的。

在函数外部定义的变量称为global（全局）变量。全局变量很危险，应该尽可能在函数内部创建变量。这保证我们的脚本与可以应用到网页的其他脚本都能顺利地运行。许多脚本都使用一般性的变量名称，如navigation或currentSection。如果它们被定义为全局变量，那么脚本可能覆盖彼此的值。试着运行下面的函数，看一下忽略一个var关键字可能会引起什么情况：

```
<script type="text/javascript">
    var demoVar=1 // Global variable
    alert('Before withVar demoVar is' +demoVar);
    function withVar()
    {
        var demoVar=3;
    }
    withVar();
    alert('After withVar demoVar is' +demoVar);
    function withoutVar()
    {
        demoVar=3;
    }
    withoutVar();
    alert('After withoutVar demoVar is' +demoVar);
</script>
```

withVar函数没有使变量变化，withoutVar却改变了它：

```
Before withVar demoVar is 1
After withVar demoVar is 1
After withoutVar demoVar is 3
```

3.3.8 使用对象字面量保证脚本安全

前面我们谈到了通过var关键字定义局部变量实现变量的安全。原因是为了避免其他函数使用相同的变量名称，造成函数之间互相覆盖变量值的问题。函数也是如此。在同一个HTML中的不同脚本元素里可能会包含多个JavaScript文件，函数功能可能由于其他JavaScript文件当中有同名函数而失败。可以通过一个命名规则来避免这个问题，如myscript_init()和myscript_validate()作为函数名。然而，这显得有些笨拙。JavaScript提供了一种更好的、以对象形式解决的办法。

可以定义一个新的对象，将函数作为这个对象的方法使用——这就是Date和Math等JavaScript对象的工作原理。看下面的例子：

```
<script type="text/javascript">
    myscript=new Object();
    myscript.init=function()
    {
        // Some code
    };
    myscript.validate=function()
    {
        // Some code
    };
</script>
```

注意，如果你试着调用函数init()和validate()，会出现错误，因为它们根本不存在了。相反，你要使用myscript.init()和myscript.validate()。

将所有的函数封装在一个对象里（成为对象的方法），与其他一些语言（如C++和Java）编程

中使用的类是相似的。在这些语言中，将应用于相同任务的函数放入相同任务的类中，这使得创建大量的代码更加容易，并且不会被上百个函数弄得混乱。

我们使用的这种语法仍然有点笨拙，因为不得不一遍又一遍地重复对象的名称。有一个简单的表示方法叫作对象字面量（object literal），它使这一切更加简单。

对象字面量已经存在很长一段时间了，但未被充分使用，今天它已经变得越来越流行了。甚至可以这样认为，使用对象字面量的脚本都是优秀的、现代的JavaScript。

对象字面量所做的是使用一种简写方式来创建对象，并将每个函数作为对象的方法（不再是独立的函数）。让我们看下面的例子，其中大对象dynamicLinks使用对象字面量将3个函数封装起来：

```
var dynamicLinks={  
    linksInit:function()  
    {  
        if (!document.getElementById || !document.createTextNode)⇒  
            { return; }  
        var openLink=dynamicLinks.createLink('#','open');  
        dynamicLinks.appendLink(openLink);  
        var closeLink=dynamicLinks.createLink('closed.html','close');  
        dynamicLinks.appendLink(closeLink,'main');  
    },  
    createLink:function(linkTarget,linkName)  
    {  
        if (linkTarget == null) { linkTarget = '#'; }  
        if (linkName == null) { linkName = 'dummy'; }  
        var tempLink=document.createElement('a');  
        tempLink.setAttribute('href',linkTarget);  
        tempLink.appendChild(document.createTextNode(linkName));  
        return tempLink;  
    },  
    appendLink:function(sourceLink,elementId)  
    {  
        var element=false;  
        if (elementId==null || !document.getElementById(elementId))  
        {  
            element=document.body;  
        }  
        if(!element){element=document.getElementById(elementId)}  
        element.appendChild(sourceLink);  
    }  
}  
  
window.onload=dynamicLinks.linksInit;
```

可以看到，所有的函数都作为方法包含在dynamicLinks对象内，这意味着如果想要调用它们，我们需要将对象的名字添加到函数名称之前。

语法有点不同，不是将function关键字放在函数名之前，而是将其添加到函数名之后，其前用冒号。另外，除了最后一个结束大括号外，其他结束大括号之后都要跟一个逗号。

如果想使用可被某对象内部所有方法访问的变量，可通过下面的语法实现：

```

var myObject=
{
  objMainVar:'preset',
  objSecondaryVar:0,
  objArray:['one','two','three'],
  init:function(){},
  createLinks:function(){},
  appendLinks:function(){}
}

```

可能讲的内容有点多了，但不用担心，本章可是良好编程实践的参考文档，可以以后回头再来阅读。下一章中我们将继续学习更多实例，剖析一个HTML文档，并处理各个部分。

3.4 小结

学完了本章，你应该可以区分出网上看到的现代脚本和旧脚本。旧脚本的主要表征是：

- 使用大量的`document.write()`。
- 检查浏览器和版本，而不是对象。
- 写出大量的HTML而不是访问文档中已经存在的内容。
- 使用专有的DOM对象，如微软IE使用的`document.all`或Netscape Navigator使用的`document.layers`。
- 在文档中随处出现（而不是在`<head>`中，或通过`<script src="---.js">`被包含在文档中）并且依赖于`javascript:链接`而不是赋值事件。

你还了解到应该将JavaScript放入到一个js文档而不是将其嵌入到HTML中，从而将行为从结构中分离出来。

然后你学习了应该使用对象检测而不是依赖于浏览器名称，渐进增强的含义以及如何将其应用于Web开发。测试用户代理的能力而不是测试浏览器的名称和版本，这使得你的脚本语言在不方便测试用户代理的情况下（如用户代理不在手边）也可以运行。这也意味着你不必为每次新的浏览器的发布而担心了——如果它支持相关标准，那就好。

我们谈到了可访问性和它对JavaScript意味着什么，还浏览了大量的代码。主要需要记住：

- 对脚本中要使用的对象进行测试。
- 对运行良好的而且不用客户端编程的既有网站进行改进，而不是首先添加脚本然后再添加非脚本的备用方案。
- 保持代码独立，不要使用任何可能与其他脚本冲突的全局变量。
- 编码时要记住，你需要把代码交给其他人去维护，在未来的3个月内这个人可能就是你，你应该能够立即想起它的作用。
- 注释出你代码的功能并且使用便于读取的格式，这会使寻找bug和修改函数变得容易。

够多的了——除了我在前面提到的但是没有准确定义的事件处理程序（event handler）之外。后者我将在第5章中讲述。但现在，回到座位，端起一杯咖啡或茶，放松一下，准备好学习JavaScript是与HTML和CSS的交互。

HTML与JavaScript

在本章中，你终于可以着手编写真正的JavaScript代码了。你会学到JavaScript如何和（在HTML中定义的）网页结构交互，如何接收数据并把信息反馈给你的访问者。首先我会阐述一下什么是HTML文档，它是如何构成的，并讲述几种通过JavaScript创建网页内容的方式。你接着将会了解到JavaScript开发人员的“瑞士军刀”——DOM（文档对象模型），以及如何把JavaScript和HTML分开来，以创建开发人员过去使用DHTML无法顺利创建的平滑特效。

4.1 HTML 文档剖析

在用户代理上显示的文档一般为HTML文档。即使你使用了一种服务器端语言，如ASP.NET、PHP、ColdFusion或Perl，但是如果你使用浏览器执行它们，结果仍然是HTML文档。现代的浏览器（如Mozilla或Safari）也支持XML、SVG和其他格式，但对于99%的日常网络工作，你仍然要使用HTML或XHTML。

HTML文档是一种文本文档，它以DOCTYPE开头，告诉用户代理这个文档是什么以及它应该如何被处理。接着使用了一个HTML元素，它包含了所有的其他元素以及文本的内容。HTML元素中应包含一个lang属性和一个dir属性，lang属性定义了所使用的语言（人类语言，而非编程语言）；dir属性定义了文本的读取顺序（印欧语系从左到右，闪族语系则从右到左）。在HTML元素中需要一个带有TITLE元素的HEAD元素。可以添加一个可选的META元素，它决定了使用什么编码来显示文本——如果你还没有在服务器上设置编码。与HEAD元素处于同一级别，但在<head>的结束标签之后，就是BODY——这个元素包含了所有的页面内容。

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

类似这样的标记文档是由许多标签（tag，用被置于标签括弧内的单词或字母表示，如

）和文本内容构成的。文档应该是合式的（意味着对于每一个像

这样的开始标签都必须与一个

这样的结束标签相匹配）并验证是否符合提供的DTD。可以在W3C网站（<http://validator.w3.org/>）上验证文档。

HTML元素是指在尖括号<>中的所有东西，它以一个

这样的标签开头，后面紧跟着内容和一个相同名字的结束标签——如

。每个元素的开始和结束标签之间可以有内容。每个元素可以有几个属性（attribute）。在DOCTYPE上链接的DID（Document Type Definition）决定了允许使用的标签集合、它们可能的嵌套方式以及每个标签可以拥有的属性。下面的例子为一个属性名为class的p元素，这个属性的值为intro，p包含的文本为“Lorem Ipsum”。

```
<p class="intro">Lorem Ipsum</p>
```

浏览器会检查DOCTYPE，并把它遇到的元素与DTD相比较。HTML 4.01的DTD定义告诉浏览器p表示一个段落以及class属性对这个元素是有效的。它还知道class属性应该检查关联的CSS样式表，获取带有那个class的p的定义，并把它相应地呈现出来。现在，如果你使用一个在DTD未被定义的属性（如myattribute）会有什么情况发生？

```
<p class="intro" myattribute="left">Lorem Ipsum</p>
```

什么也没有发生——尽管你犯了一个技术性的错误。浏览器是非常宽容的，它们不会停止呈现，即使遇到一些类似这样的不认识的东西，它们仍会将该属性放入DOM树中。这样对于用户和开发人员当然方便，但却使得提倡正确的HTML语法和标准兼容变得困难了。然而，有几个我们应该力争标准兼容的原因——甚至是在通过JavaScript生成的HTML当中：

- 当我们知道HTML有效的时候，可以更容易追踪错误。
- 遵从规则的文档更容易维护——因为可以使用验证程序来测量它的质量。
- 当你按照一个达成共识的标准从事开发工作的时候，用户代理更有可能适当地呈现或转化你的页面。
- 如果它们是有效的HTML文档，最终的文档可以很容易地转化为其他格式。

现在，如果为我们的示例HTML添加更多的元素并把它在浏览器中打开，将会得到如图4-1所示的输出结果：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
  "http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
  <head>
    <meta http-equiv="Content-Type">
      content="text/html; charset=utf-8" />
    <title>DOM Example</title>
  </head>
  <body>
    <h1>Heading</h1>
    <p>Paragraph</p>
    <h2>Subheading</h2>
```

```

<ul id="eventsList">
  <li>List 1</li>
  <li>List 2</li>
  <li><a href="http://www.google.com">Linked List Item</a></li>
  <li>List 4</li>
</ul>
<p>Paragraph</p>
<p>Paragraph</p>
</body>
</html>

```

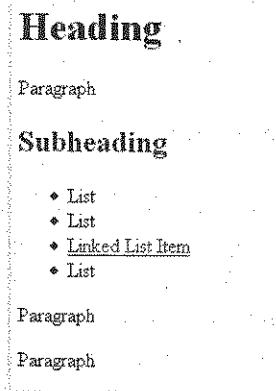


图4-1 浏览器呈现的一个HTML文档

关于XHTML的简短说明

有读者可能会对前面的代码例子中我使用HTML 4.01 STRICT而没用更加流行的XHTML会感到不解。原因是这样的：更多的浏览器支持HTML 4.01 STRICT，并且在语法上它和XHTML一样清晰。真正的XHTML文档在服务器上应该以application /XML+XHTML的形式被传送，而不是text/HTML——这是很多开发人员不敢走的一步，因为这意味着在IE中该页将无法显示。一旦最常用的浏览器都支持真正的XHTML，转化文档就不会困难了。现在4.01 STRICT使用这个窍门，使浏览器按W3C要求的那样显示文档——大多数浏览器都是这样做的。还要注意，在本书正文叙述中HTML元素的名称我可能使用的是大写，如H1，而代码例子中所有的元素都是小写——因为在XHTML中这是必需的。XML和XHTML都区分大小写，然而老式的和不太严格的HTML版本是不区分大小写的。

用户代理所“看到的”文档有些不同，DOM将文档视为一些节点的集合，包括元素节点、文本节点和属性节点。元素与它们的文本内容都是独立的节点（node）。属性节点是元素的属性。针对文档的其他部分，DOM还包含了其他种类的节点，但这3种节点——元素节点、文本节点和属性节点——是JavaScript和HTML中最重要的。如果想从浏览器的角度看一下文档，可以使用类似Firefox的DOM Inspector（检查器）这样的工具，它会把文档显示为树状结构，如图4-2所示：

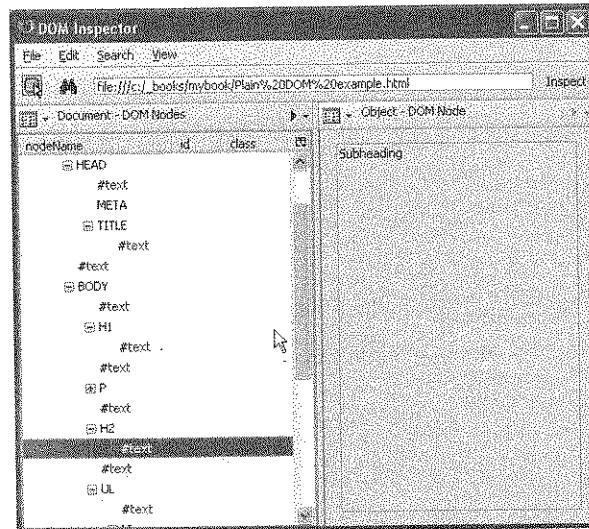


图4-2 在Mozilla DOM检查器中所展示的一个文档的DOM表示

提示 在Firefox中可以通过Tools→DOM Inspector或直接按下Ctrl+Shift+I来访问DOM检查器^①。它允许检查文档每个部分的细节，甚至移除元素，这在打印页面的时候是非常方便的。在附录（包括验证和调试）里可以学习到更多有关DOM检查器的内容。

注解 注意到元素之间的所有#text节点了吗？这并不是我们添加到文档的文本，而是我们在每行结尾添加的行结束符。有些浏览器把它视为文本节点，而有的则不是这样——稍后我们通过JavaScript访问文档内元素的时候，这将是一件非常讨厌的事情。

图4-3展示了文档树的另一种表示方式。

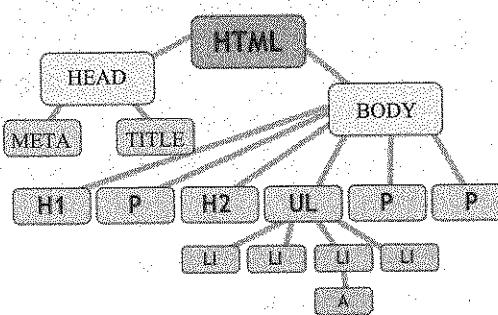


图4-3 HTML文档的结构

^① DOM Inspector是Firefox插件，需要另外安装。——编者注

知道HTML究竟是什么非常重要：HTML是结构化的内容，而不是像放在不同坐标系中的带元素的图形那样可见的结构^①。当你有一个正确的、有效的HTML文档的时候，就可以通过JavaScript不受任何限制地访问和修改它。无效的HTML文档可能会导致你的脚本失败，无论你做多少测试。一个比较典型的错误就是在同一个文档当中使用了两次同一个id属性值，这违背了唯一标识符（ID）的功用。

4.2 在网页中使用 JavaScript 提供反馈信息：老的方式

我们已经使用过一种在HTML文档中为用户提供反馈信息的方式——用document.write()写出内容。我们也讨论了这个方法存在的问题，也就是混合了结构层和表现层，并且失去了将所有JavaScript放入一个单独文件所具有的易维护的优点。

使用 window 方法：prompt()、alert()和confirm()

为用户提供反馈信息和接收用户输入数据的另一种方式是使用浏览器提供的window对象的方法，即prompt()、alert()和confirm()。

最常用的window方法就是alert()，图4-4所示的就是使用它的一个例子。它所做的就是在在一个对话框中显示一个值（如果硬件支持的话，它可能还会播放一个声音）。用户需要激活（用鼠标单击或是在键盘上按下回车）OK按钮来关闭这个消息。



图4-4 一个JavaScript的警告窗口（Firefox, Windows XP）

警告窗口在不同的浏览器和操作系统上界面外观是不一样的。

作为一种用户反馈机制，alert()拥有绝大多数用户代理都支持这一优点，但它也是一种给用户传递信息非常有争议而且笨拙的方式。警告窗口通常显示的是坏消息或提前警告某人有危险的信息——这可能并不是你的意图。

举个例子来说，你希望告诉访问者在提交表单之前，在搜索文本域需要输入内容，那么可以使用一个警告窗口来做：

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

^① 指图4-2、图4-3中的图形表示所显示的是文档结构，而不是文档本身。——编者注

```

<script type="text/javascript">
    function checkSearch()
    {
        if(!document.getElementById || !document.createTextNode){return;}
        if(!document.getElementById('search')){return;}
        var searchValue=document.getElementById('search').value;
        if(searchValue=='')
        {
            alert('Please enter a search term');
            return false;
        }
        else
        {
            return true;
        }
    }
</script>
</head>
<body>
    <form action="sitesearch.php" method="post">
        onsubmit="return checkSearch();"
    <p>
        <label for="search">Search the site:</label>
        <input type="text" id="search" name="search" />
        <input type="submit" value="Search"/>
    </p>
    </form>
</body>
</html>

```

如果访问者试着通过提交按钮提交表单，他就会得到警告信息，并且浏览器也不会在他激活OK按钮后将表单提交给服务器。在Windows XP上的Mozilla Firefox浏览器中，警告信息如图4-5所示：

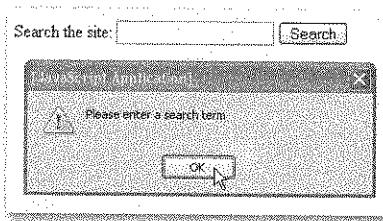


图4-5 通过警告窗口给出一个表单错误的反馈信息

警告并不会给脚本返回任何信息——它们只是简单地给用户一个信息，并在OK按钮激活之前停止代码的进一步执行。

这与prompt()和confirm()是不同的，前者允许访问者输入一些东西，但后者要求用户确认一个动作。

提示 从调试的角度看，`alert()`使用起来很方便。只要在代码中需要知道变量值的地方添加`alert(VariableName)`。你会得到变量的信息，并停止其余代码的执行直到OK按钮被激活——这对于跟踪调试你的脚本在何处失败以及如何失败非常有用。可是在循环中使用它要小心——没有办法停止循环，而且在回到你的编辑之前，恐怕要按上百次的回车。还有一些其他的调试工具，如Mozilla、Opera和Safari的JavaScript控制台以及Mozilla Venkman，在附录中有更多这方面的介绍。

可以扩展一下上面的例子，在用户对于常见词JavaScript进行搜索的时候，要求用户进行确认（如图4-6所示）：

```
function checkSearch()
{
    if(!document.getElementById || !document.createTextNode){return;}
    if(!document.getElementById('search')){return;}
    var searchValue=document.getElementById('search').value;
    if(searchValue=='')
    {
        alert('Please enter a search term before sending the form');
        return false;
    }
    else if(searchValue=='JavaScript')
    {
        var really=confirm('"JavaScript" is a very common term.\n'+
        'Do you really want to search for this?');
        return really;
    }
    else
    {
        return true;
    }
}
```

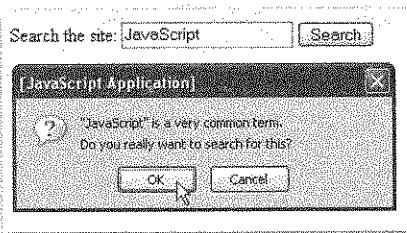


图4-6 需要用户通过`confirm()`进行确认的例子

注意`confirm()`是一种根据用户激活OK或Cancel按钮而返回布尔型值(`true`或`false`)的方法。确认对话框是一种可以防止用户在Web应用程序中执行错误步骤的简单方法。虽然这不是要求用

户进行确认的最恰当的方法，但它们是非常稳定的，并且提供了一些你自己编写的确认函数可能不具备的功能——如播放警告声音。

`alert()`和`confirm()`都发送信息给用户，但如何检索信息呢？检索用户输入的一种简单方法是`prompt()`。它有2个参数，第一个是显示在输入框上面的作为标签的字符串，第二个是输入框的预设值。`OK`和`Cancel`按钮（或其他类似东西）在文本和输入框下面显示，如图4-7所示。

```
var user=prompt('Please choose a name','User12');
```

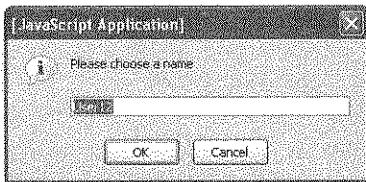


图4-7 在提示中允许用户输入数据

当用户激活`OK`按钮的时候，变量`user`的值可能是`User12`（如果用户没有更改预设值）或用户输入的任何值。当用户激活的是`Cancel`按钮的时候，这个变量的值就会是`null`。

可以使用此功能在用户将表单提交到服务器之前修改某个值：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
  <head>
    <meta http-equiv="Content-Type">
      content="text/html; charset=utf-8" />
  <title>Date example</title>
  <script type="text/javascript">
    function checkDate()
    {
      if (!document.getElementById || !
          !document.createTextNode){return;}
      if (!document.getElementById('date')){return;}
      // Define a regular expression to check the date format
      var checkPattern=new
        RegExp("\d{2}/\d{2}/\d{4}");
      // Get the value of the date entry field
      var dateValue=document.getElementById('date').value;
      // If there is no date entered, don't send the form
      if(dateValue=='')
      {
        alert('Please enter a date');
        return false
      }
      else
      {
        // Tell the user to change the date syntax either until
```

```

// she presses Cancel or entered the right syntax
while(!checkPattern.test(dateValue) && dateValue!=null)
{
    dateValue=prompt('Your date was not in the right format. ' +
        + 'Please enter it as DD/MM/YYYY.', dateValue);
}
return dateValue!=null;
}
}
</script>
</head>
<body>
<h1>Events search</h1>
<form action="eventssearch.php" method="post">
    onsubmit="return checkDate();">
    <p>
        <label for="date">Date in the format DD/MM/YYYY:</label><br />
        <input type="text" id="date" name="date" />
        <input type="submit" value="Check " />
        <br />(example 26/04/1975)
    </p>
    </form>
</body>
</html>

```

如果现在对正则表达式和test()方法不理解，也不用着急，这些东西会在第9章中讲。现在最重要的是我们在while循环内使用了prompt()。while循环会一遍又一遍地显示相同的提示窗口，直到用户按下Cancel按钮（意味着dateValue会变为null）或输入了正确格式的一个日期（这满足了正则表达式checkPattern的测试条件）。

小结

可以使用prompt()、alert()和confirm()方法创建非常不错的JavaScript脚本，它们有这样的特点：

- 很容易理解，这些方法使用了浏览器的界面外观和功能，并且提供了比HTML更友好的用户界面（特别是警告声音，播放的时候可以帮助很多的用户）。
- 很容易实现，只需要JavaScript，无需用到HTML元素，这也是它们为什么被用在bookmarklet/favelet（可以通过书签和收藏夹调用的小脚本，它们用来改变当前的文档——基本在所有支持JavaScript的浏览器中都可以对浏览器进行扩展，参见<http://www.bookmarklets.com/>）中的原因。
- 它们出现在当前文档的外部和上方，这一点更凸显出它们的重要性。

然而也有一些观点反对使用这些方法获取数据和提供反馈信息：

- 无法给这些消息应用样式，并且它们阻碍了网页，这点确实使它们显得很重要，但从设计角度来起看显得很笨拙。因为它们是用户的操作系统或浏览器界面的一部分，虽然它们很容易得到用户的认可，但却会破坏产品可能必须坚持的设计准则和方针。

- 反馈并不会发生在网站本身的界面外观当中——它降低了网站的重要性，并且会严重妨碍用户使用我们辛辛苦苦改进了的易用设计元素。
- 它们依赖于JavaScript——而按要求在JavaScript关闭的时候反馈应该依然可用。
- 除非你使用的是Mozilla或老版本的Netscape浏览器，否则没有办法可以告诉你alert()或prompt()是从这个网站还是另一个不同的网站发出的。这也是一个安全问题，因为可以让prompt()从第三个站点上发起，读取并把用户的输入传给你。这是一种叫做网络钓鱼(phishing)的技术现象，它是一个很严重的安全威胁。随着一些从事安全方面的公司发布关于这个问题的主题信息，如果网站使用这些反馈和输入机制，用户可能不会信任它(<http://secunia.com/advisories/15489/>)。

4.3 通过DOM访问文档

除了现在知道的Window方法，你还可以通过DOM访问网页文档。从某种意义上说，我们已经在document.write()的例子中这样做了，document对象是我们需要改变和添加内容的东西，write()是用来完成这些操作的方法。然而document.write()只是向文档中添加一些字符串而不是一些节点和属性，你不能把这个JavaScript语句分离出来放到一个单独的文件里——document.write()只在HTML中你把它放入的地方才有效。我们现在需要的是一种方法，用来找到文档中我们要进行更改或增加内容的位置，这正是DOM及其方法为我们提供的功能。通过前面的例子可以知道，浏览器将文档看作节点和属性的集合，DOM也为我们提供了获取这些节点和属性的工具。可以通过2种方法得到文档的元素：

- document.getElementsByTagName('p');
- document.getElementById('id').

document.getElementsByTagName('p')方法返回所有名称为p的对象集合（p可以为任意的HTML元素），document.getElementById('id')返回带有这个ID的元素。如果你已熟悉CSS，可以将这两种方法与CSS的元素选择器和ID选择器——tag{}和#id{}联系起来理解。

注解 与CSS的相似之处仅限于此，因为没有DOM对象等价于class选择器.class{}。然而，由于这是个非常便利的方法，一些开发人员对这个问题提出了他们自己的解决办法，并创建了getElementsByClassName()函数。

如果回头看一下我们前面用到的HTML示例，你就可以编写一小段JavaScript代码来说明这两种方法的用法：

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

```

<meta http-equiv="Content-Type"→
    content="text/html; charset=utf-8" />
<title>DOM Example</title>
<script type="text/JavaScript" src="exampleFindElements.js">
</script>
</head>
<body>
<h1>Heading</h1>
<p>Paragraph</p>
<h2>Subheading</h2>
<ul id="eventsList">
    <li>List 1</li>
    <li>List 2</li>
    <li><a href="http://www.google.com">Linked List Item</a></li>
    <li>List 4</li>
</ul>
<p>Paragraph</p>
<p>Paragraph</p>
</body>
</html>

```

现在我们的脚本可以通过调用`getElementsByTagName()`方法，读出文档中列表项和段落的数量，并将返回的值赋给变量——第一个标签名称为`li`，另一个名称为`p`:

```

var listElements=document.getElementsByTagName('li');
var paragraphs=document.getElementsByTagName('p');
var msg='This document contains '+listElements.length+' list items
msg+='and '+paragraphs.length+' paragraphs.';
alert(msg);

```

如图4-8所示，如果在浏览器中打开这个HTML文档，会发现这两个值都为0！

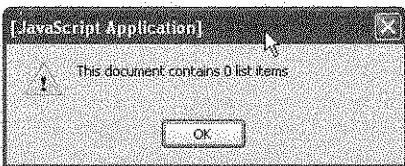


图4-8 在网页被显示出来之前访问网页元素得到的意想不到的结果

集合中没有任何元素，因为在读取文档内容时文档还未被浏览器呈现出来。我们需要延迟读取直到文档完全被加载和呈现出来。

可以通过在系统完成加载之后调用一个函数来达到这个目的。当`window`对象的`onload`事件触发后，文档就完成了加载，第5章会介绍更多有关事件的东西。现在，我们只是使用`window`对象的`onload`事件处理程序来触发这个函数:

```

function findElements()
{
    var listElements = document.getElementsByTagName('li');
}

```

```

var paragraphs = document.getElementsByTagName('p');
var msg = 'This document contains ' + listElements.length + ' list items\n';
msg += 'and ' + paragraphs.length + ' paragraphs.';
alert(msg);
}
window.onload = findElements;

```

现在如果在浏览器中打开这个HTML文档，你会看到如图4-9中的警告窗口，上面有正确的列表元素和段落的数量。



图4-9 表示查找的元素数量的警告窗口

可以像访问一个数组那样访问每一个有特定名称的元素——再次提醒你注意，数组的计数是从0而不是从1开始的。

```

// Get the first paragraph
var firstpara = document.getElementsByTagName('p')[0];
// Get the second list item
var secondListItem = document.getElementsByTagName('p')[1];

```

可以将几个getElementsByTagName()方法调用结合起来，直接读取子节点。比如，要访问第3个列表项中的第一个链接，可以使用：

```

var targetLink=document.getElementsByTagName('li')[2].getElementsByTagName('a')[0];

```

虽然这种方法显得有些乱，但有更巧妙的方法可以访问子节点——我们很快就会看到。如果想访问最后一个元素，可以使用数组的length属性：

```
var lastListElement = listElements[listElements.length - 1];
```

length属性也允许用来循环所有的元素，并一个接一个地进行更改：

```

var linkItems = document.getElementsByTagName('li');
for(var i = 0; i < linkItems.length; i++)
{
    // Do something...
}

```

元素ID在整个文档中应该是唯一的，所以getElementById()的返回值是单独一个对象而不是对象的数组。

```
var events = document.getElementById('eventsList');
```

可以将2种方法混合起来使用，以减少循环时所遍历元素的数量。前面的for循环访问文档中

所有的LI元素，下面的例子只会遍历ID为eventsList（特定ID的对象名代替document对象）的元素内部：

```
var events = document.getElementById('eventsList');
var eventLinkItems = events.getElementsByTagName('li');
for(var i = 0; i < eventLinkItems.length; i++)
{
    // Do something...
}
```

有了getElementsByTagName()和getElementById()的帮助，你就可以访问文档中的所有元素或特定目标的单个元素。如前所述，getElementById()是document的一个方法，getElementsByTagName()是每个元素的一个方法。现在该看一看到达该元素后如果在文档中进行导航了。

4.4 元素的子节点、父节点、兄弟节点和值

你已经知道，通过拼接getElementsByTagName()方法可以访问到其他元素内部的元素。然而，这种方式相当笨拙，它意味着你要知道所更改的HTML文档。有时候那是不可能的，因此你必须找到一种更通用的方式来遍历操纵HTML文档。DOM已经提供了这个方法，就是通过子节点(children)、父节点(parent)和兄弟节点(sibling)。

这种关系描述了当前元素在DOM树中的位置，以及它是否包含其他的元素。再来看一下我们简单的HTML示例，关注一下文档的主体部分：

```
<body>
    <h1>Heading</h1>
    <p>Paragraph</p>
    <h2>Subheading</h2>
    <ul id="eventsList">
        <li>List 1</li>
        <li>List 2</li>
        <li><a href="http://www.google.com">Linked List Item</a></li>
        <li>List 4</li>
    </ul>
    <p>Paragraph</p>
    <p>Paragraph</p>
</body>
```

所有缩进的元素都是body的子节点。H1、H2、UL和P是兄弟节点，而LI元素是UL元素的子节点，是其他LI元素的兄弟节点。这个链接是第3个LI元素的子节点。总之，它们是一个快乐的大家庭。

然而，可能存在更多的子节点。段落、标题、列表项以及链接内部的文本也是由节点组成的，可以回想一下前面的图4-2，虽然它们不是元素，但它们仍然遵从同样的关系规则。

文档中的每个节点都有一些重要的属性：

- 最重要的是nodeType，它描述该节点是什么——元素(element)、属性(attribute)、注释(comment)、文本(text)或其他几种类型(共12个)。在我们的HTML例子中，nodeType的值只有1和3是重要的，因为1表示元素节点而3表示文本节点。

- 另一个重要的性质是nodeName，它表示元素的名字，如果是文本节点的话则表示#text。根据文档类型和用户代理，nodeName可以是大写的，也可以是小写的，正是由于这个原因，在测试特定名字的时候最好将其转成小写。可以使用string对象的toLowerCase()方法来实现：if(obj.nodeName.toLowerCase()=='li'){...}。对于元素节点，可以使用tagName属性。
- nodeValue是节点的值：如果节点是个元素，则它为null；如果元素是文本节点，则它为文本内容。

对于文本节点，可以读取或设置nodeValue，从而能够更改元素的文本内容。举例来说，如果想更改上面例子第一个段落的文本值，你可能认为下面这样对于设置它的nodeValue就足够了：

```
document.getElementsByTagName('p')[0].nodeValue='Hello World';
```

可是，这并不起作用（尽管非常奇怪，它并没有产生错误），因为第一个段落是一个element节点。如果你想更改段落内部的文本，就需要访问它内部的文本节点，换句话说，就是这个段落的第一个子节点。

```
document.getElementsByTagName('p')[0].firstChild.nodeValue='Hello World';
```

4.4.1 从父节点到子节点

这个firstChild属性是个简便用法，每个元素都可以有任意数量的子节点，可以通过childNodes属性列出来。

- childNodes是该元素所有第一层子节点的列表——并不包括向下更深的层次。
- 可以通过数组计数器或item()方法访问当前元素的子元素。
- 简便用法yourElement.firstChild和yourElement.lastChild是yourElement.childNodes[0]和yourElement.childNodes[yourElement.childNodes.length-1]的简化版本，可以使访问更快捷一些。
- 可以通过方法hasChildNodes()检查一个元素是否有子节点，它会返回一个布尔值。

回到上面的例子，你可以按照下面的方式访问UL元素并获取其子节点的信息：

HTML

```
<ul id="eventsList">
  <li>List 1</li>
  <li>List 2</li>
  <li><a href="http://www.google.com">Linked List Item</a></li>
  <li>List 4</li>
</ul>
```

JavaScript

```
function myDOMinspector()
{
  var DOMstring='';
  if(!document.getElementById || !document.createTextNode){return;}
```

```

var demoList=document.getElementById('eventsList');
if (!demoList){return;}
if(demoList.hasChildNodes())
{
    var ch=demoList.childNodes;
    for(var i=0;i<ch.length;i++)
    {
        DOMstring+=ch[i].nodeName+'\n';
    }
    alert(DOMstring);
}
}

```

我们创建一个名为DOMstring的空字符串，检查是否支持DOM以及具有给定id属性的UL元素是否定义过。接着检查这个元素是否存在子节点，如果有，则把它们存在一个名为ch的变量中。循环遍历这个变量（它自动转变为一个数组），将每个子节点的nodeName添加到DOMstring上，后面再跟上一个换行符（\n）。然后使用alert()方法查看输出的结果。

如果在浏览器中运行这个脚本，你会看到IE与其他现代的浏览器最主要的一个区别。IE只显示4个元素，而其他浏览器如Firefox将元素之间的换行符作为文本节点也计算在内了，如图4-10所示：

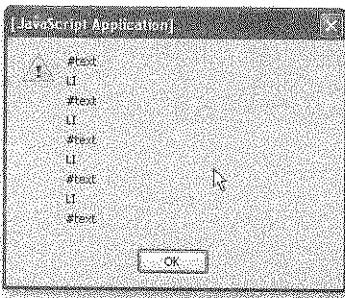


图4-10 脚本查找的节点，包括实际上是换行符的文本节点

4.4.2 从子节点到父节点

通过parentNode属性，可以从子节点回到父节点。首先，可以通过给链接添加一个ID，使我们可以更容易地访问它：

```

<ul id="eventsList">
    <li>List</li>
    <li>List</li>
    <li>
        <a id="linkedItem" href="http://www.google.com">
            Linked List Item
        </a>
    </li>
    <li>List</li>
</ul>

```

现在我们给这个链接对象赋予一个变量，并读出其父节点的名称：

```
var myLinkItem=document.getElementById('linkedItem');
alert(myLinkItem.parentNode.nodeName);
```

结果是LI，如果我们为对象引用再添加一个parentNode，就会得到UL，它是这个链接的祖父元素。

```
alert(myLinkItem.parentNode.parentNode.nodeName);
```

可以添加任意多个父元素——如果在文档树上有父元素，而且还没有到达顶层的话。如果在循环中使用parentNode，那么检查nodeName并在到达BODY后终止循环很重要。举例来说，你想检查一个对象是否在一个具有dynamic类的元素之内，那么可以使用while循环：

```
var myLinkItem = document.getElementById('linkedItem');
var parentElm = myLinkItem.parentNode;
while(parentElm.className != 'dynamic')
{
    parentElm = parentElm.parentNode;
}
```

然而，当没有符合拥有给定类的元素的时候，这个循环会以一个“对象请求”错误中止。如果告诉循环在遍历到主体时停止，就可以避免这个错误：

```
var myLinkItem = document.getElementById('linkedItem');
var parentElm = myLinkItem.parentNode;
while(!parentElm.className != 'dynamic' && parentElm != 'document.body')
{
    parentElm=parentElm.parentNode;
}
alert(parentElm);
```

4.4.3 兄弟节点之间

兄弟节点之间类似于家庭中的兄弟关系，指的是同一年级上的元素（可是它们并不区分性别如兄弟或姐妹）。可以通过一个节点的previousSibling和nextSibling属性访问同一年级上的不同子节点。回到我们列表的例子：

```
<ul id="eventsList">
    <li>List Item 1</li>
    <li>List Item 2</li>
    <li>
        <a id="linkedItem" href="http://www.google.com/">
            Linked List Item
        </a>
    </li>
    <li>List Item 4</li>
</ul>
```

可以通过getElementById()方法获取链接，并通过parentNode获取包含这个链接的LI元素。

previousSibling和nextSibling属性分别可以使你获得List Item 2和List Item 3:

```
var myLinkItem = document.getElementById('linkedItem');
var listItem = myLinkItem.parentNode;
var nextListItem = myLinkItem.nextSibling;
var prevListItem = myLinkItem.previousSibling;
```

注解 这是一个只能在IE上运行的简单例子；由于浏览器实现上的差别，在现代的浏览器中下一个和上一个兄弟并不是LI元素，而是将换行符作为内容的文本节点。

如果当前对象是父节点的最后一个子节点，那么nextSibling就会是undefined，而且如果没有进行适当的检查就会引起一个错误。与childNodes不同，对于第一个兄弟节点和最后一个兄弟没有简便的方法，但你可以写一些实用方法来查找它们。举例来说，你想在上面HTML例子中找到第一个和最后一个LI元素：

```
window.onload=function()
{
    var myLinkItem=document.getElementById('linkedItem');
    var first=firstSibling(myLinkItem.parentNode);
    var last=lastSibling(myLinkItem.parentNode);
    alert(getTextContent(first));
    alert(getTextContent(last));
}

function lastSibling(node){
    var tempObj=node.parentNode.lastChild;
    while(tempObj.nodeType!=1 && tempObj.previousSibling!=null)
    {
        tempObj=tempObj.previousSibling;
    }
    return (tempObj.nodeType==1)?tempObj:false;
}

function firstSibling(node)
{
    var tempObj=node.parentNode.firstChild;
    while(tempObj.nodeType!=1 && tempObj.nextSibling!=null)
    {
        tempObj=tempObj.nextSibling;
    }
    return (tempObj.nodeType==1)?tempObj:false;
}

function getTextContent(node)
{
    return node.firstChild.nodeValue;
}
```

注意需要检查一下nodeType，因为parentNode的第一个和最后一个子节点可能是文本节点而不是一个元素。

我们来使检查日期格式的脚本不那么唐突，通过DOM的方法来提供文本反馈，而不是给用户一个警告信息。首先，需要一个显示错误消息的容器：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
    <meta http-equiv="Content-Type">
        content="text/html; charset=utf-8" />
    <title>Date example</title>
    <style type="text/css">
        .error{color:#c00;font-weight:bold;}</style>
    </style>
    <script type="text/javascript" src="checkdate.js"></script>
</head>
<body>
    <h1>Events search</h1>
    <form action="eventssearch.php" method="post">
        onsubmit="return checkDate();">
            <p>
                <label for="date">Date in the format DD/MM/YYYY:</label><br />
                <input type="text" id="date" name="date" />
                <input type="submit" value="Check " />
                <br />(example 26/04/1975) <span class="error"> </span>
            </p>
        </form>
    </body>
</html>
```

这个检查脚本和以前的基本差不多——不同之处在于我们使用SPAN作为一种显示错误的手段：

```
function checkDate(){
    if(!document.getElementById || !document.createTextNode){return;}
    var dateField=document.getElementById('date');
    if(!dateField){return;}
    var errorContainer=dateField.parentNode.getElementsByName('span')[0];
    if(!errorContainer){return;}
    var checkPattern=new RegExp("\d{2}/\d{2}/\d{4}");
    var errorMessage='';
    errorContainer.firstChild.nodeValue=' ';
    var dateValue=dateField.value;
    if(dateValue=='')
    {
        errorMessage='Please provide a date.';
    }
    else if(!checkPattern.test(dateValue))
    {
        errorMessage='Please provide the date in the defined format.';
    }
```

```

if(errorMessage!="")
{
    errorContainer.firstChild.nodeValue=errorMessage;
    dateField.focus();
    return false;
}
else
{
    return true;
}
}

```

首先，检查是否支持DOM以及所有需要的元素是否存在：

```

if(!document.getElementById || !document.createTextNode){return;}
var dateField=document.getElementById('date');
if(!dateField){return;}
var errorContainer=dateField.parentNode.getElementsByTagName('span')[0];
if(!errorContainer){return;}

```

接着定义测试的模式和一个空的错误信息。设置错误Span的文本值为一个单空格，这是非常必要的，可以避免用户在没有修改错误输入并且再次提交表单时，显示多个错误信息。

```

var checkPattern=new RegExp("\d{2}/\d{2}/\d{4}");
var errorMessage=' ';
errorContainer.firstChild.nodeValue=' ';

```

然后是域的校验。读取日期域的值并检查是否有输入。如果没有，错误消息显示用户应该输入一个日期。如果有一个日期，但它的格式是错误的，错误消息将指出该错误。

```

var dateValue=dateField.value;
if(dateValue=='')
{
    errorMessage='Please provide a date.';
}
else if(!checkPattern.test(dateValue))
{
    errorMessage='Please provide the date in the defined format.';
}

```

现在剩下的事情就是检查一下初始为空的错误信息是否被修改了。如果它没有被修改，脚本应该向表单返回true; onsubmit="return checkDate();"—这样就提交了表单，可以让后台接手这个工作。如果错误信息被修改了，脚本就将错误信息添加到错误SPAN内的文本内容中（第一个子节点的nodeValue），并且不提交表单，把文档的焦点设置回日期输入域上。

```

if(errorMessage!="")
{
    errorContainer.firstChild.nodeValue+=errorMessage;
    dateField.focus();
    return false;
}

```

```

    }
else
{
    return true;
}

```

如图4-11所示，最终结果看起来比那个警告消息更加吸引人，并且可以随意设置它的样式。

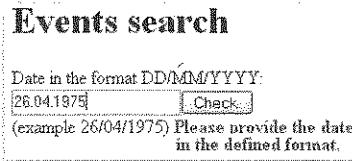


图4-11 显示一个动态错误信息

现在你已经知道了如何访问和修改现存元素的文本值。但是，如果需要修改其他属性或是需要没有为你提供的HTML，那么该怎么办呢？

4.5 修改元素属性

一旦获得了你要修改的元素，可以有2种方式来读取和修改它的属性：一种老的方式（它被更多的用户代理所支持）和一种使用新的DOM方法的方式。

老的和新的用户代理都允许你以对象属性的方式获取和设置元素的属性：

```

var firstLink=document.getElementsByTagName('a')[0];
if(firstLink.href=='search.html')
{
    firstLink.href='http://www.google.com';
}
var mainImage=document.getElementById('nav').getElementsByTagName('img')[0];
mainImage.src='dynamiclogo.gif';
mainImage.alt='Generico Corporation - We do generic stuff';
mainImage.title='Go back to Home';

```

在HTML规范中定义的所有属性都是可用的而且可以被访问。一些属性由于安全的原因是只读的，但大多数都是可以设置和读取的。也可以提出自己的属性——JavaScript是不会介意的。有时把值存储在元素的属性中可以节省大量的测试和循环。

警告 小心那些与JavaScript命令有着相同名称的属性——例如for。试图设置element.for='something'将会导致错误。浏览器开发商为此提出了一些权宜之计；对于for的这种情况（它是label元素的有效属性），规定其属性名为htmlFor。更奇的是class属性，第5章会大量使用它。这是一个保留字，需要使用className来代替。

DOM规范提供了2种方法来读取和设置属性值——`getAttribute()`和`setAttribute()`。`getAttribute()`方法有一个参数——属性名；`setAttribute()`方法需要2个参数——属性名和新的属性值。

上面的例子使用了新方法后将会是这样：

```
var firstLink=document.getElementsByTagName('a')[0];
if(firstLink.getAttribute('href') =='search.html')
{
    firstLink.setAttribute('href') ='http://www.google.com';
}
var mainImage=document.getElementById('nav').getElementsByTagName('img')[0];
mainImage.setAttribute('src') ='dynamiclogo.gif';
mainImage.getAttribute('alt') ='Generico Corporation - We do generic stuff';
mainImage.getAttribute('title') ='Go back to Home';
```

这样看起来有些臃肿，但好处是这更贴近其他更高级的编程语言。比起直接为元素指定属性值，这样更可能被未来的用户代理所支持，并且它们可以很容易地处理任意的属性名。

4.6 创建、移除和替换元素

在HTML/JavaScript环境中，DOM也提供了修改文档结构的方法（通过JavaScript进行XML转换的方法更多）。不只可以修改已存在的元素，还可以创建新元素、移除和替换旧元素。这些方法如下。

- `document.createElement('element')`: 创建一个标签名为`element`的新元素。
- `document.createTextNode('string')`: 创建一个节点值为`string`的文本节点。
- `node.appendChild(newNode)`: 将`newNode`作为子节点，添加在`node`的所有子节点之后。
- `newNode=node.cloneNode(bool)`: 创建`newNode`节点作为`node`的副本（克隆）。如果`bool`值为`true`，这个克隆就包括所有原节点的子节点及其属性的克隆。
- `node.insertBefore(newNode,oldNode)`: 把`newNode`作为一个`node`的新节点插入到`oldNode`之前。
- `node.removeChild(oldNode)`: 移除`node`节点的子节点`oldNode`。
- `node.replaceChild(newNode, oldNode)`: 使用节点`newNode`替换`node`节点的子节点`oldNode`。

注解 `createElement()`和`createTextNode()`都是`document`对象的方法，所有其他的都是节点的方法。

当你想要创建有JavaScript增强的Web产品时，所有的这些都是必不可少的，但不要完全依赖它。除非你向用户反馈的所有信息都是通过警告、确认、提示这些弹出信息实现的，否则你会不得不依赖于给定的HTML元素——例如前面例子中的错误信息SPAN。然而，由于JavaScript运行所需要的HTML只有在JavaScript可用的前提下才有意义，因此当不支持脚本的时候，它就不会是可

用的。一个多余的SPAN并不会有多大伤害——然而，提供给用户许多功能（如日期选择工具）的表单控件不能工作却是一个问题。

我们来用这些方法解决创建Web应用时常见的一个问题：使用链接替换提交按钮。有2种方式可以提交表单数据：提交按钮或一个图片按钮（事实上，有3种——如果把单击回车键也算上）。

提交按钮对很多设计师来说都是非常头痛的，因为在不同的操作系统和浏览器下，无法使它具有一致的样式。图片按钮也有一个问题，因为在站点上定位的时候需要大量的工作，并且当访问者使用较大字体设置的时候，它们不会调整大小（除非使用em定义它们的大小，但它又会导致很难看的像素化问题）。

链接就好多了，因为可以通过CSS设置它们的样式，调整大小，很容易地从本地化的数据集中进行组装。使用链接的问题是，需要使用JavaScript提交相关的表单。这也是比较懒或繁忙的开发人员要使用一个链接来解决这个两难问题的原因，它使用javascript:协议（许多代码生成器和框架都会提出类似的东西）来提交表单：

```
<a href="javascript:document.forms[0].submit()">Submit</a>
```

在本书前面我说过，这并不是一个好的选择——因为没有启用或支持JavaScript时，这将成为一个死链接，并导致没有办法提交表单。如果想使这两种状态都好用——为非JavaScript用户提供简单的提交按钮，为启用了脚本的用户提供链接——你需要这样做：

- (1) 循环遍历文档中所有INPUT元素。
- (2) 检查type值是否为submit。
- (3) 如果不是的话，continue（继续）这个循环，跳过其余的代码。
- (4) 如果是的话，创建一个新的文本节点链接。
- (5) 将文本节点的值设置为INPUT元素的值。
- (6) 设置链接的href为javascript:document.forms[0].submit()。
- (7) 使用链接替换input元素。

注解 设置href属性值为javascript:，这并不是实现此功能最巧妙的方式。在第5章中，你会学到事件处理程序——实现这个解决方案更好的方式。

代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Example: Submit buttons to links</title>
    <style type="text/css"></style>
    <script type="text/javascript" src="submitToLinks.js"></script>
  </head>
```

```

<body>
  <form action="nogo.php" method="post">
    <label for="Name">Name:</label>
    <input type="text" id="Name" name="Name" />
    <input type="submit" value="send" />
  </form>
</body>
</html>
function submitToLinks()
{
  if(!document.getElementById || !document.createTextNode){return;}
  var inputs,i,newLink,newText;
  inputs=document.getElementsByTagName('input');
  for (i=0;i<inputs.length;i++)
  {
    if(inputs[i].getAttribute('type').toLowerCase()!='submit')➥
      {continue;i++}
    newLink=document.createElement('a');
    newText=document.createTextNode(inputs[i].getAttribute('value'));
    newLink.appendChild(newText);
    newLink.setAttribute('href','javascript:document.forms[0]➥
      .submit()');
    inputs[i].parentNode.replaceChild(newLink,inputs[i]);
  }
}
window.onload=submitToLinks;

```

当JavaScript可用的时候，用户得到一个可以提交表单的链接；否则会得到一个提交按钮，如图4-12所示。

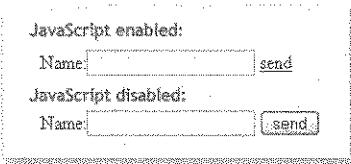


图4-12 根据JavaScript的可用性，用户会得到一个可以提交表单的链接或按钮

然而，这个函数有个主要的缺点：在提交按钮之后有多个输入元素时，函数执行就会失败。修改一下HTML，在提交按钮之后增加一个输入元素：

```

<form action="nogo.php" method="post">
  <p>
    <label for="Name">Name:</label>
    <input type="text" id="Name" name="Name" />
    <input type="submit" value="check" />
    <input type="submit" value="send" />
  </p>
  <p>
    <label for="Email">email:</label>
  
```

```

<input type="text" id="Email" name="Email" />
</p>
</form>
```

你会发现“send”提交按钮并没有被链接所替换。这是因为你移除了一个输入元素，它改变了数组的长度，使循环没能访问到它应该访问到的元素。这个问题的修改办法是，每移除一个元素后都将循环计数器减1。然而，你需要通过比较循环计数器和数组的长度以检查是否为最后一次循环（直接给计数器减1会导致脚本失败，因为它试着访问一个并不存在的元素）。

```

function submitToLinks()
{
    if(!document.getElementById || !document.createTextNode){return;}
    var inputs,i,newLink,newText;
    inputs=document.getElementsByTagName('input');
    for (i=0;i<inputs.length;i++)
    {
        if(inputs[i].getAttribute('type').toLowerCase()!='submit')➥
            {continue;i++}
        newLink=document.createElement('a');
        newText=document.createTextNode(inputs[i].getAttribute('value'));
        newLink.appendChild(newText);
        newLink.setAttribute('href','javascript:document.forms[0]➥
            .submit()');
        inputs[i].parentNode.replaceChild(newLink,inputs[i]);
        if(i<inputs.length){i--};
    }
}
window.onload=submitToLinks;
```

这个版本没有问题，它将会用链接替换掉这两个按钮。

注解 这个脚本会破坏表单的可用性：当有提交按钮的时候，可以通过按回车键来提交表单。当我们移除所有提交按钮的时候，这就不能再用了。解决办法是添加一个空的图片按钮或是将提交按钮隐藏而不要移除。第5章我们会回到这个话题。另一个可用性涉及你是否应该完全更改表单的界面外观——因为将无法马上识别出表单元素。用户已经习惯了在其操作系统和浏览器上的表单风格——如果你更改了，他们不得不去寻找可交互的元素，并可能期望其他的功能。在私人数据和金融交易方面，人们都相信表单——任何含混的东西都很可能被认为是安全问题。

4.6.1 避免NOSCRIPT

SCRIPT元素有一个对立的NOSCRIPT。这个元素最初的目的就是在JavaScript不可用的时候为用户提供可选择的内容。语义HTML^①并不赞成将脚本块嵌入到文档当中（主体当中应只包括有关文档结构的东西，而SCRIPT并不是）；这样NOSCRIPT也遭到了废弃。可是，你会发现网上有大量的有

① 语义（Semantic）HTML指正确布局的(X)HTML标记。——编者注

关于可访问性的指南，它们都赞成使用NOSCRIPT作为一个保险的措施。它只是通过NOSCRIPT标签向网页内添加一条简单的消息，来提示如果想要完美地使用这个网站就需要使用JavaScript，这看起来是解决问题的一个非常简单的方法。因为如此，许多开发人员感到不解，为什么W3C不赞成NOSCRIPT或为什么要为这个原因就牺牲HTML的有效性。然而，通过使用DOM的方法，你可以解决这个问题。

理想状态下，没有任何网站的运行必须需要JavaScript——只是有些时候使用脚本可以使网站运行起来更快或是操作起来更方便。可是，在现实世界中，有时你不得不使用一些现成的产品或框架（它们能够生成依赖于脚本支持的代码）。当无法重新设计或替换易用性差的老系统时，你需要告诉用户此网站运行需要脚本支持。

使用NOSCRIPT，这很容易做到：

```
<script type="text/javascript">myGreatApplication();</script>
<noscript>
    Sorry but you need to have scripting enabled to use this site.
</noscript>
```

这样一条信息帮助不是太大——至少你应该允许那些不能使用脚本的用户（例如，银行或财务公司的工作人员，他们会由于脚本的安全隐患而将脚本关闭）可以与你联系。

现代的脚本使用另外一种方法解决这个问题：给出一些信息，在脚本可用的时候就替换它。在依赖于脚本支持的应用程序中，可能会是这样的：

```
<p id="noscripting">
    We are sorry, but this application needs JavaScript
    to be enabled to work. Please <a href="contact.html">contact us</a>
    If you cannot enable scripting and we will try to help you in other
    ways.
</p>
```

然后，编写一段简单的脚本将其移除，同时还可以检查是否支持DOM：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Example: Replacing noscript</title>
    <script type="text/javascript">
        function noscript()
        {
            if(!document.getElementById || !
                !document.createTextNode){return;}
            // Add more tests as needed (cookies, objects...)
            var noJSmsg=document.getElementById('noscripting');
            if(!noJSmsg){return;}
            var headline='Browser test succeeded';
            replaceMessage='We tested if your browser is capable of ';
```

```

replaceMessage+='supporting the application, and all checked';
out fine. ';
replaceMessage+='Please proceed by activating the following';
link.';

var linkMessage='Proceed to application.';
var head=document.createElement('hi');
head.appendChild(document.createTextNode(headline));
noJSmsg.parentNode.insertBefore(head,noJSmsg);
var infoPara=document.createElement('p');
infoPara.appendChild(document.createTextNode(replaceMessage));
noJSmsg.parentNode.insertBefore(infoPara,noJSmsg);
var linkPara=document.createElement('p');
var appLink=document.createElement('a');
appLink.setAttribute('href','application.aspx');
appLink.appendChild(document.createTextNode(linkMessage));
linkPara.appendChild(appLink);
noJSmsg.parentNode.replaceChild(linkPara,noJSmsg);
}

window.onload=noscript;
</script>
</head>
<body>
<p id="noscript">
    We are sorry, but this application needs JavaScript to be
    enabled to work. Please <a href="contact.html">contact us</a>
    if you cannot enable scripting and we will try to help you in
    other ways
</p>
</body>
</html>

```

可以看到，通过DOM生成大量的内容是相当麻烦的，这种情况下并不需要把生成的每个节点都作为一个变量，很多开发人员都使用innerHTML来代替。

4.6.2 通过innerHTML简化脚本

微软在早期的IE开发中就实现了innerHTML方法，现在大多数浏览器都对它提供支持，甚至有人建议将其添加到DOM标准当中。它允许你定义一个包含HTML的字符串，并将其赋给一个对象。用户代理接着会帮你做剩下的事情：所有节点的生成和子节点的添加。使用innerHTML的NOSCRIPT例子会简短许多：

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
<meta http-equiv="Content-Type">
    content="text/html; charset=utf-8">
<title>Example: Replacing noscript</title>
<script type="text/javascript">

```

```

function noscript()
{
    if(!document.getElementById || !document.createTextNode){return;}
    // Add more tests as needed (cookies, objects...)
    var noJSmsg=document.getElementById('noscripting');
    if(!noJSmsg){return;}
    var replaceMessage='<h1>Browser test succeeded</h1>';
    replaceMessage='<p>We tested if your browser is capable of ';
    replaceMessage+='supporting the application, and all checked';
    out fine. ';
    replaceMessage+='Please proceed by activating the following';
    link.</p>';
    replaceMessage+='<p><a href="application.aspx">';
    Proceed to application</a></p>';
    noJSmsg.innerHTML=replaceMessage;
}
</script>
</head>
<body>
<p id="noscripting">
    We are sorry, but this application needs JavaScript to be
    enabled to work. Please <a href="contact.html">contact us</a>
    if you cannot enable scripting and we will try to help you in
    other ways.
</p>
</body>
</html>

```

也可以读出一个元素的innerHTML属性，这在调试代码的时候会非常方便——因为不是所有浏览器都有“查看源文件”的功能。使用其他HTML替换整篇的HTML也非常容易，当我们显示使用Ajax从后台获取的数据时，我们会经常这样做。

4.6.3 DOM 小结：你的备忘单

许多东西都需要理解，而且把你需要的DOM特性放到一块可以方便复制和随时使用，那么我们从这里开始吧！

1. 访问文档中元素

- ❑ document.getElementById('id'): 获取给定id的元素，并将其作为对象。
- ❑ document.getElementsByTagName('tagname'): 获取所有标签名为tagname的元素，并把它保存在一个类似数组的列表中。

2. 读取元素的属性、节点值及其他节点数据

- ❑ node.getAttribute('attribute'): 获取属性名为attribute的值。
- ❑ node.setAttribute('attribute', 'value'): 设置属性名为attribute的值为value。
- ❑ node.nodeType: 读取节点类型 (1=元素, 3=文本节点)。

- node.nodeName: 读取节点名称(元素名字或#textNode)。
- node.nodeValue: 读取或设置节点的值(文本节点的情况下则为文本内容)。

3. 节点之间操作

- node.previousSibling: 获取上一个兄弟节点，并将它保存为一个对象。
- node.nextSibling: 获取下一个兄弟节点，并将它保存为一个对象。
- node.childNodes: 获取对象的所有子节点，并把它们存储到一个列表中。对于第一个和最后一个子节点，还可以使用node.firstChild和node.lastChild进行简写。
- node.parentNode: 获取包含node的节点。

4. 创建新节点

- document.createElement(element): 创建一个名字为element的新元素，需要提供一个字符串形式的元素名。
- document.createTextNode(string): 创建一个节点值为string的文本节点。
- newNode = node.cloneNode(bool): 创建newNode节点作为node的副本(克隆)。如果bool值为true，这个克隆将包括原节点的所有子节点和属性的克隆。
- node.appendChild(newNode): 将newNode作为子节点，添加在node所有子节点之后。
- node.insertBefore(newNode, oldNode): 在node节点的子节点oldNode之前插入newNode。
- node.removeChild(oldNode): 移除node节点的子节点oldNode。
- node.replaceChild(newNode, oldNode): 使用节点newNode替换node节点的子节点oldNode。
- element.innerHTML: 读写给定element的HTML内容，它是一个字符串，包括所有子节点及它们的属性和文本内容。

4.6.4 DOMhelp: 我们自己的辅助函数库

使用DOM最令人厌烦的事情就是浏览器的不一致性，尤其是在每次使用nextSibling的时候都要检查一下nodeType，因为用户代理可能会也可能不会把换行符作为它自己的文本节点。

因此用一组方便的工具函数来解决这些问题，使你能够专注于主要脚本的逻辑将是一个非常好的主意。在网上有许多可用的JavaScript框架和库，最大的而且经常更新的可能就是Prototype了(<http://prototype.js.org/>)。

这里我们来开始编写自己的辅助方法函数库，并顺带说明在不使用这个函数库的情况下必须面对的问题。

注解 在本书附带代码演示zip文件中你可以找到DOMhelp.js文件和一个测试用的HTML文件——压缩文件中的版本有许多方法，我们会在第5章中进行讨论，所以不要感到太困惑。

该库是由一个名为DOMhelp的对象和一些实用的方法组成。下面是它的骨架，我们会在本章和第5章中对其进行充实：

```

DOMhelp=
{
    // Find the last sibling of the current node
    lastSibling:function(node){},
    // Find the first sibling of the current node
    firstSibling:function(node){},
    // Retrieve the content of the first text node sibling of the current node
    getText:function(node){},
    // Set the content of the first text node sibling of the current node
    setText:function(node,txt){},
    // Find the next or previous sibling that is an element
    // and not a text node or line break
    closestSibling:function(node,direction){},
    // Create a new link containing the given text
    createLink:function(to,txt){},
    // Create a new element containing the given text
    createTextElm:function(elm,txt){},
    // Simulate a debugging console to avoid the need for alerts
    initDebug:function(){},
    setDebug:function(bug){},
    stopDebug:function(){}
}

```

在本章中你已经在前面遇到了有关第一个和最后一个兄弟节点的函数；唯一的问题是在那些函数中，在为临时对象赋值前没有检查是否真的存在上一个或下一个兄弟节点。这里的两个方法都检查了兄弟节点的存在，如果不可用的时候会返回false：

```

lastSibling:function(node)
{
    var tempObj=node.parentNode.lastChild;
    while(tempObj.nodeType!=1 && tempObj.previousSibling!=null)
    {
        tempObj=tempObj.previousSibling;
    }
    return (tempObj.nodeType==1)?tempObj:false;
},
firstSibling:function(node)
{
    var tempObj=node.parentNode.firstChild;
    while(tempObj.nodeType!=1 && tempObj.nextSibling!=null)
    {
        tempObj=tempObj.nextSibling;
    }
    return (tempObj.nodeType==1)?tempObj:false;
},

```

接下来是getText方法，它读出元素第一个文本节点的文本值：

```
getText:function(node)
{
    if(!node.hasChildNodes()){return false;}
    var reg=/^\s+$/;
    var tempObj=node.firstChild;
    while(tempObj.nodeType!=3 && tempObj.nextSibling!=null || ▶
        reg.test(tempObj.nodeValue))
    {
        tempObj=tempObj.nextSibling;
    }
    return tempObj.nodeType==3?tempObj.nodeValue:false;
},
```

我们可能遇到的第一个问题是该节点不包含子节点，所以检查一下hasChildNodes()。另一个问题是节点内所嵌套的元素和空白字符（如换行符或tab空格）被除IE以外的浏览器当作节点读取。所以，我们从一个节点跳到下一个节点，直到nodeType为text(3)，并且该节点不是完全由空白字符组成的（由正则表达式检查）。在将下一个兄弟节点赋给tempObj之前，我们也检查了是否存在下一个兄弟节点。如果这些都满足，该方法返回第一个文本节点的nodeValue；否则它返回false。

setText函数需要同样的模式作检查，它使用新文本替换节点的第一个文本子节点，它需要避免换行符和制表符：

```
setText:function(node,txt)
{
    if(!node.hasChildNodes()){return false;}
    var reg=/^\s+$/;
    var tempObj=node.firstChild;
    while(tempObj.nodeType!=3 && tempObj.nextSibling!=null || ▶
        reg.test(tempObj.nodeValue))
    {
        tempObj=tempObj.nextSibling;
    }
    if(tempObj.nodeType==3){tempObj.nodeValue=txt}else{return false;}
},
```

下面两个方法帮助我们处理创建一个链接时比较常见的问题，这个链接包含一个目标和其中的文本，我们使用其中的文本创建一个元素：

```
createLink:function(to,txt)
{
    var tempObj=document.createElement('a');
    tempObj.appendChild(document.createTextNode(txt));
    tempObj.setAttribute('href',to);
    return tempObj;
},
createTextElm:function(elm,txt)
{
```

```

var tempObj=document.createElement(elm);
tempObj.appendChild(document.createTextNode(txt));
return tempObj;
},

```

你在这里已看到它们在前面的例子中并不起任何作用，但在有的地方它们却是非常方便的。事实上，一些浏览器会将换行符看作文本节点，而其他的则不这样，这意味着你不能完全相信nextSibling或previousSibling会为你返回的下一个元素——例如，在一个无序的列表当中。实用方法closestSibling()可以解决这个问题。它需要node和direction（1表示下一个兄弟节点，-1表示上一个兄弟节点）两个参数。

```

closestSibling:function(node,direction)
{
    var tempObj;
    if(direction== -1 && node.previousSibling!=null)
    {
        tempObj=node.previousSibling;
        while(tempObj.nodeType!=1 && tempObj.previousSibling!=null)
        {
            tempObj=tempObj.previousSibling;
        }
    }
    else if(direction==1 && node.nextSibling!=null)
    {
        tempObj=node.nextSibling;
        while(tempObj.nodeType!=1 && tempObj.nextSibling!=null)
        {
            tempObj=tempObj.nextSibling;
        }
    }
    return tempObj.nodeType==1?tempObj:false;
},

```

最后几个方法是模拟一个JavaScript编程调试控制台。使用alert()作为显示值的一种方法固然方便，但是如果你想在一个相当大的循环中观察值的变化情况，这就变得非常痛苦了——谁愿意按上200次回车呢？不使用alert()，我们可以向文档添加新的DIV元素，将我们想要检查的数据作为其新的子节点输出。通过适当的样式表，我们可以让DIV飘浮在内容之上。开始是一个初始化方法，它检查控制台是否已存在，如果存在则将其移除。要避免同时存在几个控制台这是必需的。接着创建一个DIV元素，给它一个ID来调整样式，并将其添加到文档。

```

initDebug:function()
{
    if(DOMhelp.debug){DOMhelp.stopDebug();}
    DOMhelp.debug=document.createElement('div');
    DOMhelp.debug.setAttribute('id',DOMhelp.debugWindowId);
    document.body.insertBefore(DOMhelp.debug,document.body.firstChild);
},

```

setDebug方法接受一个字符串参数bug。它会检查控制台是否存在，如果需要，它将调用初始化方法来创建控制台。然后在bug字符串后面加上换行符并将它添加到控制台的HTML内容中。

```
setDebug:function(bug)
{
    if(!DOMhelp.debug){DOMhelp.initDebug();}
    DOMhelp.debug.innerHTML+=bug+'\n';
},
```

最后一个方法是将控制台从文档中移除，如果它存在的话。注意，我们既需要移除元素，还需要把对象属性设为null；否则的话，即使没有控制台可以写入，DOMhelp.debug的值也会是true。

```
stopDebug:function()
{
    if(DOMhelp.debug)
    {
        DOMhelp.debug.parentNode.removeChild(DOMhelp.debug);
        DOMhelp.debug=null;
    }
}
```

我们会在第5章继续扩展这个辅助函数库。

4.7 小结

读过本章之后，你应该可以处理任何HTML文档了，可以获取你需要的部分，并通过DOM修改甚至创建新的标记。

你已经了解了HTML文档的内部结构，以及DOM如何获取文档的元素、属性和文本节点的集合，也学到了Window方法alert()、confirm()和prompt()，这些都很容易使用并得到了广泛的支持，虽然不太安全且略显笨拙，但毕竟是获取数据和提供反馈信息的方法。

接着学习了DOM，以及如何访问元素，如何在元素之间进行导航，还有如何创建新内容。

第5章将介绍如何处理一些表现层的问题，如何跟踪用户如何与其浏览器中的文档进行交互，并通过事件处理程序做出相应的处理。

表现与行为（CSS与事件处理）

第4章对HTML文档进行了剖析。还开发了一个原始的库。现在我们该看一下如何使用CSS来为文档增加一种新的显示效果，并使用事件来驱动它。如果你也喜欢美的东西，那么请打起精神，因为我们要开始学习表现层了。

5.1 通过JavaScript改变表现层

HTML文档中的每个元素都有一个style属性，它是元素本身的所有外观属性的一个集合。可以读写属性的值，而且如果你把一个值写到这个属性中，就会立即改变这个元素的界面外观。

注解 注意我们在本章中会经常使用前一章中创建的DOMhelp库（当然，在本书的其他章节中也会用到）。

首先试一下这个脚本：

exampleStyleChange.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Example: Accessing the style collection</title>
  <style type="text/css">
  </style>
  <script type="text/javascript" src="DOMhelp.js"></script>
  <script type="text/javascript" src="styleChange.js"></script>
</head>
<body>
<h3>Contact Details</h3>
<address>
  Awesome Web Production Company<br />
```

```

Going Nowhere Lane 0<br />
Catch 22<br />
N4 2XX<br />
England<br />
</address>
</body>
</html>

styleChange.js

sc = {
  init:function(){
    sc.head = document.getElementsByTagName('h3')[0];
    if(!sc.head){return;}
    sc.ad = DOMhelp.closestSibling(sc.head,1);
    sc.ad.style.display='none';
    var t = DOMhelp.getText(sc.head);
    var collapseLink = DOMhelp.createLink('#',t);
    sc.head.replaceChild(collapseLink,sc.head.firstChild);
    DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false)
    collapseLink.onclick = function(){return;} // Safari fix
  },
  peekaboo:function(e){
    sc.ad.style.display=sc.ad.style.display=='none'?'none':'';
    DOMhelp.cancelClick(e);
  }
}
DOMhelp.addEvent(window,'load',sc.init,false);

```

注解 请稍微耐心一些，`addEvent()`和`cancelClick()`部分会在5.2节中讲解。现在，集中关注一下上面脚本的黑体字部分。

上面的脚本首先获取了文档中的第一个H3元素，并通过Domhelp库中的辅助方法`closestSibling`得到ADDRESS元素（这个方法确保它可以获得更多一个元素，并且换行不会被当作文本节点）。接下来它修改了style集合中的display属性以隐藏地址。它还使用了一个指向函数`peekaboo`的链接替换了标题里的文本。这个链接是键盘用户展开或折叠地址信息所必需的。虽然鼠标用户可以很容易地单击这个标题，但使用`tab`键来遍历整个文档就不可理解了。`peekaboo()`函数会读取地址style集合的display属性值，如果display属性被设置为`none`，则使用一个空字符串替换它；当display属性被设置为其他的时候，则不是使用空字符串，而使用`none`替换它。这样可以有效地隐藏和显示地址，如图5-1所示：

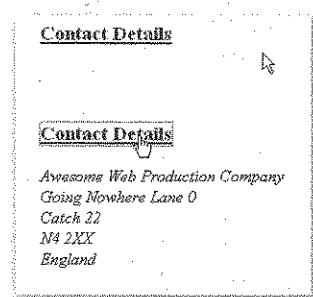


图5-1 地址的两种状态（折叠和展开）

注解 你以前可能碰到过一些脚本，它们使用`element.style.display='block'`达到与`none`相反的效果。它对大多数元素都起作用，但只简单地设置显示值为空会把它重置为初始的显示值——它并不必须是`block`；它可以是`inline`或`table-row`。如果你添加了一个空字符串，就会把问题留给浏览器，它会设置一个合适的值；否则你不得不为不同的元素添加一个`switch`块或多个`if`条件。

样式集合包含了当前元素的所有样式设置，可以使用各种CSS选择器属性符号修改。属性符号一般去掉CSS选择器中的连字符并应用骆驼命名法即可得到。例如，`line-height`变成为`lineHeight`，`border-right`变成为`borderRight`。下面是所有可用属性的列表（注意一下`float`要变为`cssFloat`）：

- `background`、`backgroundAttachment`、`backgroundColor`、`backgroundImage`、`backgroundPosition`和`backgroundRepeat`；
- `border`、`borderBottom`、`borderTop`、`borderLeft`、`borderRight`及它们的`style`、`width`和`color`；
- `color`、`direction`、`display`、`visibility`、`letterSpacing`、`lineHeight`、`textAlign`、`textDecoration`、`textIndent`、`textTransform`、`wordSpacing`、`letterSpacing`；
- `margin`、`padding`（及它们的`top`、`left`、`bottom`和`right`）；
- `width`、`height`、`minWidth`、`maxWidth`、`minHeight`、`maxHeight`；
- `captionSide`、`emptyCells`、`tableLayout`、`verticalAlign`；
- `top`、`bottom`、`left`、`right`、`zIndex`、`cssFloat`、`position`、`overflow`、`clip`、`clear`；
- `listStyle`、`listStyleImage`、`listStylePosition`、`listStyleType`；
- `font`、`fontFamily`、`fontSize`、`fontStretch`、`fontStyle`、`fontVariant`、`fontWeight`。

可以使用`getAttribute()`和`setAttribute()`来读写所有的这些属性；可是，如果要设置这些属性，使用JavaScript对象的属性语法将元素的`style`属性设置为一个字符串值，可能更快捷。对于浏览器来说，下面的两个示例是一样的，但是后一种可能会显示得快一点，而且可以使你的JavaScript代码更简短。例如，下面的代码：

```

var warning=document.createElement('div');

warning.style.borderColor='#c00';
warning.style.borderWidth='1px';
warning.style.borderStyle='solid';
warning.style.backgroundColor='#fcc';
warning.style.padding='5px';
warning.style.color='#c00';
warning.style.fontFamily='Arial';

// 与下面的等同：
warning.setAttribute( 'style' , 'font-family:arial;color:#c00;
padding:5px;border:1px solid #c00;background:#fcc' );

```

虽然在现代Web设计中不赞成直接设置style属性（这样会把行为和表现混合在一起，而且使维护更困难），但是有些情况下你不得不使用JavaScript直接设置style属性的值，例如：

- 弥补浏览器对CSS支持不够的缺点；
- 动态地改变元素的尺寸大小来修改布局上的小问题；
- 显示网页文档的动画部分；
- 创建具有拖放功能的富用户界面。

注解 本章稍后会介绍有关前两点；可是，将不会讲述有关动画和拖放的示例，因为它们属于高级的JavaScript主题，需要许多超出本书范围的解释。在11章中你会读到一些使用现成库的示例。

对于简单的样式设置，为了简化脚本的维护，你应该避免在JavaScript中定义界面外观。在第3章中，我们已讨论了现代Web设计的主要特性：开发中不同层之间的分离。

如果在JavaScript中使用了许多样式定义，那么就混淆了表现层和行为层。如果几个月后应用程序的界面外观发生了变化，你或者一些第三方开发人员就不得不找出这些脚本代码并修改其中的所有设置。这既无必要，也不可取，因为你可以把界面外观的代码分离出来放到CSS文档中。

可以通过动态地改变网页元素的class属性来实现这种分离。通过这种方式，可以应用或移除在站点的样式表中定义的样式设置。CSS设计师不需要关注你的脚本代码，而且你也没有必要知道所有的浏览器对CSS是否支持的问题。你需要知道的就是这些样式类的名字。

例如，把一个叫做dynamic的类应用到一个ID为nav的元素上，可以改变它的className属性：

```
var n=document.getElementById('nav');  
n.className='dynamic';
```

注解 逻辑上，你也可以通过setAttribute()方法来改变类，但是浏览器对它的支持上是很奇怪的（IE不允许使用class或style作为属性），这就是当时坚持使用className的原因。属性的名字是className而不是class，因为class在JavaScript中是一个保留字，把它作为一个属性使用的时候会导致错误。

可以通过把className的值设置为一个空字符串来移除类。再强调一下，removeAttribute()在不同的浏览器上并不会可靠地工作。

你可能已经注意到，HTML元素可以拥有不止一个CSS类。下面这种构造是一个有效的HTML，有时它还是一种很不错的方法：

```
<p class="intro special kids">Lorem Ipsum</p>
```

在JavaScript中，可以通过把空格开头的值添加到className的值前来实现同样的功能。可是，这样存在一个危险，就是浏览器不会正确地显示你设置的类，尤其是当添加或移除导致在

className值的开始或结尾处为空格的时候。下面的两个例子在一些浏览器中就不能正确地显示出来（尤其是在Mac平台的IE 5上面）：

```
<p class="intro special kids ">Lorem Ipsum</p>
<p class=" intro special kids">Lorem Ipsum</p>
```

可以使用一种辅助方法来解决这个问题。编写这个动态添加和删除类的辅助方法应该很容易：如果className属性不为空，则在它之前加上一个空格开头的类值；如果为空，则不加空格。从初始值中移除类名就如同从一个字符串中移除一个字。然而，由于你需要解决这个单独空格的浏览器问题，所以它会比这个更复杂一些。下面的工具方法包含在DOMhelp中，它可以用动态地添加和移除一个元素的类。它还可以检查某个类是否已被添加到它上面了。

```
function cssjs(a,o,c1,c2){
    switch (a){
        case 'swap':
            if(!domtab.cssjs('check',o,c1)){
                o.className.replace(c2,c1)
            }else{
                o.className.replace(c1,c2);
            }
            break;
        case 'add':
            if(!domtab.cssjs('check',o,c1)){
                o.className+=o.className?' '+c1:c1;
            }
            break;
        case 'remove':
            var rep=o.className.match(' '+c1)?' '+c1:c1;
            o.className=o.className.replace(rep,'');
            break;
        case 'check':
            var found=false;
            var temparray=o.className.split(' ');
            for(var i=0;i<temparray.length;i++){
                if(temparray[i]==c1){found=true;}
            }
            return found;
            break;
    }
}
```

不要太关注这个方法的内部工作机制，只要掌握了match()和replace()方法（这两个方法会在第8章中讲到），你就会很容易明白它。目前而言，所有你需要知道的就是如何使用它们，以及为此如何使用这个方法的4个参数。

□ a是需要进行的操作，它有以下几个可选项。

- swap：把一个类替换为另一个。
- add：添加一个新类。

- remove: 移除一个类。
- check: 检查这个类是否已经应用了或没有应用。

o 是你需要为其添加类或从中删除类的对象。

c1和c2都是类的名字，且c2只有在swap动作中才需要。

使用这个方法重新编写一下前面的例子，这次通过动态地应用或删除一个类来隐藏和显示地址。

exampleClassChange.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Example: Dynamically changing classes</title>
<style type="text/css">
    @import "dynamic.css";
</style>
<script type="text/javascript" src="DOMhelp.js"></script>
<script type="text/javascript" src="classChange.js"></script>
</head>
<body>
<h3>Contact Details</h3>
<address>
    Awesome Web Production Company<br />
    Going Nowhere Lane 0<br />
    Catch 22<br />
    N4 2XX<br />
    England<br />
</address>
</body>
</html>
```

这个样式表中包含一个叫做hide的类，它会隐藏所有应用了它的元素。在这个例子中，为了达到隐藏对象的目的，使用了左移(off-left)技术(<http://css-discuss.incutio.com/?page=screenReader-Visibility>)，它可能是隐藏内容最容易理解的方式。改变visibility或display属性来隐藏元素的问题是：尽管在浏览器中它是可见的，但是盲人用户却无法通过屏幕阅读器访问。

classChange.css (摘录)

```
.hide{
    position: absolute;
    top: 0;
    left: -9999px;
    height: 0;
    overflow: hidden;
}
```

在脚本的开始处指定了类的名字为一个参数，这意味着如果有人在后续阶段需要改变这个名

字，他就不必去检查所有的脚本。

如果你开发了一个很复杂的站点，它需要添加和移除很多不同的类，那么可以把它们移出来放到包括自己对象的JavaScript脚本中。对这个例子来讲，这样的移动可能有点太大动干戈了，但后面我们还会回来讨论这个选择上来。

注解 注意DOMhelp中已经包含了cssjs()方法，因此不需要在这个例子中再包含它了。

```
classChange.js

sc={

    // CSS classes
    hidingClass:'hide', // Hide elements

    init:function(){
        sc.head=document.getElementsByTagName('h3')[0];
        if(!sc.head){return;}
        sc.ad=DOMhelp.closestSibling(sc.head,1);

        DOMhelp.cssjs('add',sc.ad,sc.hidingClass);

        var t=DOMhelp.getText(sc.head);
        var collapseLink=DOMhelp.createLink('#',t);
        sc.head.replaceChild(collapseLink,sc.head.firstChild);
        DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false)
        collapseLink.onclick=function(){return;} // Safari fix
    },
    peekaboo:function(e){

        if(DOMhelp.cssjs('check',sc.ad,sc.hidingClass)){
            DOMhelp.cssjs('remove',sc.ad,sc.hidingClass)
        } else {
            DOMhelp.cssjs('add',sc.ad,sc.hidingClass)
        }

        DOMhelp.cancelClick(e);
    }
}
DOMhelp.addEvent(window,'load',sc.init,false);
```

辅助 CSS 设计者

通过DOM编程以及将CSS分离出来放到可以动态应用和移除的类中，Web设计师的工作变得更加轻松了。使用DOM和JavaScript可以比CSS选择器访问文档中的更多东西。例如，我们经常需要访问CSS中父元素以实现悬停特效。在CSS中这是不可能实现的；而在JavaScript中通过parentNode可以很容易地实现。使用JavaScript和DOM，可以通过改变应用了类的HTML内容和ID

来为设计师提供样式表动态的效果，还可以生成内容，甚至通过添加或移除STYLE与LINK元素来添加和删除整个样式表。

1. 动态给网页添加样式更加容易

使设计师为一个站点的脚本增强版本与非脚本版本创建不同的样式尽可能地容易，是非常重要的。非脚本版本可能很简单，而且它需要的样式也更少（例如，在上面那个地址例子中，你只需要在JavaScript启用的时候，链接定义的样式到H3的内部即可，因为链接是通过JavaScript生成的）。当脚本启用的时候，给设计师提供一个唯一标识的最简单选择，就是给主体或布局的主要元素应用一个类。

```
dynamicStyling.js——用于exampleDynamicStyling.html

sc={

    // CSS classes
    hidingClass:'hide', // Hide elements
    DOMClass:'dynamic', // Indicate DOM support

    init:function(){
        // Check for DOM and apply a class to the body if it is supported
        if(!document.getElementById || !document.createElement){return;}
        DOMhelp.cssjs('add',document.body,sc.DOMClass);

        sc.head=document.getElementsByTagName('h3')[0];
        if(!sc.head){return;}
        sc.ad=DOMhelp.closestSibling(sc.head,1);
        DOMhelp.cssjs('add',sc.ad,sc.hidingClass);
        var t=DOMhelp.getText(sc.head);
        var collapseLink=DOMhelp.createLink('#',t);
        sc.head.replaceChild(collapseLink,sc.head.firstChild);
        DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false)
        collapseLink.onclick=function(){return;} // Safari fix
    },
    peekaboo:function(e){
        if(DOMhelp.cssjs('check',sc.ad,sc.hidingClass)){
            DOMhelp.cssjs('remove',sc.ad,sc.hidingClass)
        } else {
            DOMhelp.cssjs('add',sc.ad,sc.hidingClass)
        }
        DOMhelp.cancelClick(e);
    }
}
DOMhelp.addEvent(window,'load',sc.init,false);
```

通过这种方式，CSS设计师可以在样式表中定义，在JavaScript被禁用的时候要应用何种设置，而在JavaScript启用的时候，通过在后代选择器中使用带有类名的主体，从而用其他样式来覆盖它们。

dynamicStyling.css

```
*{  
    margin:0;  
    padding:0;  
}  
body{  
    font-family:Arial,Sans-Serif;  
    font-size:small;  
    padding:2em;  
}  
/* JS disabled */  
address{  
    background:#ddd;  
    border:1px solid #999;  
    border-top:none;  
    font-style:normal;  
    padding:.5em;  
    width:15em;  
}  
h3{  
    border:1px solid #000;  
    color:#fff;  
    background:#369;  
    padding:.2em .5em;  
    width:15em;  
    font-size:1em;  
}  
/* JS enabled */  
body.dynamic address{  
    background:#fff;  
    border:none;  
    font-style:normal;  
    padding:.5em;  
    border-top:1px solid #ccc;  
}  
body.dynamic h3{  
    padding-bottom:.5em;  
    background:#fff;  
    border:none;  
}  
body.dynamic h3 a{  
    color:#369;  
}  
/* dynamic classes */  
.hide{  
    position:absolute;  
    top:0;
```

```
left:-9999px;
height:0;
}
```

如图5-2所示，根据JavaScript和DOM是否可用，这个地址的例子现在有两种完全不同的风格（其中一个有两种状态）。



图5-2 地址的3种状态（非动态版本、折叠版本、展开版本）

如果你的站点不是特别复杂且没有太多动态元素的话，它会工作得非常好。对于更复杂的站点，你应该对非JavaScript和JavaScript版本使用不同的样式表，并通过JavaScript来添加后一个样式表。这样又多了一个好处：低级用户不需要加载对他们毫无用处的样式表。可以通过在文档的标题中创建一个新的LINK元素来添加动态样式表。在这个例子中，开头就包含了一个低级样式表。

```
exampleStyleSheetChange.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Example: Dynamically applying new Style Sheets </title>
  <style type="text/css">
    @import "lowlevel.css";
  </style>
  <script type="text/javascript" src="DOMhelp.js"></script>
  <script type="text/javascript" src="styleSheetChange.js"></script>
</head>
<body>
<h3>Contact Details</h3>
<address>
```

```
Awesome Web Production Company<br />
Going Nowhere Lane 0<br />
Catch 22<br />
N4 2XX<br />
England<br />
</address>
</body>
</html>
```

下面的脚本检查是否支持DOM，并添加一个指向高级样式表的新链接元素：

styleSheetChange.js

```
sc={
  // CSS classes

  hidingClass:'hide', // Hide elements
  highLevelStyleSheet:'highlevel.css', // Style sheet for dynamic site

  init:function(){

    // Check for DOM and apply a class to the body if it is supported
    if(!document.getElementById || !document.createElement){return;}

    var newStyle=document.createElement('link');
    newStyle.setAttribute('type','text/css');
    newStyle.setAttribute('rel','StyleSheet');
    newStyle.setAttribute('href',sc.highLevelStyleSheet);
    document.getElementsByTagName('head')[0].appendChild(newStyle);

    sc.head=document.getElementsByTagName('h3')[0];
    if(!sc.head){return;}
    sc.ad=DOMhelp.closestSibling(sc.head,1);
    DOMhelp.cssjs('add',sc.ad,sc.hidingClass);
    var t=DOMhelp.getText(sc.head);
    var collapseLink=DOMhelp.createLink('#',t);
    sc.head.replaceChild(collapseLink,sc.head.firstChild);
    DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false)
    collapseLink.onclick=function(){return;} // Safari fix
  },
  peekaboo:function(e){
    if(DOMhelp.cssjs('check',sc.ad,sc.hidingClass)){
      DOMhelp.cssjs('remove',sc.ad,sc.hidingClass)
    } else {
      DOMhelp.cssjs('add',sc.ad,sc.hidingClass)
    }
    DOMhelp.cancelClick(e);
  }
}
DOMhelp.addEvent(window,'load',sc.init);
```

下面是脚本执行后的HTML示例的头部（你可以在Firefox中测试一下，使用Ctrl+A或Cmd+A选择文档的全部，然后右键单击任意地方并选择View Selected Source）。

脚本执行后的exampleStyleSheetChange.html（摘录）

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Example: Using dynamic classes</title>
  <style type="text/css">
    @import "lowlevel.css";
  </style>
  <script type="text/javascript" src="DOMhelp.js"></script>
  <script type="text/javascript" src="styleSheetChange.js"></script>
  <link href="highlevel.css" rel="StyleSheet" type="text/css">
</head>
```

你可能在很早的时候就碰到过样式的动态改变。早在2001年的时候，样式切换就非常流行了。有一些小的网页工具允许用户通过从给定的一个列表中选择一种样式来选定一种页面外观。现代的浏览器已经内置包含了这个选项，例如在Firefox中，可以选择View→Page Style得到所有可选的样式风格。对于IE，JavaScript开发人员想出了一个巧妙的办法。如果你设置了一个链接元素的disabled属性，IE浏览器就不会应用它。因此你所要做的就是循环遍历文档的所有链接元素，除所选的那个以外，都设置为不可用。

演示例子exampleStyleSwitcher.html展示了样式切换是如何来实现的。在HTML中，定义了一个主要的样式表，并将样式表改变为大字体和高对比度。

exampleStyleSwitcher.html（摘录）

```
<link href="demoStyles.css" title="Normal"
      rel="stylesheet" type="text/css">
<link href="largePrint.css" title="Large Print"
      rel="alternate stylesheet" type="text/css">
<link href="highContrast.css" title="High Contrast"
      rel="alternate stylesheet" type="text/css">
```

这个脚本并不复杂。遍历文档中的所有LINK元素，并对每个元素判断其属性是stylesheet还是alternate stylesheet。然后创建一个新列表（带有指向函数的链接），其中链接只有当前选择的样式表可用，并把这个列表加到文档中。

开始需要两个属性：一个用来存储显示CSS样式的“样式菜单”的ID，另一个用来保存所有可用样式前面的第一个列表项的标签。

styleSwitcher.js

```
switcher={
  menuID:'styleSwitcher',
  chooseLabel:'Choose Style:',
```

一个名为init()的初始化方法创建了一个新的HTML列表，并添加了一个使用chooseLabel标签作为文本内容的列表项。设置列表的ID为属性中定义的那个。

styleSwitcher.js (续)

```
init:function(){
    var templI,tempA,styleTitle;
    var stylemenu=document.createElement('ul');
    templI=document.createElement('li');
    templI.appendChild(document.createTextNode(switcher.chooseLabel));
    stylemenu.appendChild(templI);
    stylemenu.id=switcher.menuID;
```

在文档中循环遍历所有的LINK元素。对于每一个元素，检查它的rel属性的值。如果这个值既不是stylesheet，也不是alternate stylesheet，那么跳过这个LINK元素。这是非常有必要的，可以避免其他通过LINK标签提供的选择性内容，如RSS提要（RSS feed）被禁止。

styleSwitcher.js (续)

```
var links=document.getElementsByTagName('link');
for(var i=0;i<links.length;i++){
    if(links[i].getAttribute('rel')!='stylesheet' &&
    links[i].getAttribute('rel')!='alternate stylesheet'){
        continue;
    }
}
```

为每种样式使用一个链接创建一个新的列表项，并把这个链接的文本值设置为LINK元素的title属性的值。设置一个虚假的href属性使这个链接看起来更像是一个链接，不然的话用户可能不会把这个新链接看作是个可交互的元素。

styleSwitcher.js (续)

```
tempLI=document.createElement('li');
tempA=document.createElement('a');
styleTitle=links[i].getAttribute('title');
tempA.appendChild(document.createTextNode(styleTitle));
tempA.setAttribute('href','#');
```

为链接应用一个事件处理程序，可以触发setSwitch()方法的执行，并通过this关键字把链接本身作为一个参数传递过去。接着可以继续添加新的列表项到菜单列表中，并在循环完成的时候把这个列表追加到文档的主体中。

styleSwitcher.js (续)

```
tempA.onclick=function(){
    switcher.setSwitch(this);
}
tempLI.appendChild(tempA);
stylemenu.appendChild(tempLI);
```

```

    }
    document.body.appendChild(stylemenu);
},

```

在setSwitch()方法中，你会通过参数o重新获得这个已被激活的链接。循环遍历所有的LINK元素，检查每一个链接以查看title属性是否和链接的文本内容是一样的（可以使用firstChild.nodeValue安全地读取链接的文本而不用检查结点的类型，因为是你生成了这些链接）。如果title不一样，那么设置LINK的disabled属性为true；如果它是相同的，设置disabled属性为false并设置rel属性为stylesheet而不是alternate stylesheet。然后通过返回false阻止这个链接被跟踪。

styleSwitcher.js（续）

```

setSwitch:function(o){
    var links=document.getElementsByTagName('link');
    for(var i=0;i<links.length;i++){
        if(links[i].getAttribute('rel')!='stylesheet' &&
        links[i].getAttribute('rel')!='alternate stylesheet'){
            continue;
        }
        var title=o.firstChild.nodeValue;
        if(links[i].getAttribute('title')!=title){
            links[i].disabled=true;
        } else {
            links[i].setAttribute('rel','stylesheet');
            links[i].disabled=false;
        }
    }
    return false;
}
}

```

可以在启用了JavaScript和CSS的浏览器中打开exampleStyleSwitcher.html来测试它的功能。

Paul Sowden于2001年在网站Alistapart.com上发表的文章*Alternative Style: Working With Alternate Style Sheets* (<http://www.alistapart.com/articles/alternate/>) 中首次提出了这个技巧。由于Netscape 4（记得是在2001的时候）当时不能很好地支持这个方法，所以Daniel Ludwin在2002年提出了*Backward Compatible Style Switcher* (<http://www.alistapart.com/articles/n4switch/>)，其中使用了document.write（从编码风格的角度看，向后退了一小步）。

关于JavaScript样式切换不可访问性的许多评论，2002年Chris Clark在他的文章*Build a PHP Switch* (<http://www.alistapart.com/articles/phpswitch/>) 中把这个技巧移植到了服务器端。这个方法更加稳定，但是它需要重新加载页面来应用新的样式。

随后出现了许多这种同一思想的不同变形，在2005年，Dustin Diaz在他的文章*Unobtrusive Degradable Ajax Style Sheet Switcher* (<http://24ways.org/advent/introducing-udasss>) 中提出了这样一种思想，使用Ajax很好地结合了PHP样式切换的稳定性和JavaScript增强的界面平滑性。

样式切换是个非常有用的功能，尤其是当你提供一些如大字体或前后背景之间的高对比度样

式来帮助用户克服类似视力不好这样的问题的时候。另一方面，如果你使用它们只为了提供不同样式，就变得没有多大意义了。

样式切换思想的演变说明了JavaScript的解决方案从来就不是固定不变的，而是需要在现实的世界中通过用户和其他开发人员的反馈不停地接受考验，以便真正地应用到产品环境或运行的网站中。如果你常在网上冲浪，会发现许多“实验性的”脚本，它们说能做很多事情，但切身体会一下就会发现要么非常慢、不稳定，要么只是一个用其他技术可以更好实现的小技巧。我们可以在JavaScript做任何事情并不意味着我们都应该这样去做。

2. 维护简化

把所有的界面外观代码从脚本中提取出来放到样式表中（从而把它变成了CSS设计者的责任）只能算做完了一半。在一个项目的维护过程中，CSS类名可能必须要改变——例如，为了支持某种后台或内容管理系统(CMS)。有一个例子就是Adobe公司的Contribute(一个轻量级的CMS，可以进行所见即所得的网站编辑)，在标准的配置中，它要求有一些类不在编辑器的样式选择器中出现，这些类以mmhide_开头。因此，使设计者可以更容易地改变动态应用的类名非常重要。最基本的办法就是把类名放到它们自己的变量或参数中^①。在前面的例子中我们已经这样做了。可以直接地应用类名：

```
sc={
    init:function(){
        // Check for DOM and apply a class to the body if it is supported
        if(!document.getElementById || !document.createElement){return;}
        DOMhelp.cssjs('add',document.body, 'dynamic');
        sc.head=document.getElementsByTagName('h3')[0];
        if(!sc.head){return;}
        sc.ad=DOMhelp.closestSibling(sc.head,1);
        DOMhelp.cssjs('add',sc.ad, 'hide');
        var t=DOMhelp.getText(sc.head);
        var collapseLink=DOMhelp.createLink('#',t);
        sc.head.replaceChild(collapseLink,sc.head.firstChild);
        DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false);
        collapseLink.onclick=function(){return;} // Safari fix
    },
    peekaboo:function(e){
        if(DOMhelp.cssjs('check',sc.ad,sc.hidingClass)){
            DOMhelp.cssjs('remove',sc.ad,sc.hidingClass)
        } else {
            DOMhelp.cssjs('add',sc.ad,sc.hidingClass)
        }
        DOMhelp.cancelClick(e);
    }
}
DOMhelp.addEvent(window,'load',sc.init,false);
```

此外，也可以把它们移出方法作为主要对象的属性，并给它们加上注释，可以让不了解

^① 代码中为'dynamic'和'hide'。——编者注

JavaScript的人改变类名而不影响你的方法的质量或功能。

```
sc={

    // CSS classes
    hidingClass:'hide',           // Hide elements
    DOMClass:'dynamic', // Indicate DOM support

    init:function(){
        if(!document.getElementById || !document.createElement){return;}
        DOMhelp.cssjs('add',document.body,sc.DOMClass);
        sc.head=document.getElementsByTagName('h3')[0];
        if(!sc.head){return;}
        sc.ad=DOMhelp.closestSibling(sc.head,1);
        DOMhelp.cssjs('add',sc.ad,sc.hidingClass);
        var t=DOMhelp.getText(sc.head);
        var collapseLink=DOMhelp.createLink('#',t);
        sc.head.replaceChild(collapseLink,sc.head.firstChild);
        DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false);
        collapseLink.onclick=function(){return;} // Safari fix
    },
    peekaboo:function(e){
        // More code snipped
    }
}
DOMhelp.addEvent(window,'load',sc.init,false);
```

对于更小的脚本和没有包含很多不同的JavaScript的项目，这已经足够了，如果加一些文档就更好了。如果有许多动态的类分布在几个文档中，或者你非常想让非编码人员也可以修改你的代码，那么应该使用一个独立的JavaScript包含文件，它包括一个叫CSS的对象，所有的类都是它的参数。显然可以将它命名为cssClassNames.js，并在项目的文档中记录它的存在。

```
cssClassNames.js
css={
    // Hide elements
    hide:'hide',

    // Indicator for support of dynamic scripting
    // will be added to the body element
    supported:'dynamic'
}
```

可以像使用其他脚本一样在网页文档中应用它：

```
exampleDynamicStylingCSSObject.html
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Example: Importing class names from a CSS names object</title>
    <style type="text/css">
```

```

@import "dynamicStyling.css";
</style>
<script type="text/javascript" src="DOMhelp.js"></script>
<script type="text/javascript" src="cssClassNames.js"></script>
<script type="text/javascript" src="dynamicStylingCSSObject.js"></script>
</head>

```

这个方法的实际应用结果是你不必为不同的CSS类名（通常包含“class”，因此会使程序员感到迷惑）提供许多参数名。可以使用下面的代码：

```

dynamicStylingCSSObject.js

sc={
  init:function(){
    if(!document.getElementById || !document.createElement){return;}
    DOMhelp.cssjs('add',document.body,css.supported);

    sc.head=document.getElementsByTagName('h3')[0];
    if(!sc.head){return;}
    sc.ad=DOMhelp.closestSibling(sc.head,1);

    DOMhelp.cssjs('add',sc.ad,css.hide);
    var t=DOMhelp.getText(sc.head);
    var collapseLink=DOMhelp.createLink('#',t);
    sc.head.replaceChild(collapseLink,sc.head.firstChild);
    DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false);
    collapseLink.onclick=function(){return;} // Safari fix
  },
  peekaboo:function(e){
    // More code snipped
  }
}
DOMhelp.addEvent(window,'load',sc.init,false);

```

在这个例子中，cssClassNames.js文件使用了字面量对象符号（object literal notation）^①。如果使用JSON (<http://www.json.org/>)，可以向前走得更远并去掉注释。JSON是一种用来转换数据的格式，可以把一个程序或系统中的数据转换为另一个程序或系统中的数据。你会在第7章中了解到更多关于JSON及其优点的知识。目前而言，知道JSON可以使带有类名的文件更有可读性就足够了：

```

cssClassNameJSON.js

css={
  'hide elements' : 'hide',
  'dynamic scripting enabled' : 'dynamic'
}

```

^① 指上页代码中CSS这样的对象。——编者注

不用使用前面的属性符号，现在读取数据时就好像是关联数组一样：

```
dynamicStylingJSON.js

sc={
  init:function(){
    if(!document.getElementById || !document.createElement){return;}
    DOMhelp.cssjs('add',document.body,
      css['dynamic scripting enabled']);
    sc.head=document.getElementsByTagName('h3')[0];
    if(!sc.head){return;}
    sc.ad=DOMhelp.closestSibling(sc.head,1);

    DOMhelp.cssjs('add',sc.ad,css['hide elements']);
    var t=DOMhelp.getText(sc.head);
    var collapseLink=DOMhelp.createLink('#',t);
    sc.head.replaceChild(collapseLink,sc.head.firstChild);
    DOMhelp.addEvent(collapseLink,'click',sc.peekaboo,false);
    collapseLink.onclick=function(){return;} // Safari fix
  },
  peekaboo:function(e){
    // More code snipped
  }
}
DOMhelp.addEvent(window,'load',sc.init,false);
```

是否从行为中进一步把表现分离完全是由你决定的，但是从项目的复杂性和维护人员的角度来看，它将预防许多可避免的错误。

3. 克服CSS支持性问题

在过去的几年中，CSS在Web开发中变得越来越重要。复杂的嵌套表布局方式一直在走下坡路，正逐步被轻量级的CSS布局所代替。而且，过去由JavaScript处理的许多特效，如图像轮转和折叠式菜单，它们的CSS解决方案已变得非常流行，并给设计社区一个新希望，可以不用了解JavaScript也能够创建出平滑流畅、交互性的界面。

许多CSS开发人员迟早都不得不面对的是，老浏览器支持得不够或者新一代浏览器很难理解的难题。因为CSS不是一种编程语言，无法用它来循环、条件分支或对象测试，为了支持某些特效，你既需要同时使用CSS和JavaScript。使用伪类、生成的内容以及CSS补丁（hack）去解决某些浏览器问题在CSS 2和CSS 3中是可以的，但是你还不能依赖它们能被支持——尤其是对于像IE 6这样对CSS支持有问题的浏览器，即使未必是市场占有量最大的，但仍然会很强大。CSS补丁与浏览器检测会遇到同样的问题：当一种新的浏览器出来的时候，你不得不检查它是否仍然遵从这些补丁所需要的规则。现在，IE 7的beta版本就需要许多CSS开发人员去重写它们以前的代码。

许多非常有趣的CSS概念需要依赖JavaScript手段来支持类似IE 6这样的浏览器，从而解决上述问题。

- 等高的多列布局

对于以前只使用表布局的设计者，关于CSS布局最头疼的一件事情就是，如果对列使用CSS的浮动（float）技术，它们的高度会不一样，如图5-3所示。

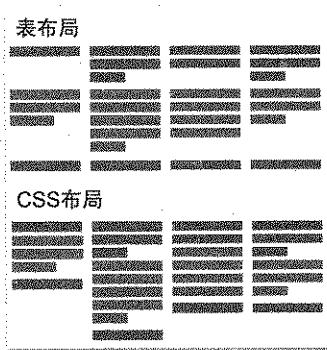


图5-3 多列布局的高度问题

来看一个新闻项列表，每个新闻项包含一个标题、一个简单段落和一个“more”链接。

exampleColumnHeightIssue.html（使用了虚构的内容）

```
<ul id="news">
  <li>
    <h3><a href="news.php?item=1">News Title 1</a></h3>
    <p>Description 1</p>
    <p class="more"><a href="news.php?item=1">more link 1</a></p>
  </li>
  <li>
    <h3><a href="news.php?item=2">News Title 2</a></h3>
    <p>Description 2</p>
    <p class="more"><a href="news.php?item=2">more link 2</a></p>
  </li>
  <li>
    <h3><a href="news.php?item=3">News Title 3</a></h3>
    <p>Description 3</p>
    <p class="more"><a href="news.php?item=1">more link 3</a></p>
  </li>
  <li>
    <h3><a href="news.php?item=1">News Title 1</a></h3>
    <p>Description 4</p>
    <p class="more"><a href="news.php?item=4">more link 4</a></p>
  </li>
</ul>
```

如果你现在应用了一种样式表，它把列表项和主列表浮动到左边，并设置更多文本和布局样式，就会得到一个多列的布局。用CSS样式来实现这种效果是非常容易的：

columnHeightIssue.css

```
#news{
    width:800px;
    float:left;
}
#news li{
    width:190px;
    margin:0 4px;
    float:left;
    background:#eee;
}
#news h3{
    background:#fff;
    padding-bottom:5px;
    border-bottom:2px solid #369;
}
#news li p{
    padding:5px;
}
```

在这个例子中可以看到，每列的高度都不相同，段落和“more”链接的位置也都不相同。这样的设计看上去很不平衡，常使读者感到很迷惑。也许有一种CSS方式能够修正这个问题（人们似乎总是能找到CSS补丁或权宜的解决方式），但这里我们使用JavaScript来解决这个问题。

下面的脚本——在文档的HEAD中调用，会修正这个问题：

fixColumnHeight.js——在exampleFixedColumnHeightIssue.html中会用到

```
fixcolumns={

    highest:0,
    moreClass:'more',

    init:function(){
        if(!document.getElementById || !document.createTextNode){return;}
        fixcolumns.n=document.getElementById('news');
        if(!fixcolumns.n){return;}
        fixcolumns.fix('h3');
        fixcolumns.fix('p');
        fixcolumns.fix('li');
    },
    fix:function(elm){
        fixcolumns.getHighest(elm);
        fixcolumns.fixElements(elm);
    },
    getHighest:function(elm){
        fixcolumns.highest=0;
```

```

var temp=fixcolumns.n.getElementsByTagName(elm);
for(var i=0;i<temp.length;i++){
    if(!temp[i].offsetHeight){continue;}
    if(temp[i].offsetHeight>fixcolumns.highest){
        fixcolumns.highest=temp[i].offsetHeight;
    }
}
},
fixElements:function(elm){
    var temp=fixcolumns.n.getElementsByTagName(elm);
    for(var i=0;i<temp.length;i++){
        if(!DOMhelp.cssjs('check',temp[i],fixcolumns.moreClass)){
            temp[i].style.height=parseInt(fixcolumns.highest)+'px';
        }
    }
}
}
DOMhelp.addEvent(window, 'load', fixcolumns.init, false);

```

首先，定义一个存储最高元素高度的参数和一个“more”链接要用到的类。（后者非常重要，我们会在后面解释。）在init()方法中，首先检查是否支持DOM以及ID为news的必需元素是否可用。把这个news元素存储在属性n中以供在其他的方法中再使用。我们对列表中包含的每个元素——标题、段落和列表项，分别调用fix()方法。最后修改列表项非常重要，因为它们的最大高度可能在其他元素被修改的时候已经发生变化了。

fixColumnHeight.js（摘录）

```

fixcolumns={

    highest:0,
    moreClass:'more',

    init:function(){
        if(!document.getElementById || !document.createTextNode){return;}
        fixcolumns.n=document.getElementById('news');
        if(!fixcolumns.n){return;}
        fixcolumns.fix('h3');
        fixcolumns.fix('p');
        fixcolumns.fix('li');
    },
}

```

fix()方法调用了另外两个方法，其中一个找出应用到每个项的最大高度，另外一个应用这个高度。

fixColumnHeight.js（摘录）

```

fix:function(elm){
    fixcolumns.getHighest(elm);
    fixcolumns.fixElements(elm);
},

```

`getHighest()`方法首先设置参数`highest`为0，然后循环遍历列表中元素名与传递的参数`elm`匹配的所有元素。接着通过读取`offsetHeight`属性重新获得该元素的高度。这个属性记录着这个元素在浏览器上呈现出来的高度。接着这个方法检查元素的高度是否大于属性`highest`。如果大于，则把这个属性设置为新的值。通过这种方式就会找出最高的元素。

`fixColumnHeight.js`（摘录）

```
getHighest:function(elm){
    fixcolumns.highest=0;
    var temp=fixcolumns.n.getElementsByTagName(elm);
    for(var i=0;i<temp.length;i++){
        if(!temp[i].offsetHeight){continue;}
        if(temp[i].offsetHeight>fixcolumns.highest){
            fixcolumns.highest=temp[i].offsetHeight;
        }
    }
},
```

警告 这里重新把参数`highest`设置为0非常重要，因为`getHighest()`需要查找被作为参数传递过去的最高元素，不是你修改的所有元素。如果在某些反常情况下H3比最高的段落还高，那么在段落和“more”链接之间会有空隙。

`fixElements()`方法接着把最大的高度应用到给定名字的所有元素上。注意你需要检查决定“more”链接的类，否则的话链接的高度会与最高的内容段落的高度一样高。

```
fixElements:function(elm){
    var temp=fixcolumns.n.getElementsByTagName(elm);
    for(var i=0;i<temp.length;i++){
        if(!DOMhelp.cssjs('check',temp[i],fixcolumns.moreClass)){
            temp[i].style.height = fixcolumns.highest + 'px';
        }
    }
}
```

注解 注意，把`highest`应用到元素的高度前，需要把它转换为数字并加上一个“px”后缀；对于元素的CSS尺度，你不能只简单地赋给它一个不带单位的数字。

- 缺乏对`:hover`的支持

CSS规范允许对文档的任意元素使用伪类`:hover`，并且许多浏览器都支持这个功能。它可以让设计师对文档的较大部分突出显示，甚至模拟过去只有使用JavaScript才可以实现的动态折叠导航菜单。虽然一些没有使用CSS和JavaScript的非交互元素在鼠标悬停到它上面的时候，是否也应该有一个不同的状态，还需要讨论，但是这是一个设计师可以大量使用的特性——毕竟对文档的当前部分突出显示会更易读。

来看一个例子，再次拿出新闻项列表并给它应用一种不同的样式表。如果想在CSS-2兼容的浏览器中突出显示整个列表项，所有你需要做的就是为列表项定义一个悬停的状态：

```
listItemRolloverCSS.css (摘录) ——与exampleListItemRollover.html一起使用

#news{
    font-size:.8em;
    background:#eee;
    width:21em;
    padding:.5em 0;
}
#news li{
    width:20em;
    padding:.5em;
    margin:0;
}
#news li:hover{
    background:#fff;
}
```

在浏览器Firefox 1.5上，这个特效看起来如图5-4所示：

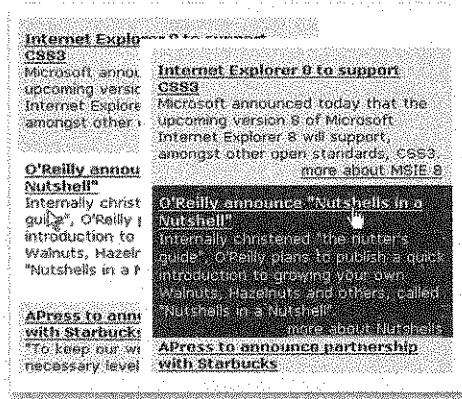


图5-4 在CSS中使用伪类:hover的翻转特效

在IE 6中，不会得到这种效果，因为它对列表项不支持:hover。可是，它支持JavaScript，这意味着你可以使用cssjs方法在用户悬停在列表项的时候动态地添加一个类：

listItemRollover.css (摘录)

```
#news{
    font-size:.8em;
    background:#eee;
    width:21em;
    padding:.5em 0;
}
```

```

#news li{
    width:20em;
    padding:.5em;
    margin:0;
}
#news li:hover{
    background:#fff;
}
#news li.over{
    background:#fff;
}

```

可以通过onmouseover和onmouseout事件处理程序并使用this关键字来添加这个类，this关键字我们稍后会在本章做进一步研究。

listItemRollover.js（摘录）

```

newshl={
    overClass:'over',
    init:function(){
        alert('d');
        if(!document.getElementById || !document.createTextNode){return;}
        var newsList=document.getElementById('news');
        if(!newsList){return;}
        var newsItems=newsList.getElementsByTagName('li');
        for(var i=0;i<newsItems.length;i++){
            newsItems[i].onmouseover=function(){
                DOMhelp.cssjs('add',this,newshl.overClass);
            }
            newsItems[i].onmouseout=function(){
                DOMhelp.cssjs('remove',this,newshl.overClass);
            }
        }
    }
}
DOMhelp.addEvent(window,'load',newshl.init,false);

```

如果在IE 6中检查一下这个例子，你会得到与大多数现代浏览器一样的效果。

可以使用CSS的伪类选择器来实现动态特效（:hover、:active和:focus），但它们只能将其设置应用到包含在当前元素之中的元素上（使用Opera 8或Safari的CSS高手把这些功能和兄弟选择器结合起来使用可以实现更多的功能，但它属于高级的CSS范畴，已超出了本书的范围）。

使用JavaScript，可以处理所有的DOM对象（包括parentNode、nextSibling、firstChild，等等）。

如果想在用户悬停到链接上的时候有一个不同的翻转状态，扩展这个脚本就很容易实现。首先需要一个激活状态的新类：

listItemDoubleRollover.css——在listItemDoubleRollover.html中使用

```
#news{
    font-size:.8em;
    background:#eee;
    width:21em;
    padding:.5em 0;
}
#news li{
    width:20em;
    padding:.5em;
    margin:0;
}
#news li:hover{
    background:#fff;
}
#news li.over{
    background:#fff;
}
#news li.active{
    background:#ffc;
}
```

接下来需要对列表项内的链接应用事件，并改变它们父节点的父节点所用的类（因为这个例子中的链接是在标题或段落中）：

listItemDoubleRollover.js

```
newshl={
    // CSS classes
    overClass:'over', // Hover state of list item
    activeClass:'active', // Hover state on a link

    init:function(){
        if(!document.getElementById || !document.createTextNode){return;}
        var newsList=document.getElementById('news');
        if(!newsList){return;}
        var newsItems=newsList.getElementsByTagName('li');
        for(var i=0;i<newsItems.length;i++){
            newsItems[i].onmouseover=function(){
                DOMhelp.cssjs('add',this,newshl.overClass);
            }
            newsItems[i].onmouseout=function(){
                DOMhelp.cssjs('remove',this,newshl.overClass);
            }
        }
        var newsItemLinks=newsList.getElementsByTagName('a');
        for(i=0;i<newsItemLinks.length;i++){
```

```

newsItemLinks[i].onmouseover=function(){
    var p=this.parentNode.parentNode;
    DOMhelp.cssjs('add',p,newshl.activeClass);
}
newsItemLinks[i].onmouseout=function(){
    var p=this.parentNode.parentNode;
    DOMhelp.cssjs('remove',p,newshl.activeClass);
}
}
}
}
DOMhelp.addEvent(window, 'load', newshl.init, false);

```

依据用户鼠标停放在文本还是链接上，执行的结果是新闻项的两种不同的状态，如图5-5所示。

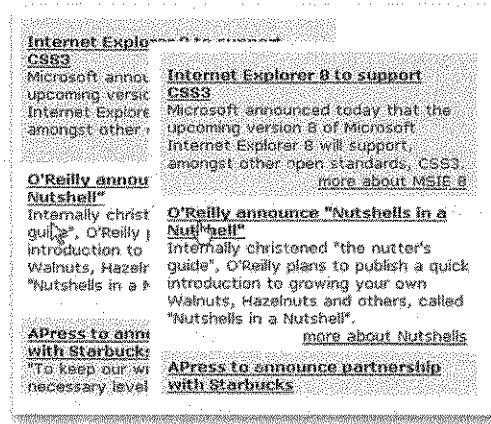


图5-5 单一元素的不同翻转状态

使用JavaScript，还可以使整个新闻项都可以单击，多应用几个事件就可以了。

- 如果你有一把锤子，所有的东西看起来都像一颗钉子

通过这几个例子你会领悟到，JavaScript和DOM在控制浏览器行为的时候非常强大，但在处理特效的时候却不行，因为浏览器对CSS的支持性不是很够。

然而，问题是，是否值得我们花精力去划清这个界限。在Web有许多脚本讲述如何在IE 6上解决CSS的支持性问题，但它们可能引来许多其他的问题——从减慢页面的下载到带来许多安全隐患，等等。

你经常会发现，自己在使用CSS做一些本应该用JavaScript做的事情。比如多级的下拉菜单就是如此。与CSS2兼容的浏览器允许对所有的对象使用hover特效，所以可以在鼠标移动到父元素的时候隐藏和显示嵌套的列表元素。然而，它的整体支持性还是不够，尽管一些CSS设计师如Stu Nicholls (<http://www.cssplay.com>) 设法通过:active和:focus伪类选择器，在浏览器Opera、Safari和Mozilla上启用键盘来操作特效，但要在IE 6上也做同样的事情，你仍然不得不使用脚本。

JavaScript相对于CSS有一个很大的优点，就是它和文档的交互是双向的。相对于CSS只能设置样式或处理一些给定的东西，JavaScript可以读取值、检查支持性、检查对象是否可用以及它们是什么，甚至在需要的时候可以不受限制地创建对象。CSS只能读取网页文档，就像你只能阅读报纸，但是JavaScript不只能够读取文档，还能修改它。

JavaScript的这一功能被许多CSS技巧使用。许多特效也只有通过扩展标记才能实现——嵌套元素、清除元素、灵活的圆角图片，等等。因此，开发人员开始使用JavaScript生成这些而不是把它们放到HTML中。这使源文档显得更加简洁，但是对于终端用户，DOM树（包括所有生成内容）最后还是必须要处理。臃肿的HTML并不因为转为通过JavaScript生成而好多少。有时候考虑简化界面或在开始的时候就使它更加灵活，比通过许多CSS和JavaScript高级技巧使它的行为表现类似于表格可能要更好。

关于表现还要注意的是，一个灵活的DOM脚本有时会在意想不到的地方帮助你。Flash影片就是例子。多年来Flash一直被作为JavaScript杀手来宣传，而且最近Bobby van der Sluis已发布了一种脚本，叫做Unobtrusive Flash Object (<http://www.bobbyvandersluis.com/ufo/>)，即UFO，它使Flash开发人员的日子更轻松了些。

使用Flash的问题是，一些浏览器将Flash影片嵌入文档中的方式是和标准不兼容的。XHTML需要一个`<object>`标签来添加这样的多媒体内容；而为了适应老的浏览器，许多开发人员也使用另外一个`<embed>`标签来添加Flash影片，这不是HTML标准的一部分。早在2002年，Drew McLellan为此就提出了一种只适用于Flash的解决方案，把它叫做Flash Satay (<http://www.alistapart.com/articles/flashsatay/>)，但这个修正并不是总是适用的（这个名字令人困惑，如果Drew把它叫做“Embedding Flash the XHTML way”，我相信许多人将已经学习到它）。

Bobby的解决方案使用了分离式JavaScript，在可能的情况下可以使用Flash影片来取代替换内容。这还提高了安全性，如果微软打算关闭Flash内容的自动执行，也不会出问题。

微软计划让Flash内容只有在用户单击主要影片的时候启动它而不是自动地启动它。在IE行为中的这个改变是由于微软与Eolas公司和加利福尼亚大学之间的版权诉讼引起的。对这个问题微软自己的解决方案的代码质量是非常可疑的 (http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/overview/activating_activex.asp)，因为它使用了`document.write()`，UFO是关于这个问题的一种简洁灵活的方式。

5.2 通过事件处理改变文档的行为

事件处理可能是JavaScript提供给以用户界面为中心的开发人员最好的东西了。它也是最容易搞混的一个JavaScript主题，并不是因为它复杂，而是因为不同的浏览器实现事件处理的方式不同。我会很快给你解释事件是什么；展示一个成熟的、好用的处理事件的方法；接下来会讲一个W3C推荐的新方法。最后，你会学到如何调整与W3C兼容的方法，让不支持它的浏览器也可以理解你的脚本。

事件可以是各种各样的事情，例如：

- 初始化的加载和网页文档的呈现；
- 图片的加载；
- 用户单击一个按钮；
- 用户按一个键；
- 用户把鼠标移动到某个元素上。

注解 可以把事件处理想象成一个移动探测器或门铃装置——如果有人移动到门跟前，灯就会打开；如果有人按下门铃的按钮，电路就会闭合，铃声装置就会被触发并发出声音。同样的原理，当用户把鼠标放到一个链接上去触发一个函数或当用户单击这个链接去触发另一个函数的时候，你也可以检测到。

可以用好几种方式应用事件处理，使你的脚本知道正在发生什么。最直接的方式就是在HTML中使用内置的事件：

```
<a href="moreinfo.html" onclick="return infoWindow(this.href)">more information</a>
```

一个更巧妙的方式在前面的章节中已经提到了：通过一个类或ID标识一个对象，然后在脚本中设置事件处理程序。最简单也是支持最广泛的方式是直接用属性为对象应用事件处理程序：

HTML：
`more information`

JavaScript：
`var triggerLink=document.getElementById('info');
triggerlink.onclick=infoWindow;`

注解 注意没有必要去检查ID为info的元素，因为这个函数只能被它调用。

以这种方式触发事件有几个问题：

- 没有把这个元素传给函数；需要再次查找这个对象。
- 只能一次指派一个函数。
- 你的脚本以独占的方式劫持了这个元素的事件——其他脚本的方法试图在这个对象上使用其他事件的时候就不再起作用了。

除非你已定义triggerLink作为一个全局变量或对象属性，不然函数infoWindow()需要在使用它前找出触发的元素。

```
function infoWindow(){  

  var url=document.getElementById('info').getAttribute('href');  

  // Other code  

}
```

这个问题与“关联一个事件的多个函数”的问题可以通过应用一个调用真实函数的匿名函数

来解决，它同样也允许你通过关键字this来传送当前的对象：

```
var triggerLink=document.getElementById('info');
triggerlink.onclick=function(){
    showInfoWindow(this.href);
    highlight(this);
    setCurrent(this);
}
function showInfoWindow(url){
    // Other code
}
```

第三个问题仍然没有解决。在网页文档中的最后一个脚本，会覆盖其他脚本的事件触发，这意味着它不能和其他的脚本一起工作。因此你需要一种不覆盖其他脚本的指定事件处理的方式。在文档被加载后你需要调用几个不同函数的时候，这个尤为重要。

Simon Willison (<http://simon.incutio.com/>) 为此建立了一个解决办法，它使用下面的这个方法来进行事件处理：

```
function addLoadEvent(func) {
    var oldonload = window.onload;
    if (typeof window.onload != 'function') {
        window.onload = func;
    } else {
        window.onload = function() {
            oldonload();
            func();
        }
    }
}
```

如果使用这个工具函数把函数添加到网页中，就不会覆盖其他被网页的load事件调用的函数。这在过去和现在都是一个很大的进步。

据我所知，前面的方法在所有支持JavaScript的浏览器中都好用。在特殊情况下，如果你不得不支持Mac平台上的IE 5，那么还有很多事情要做。新的浏览器支持改进了的事件模型，它可以清楚区分事件处理的不同方面。

5.2.1 W3C 标准兼容的事件

注解 本节中的例子需要在DOM-2兼容的浏览器上执行。推荐使用Mozilla Firefox的1.5版本。IE 6不是DOM-2兼容的浏览器，这些例子在它上面不会正常工作。本节会讲解W3C是如何定义事件的以及它的工作原理。然后，我们会扩展这个方法以在IE以及其他不支持的浏览器上执行。

W3C中的DOM-2和即将发布的DOM-3规范对于事件处理的方法有所不同。首先，它们详细定义了处理事件时要用到的几个概念：

- 事件（event）是发生动作，例如，click。
- 事件处理器（event handler），例如onclick，在DOM-1中是注册事件的地方。
- 在DOM-2中，这点略有不同——你处理的是事件目标和事件监听器：
 - 事件目标（event target）是事件发生的地方，大多数情况下就是一个HTML元素。
 - 事件监听器（event listener）是一个函数，它用来处理事件。
- DOM-3引入了事件捕捉（event capturing）的概念，现在还没有浏览器支持它。如果想学习这些东西，可以查看W3C规范(<http://www.w3.org/TR/DOM-Level-3-Events/events.html>)；这里我不会涉及事件捕捉，因为它现在还不是可用的JavaScript的一部分。

1. 应用事件

可以通过addEventListener()方法来应用一个事件。这个方法有3个参数：一个是事件，它是一个不带“on”前缀的字符串，一个是事件监听器函数的名字（不带括号），还有一个是Boolean值，叫useCapture，它定义了是否使用事件捕捉。就目前而言，把useCapture设置为false是比较安全的。

如果想通过addEventListener()把函数infoWindow()应用到链接上，那么可以使用下面的代码：

```
var triggerLink=document.getElementById('info');
triggerLink.addEventListener( 'click', infoWindow, false);
```

如果需要添加一个悬停特效，在鼠标移动到链接上时调用highlight()函数，在鼠标离开链接时调用unhighlight()函数，可以多加两行代码：

```
var triggerLink=document.getElementById('info');
triggerLink.addEventListener( 'click', infoWindow, false);
triggerLink.addEventListener( 'mouseout', highlight, false);
triggerLink.addEventListener( 'mouseover', unhighlight, false);
```

2. 检查触发的事件名称、地方和方式

从开发便利性的角度来看，好像又回到了以前的地方：你必须再次在函数infoWindow()中查找元素来读取href。的确是这样的，然而，通过使用addEventListener，标准兼容的浏览器会给你提供事件对象（event object），可以通过一个叫e的参数来读出它。

你以前可能见到过e，并想知道它到底是什么，是否应该相信一个不知道从哪儿来的e。应用事件时，使用一个没有传递的参数刚开始会使你很困惑，但你一旦学会了事件对象，就再也不会回去使用onevent属性了。事件对象拥有许多你可以在事件监听器函数中使用的属性。

- target：触发事件的元素。
- type：被触发的事件（例如，click）。
- button：被按下的鼠标按钮。0代表左按钮，1代表中间按钮，2表示右按钮。
- keyCode/data/charCode：被按下的键的字符编码。W3C规范使用data；可是，它并没有获得广泛的支持。相反大多数浏览器都使用charCode，而keyCode是IE的实现。幸好所有的现代浏览器都能理解keyCode，这意味着你现在可以使用它，直到data在用户代理中获得更广泛的支持。

- shiftKey/ctrlKey/altKey: 一个布尔值——如果Shift、Ctrl或Alt键（分别地）被按下则返回true。

可供使用的完整属性列表取决于你所监听的事件。可以在DOM-2规范中查找所有的属性列表：<http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-Registration-interfaces>。

使用Event对象，可以很容易地使用一个函数来处理几个事件：

```
var triggerLink=document.getElementById('info');
triggerLink.addEventListener( 'click', infoWindow, false);
triggerLink.addEventListener( 'mouseout', infoWindow, false);
triggerLink.addEventListener( 'mouseover', infoWindow, false);
```

对于这3种事件，可以使用同样的函数并检查它们的事件类型：

```
function infoWindow(e){
  switch(e.type){
    case 'click':
      // Code to deal with the user clicking the link
      break;
    case 'mouseover':
      // Code to deal with the user hovering over the link
      break;
    case 'mouseout':
      // Code to deal with the user leaving the link
      break;
  }
}
```

也可以通过检查事件目标的nodeName来检查事件发生所在的元素。再次提醒你一下，必须使用toLowerCase()以避免跨浏览器平台的问题：

```
function infoWindow(e){
  targetElement=e.target.nodeName.toLowerCase();
  switch(targetElement){
    case 'input':
      // Code to deal with input elements
      break;
    case 'a':
      // Code to deal with links
      break;
    case 'h1':
      // Code to deal with the main heading
      break;
  }
}
```

3. 停止事件传播

指派事件与使用事件监听器截取它们也意味着你需要注意两个问题：一个是许多事件都有默认的动作——举个例子，click可能使浏览器打开一个链接或提交一个表单，而keyup则可能添加一个字符到表单域中。

另外一个问题叫做事件冒泡（event bubbling），它主要意味着当一个事件发生在一个元素上的时候，它也发生在这个元素的所有父元素上。

● 事件冒泡

回到新闻列表的HTML标记：

```
exampleEventBubble.html
<ul id="news">
  <li>
    <h3><a href="news.php?item=1">News Title 1</a></h3>
    <p>Description 1</p>
    <p class="more"><a href="news.php?item=1">more link 1</a></p>
  </li>
  <!-- and so on -->
</ul>
```

如果为列表中的链接指派一个mouseover事件，悬停在它们上面也会触发其他的事件监听，它们可能是在段落上、列表项上、列表上，以及节点树直到文档的主体上的所有其他元素。通过下面这个例子，你会明白如何把事件监听器附加到指向适当函数的每个对象上：

```
eventBubble.js
bubbleTest={
  init:function(){
    if(!document.getElementById || !document.createTextNode){return;}
    bubbleTest.n=document.getElementById('news');
    if(!bubbleTest.n){return;}
    bubbleTest.addMyListeners('click',bubbleTest.liTest,'li');
    bubbleTest.addMyListeners('click',bubbleTest.aTest,'a');
    bubbleTest.addMyListeners('click',bubbleTest.pTest,'p');

  },
  addMyListeners:function(eventName,functionName,elements){
    var temp=bubbleTest.n.getElementsByTagName(elements);
    for(var i=0;i<temp.length;i++){
      temp[i].addEventListener(eventName,functionName,false);
    }
  },
  liTest:function(e){
    alert('li was clicked');
  },
  pTest:function(e){
    alert('p was clicked');
  },
  aTest:function (e){
    alert('a was clicked');
  }
}
window.addEventListener('load',bubbleTest.init,false);
```

现在在它们被单击的时候，所有的列表项都会触发liTest()方法，所有的段落会触发pTest()方法，所有的链接会触发aTest()方法。

可是，如果单击段落，你会得到两个警告：

```
p was clicked
li was clicked
```

可以使用e.stopPropagation()方法来阻止这种情况的发生，它可以确保只有应用到链接上的事件监听器会得到这个事件。如果把pTest()方法修改为下面的样子：

stopPropagation.js——在exampleStopPropagation.html中使用

```
pTest:function(e){
    alert('p was clicked');
    e.stopPropagation();
},
```

输出会是：

```
p was clicked
```

事件冒泡事实上问题并不大，因为你不会常把不同的监听器指派给内嵌元素而不指派给它们的父元素。如果想学习更多有关事件冒泡以及某个事件发生后事件监听器发生顺序的内容，可以参考Peter-Paul Koch写过的一篇非常优秀的文章，参见http://www.quirksmode.org/js/events_order.html。

- 阻止默认的动作

你可能还会遇到的一些其他问题是，某些元素的事件拥有自己默认的动作。例如，链接会载入另一个文档。你可能不希望它发生，因此必须在执行你的事件监听器函数后停止默认的动作。

在DOM-1事件处理程序模型中，可以通过在被调用的函数中返回一个false来实现：

```
element.onclick=function(){
    // Do other code
    return false;
}
```

如果单击前面例子中的任何一个链接，它们都会加载链接的网页文档。可以使用DOM-2的preventDefault()方法来覆盖它。我们把它添加到aTest方法中测试一下：

preventDefault.js——在examplePreventDefault.html中使用

```
aTest:function (e){
    alert('a was clicked');
    e.stopPropagation();
    e.preventDefault();
}
```

现在单击这个链接就只会显示警告窗口：

```
a was clicked
```

链接的页面不会被打开，而且你会停留在原页面，对链接数据进行其他处理。

例如，你可以开始的时候只显示大标题，并在单击它们时展开要显示的内容。首先，你的样式表中还需要一些类来处理这些改变。

listItemCollapse.css（摘录）

```
.hide{
    display:none;
}
li.current{
    background:#ccf;
}
li.current h3{
    background:#69c;
}
```

折叠元素的脚本并不复杂，复杂的是使用我们讲过的所有处理对象的事件：

newsItemCollapse.js

```
newshl={
    // CSS classes
    overClass:'over', // Rollover effect
    hideClass:'hide', // Hide things
    currentClass:'current', // Open item

    init:function(){
        var ps,i,hl;
        if(!document.getElementById || !document.createTextNode){return;}
        var newsList=document.getElementById('news');
        if(!newsList){return;}
        var newsItems=newsList.getElementsByTagName('li');
        for(i=0;i<newsItems.length;i++){
            hl=newsItems[i].getElementsByTagName('a')[0];
            hl.addEventListener('click',newshl.toggleNews,false);
            hl.addEventListener('mouseover',newshl.hover,false);
            hl.addEventListener('mouseout',newshl.hover,false);
        }
        var ps=newsList.getElementsByTagName('p');
        for(i=0;i<ps.length;i++){
            DOMhelp.cssjs('add',ps[i],newshl.hideClass);
        }
    },
    toggleNews:function(e){
```

```

var section=e.target.parentNode.parentNode;
var first=section.getElementsByTagName('p')[0];
var action=DOMhelp.cssjs('check',first,newshl.hideClass)?'remove':'add';
var sectionAction=action=='remove'? 'add': 'remove';
var ps=section.getElementsByTagName('p');
for(var i=0;i<ps.length;i++){
    DOMhelp.cssjs(action,ps[i],newshl.hideClass);
}
DOMhelp.cssjs(sectionAction,section,newshl.currentClass);
e.preventDefault();
e.stopPropagation();
},
hover:function(e){
    var hl=e.target.parentNode.parentNode;
    var action=e.type=='mouseout'? 'remove': 'add';
    DOMhelp.cssjs(action,hl,newshl.overClass);
}
}
window.addEventListener ('load',newshl.init,false);

```

结果是可以单击的新闻标题，单击它们时会显示相关的新闻摘要。“more”链接并没有受影响，它在被单击的时候会把完整的新闻文章发送给用户（参见图5-6）。

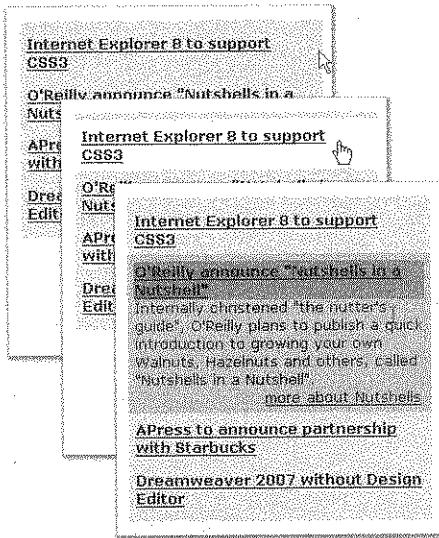


图5-6 通过单击标题显示新闻项

我们一步一步来看所有的脚本。在定义了CSS类属性和检查了必需的元素后，开始循环遍历所有的列表项，获取第一个链接（它是标题中的一个链接），并为click、mouseover和mouseout指派事件监听程序。click事件应该触发newshl.toggleNews()方法，而mouseout和mouseover应该触发newshl.hover()方法。

newsItemCollapse.js（摘录）

```
for(i=0;i<newsItems.length;i++){
    hl=newsItems[i].getElementsByTagName('a')[0];
    hl.addEventListener('click',newshl.toggleNews,false);
    hl.addEventListener('mouseover',newshl.hover,false);
    hl.addEventListener('mouseout',newshl.hover,false);
}
```

可以通过在段落上应用隐藏类来隐藏列表项中的所有段落：

newsItemCollapse.js（摘录）

```
var ps=newsList.getElementsByTagName('p');
for(i=0;i<ps.length;i++){
    DOMhelp.cssjs('add',ps[i],newshl.hideClass);
}
```

`toggleNews()`方法通过读取事件对象的目标来获取当前的元素。这个目标是链接，它意味着如果你想访问列表项，需要访问两次父节点。

newsItemCollapse.js（摘录）

```
toggleNews:function(e){
    var section=e.target.parentNode.parentNode;
```

读取列表项中的第一个段落并检查它是否已经有指派给它的隐藏类。如果已经有指派的隐藏类，定义变量`action`为`remove`；否则定义`action`为`add`。设置另一个叫做`sectionAction`的变量，把它定义为与`action`有相同的选项但对立的变量。

newsItemCollapse.js（摘录）

```
var first=section.getElementsByTagName('p')[0];
var action=DOMhelp.cssjs('check',first,newshl.hideClass)?'remove':'add';
var sectionAction=action=='remove'? 'add': 'remove';
```

循环遍历所有的段落，根据`action`决定是删除`hideClass`还是添加它。对于`section`和`currentclass`使用同样的处理，但是这次使用`sectionAction`。这样就可以有效地控制段落的可见性和标题的样式。

newsItemCollapse.js（摘录）

```
var ps=section.getElementsByTagName('p');
for(var i=0;i<ps.length;i++){
    DOMhelp.cssjs(action,ps[i],newshl.hideClass);
}
DOMhelp.cssjs(sectionAction,section,newshl.currentClass);
```

通过调用`preventDefault()`阻止最初单击的链接被打开，并调用`stopPropagation()`方法禁止事件冒泡。

newsItemCollapse.js (摘录)

```
e.preventDefault();
e.stopPropagation();
},
```

hover方法通过parentNode获取列表项并检查用来调用这个方法的事件类型。如果事件是mouseout, 定义动作为remove; 否则的话定义为add。然后应用这个CSS类或从列表项中删除这个类。

newsItemCollapse.js (摘录)

```
hover:function(e){
    var hl=e.target.parentNode.parentNode;
    var action=e.type=='mouseout'?'remove':'add';
    DOMhelp.cssjs(action,hl,newshl.overClass);
}
```

最后, 给window对象添加一个事件监听程序, 它会在窗口完成加载后触发newshl.init()方法。

newsItemCollapse.js (摘录)

```
}
```

```
window.addEventListener ('load',newshl.init,false);
```

现在你知道了在DOM-2兼容的浏览器中如何处理单击后的变化了。现在该考虑一下其他的浏览器, 确保这些功能也可以得到支持。

5.2.2 修正事件以适应W3C不兼容的浏览器

现在你已了解了事件处理的原理(在许多新的浏览器中得到了支持, 希望在这些不兼容W3C的浏览器上也好用), 我们来看看这些违反统一标准的浏览器并学习一下如何来处理它们。

注解 这些帮助方法已经包含在DOMhelp.js里了; 如果你想不使用DOMhelp而直接使用这些方法, 那么可以在源代码中找到它们。

它们其中之一就是IE 6, 考虑到它的年龄和它要扮演的角色(浏览器和操作系统文件管理器), 实际上这也是可理解的。IE有它自己的事件模型, 因此等价的标准方法不仅命名不同而且行为表现也不同。

IE中使用了方法attachEvent()来代替addEventListener(); IE还在window.event中保留了一个全局的事件对象来代替为每个监听程序传递事件对象。

一位名叫Scott Andrew的开发人员提出了一个更方便的函数, 叫做addEvent(), 它解决了添加事件方式不同的问题:

```
function addEvent(elm, evType, fn, useCapture) {
// Cross-browser event handling for IE5+, NS6+ and Mozilla/Gecko
// By Scott Andrew
```

```

if (elm.addEventListener) {
    elm.addEventListener(evType, fn, useCapture);
    return true;
} else if (elm.attachEvent) {
    var r = elm.attachEvent('on' + evType, fn);
    return r;
} else {
    elm['on' + evType] = fn;
}
}

```

这个函数比`addEventListener()`多用了一个参数，这个参数代表元素本身。它检查是否支持`addEventListener()`，而且它如果可以用W3C兼容的方式绑定事件的话就返回`true`。

否则的话，它会检查是否支持`attachEvent()`（意味着使用的是IE），并试图用`attachEvent`绑定事件。注意对于事件`attachEvent()`不需要“on”前缀。对于既不支持`addEventListener()`也不支持`attachEvent()`的浏览器（如Mac平台上的IE），这个函数则把DOM-1属性指向了函数。在Mac平台的IE上，这样做会覆盖附加到这个元素上的其他事件，但至少它可以用了。

注解 关于如何改进`addEvent()`一直存在争议——例如，为了支持保留使用`this`来发送当前对象的选择——到目前为止已出现了许多灵活的解决方案。但由于它们每一个都存在不同的缺点，所以这里就不做详细介绍了，但如果你特别感兴趣，可以到网站`quirksmode.org`（http://www.quirksmode.org/blog/archives/2005/10/_and_the_winner_1.html）的`addEvent()`编码大赛页面上查看相关内容。

由于IE使用了一个全局事件，所以你不能依赖传送给监听程序的事件对象，需要编写一个不同的函数来获得被激活的元素。问题变得更加混乱了，因为`window.event`的属性和W3C事件对象中的任何一个都有所不同：

- 在IE中，`target`被`srcElement`取代了。
- `button`返回的值也不同。在W3C模型中，0表示左按钮，1表示中间按钮，2表示右按钮；可是IE左按钮返回1，右按钮返回2，中间按钮返回4。它同样也返回3，但是在左按钮和右按钮同时被按下时候，并且在三个按钮同时被按下时候返回7。

为了适应这些变化，可以使用下面的这个函数：

```

function getTarget(e){
    var target;
    if(window.event){
        target = window.event.srcElement;
    } else if (e){
        target = e.target;
    } else {
        target = null ;
    }
    return target;
}

```

或更简单一些，可以使用三元运算符：

```
getTarget:function(e){
    var target = window.event ? window.event.srcElement :
        e ? e.target : null;
    if (!target){return false;}
    return target;
}
```

Safari^①有一个令人讨厌的bug（或特性——这个无法确定）：如果你单击一个链接，它不把这个链接作为目标，而是把包含在链接里的文本节点作为目标。一个变通的办法就是检查这个元素的节点名是否真的是一个链接：

```
getTarget:function(e){
    var target = window.event ? window.event.srcElement : e ? e.target : null;
    if (!target){return false;}
    if (target.nodeName.toLowerCase() != 'a'){target = target.parentNode;}
    return target;
}
```

阻止默认的动作和事件冒泡也需要使其适应不同的浏览器实现。

- stopPropagation()不是IE中的一个方法，而是窗口事件的一个叫做cancelBubble的属性。
- preventDefault()也不是IE中的一个方法，而是一个叫做returnValue的属性。

这意味着你不得不编写自己的stopBubble()和stopDefault()方法：

```
stopBubble:function(e){
    if(window.event && window.event.cancelBubble){
        window.event.cancelBubble = true;
    }
    if (e && e.stopPropagation){
        e.stopPropagation();
    }
}
```

注解 Safari支持stopPropagation()方法，但是使用它却什么也做不了。这意味着它不会阻止事件继续冒泡，因此对于此没有很好的解决办法。希望这个问题会在未来的版本中得到解决。

```
stopDefault:function(e){
    if(window.event && window.event.returnValue){
        window.event.cancelBubble = true;
    }
    if (e && e.preventDefault){
        e.preventDefault();
    }
}
```

① 苹果公司的一款开源浏览器。——译者注

由于一般情况下都需要阻止这两件事情发生，所以把它们集中放到一个函数中可能会更有意义：

```
cancelClick:function(e){
    if (window.event && window.event.cancelBubble
        && window.event.returnValue){
        window.event.cancelBubble = true;
        window.event.returnValue = false;
        return;
    }
    if (e && e.stopPropagation && e.preventDefault){
        e.stopPropagation();
        e.preventDefault();
    }
}
```

使用这些帮助方法可以使你很容易处理事件和跨浏览器的问题，但是有一个例外：

Safari可以理解这个preventDefault()方法，但是它没有实现它应该完成的功能。因此，如果你给链接添加一个事件处理，并在监听程序的方法中调用cancelClick()，这个链接依然会被打开。

对于Safari这个问题的解决方法是，使用老版本的onevent语法另外添加一个哑函数来阻止这个链接被打开。你很快就会在脚本中看到这个修改。再看一下折叠标题的例子，把与DOM-2兼容的方法和属性替换为跨浏览器平台的帮助方法：

xBrowserListItemCollapse.js —— 在exampleXBrowserListItemCollapse.html中使用

```
newshl = {
    // CSS classes
    overClass:'over', // Rollover effect
    hideClass:'hide', // Hide things
    currentClass:'current', // Open item

    init:function(){
        var ps,i,hl;
        if(!document.getElementById || !document.createTextNode){return;}
        var newsList = document.getElementById('news');
        if(!newsList){return;}
        var newsItems = newsList.getElementsByTagName('li');
        for(i = 0;i<newsItems.length;i++){
            hl = newsItems[i].getElementsByTagName('a')[0];
            DOMhelp.addEvent(hl,'click',newshl.toggleNews,false);
            hl.onclick = DOMhelp.safariClickFix;
            DOMhelp.addEvent(hl,'mouseover',newshl.hover,false);
            DOMhelp.addEvent(hl,'mouseout',newshl.hover,false);
        }
        var ps = newsList.getElementsByTagName('p');
        for(i = 0;i<ps.length;i++){
            DOMhelp.cssjs('add',ps[i],newshl.hideClass);
        }
    }
}
```

```

    }
},
toggleNews:function(e){
    var section = DOMhelp.getTarget(e).parentNode.parentNode;
    var first = section.getElementsByTagName('p')[0];
    var action = DOMhelp.cssjs('check',first,newshl.hideClass)?'remove':'add';
    var sectionAction = action == 'remove'? 'add': 'remove';
    var ps = section.getElementsByTagName('p');
    for(var i = 0;i<ps.length;i++){
        DOMhelp.cssjs(action,ps[i],newshl.hideClass);
    }
    DOMhelp.cssjs(sectionAction,section,newshl.currentClass);
    DOMhelp.cancelClick(e);
},
hover:function(e){
    var hl = DOMhelp.getTarget(e).parentNode.parentNode;
    var action = e.type == 'mouseout'? 'remove':'add';
    DOMhelp.cssjs(action,hl,newshl.overClass);
}
}
DOMhelp.addEvent(window,'load',newshl.init,false);

```

注解 `hl.onclick = DOMhelp.safariClickFix;` 可以简化为 `hl.onclick = function(){return false;};` 可是，一旦 Safari 开发组把这个问题排上了日程，搜索和替换这个修改会更容易些。

可以单击的标题现在在所有现代的浏览器上都可以工作了；然而，看上去可以使这个脚本更简化一些。现在的例子作了许多循环，实际上并不真的有必要。可以不用在列表项中单独地隐藏所有的段落，只要给列表项添加一个类，让 CSS 引擎来隐藏所有的段落就会更容易：

`listItemCollapseShorter.css`（摘录）——在 `exampleListItemCollapseShorter.html` 中使用

```

#news li.hide p{
    display:none;
}
#news li.current p{
    display:block;
}

```

使用这种方式可以去除 `init()` 方法中的遍历所有段落的内部循环，并且使用一行代码来替换它，这行代码把 `hide` 类应用到了列表项上，如下：

`listItemCollapseShorter.js`（摘录）——在 `exampleListItemCollapseShorter.html` 中使用

```

newshl={
    // CSS classes
    overClass:'over', // Rollover effect
    hideClass:'hide', // Hide things
    currentClass:'current', // Open item
}

```

```

init:function(){
    var hl;
    if(!document.getElementById || !document.createTextNode){return;}
    var newsList=document.getElementById('news');
    if(!newsList){return;}
    var newsItems=newsList.getElementsByTagName('li');
    for(var i=0;i<newsItems.length;i++){
        hl=newsItems[i].getElementsByTagName('a')[0];
        DOMhelp.addEvent(hl,'click',newshl.toggleNews,false);
        DOMhelp.addEvent(hl,'mouseover',newshl.hover,false);
        DOMhelp.addEvent(hl,'mouseout',newshl.hover,false);
        hl.onclick = DOMhelp.safariClickFix;
        DOMhelp.cssjs('add',newsItems[i],newshl.hideClass);
    }
},

```

下一个修改是在toggleNews()方法中。在那里可以使用一个简单的if条件来代替循环，它会检查当前的类是否被应用到列表项上，如果应用了，则使用hide来代替current；如果没有，则使用current来代替hide，这样就可以显示或隐藏列表项中的所有段落了。

listItemCollapseShorter.js（摘录）——在exampleListItemCollapseShorter.html中使用

```

toggleNews:function(e){
    var section=DOMhelp.getTarget(e).parentNode.parentNode;
    if(DOMhelp.cssjs('check',section,newshl.currentClass)){
        DOMhelp.cssjs('swap',section,newshl.currentClass,
                      newshl.hideClass);
    }else{
        DOMhelp.cssjs('swap',section,newshl.hideClass,
                      newshl.currentClass);
    }
    DOMhelp.cancelClick(e);
},

```

剩下的部分仍然是一样的：

listItemCollapseShorter.js（摘录）——在exampleListItemCollapseShorter.html中使用

```

hover:function(e){
    var hl = DOMhelp.getTarget(e).parentNode.parentNode;
    var action = e.type == 'mouseout'?'remove':'add';
    DOMhelp.cssjs(action,hl,newshl.overClass);
}
}
DOMhelp.addEvent(window,'load',newshl.init,false);

```

5.2.3 永不停止优化

分析你的代码以判断哪些地方可以优化，这种工作应该永远不要停止，即使在你热烈编码并

创建更复杂功能的时候也不例外。

往后退一步，分析一下你需要解决的问题，并重新评估一下已完成的东西，有时候比只是不断地编写代码更有好处。在这个例子中，优化就在于把隐藏元素留给了CSS的层叠，而不是循环遍历所有的元素并逐一地隐藏它们。

当你再看一遍以前创建的代码的时候，下面的几个要点需要你记住：

- 任何可以避免嵌套循环的思想都是好的。
- 主要对象的属性能很好地存储对几个方法都重要的信息——例如，站点导航中被激活的元素。
- 如果你发现自己有很多代码重复了，那么可以创建一个新的方法来完成这个任务——如果将来需要修改这些代码，你只需要在一个地方修改就可以了。
- 不要过多使用节点树来访问。如果许多元素需要知道其他的元素，一旦查出了它就把它存到一个属性中。这样可以使代码更简短，因为类似contentSection的东西要比elm.parentNode.parentNode.nextSibling更简短一些。
- 一个很长的if和else语句列表作为一个switch/case块处理可能会更好一些。
- 如果一些代码将来很可能要修改，如Safari的stopPropagation()，那么最好把它们放到自己的方法中。下次你看到这些代码并发现这些表面上看起来没什么用处的方法，你会记得它们是要干什么的。
- 不要过分依赖于HTML。它一直是首先要修改的东西（尤其是在CMS中）。

5.2.4 页面加载问题及其解决方案

当开发人员开始大量使用CSS的时候，很快就会碰见许多令人厌烦的浏览器bug。其中之一就是无样式内容瞬间（flash of unstyled content），也就是大家所知道的FOUC（<http://www.bluerobot.com/web/css/fouc.asp>）。在应用样式之前，显示的网页会有一小段时间没有样式表。

我们现在面临着和JavaScript增强的网页同样的问题。如果加载折叠新闻项的例子，你会看到所有的新闻需要一个瞬间展开。这个瞬间是文档和它的所有依赖内容如图片和第三方内容完成加载过程需要的时间。

这个问题使追求完美效果的开发人员郁闷了很长一段时间，因为在页面和所有被包含的媒体如图片被加载的时候会触发onload事件。不过，许多聪明的DOM程序员集中智慧解决了这个问题。

Dean Edwards（高科技领域中需要记住的一个名字）编写了一个脚本，可以在页面完成加载所有的内容元素前执行代码，这样就可以实现没有跳跃的平滑界面。可以在<http://dean.edwards.name/weblog/2005/09/busted/>上测试他的解决方案。

这种解决办法的问题是它严重依赖于特定的浏览器代码而且在Safari和Opera上不起作用。这个方法未来可能会修改，所以要不时访问一下Dean的网站。

还有一个不同的解决方案，它可以被所有启用了JavaScript的浏览器支持，但在结构、表现和行为的分离方面做得不够好。它通过document.write()把隐藏元素必需的CSS写入了文档的HEAD中。如果你想把这个应用到新闻标题的例子中，所要做的就是在HTML文档的头部添加这个列表

项的段落样式。

exampleListItemCollapseNoFlash.html（摘录）

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Example: Collapsing List Items without Flashing</title>
  <style type="text/css">
    @import 'listItemCollapseNoFlash.css';
  </style>
  <script type="text/javascript" src="DOMhelp.js"></script>
  <script type="text/javascript" src="listItemCollapseNoFlash.js"></script>
  <script type="text/javascript">
    document.write('<'+style type="text/css">');
    document.write('#news li p{display:none;}');
    document.write('<'+/style>');
  </script>
</head>
```

注解 样式标签的连接使用是非常有必要的，可以避免验证程序提示HTML不合法。

这两个版本的解决方案都是切实可行的，但都有点美中不足，到目前为止，我没有看到其他有关这个问题的更好解决方案。

5.2.5 读取和过滤键盘输入

最常用的Web事件可能要算是click了，因为它有所有元素都支持的优点而且可以同时用键盘和鼠标触发，如果这个正在处理的元素可以通过键盘访问到的话。

然而，并没有什么因素阻止我们在JavaScript中使用keyup或keypress处理器来检查键盘的输入。前者是W3C标准；后者不在标准中，它发生在keydown和keyup之后，但是在浏览器中它得到了广泛的支持。

我们来编写一个如何读取和使用键盘输入脚本，它会检查在一个表单框中输入的数据是否全部是数字。在第2章中，你已检查过输入的内容，并把它转换为了数字，但是这次你要在输入的时候检查它而不是在表单提交之后。如果用户输入了一个非数字字符，脚本程序应该设置表单的提交按钮为不可用并给出错误信息提示。

来看一个只有一个输入框的简单HTML表单：

exampleKeyChecking.html

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

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Example: Checking keyboard entry</title>
<style type="text/css">
    @import 'keyChecking.css';
</style>
<script type="text/javascript" src="DOMhelp.js"></script>
<script type="text/javascript" src="keyChecking.js"></script>
</head>
<body>
<p class="ex">Keychecking example, try to enter anything but
numbers in the form field below.</p>
<h1>Get Chris Heilmann's book cheaper!</h1>
<form action="nothere.php" method="post">
<p>
    <label for="Voucher">Voucher Number</label>
    <input type="text" name="Voucher" id="Voucher" />
    <input type="submit" value="redeem" />
</p>
</form>
</body>
</html>

```

在应用了下面的脚本后，浏览器会检查用户输入的内容，并在输入的内容不是一个数字的时候，给出一个错误提示信息，设置提交按钮为不可用，如图5-7所示。

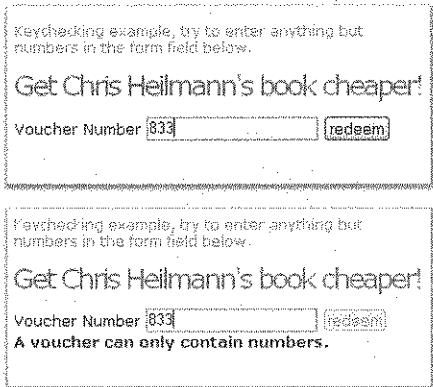


图5-7 在按键的时候检查输入的内容

keyChecking.js

```

voucherCheck={
    errorMessage:'A voucher can contain only numbers.',
    error:false,
    errorClass:'error',
    init:function(){
        if (!document.getElementById || !document.createTextNode) { return; }
        var voucher=document.getElementById('Voucher');
    }
}

```

```

if(!voucher){return;}
voucherCheck.v=voucher;
DOMhelp.addEvent(voucher, 'keyup', voucherCheck.checkKey, false);
},
checkKey:function(e){
if(window.event){
var key = window.event.keyCode;
} else if(e){
var key=e.keyCode;
}
var v=document.getElementById('Voucher');
if(voucherCheck.error){
v.parentNode.removeChild(v.parentNode.lastChild);
voucherCheck.error=false;
DOMhelp.closestSibling(v,1).disabled='';
}
if(key<48 || key>57){
v.value=v.value.substring(0,v.value.length-1);
voucherCheck.error=document.createElement('span');
DOMhelp.cssjs('add', voucherCheck.error, voucherCheck.errorClass);
var message = document.createTextNode(voucherCheck.errorMessage)
voucherCheck.error.appendChild(msg);
v.parentNode.appendChild(voucherCheck.error);
DOMhelp.closestSibling(v,1).disabled='disabled';
}
}
}
DOMhelp.addEvent(window, 'load', voucherCheck.init, false);

```

首先，你需要定义一些属性，如一个错误信息、一个表示是否已显示了错误信息的布尔值以及一个应用到错误信息上的类。需要对必需的元素进行检查，并附加一个指向checkKey()方法的keyup事件。

keyChecking.js（摘录）

```

voucherCheck={
errorMessage:'A voucher can only contain numbers.',
error:false,
errorClass:'error',
init:function(){
if (!document.getElementById || !document.createTextNode) { return; }
var voucher=document.getElementById('Voucher');
if(!voucher){return;}
voucherCheck.v=voucher;
DOMhelp.addEvent(voucher, 'keyup', voucherCheck.checkKey, false);
},

```

checkKey方法会判断是否使用了window.event或事件对象并且会以适当的方式读取keyCode。

keyChecking.js (摘录)

```
checkKey:function(e){
    if(window.event){
        var key = window.event.keyCode;
    } else if(e){
        var key=e.keyCode;
    }
}
```

然后它获得这个元素（在这里我们通过getElementById()来获得，虽然你也可以使用DOMhelp.getTarget(e)轻松获得，但是没有必要使用复杂的，够用就可以了），并且检查错误属性是否为true。如果它为true，那么就已经存在一个可见的错误信息并且提交按钮是不可用的。在这种情况下，你需要删除错误信息，设置error属性为false，并且设置提交按钮可用（它是input元素的下一个兄弟节点——这里使用closestSibling()可以确保它是一个按钮而不是一个换行）。

keyChecking.js (摘录)

```
var v=document.getElementById('Voucher');
if(voucherCheck.error){
    v.parentNode.removeChild(v.parentNode.lastChild);
    voucherCheck.error=false;
    DOMhelp.closestSibling(v,1).disabled='';
}
```

需要确定按下的键不是0~9的任何一个阿拉伯数字——也就是说，它的ASCII码不在48~57之间。

提示 可以在任一的ASCII码表中获得每个键的值，例如，在网站 <http://www.whatasciicode.com/> 上。

如果这个键不是一个数字键，那么删除域值中输入的最后一个键，并创建一个错误信息。创建一个新的span元素、添加应用errorClass类、添加错误信息、把它作为一个新的子节点追加到文本输入框的父节点上并且设置表单的按钮为不可用。剩下的最后一件事情就是在页面完成加载的时候启动voucherCheck.init()方法。

keyChecking.js (摘录)

```
if(key<48 || key>57){
    v.value=v.value.substring(0,v.value.length-1);
    voucherCheck.error=document.createElement('span');
    DOMhelp.cssjs('add',
        voucherCheck.error,
        voucherCheck.errorClass);
    var message = document.createTextNode(voucherCheck.errorMessage)
    voucherCheck.error.appendChild(msg);
    v.parentNode.appendChild(voucherCheck.error);
    DOMhelp.closestSibling(v,1).disabled='disabled';
```

```

        }
    }
}

DOMhelp.addEvent(window, 'load', voucherCheck.init, false);

```

注解 通常对于每一个keyup事件检查域中输入的内容是否为一个数字已经足够了，但是这个例子给你示范了使用键盘事件的威力。

如果想读取使用了Shift、Ctrl或Alt的组合键，那么需要在你的事件监听方法中检查事件属性shiftKey、ctrlKey或altKey：

```

if(e.shiftKey && key==48){alert('shift and 0');}
if(e.ctrlKey && key==48){alert('ctrl and 0');}
if(e.altKey && key==48){alert('alt and 0');}

```

5.2.6 事件处理的危险

使用这些函数，可以监听用户引起的任何一个事件并作出反应。可以使导航条表现为翻转效果而不是链接的单击，你可以添加只在网页上有效的快捷键，而且你可以使任何元素都对鼠标的移动作出反应。

使用事件处理全部的内容，如导航内容、用户旅行线路以及使用表单和用户进行交互是非常吸引人的。问题在于，这到底是一件好事还是坏事？

这是因人而异的，有人可能会以为拖放式界面会是最好的，但是对于不能使用鼠标的用户怎么办？可以通过绑定事件使文档中的所有元素都是可交互的；可是，并不是所有的用户代理都会允许访问者不用鼠标就可以访问到元素。一个可以单击的标题对于键盘用户不是可用的，但是带有嵌入链接的标题是可以使用的，因为用户可以使用tab键来访问链接，但是不能使用它来访问标题。

基本的可访问性指南和法律都强调，如果需要使用DOM编程和HTML创建自己的富界面，你需要保留输入设备的独立性。

一个拖放式界面并没有什么过错，只要你也允许使用键盘访问它就可以。因为你不能依赖于JavaScript是可用的，所以对于可拖放的元素你需要真正的链接，它可以使用click事件，甚至键盘访问。

键盘事件处理也是一个问题。虽然大多数浏览器都支持keypress（在W3C规范中没有提到它，它们更喜欢使用keyup），但你永远无法知道你要指派给一个元素的快捷键对于用户电脑上的另一款软件是否是必需的。

键盘的访问是操作系统必不可少的一部分，需要使用某种组合键的访问者在工作中就会劫持你的这些组合键。因此灵活的网络应用程序都使得键盘快捷键是可以选择的甚至是可被用户定制的。

当使用accesskey属性的时候，在HTML中也会发生同样的问题。这个属性会告诉浏览器当定义在属性值中的键被按下的时候激活元素（在IE和Mozilla中要和Alt键一起使用，而在其他的一

些浏览器中则使用其他的组合键)。实质上,这样就是添加一个事件并指派一个事件监听程序,从而设置元素的焦点或者打开元素的默认动作。一直以来,对这些属性使用数字键一直是非常常见的而且被认为是安全的,除非有一个用户,他的名字中有特殊的字符而且需要使用Alt和这些字符的ASCII码来输入名字。

总而言之,富HTML和JavaScript界面还在研究中,可以在网站<http://www.mozilla.org/access/dhtml/>上看一看,而且可以使用IBM和Mozilla开发的丰富控件。

5.3 小结

这一章已经学完了,希望大家不会感到内容太多而消化不了。

在前半部分,我们主要讨论了CSS与JavaScript的交互,涉及了下面几个方面:

- 在JavaScript中如何通过样式集合修改表现层。
- 如何通过把脚本的风格代码放到CSS类中来帮助CSS设计师。
- 如何给CSS设计师提供方法以根据脚本是否启用为文档而设置不同的样式。
- 介绍了不同的第三方样式切换,并说明了JavaScript脚本不是固定不变的,而应该随着时间的变化不断改进和优化的。
- 如何通过介绍只包括CSS名字信息的对象来简化CSS和JavaScript一起使用的维护工作。
- 使用JavaScript修正一个CSS问题——在这一章的例子中,多列显示的高度不相同。
- 通过应用跨浏览器的悬停特效来帮助CSS设计师。
- 使用JavaScript创建许多HTML元素来支持CSS特效(而不是从头就使用JavaScript来实现这些特效)的危险。

我们接着学习了使网站可以单击的内容——换句话说,就是事件处理。我们讨论了:

- 如何通过DOM-1的onevent属性在老版本的浏览器中应用事件处理(如onclick、onmouseover等)。
- 在DOM-2规范中W3C关于事件定义了什么以及如何使用他们推荐的东西。
- 如何使不兼容的浏览器也可以实现同样的效果。
- 如何避免在页面没有完全加载的时候显示的问题。
- 如何处理键盘输入。
- 事件处理的危险。

这就是本章的全部内容了,现在你应该掌握了所有的工具,并且拥有了许多稳定的、易于维护的、平滑流畅的非常绝妙的JavaScript代码。在第6章中,我们会学习一些JavaScript最常见的用法并且使用它们开发尖端的方案来代替你在使用的老脚本。

第6章

JavaScript的常用对象： 图片和窗口

学习了前面几章后，我们已经掌握了JavaScript及其与CSS和HTML交互的知识。在本章中你会学习到有关JavaScript目前在Web上最常见的用途，我们还会分析几个例子。在这些例子中，你会了解到如何确保它们不与网页上的其他脚本发生冲突，我还会解释说明可能发生的问题。我们还会接触到一些功能，它们可以临时使用，但可能并不是最安全的选择。

注解 本章包含许多代码例子，会要求你在浏览器中打开其中的一些来测试其功能，因此如果你还没有在网站<http://www.beginningjavascript.com>下载本书的示例代码，那么现在最好去下载。

这里大多数的代码示例使用了DOM-2事件处理，比DOM-1的等价代码要复杂一些，但可以更好地和其他脚本一起工作，而且非常有可能它们在未来的浏览器中也可以正常运行。我再啰嗦一下，后面会重复地使用这些方法，这样你就会很快地掌握它们。

这些例子开发的时候也考虑到了维护性和灵活性。这意味着在下一阶段可能被不了解JavaScript的人修改的所有东西都保存在属性中，而且你可以在同一文档的几个部分中使用这些脚本的功能。这样也增加了一些代码的复杂度，但是它是符合大多数客户要求的可交付使用的代码。

6.1 图片与 JavaScript

图片的动态改变可能是用JavaScript来实现的第一个令人称奇的特效。当CSS还没有被浏览器支持的时候（而且更合理地说，是仍然在被定义的过程中），JavaScript是唯一可以在用户把鼠标移动到图片上或单击它的时候改变它的方式。在最近几年里，越来越多以前使用JavaScript完成的图片特效被可以使维护更容易的纯CSS解决方案给替代了。我们稍后会再讨论这些问题；现在，我们先来看一下JavaScript处理图片的要素。

6.1.1 图片编程基础

在JavaScript中，可以用两种方式来获取和修改图片：通过使用`getElementsByName()`和`getElementById()`的DOM-2方式，或一种更老的方式，即使用保存在`document`对象一个属性中的集合`images`。我们来看一个例子，它是带有一个照片列表的HTML文档：

```
<ul class="slides">
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

在JavaScript中可以使用这两种方式来获取需要处理的这些照片：

```
// Old DOM
var photosOldDOM=document.images;
// New DOM
var photos=document.getElementsByTagName('img');
```

这两种方法都会得到一个包含所有图片对象的数组，对于每一个对象，你都可以读取和操纵它们的属性。举个例子来说，你想知道第3个图片的可替换文本。所要做的就是读取这个对象的`alt`属性：

```
// Old DOM alt property
var photosOldDOM=document.images;
alert(photosOldDOM[2].alt);
// W3C DOM-2 alt attribute
var photos=document.getElementsByTagName('img');
alert(photos[2].getAttribute('alt'));
```

图片有好几个属性，其中的一些可能很显而易见，但是其他的你可能没有听说过。

- border：HTML中`border`属性的值；
- name：`img`标签的名字属性；
- complete：一个属性，如果图片已加载完成，则它是`true`（只读——你不能改变这个属性）；
- height：图片的高度（以像素为单位，返回一个整数）；
- width：图片的宽度（以像素为单位，返回一个整数）；
- hspace：图片周围的水平方向间隔；
- vspace：图片周围的垂直方向间隔；
- lowsrc：属性中定义的同样名字的图片预览；
- src：图片的URL地址。

可以使用这些属性来访问图片和动态改变图片。如果在浏览器中打开这个例子文档exampleImageProperties.html，你可以如图6-1所示那样读写示范图片的属性。

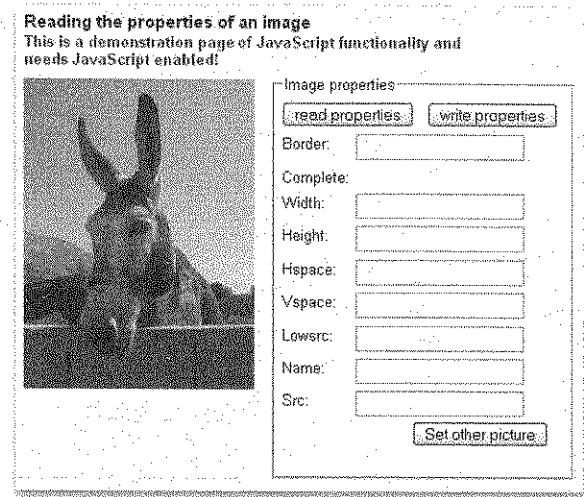


图6-1 读写图片的属性

注解 注意如果图片的尺寸已通过HTML的width和height属性定义了，然后改变它的源文件大小，那么不会自动改变它的大小。举个例子，激活例子中的Set other picture按钮。这样会导致其他图片的难看的变形，因为浏览器不会以一种特别复杂的方式去调整图片的大小。

6.1.2 预载图片

如果要在网页中动态地使用图片来实现翻转或幻灯片效果，需要把已加载过的图片放到浏览器的内存缓冲区中以给访问者一种平滑流畅的体验。可以以几种不同的方式来实现它。一种是在初始化页面的时候为每个想预载的图片创建一个新的图像对象：

```
kitten = new Image();
kitten.src = 'pictures/kittenflat.jpg';
```

你很快就会在6.1.3节中看到这样的一个例子。一些开发工具提供了遍历所有必需图片的脚本，如Macromedia的简单预载函数：

```
function simplePreload() {
    var args = simplePreload.arguments;
    document.imageArray = new Array( args.length );
    for(var i = 0; i < args.length; i++ ) {
        document.imageArray[i] = new Image();
        document.imageArray[i].src = args[i];
    }
}
```

如果使用你要预载的图片调用这个函数，它会创建一个新的数组，其中包含所有的图片，一个接一个地加载它们。例如：

```
simplePreload( 'pictures/cat2.jpg', 'pictures/dog10.jpg' );
```

一个与其不同的且不依赖于编程的预载图片的方式是，把所有的图片作为 1×1 像素的图片放到HTML的一个容器对象中，你可以通过CSS来隐藏它。这样会混合结构和行为且与其他图片预载技术一样存在相同的问题：你强迫访问者下载许多他可能不想立即看到的图片。如果你已使用了预载技术，那么让它们保持可选可能是个比较好的选择，如果用户想预载所有的图片，可以让用户自己决定。

我们这里对图片预载的介绍就简短点，因为关于图片还有很多东西要学。

6.1.3 翻转效果

当JavaScript在最常用的用户代理上开始获得广泛支持的时候，翻转或悬浮效果非常的狂热。许多脚本被编写出来，且出现了许多小工具，允许“不用编码就能即时生成翻转”。

翻转效果的思想相当的容易：你把鼠标悬停在一个图片上，图片会发生变化，表示这是一个可以单击的图片而不是只可以看得的。图6-2显示了一个翻转效果。



图6-2 翻转效果意思是，当鼠标悬停到它上面的时候元素会改变它的外观

1. 使用几个图片的翻转

可以通过在鼠标悬停到图片上的时候改变它的src属性来创建一个翻转效果。老式学院派翻转效果是附加到标签的name属性上并使用images集合。这样的构造函数在20世纪90年代的网页中并不少见：

`exampleSimpleRollover.html (摘录)`

HTML:

```
<a href="contact.html"
  onmouseover="rollover('contact','but_contact_on.gif')"
  onmouseout="rollover('contact','but_contact.gif')">
  
</a>
```

JavaScript:

```
function rollover( img, url ) {
  document.images[img].src=url;
}
```

翻转的问题过去（现在仍然）是第二个图片可能不会得到加载，这样就不会达到预期的目的。该图片作为一个可交互元素的事实就不是很明显——只有在第二个图片被显示的时候——因此在这种情况下，常让用户感到迷惑而不是帮助了他们。这就是为什么经典的翻转函数（如Macromedia Dreamweaver函数库上的那个）在连接name属性的时候使用了前面讲过的图像对象的预载技术：

examplePreloadingRollover.html (摘录)

HTML:

```
<a href="contact.html"
    onmouseover="rollover('contact',1)"
    onmouseout="rollover('contact',0)">
    
</a>
```

JavaScript:

```
contactoff = new Image();
contactoff.src = 'but_contact.gif';
contacton = new Image();
contacton.src = 'but_contact_on.gif';
function rollover( img, state ) {
    var imgState = state == 1 ? eval(img + 'on.src') : eval(img + 'off.src');
    document.images[img].src = imgState;
}
```

上面的脚本创建了两个新的图像对象，名字为contacton和contactoff。翻转图片的脚本检查了图片的名字和翻转的状态，并使用eval()获取正确的对象并读取它的src属性。接着把图片的src属性设置为从对象中获取的src属性。可以想象当你在一个页面中需要20个翻转图片的时候会拥有大量的代码，因此有必要提出一种更先进更通用的方法来创建翻转效果。

Daniel Nolan在2003年提出了一种非常巧妙的解决方案，其描述在网站http://www.dnolan.com/code/js/rollover/上可以找到。它的方案使用了图片的文件名和假设的一个翻转状态的前缀“_o”。你所要做的就是在希望有翻转效果的图片上添加一个叫imgover的类。

使用DOM-2处理器可以方便地重复同样的功能。首先你需要的就是一个HTML文档，它包含了指派了正确的类的图片：

exampleAutomatedRollover.html (摘录)

```
<ul>
    <li>
        <a href="option1.html">
            Option 1
        </a>
    </li>
```

```

<li>
  <a href="option2.html">
     Option 2
  </a>
</li>
[... code snipped ...]
</ul>

```

接着可以设计你的脚本。脚本的主要对象会被表示翻转的ro调用。由于你想让以后的维护者尽可能简单地维护，因此需要把可能发生变化的东西放到主对象的属性中。

在这个脚本中，下面是一个CSS类，它定义了哪一张图片该获得一个翻转状态以及鼠标经过图片的后缀。在这个例子中，你会分别使用“roll”和“_on”。需要两种方法，一个初始化效果，另一个来处理翻转。此外，你需要一个数组存储预载的图片。所有的这些放到一起构成了翻转脚本的大纲：

automatedRollover.js (大纲)

```

ro = {
  rollClass : 'roll',
  overSrcAddOn : '_on',
  preLoads : [],
  init : function(){}
, roll : function( e ){}
}
DOMhelp.addEvent( window, 'load', ro.init, false );

```

我们来充实一下这个大纲。首先是属性和init方法。在它里面预先定义了一个叫oversrc的变量并把文档中的所有图片都存储在一个叫imgs的数组中。循环遍历这些图片，并跳过那些没有附加正确CSS类的图片。

automatedRollover.js (摘录)

```

ro = {
  rollClass : 'roll',
  overSrcAddOn : '_on',
  preLoads : [],
  init : function() {
    var oversrc;
    var imgs = document.images;
    for( var i = 0; i < imgs.length; i++ ) {
      if( !DOMhelp.cssjs( 'check', imgs[i], ro.rollClass ) ) {
        continue;
      }
    }
}

```

如果绑定在图片上的是正确的CSS类，那么读取它的src属性，替换其中的句点为overSrcAddOn属性中定义的后缀加句点，并把结果保存到oversrc变量中。

automatedRollover.js (续)

```
oversrc = imgs[i].src.toString().replace( '.',➥
ro.overSrcAddOn + '.' );
```

注解 例如，文档中的第一个图片名字是src but_1.gif。加上这里定义的后缀属性的oversrc的值就是but_1_on.gif。

接着创建一个新的image对象并把它作为preLoads数组的一个新项存储起来。设置新图片的src属性为oversrc。使用DOMhelp库中的addEvent()方法为指向roll方法的mouseover和mouseout添加一个事件处理程序。

```
automatedRollover.js (续)
    ro.preLoads[i] = new Image();
    ro.preLoads[i].src = oversrc;
    DOMhelp.addEvent( imgs[i], 'mouseover', ro.roll, false );
    DOMhelp.addEvent( imgs[i], 'mouseout', ro.roll, false );
}
},
```

roll方法通过getTarget(e)重新获得事件发生的图片，并把它的src属性保存在一个叫s的变量中。接着通过读取事件的类型检查哪一个事件发生了。如果事件类型是mouseover，就将文件名中的句点替换为overSrcAddOn属性中定义的后缀加句点；如果事件为mouseout的话则做相反替换。可以为window对象添加一个事件处理，它在窗口完成加载后会调用ro.init()方法。

```
automatedRollover.js (续)
roll : function( e ) {
    var t = DOMhelp.getTarget( e );
    var s = t.src;
    if( e.type == 'mouseover' ) {
        t.src = s.replace( '.', ro.overSrcAddOn + '.' );
    }
    if( e.type == 'mouseout' ) {
        t.src = s.replace( ro.overSrcAddOn + '.', '.' );
    }
}
}
DOMhelp.addEvent( window, 'load', ro.init, false );
```

这个演示页面的执行结果如图6-3所示，当用户鼠标悬停到原始图片的时候，浏览器会使用已经加载到浏览器高速缓存中的高亮度图片来显示其翻转效果。

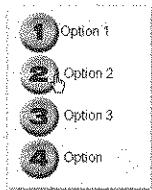


图6-3 预载且自动的翻转

可以试着使用更巧妙的脚本去预载图片，但它也可能不会一直工作。用户浏览器的高速缓存设置或他的网络连接上的特殊设置都可能使在后台悄悄进行的预载不会发生，它并没有真正在文档中添加图片。因此，使用一个单一的图片来实现翻转效果是个比较安全的选择。

2. 使用一张图片的翻转效果

当CSS设计师们开始探索使用: hover伪类选择器来做更多的事情，而不只是改变一个链接的下划线的时候，只使用CSS的翻转诞生了。这些主要意味着你可以为链接和链接的悬停状态指定不同的背景图片。

这样也会发生同样的问题——图片必须在显示出来前完成加载，否则会造成翻转效果出现闪烁或什么也没有发生。解决的方法是对于两个状态使用一张图片，并使用background-position属性来改变这个图片的位置，如图6-4所示。

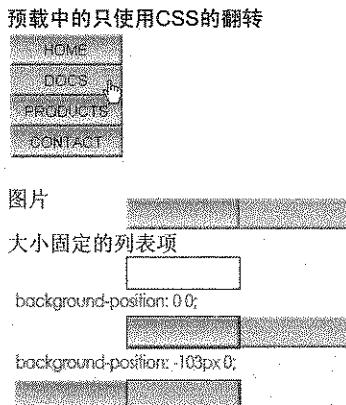


图6-4 使用背景位置和CSS的翻转效果

可以在浏览器中打开exampleCSSonlyRollover.html来看一下效果。CSS限制链接为一个固定的大小，并使用一个宽度为图片宽度一半的负background-position值，在悬停状态下通过移动背景图片来实现翻转效果：

exampleCSSonlyRollover.html（摘要）

```
#nav a{
    width:103px;
    padding-top:6px;
    height:22px;
    background:url(doublebutton.gif) top left no-repeat #ccc;
}
#nav a:hover{
    background-position:-103px 0;
}
```

可以使用JavaScript实现同样的效果；可是，我们要更有创造性，要使用JavaScript做一些CSS不能做到的效果。

3. 父元素的翻转效果

我们来拿一个HTML列表为例，通过给它添加一个好看的背景图片把它变成一个时髦的导航条，接着让链接在鼠标滑过它们的时候改变背景图片。你需要的第一个东西就是带有所有背景状态的一个背景图片，如图6-5所示。

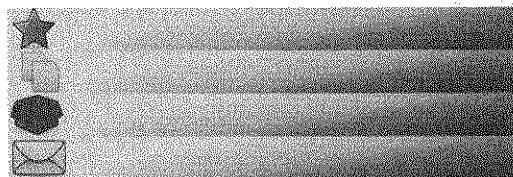


图6-5 具有所有背景状态的导航（调整大小后）

导航条的HTML是链接的一个列表。从基本的网站可用性的角度考虑，强烈建议不要链接当前页面，当前的链接使用一个``标签代替。

`exampleParentRollover.html`（摘要）

```
<ul id="nav">
<li><a href="index.html">Home</a></li>
<li><a href="documentation.html">Documentation</a></li>
<li><strong>Products</strong></li>
<li><a href="contact.html">Contact Us</a></li>
</ul>
```

可是，由于这个导航条可能是某个导航菜单的第一级，因此也有可能突出显示的不是一个`STRONG`元素而是一个作用于列表菜单项的类：

```
<ul id="nav">
<li><a href="index.html">Home</a></li>
<li><a href="documentation.html">Documentation</a></li>
<li class="current"><a href="products.html">Products</a></li>
<li><a href="contact.html">Contact Us</a></li>
</ul>
```

这两种情况都必须予以考虑。在演示的网页中解释CSS的用法并不是本书的目的；需要说明的是你可以通过修正ID为`nav`的列表的尺寸，让它浮动到左边，并浮动其中所有的列表元素。

相反，我们来直接看一下如何编制脚本。你需要为主对象（它叫做`pr`，代表父元素的翻转）定义几个属性：导航列表的ID、导航的高度（它也是每个图片的高度，且对背景图片的位置是必需的）、可能已被用来代替一个``标签突出显示当前项的可选类。

`parentRollover.js`（摘录）

```
pr = {
  navId : 'nav',
  navHeight : 50,
  currentLink : 'current',
```

刚开始需要一个初始化方法，它用来检查是否支持DOM以及对应ID的必需列表是否可用。

parentRollover.js (续)

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    pr.nav = document.getElementById( pr.navId );
    if( !pr.nav ){ return; }
```

下一步任务是循环遍历包含在这个列表中的所有列表菜单项，并检查这个菜单项中是否存在一个STRONG元素或有一个“当前”类。如果其中的一个为true，脚本就应该保存循环的计数到主对象的current属性中。这个属性会在翻转方法中被用来重新设置背景为原始的状态。

parentRollover.js (续)

```
var lis = document.getElementsByTagName( 'li' );

for(var i = 0; i < lis.length; i++)
{
    if( lis[i].getElementsByName( 'strong' ).length > 0 ||
        DOMhelp.cssjs('check', lis[i], pr.currentLink) ) {
        pr.current = i;
    }
}
```

每个列表项都获得一个叫index的新属性，它包括自己在整个列表数组中的计数。使用这个属性是个技巧，不用你去循环所有的列表项并使用事件监听程序方法中的目标来比较它们。

给roll()方法指定了两个事件处理程序：一个是当鼠标放在列表项上的时候，另一个是当鼠标离开列表项的时候。

parentRollover.js (续)

```
lis[i].index = i;
DOMhelp.addEvent( lis[i], 'mouseover', pr.roll, false );
DOMhelp.addEvent( lis[i], 'mouseout', pr.roll, false );
},
},
```

翻转方法roll()开始先预定义一个变量，叫做pos，它在后面会成为显示正确图片需要的偏移值。然后调用getTarget()方法来确定哪个元素被翻转了，并把这个目标的节点名和LI作比较。这是一个比较保险的方法，因为尽管你为LI指定了事件处理程序，但浏览器可能发送链接作为事件的目标。这个原因可能是链接是个可交互的页面元素，而LI不是，浏览器呈现引擎会认为一个链接更重要。你不会知道真正的原因，但是你应该知道事实上一些用户代理会把链接而不是列表元素看作为事件目标。

parentRollover.js (续)

```
roll : function( e ) {
    var pos;
    var t = DOMhelp.getTarget(e);
    while(t.nodeName.toLowerCase() != 'li'
        && t.nodeName.toLowerCase() != 'body') {
        t = t.parentNode;
    }
}
```

接下来，定义显示正确的背景图片所需要的位置。这个位置是列表项的index值或保存的current属性乘以每个图片的高度。

这两个中该应用哪一个取决于用户是否把鼠标放到列表项上，这个你通过比较事件类型和mouseover就可以发现。相应地设置导航条背景位置的样式，然后在页面完成加载后调用init()方法。

parentRollover.js (摘录)

```
pos = e.type == 'mouseover' ? t.index : pr.current;
pos = pos * pr.navHeight;
pr.nav.style.backgroundPosition = '0 - ' + pos + 'px';
}
}
DOMhelp.addEvent( window, 'load', pr.init, false );
```

当在浏览器中打开网页exampleParentRollover.html的时候，你可以看到滑过不同的导航条链接就会显示不同的背景图片，如图6-6所示。

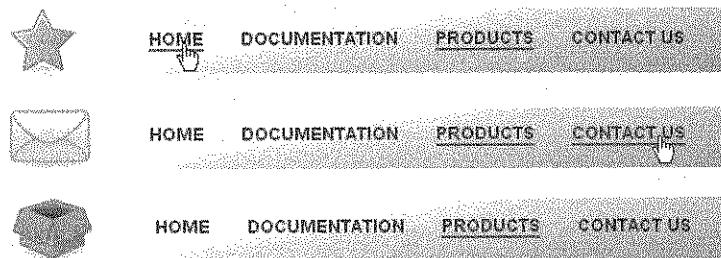


图6-6 导航条的不同翻转状态

这是翻转会影响父节点元素问题的程序解决方案。可是，它存在一个问题：如果菜单项的顺序发生了变化，维护者也必须相应地改变图片。这样的解决方案不是非常的灵活，这就是为什么给导航列表使用动态指派类来定位背景图片更好的原因了。

脚本的这个必要改变影响到属性和roll()方法；初始化的方法依然是一样的。除了currentLink和navId属性，你还需要一个添加到导航列表的类名。这个新的属性可以叫做dynamicLink。

在roll()方法中，需要再一次检查触发这个方法的事件是否mouseover，并相应地添加或删除

新的动态类。这个动态指定的类名是由dynamicLink属性值和当前的索引值相加构成的（因为把第一个类叫做item1比item0更容易使人们接受）：

parentCSSrollover.js 在 parentCSSrollover.html 中会用到（简写）

```
pr = {
  navId : 'nav',
  currentLink : 'current',
  dynamicLink : 'item',
  init : function() {
    // [... same as in parentRollover.js ...]
  },
  Roll : function( e ) {
    // [... same as in parentRollover.js ...]
    var action = e.type == 'mouseover' ? 'add' : 'remove';
    DOMhelp.cssjs( action, pr.nav, pr.dynamicLink + ( t.index + 1 ) );
  }
}
DOMhelp.addEvent( window, 'load', pr.init, false );
```

允许CSS设计师通过与定义CSS中的类一样的方式来定义翻转导航条的不同状态：

parentCSSrollover.css 在 parentCSSrollover.html 中会用到（摘录）

```
#nav.item1{
  background-position:0 0;
}
#nav.item2{
  background-position:0 -50px;
}
#nav.item3{
  background-position:0 -100px;
}
#nav.item4{
  background-position:0 -150px;
}
```

这种方法又为CSS设计师提供了一个设计导航条的陷阱：动态类可以被用作定义各菜单项都不同的链接自己的当前翻转状态或突出显示状态。

6.1.4 幻灯片显示

幻灯片显示是多张缩略图嵌在网页中，使用前一个和后一个按钮控制图片的改变，甚至有时候自动在一定时间后变换图片。它们被用来图解文本或提供一个产品的不同视图。

可以区分两种类型的幻灯片显示：一种是内嵌的，它把所有的图片都放到同样的文档中；另一个是动态的，它加载需要的所有图片。

1. 内嵌的幻灯片显示

可能最简单的在网页中添加一个幻灯片显示的方式就是把所有的图片作为一个列表进行添

加。然后可以使用JavaScript通过隐藏和显示不同嵌入图片的列表项把这个列表转变为幻灯片。演示文档examplePhotoListInlineSlideShow.html就是这样，如图6-7所示。

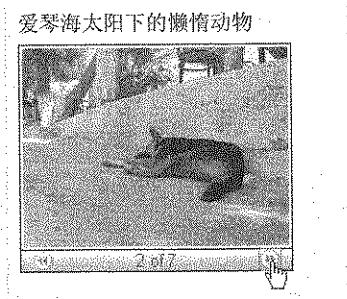


图6-7 使用JavaScript的内嵌幻灯片显示

下面的HTML是一个无序的列表，所有的图片都作为列表项。注意一下，这样还允许你为每一个图片设置一个适当的可替换文本。

```
examplePhotoListInlineSlideShow.html (摘录)
<ul class="slides">
  <li>
    
  </li>
  <li>
    
  </li>
  <li>
    
  </li>
  <li>
    
  </li>
  <li>
    
  </li>
  <li>
    
  </li>
  <li>
    
  </li>
</ul>
```

以后所有的维护人员要改变图片的顺序或添加、删除图片必须要做的就是改变HTML文档；根本没有必要去修改JavaScript。假设你应用了一个适当的样式表，不支持JavaScript的访问者会得到所有显示的图片，如图6-8所示。不支持样式表的用户会得到缩略图的一个列表。



图6-8 不使用JavaScript的内嵌幻灯片显示

嵌入所有图片到文档中的一个效果就是当访问者加载这个页面的时候所有的图片就会被加载。这可能是件好事，也可能是件坏事，取决于访问者的连接速度。稍后我们会看一个例子，只有在用户单击缩略图的时候才会加载大图片。

我们来看一下把图片列表转变成幻灯片显示的脚本。会使用到在前面几章中开发的DOMhelp库以解决浏览器的问题并使代码变得更精炼。

跟通常一样，第一件要做的事情就是设计脚本。在这个例子中，应该给出CSS设计师和HTML开发人员触发其功能或定义其界面外观的几个类：

- 一个类用来说明列表应该被转变成一个幻灯片显示。
- 一个类用来定义动态幻灯片显示列表的界面外观。
- 一个类用来显示以前隐藏的元素。
- 一个类用来定义图片计数的外观（例如，3个图片中的1个）。
- 一个类用来隐藏在某个播放状态不该显示在那里的元素。

也应该允许维护者改变外观、向前向后链接的内容以及图片计数的文本内容。

所需要的所有方法（除去DOMhelp中包括的帮助方法）有3个，分别是用来全局初始化的方法、用来初始化每个幻灯片显示的方法，以及显示一个幻灯片的方法。这些所有的东西放到一块就构成了脚本的大纲：

photoListInlineSlides.js (大纲)

```
inlineSlides = {
    // CSS classes
    slideClass : 'slides',
    dynamicSlideClass : 'dynslides',
    showClass : 'show',
    slideCounterClass : 'slidecounter',
```

```

hideLinkClass : 'hide',

// Labels
// Forward and backward links, you can use any HTML here
forwardsLabel : '' +
    alt="next" />',
backwardsLabel : '' +
    alt="previous" />',
// Counter text, # will be replaced by the current image count
// and % by the number of all pictures
counterLabel : '# of %',

init : function() {},
initSlideShow : function( o ) {},
showSlide : function( e ) {}
}
DOMhelp.addEvent( window, 'load', inlineSlides.init, false );

```

注解 注意这里为向前和向后链接在标签上提供了一个基于HTML的选项。这个是考虑到了幻灯片显示的更多灵活样式，因为维护者可以添加自己的HTML（如图片）。再者，如果想允许维护者改变像计数器这样的动态文本，那么使用像#和%这样的占位符，并解释它们将来会被什么替换是非常有益的。

我们逐个来看一下脚本中用到的这些方法。第一个是全局的初始化方法init():

- (1) 检查是否支持DOM。
- (2) 如果检查结果是成功的，则循环遍历文档中的所有UL元素。
- (3) 对于每一个UL元素，检查是否有类定义它为一个幻灯片显示（这个类保存在sideClass属性中），如果没有类定义它为一个幻灯片显示，则跳过由这个函数执行的其他步（使用continue来实现）。
- (4) 如果当前的UL元素要变为一个幻灯片显示，那么把定义它为幻灯片显示的类替换为定义动态幻灯片显示的类；在列表中添加一个新的属性currentSlide，并把它作为一个参数调用initSlideShow方法。

photoListInlineSlides.js（摘录）

```

init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var uls = document.getElementsByTagName( 'ul' );
    for( var i = 0; i < uls.length; i++ ) {
        if( !DOMhelp.cssjs( 'check', uls[i], -->
            inlineSlides.slideClass ) ) {

```

```

        continue;
    }
    DOMhelp.cssjs( 'swap', uls[i], inlineSlides.slideClass,➥
    inlineSlides.dynamicSlideClass );
    uls[i].currentSlide = 0;
    inlineSlides.initSlideShow( uls[i] );
}
},

```

使用这个方法可以减少许多循环和检查。首先，在JavaScript可用的情况下使用另一个类替换这个类，这允许你在CSS中隐藏所有的列表项而不用在initSlideShow()方法中循环遍历它们：

photoListInlineSlides.css（摘录）

```

.dynslides li{
    display:none;
    margin:0;
    padding:5px;
}

```

为了让幻灯片显示正常工作你还必须定义的其他动态指派CSS类是一个hide类和一个show类。hide类用来在第一张图片显示的时候移除向后的链接或最后一张图片显示的时候移除向前的链接；show类用来撤消使用.dynslides li选择器实现的隐藏效果。这个演示例子中的其他CSS选择器和属性都是起一个纯粹的修饰作用。

photoListInlineSlides.css（摘录）

```

.dynslides .hide{
    visibility:hidden;
}
.dynslides li.show{
    display:block;
}

```

通过把当前显示的图片保存在列表的一个属性中，就可以不必去循环遍历所有的图片，并在显示出当前图片前隐藏它们。你需要做的就是决定哪一张图片应该被显示出来，读取父列表元素的属性，并隐藏保存在这个属性中的前一张图片就可以了。

接下来重新设置这个属性为新的图片，下一次一张图片显示的时候，这个循环又重新开始了。你可能把当前照片保存到主要对象的一个属性中了，但把它保存在列表的一个属性中意味着你考虑到了同样的页面中可能存在好几个幻灯片显示。

initSlideShow()方法通过一个叫o的参数获取每个幻灯片的显示列表。首先，使用var关键字定义你会用到的变量，确保它们不会使用同样的名字覆盖全局变量。接着创建一个新的段落元素来保存向前和向后的链接以及图片计数器，并把它直接插入到列表的后面（使用o.nextSibling）。

photoListInlineSlides.js (续)

```
initSlideShow : function( o ) {
    var p, temp, count;
    p = document.createElement( 'p' );
    DOMhelp.cssjs( 'add', p, inlineSlides.slideCounterClass );
    o.parentNode.insertBefore( p, o.nextSibling );
```

再接下来，使用DOMhelp中的createLink方法创建向后的链接，并使用innerHTML来添加合适的标签。添加一个事件处理器来调用showSlide方法，通过应用适当的CSS类隐藏这个链接。把这个链接保存到列表的一个叫rew的属性中以方便以后访问它。

photoListInlineSlides.js (续)

```
o.rew = DOMhelp.createLink( '#', '' );
o.rew.innerHTML = inlineSlides.backwardsLabel;
DOMhelp.addEvent( o.rew, 'click', inlineSlides.showSlide, false );
DOMhelp.cssjs( 'add', o.rew, inlineSlides.hideLinkClass );
p.appendChild( o.rew );
```

下面是一个担当图片计数器的新的SPAN元素。获取主要对象的counterLabel属性，并使用当前列表的currentSlide属性值加上1来替换#字符（因为人们习惯从1开始计数，不像计算机那样从0开始计数）。使用列表中LI元素的数量替换%字符，并在把它作为一个新的子节点添加到段落前把结果字符串作为一个新的文本节点添加到SPAN上。

photoListInlineSlides.js (续)

```
o.count = document.createElement( 'span' );
temp = inlineSlides.counterLabel;
replace( /#/ , o.currentSlide + 1 );
temp = temp.replace( /%/ , o.getElementsByTagName( 'li' ).length );
o.count.appendChild( document.createTextNode( temp ) );
p.appendChild( o.count );
```

注解 注意到这里将计数器SPAN保存在列表的一个叫count的属性中。这是一种纯粹的偷懒省事的做法，因为它可以使你在后面不必通过getElementsByTagName('span')[0]来获取它。它同样也可以使脚本不易被后期有可能在列表项中添加其他span的维护者所破坏。

添加向前的链接和添加向后的链接类似，除了forwardsLabel属性被用作内容和一个叫fwd的属性被用作简写。

photoListInlineSlides.js (续)

```
o.fwd = DOMhelp.createLink( '#', '' );
o.fwd.innerHTML = inlineSlides.forwardsLabel;
DOMhelp.addEvent( o.fwd, 'click', inlineSlides.showSlide, false );
p.appendChild( o.fwd );
```

这个方法在结尾处，获取与currentSlide属性对应的列表项，并在它上面添加show类。你可能已使用了o.firstChild，但是未来的维护者可能想最初不显示第一张而显示另外一张不同的照片。

photoListInlineSlides.js (续)

```
temp = o.getElementsByTagName( 'li' )[o.currentSlide];
DOMhelp.cssjs( 'add', temp, inlineSlides.showClass );
},
```

showSlide()方法定义了一个叫action的变量，并使用getTarget(e)获取事件目标。因为你不知道维护者是否在链接标签中使用了图片，所以你需要通过检查目标的parentNode的nodeName是否是A来查找链接。这样也解决了Safari发送的是链接中的文本而不是链接本身的bug问题。这个方法接着通过读取目标的parentNode的closestSibling()来获取事件被触发所在的列表。

photoListInlineSlides.js (续)

```
showSlide : function( e ) {
    var action;
    var t = DOMhelp.getTarget( e );
    while( t.nodeName.toLowerCase() != 'a'
        && t.nodeName.toLowerCase() != 'body' ) {
        t=t.parentNode;
    }
    var parentList = DOMhelp.closestSibling( t.parentNode, -1 );
```

小结 访问者单击链接的内容来获得前一张或后一张图片。事件的目标可能是图片（如在这个例子中）或文本——或者是这个脚本的维护者放入forwardsLabel和backwardsLabel属性中的其他东西。也因为Safari是把发送的包含在链接中的文本（而不是链接本身）作为目标，因此你需要检查节点的名字并把它与A相比较。接着，获取这个A的父节点（新创建的段落）并获得它的前一个兄弟节点，也就是包括图片的UL。

接下来，需要从当前正讨论的列表中查找currentSlide属性，以及通过检查列表项数组的length属性来获得图片的数量。通过移除show类来隐藏前一张显示的图片。

photoListInlineSlides.js (续)

```
var count = parentList.currentSlide;
var photoCount = parentList.getElementsByTagName( 'li' ).length - 1;
var photo = parentList.getElementsByTagName( 'li' )[count];
DOMhelp.cssjs( 'remove', photo, inlineSlides.showClass );
```

通过比较目标和列表的fwd属性来判定这个被激活的链接是否为前一张的链接，然后相应地增加或减少计数。

如果这个计数器比0大，那么从向后的链接中移除hide类；否则的话添加这个类，这样可以很好地隐藏或显示这个链接。把同样的逻辑应用到向前的链接上，但这次比较的标准是计数器小

于列表项的总数。这样可以防止在第一张幻灯片上显示向后的链接以及在最后一张幻灯片上显示向前的链接。

```
photoListInlineSlides.js (续)

count = ( t == parentList.fwd ) ? count+1 : count-1;
action = ( count > 0 ) ? 'remove' : 'add' ;
DOMhelp.cssjs( action, parentList.rew,➥
  inlineSlides.hideLinkClass );
action = ( count < photoCount ) ? 'remove' : 'add';
DOMhelp.cssjs( action, parentList.fwd,➥
  inlineSlides.hideLinkClass);
```

要注意这些链接；现在你需要增加显示的计数。因为这个计数是作为列表的一个属性保存的，所以很容易读取那个属性的第一个子节点——它就是SPAN中的文本。接着可以使用String对象的replace()方法来用新的图片数字替换第一个数字输入（这里通过一个正则表达式），再次强调这里的图片数字是count+1，因为人类是从1开始计数而不是从0。接下来，重新设置currentSlide属性，获取新的照片（记住你已改变了count），并通过添加show类来显示当前的照片。所有剩下的就是在窗口加载完成后启动init()方法。

```
photoListInlineSlides.js (摘录)

photo = parentList.getElementsByTagName( 'li' )[count];
var counterText = parentList.count.firstChild
counterText.nodeValue = counterText.nodeValue.➥
  replace( /\d/, count + 1 );
parentList.currentSlide = count;
photo = parentList.getElementsByTagName( 'li' )[count];
DOMhelp.cssjs( 'add', photo, inlineSlides.showClass );
DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', inlineSlides.init, false );
```

然而，还没有彻底地完成。如果在Safari浏览器中试一下这个幻灯片显示，你会发现向前和向后的链接是隐藏了，但是它们仍然是可以单击的并且在你试着到达不存在的图片时将发生错误。

警告 这是动态网页开发中常见的一个错误，因此明显地隐藏东西没有必要让它们对于所有的用户都消失。要考虑到盲人或文本浏览器用户（比如Lynx），而且还有一些浏览器的bug和莫名其妙的东西也要考虑。

可是要防止这个问题非常的容易：你所要做的就是修正这个showSlide()方法，当被单击的目标有指定给它的hide CSS类的时候什么也不要。且当修改它的时候，你也可以添加Safari的修正来取消新产生链接的默认动作。演示例子examplePhotoListInlineSlideShowSafariFix.html对这些修改做了一下合并：

photoListInlineSlidesSafariFix.js

```

inlineSlides = {

    // CSS classes
    slideClass : 'slides',
    dynamicSlideClass : 'dynslides',
    showClass : 'show',
    slideCounterClass : 'slidecounter',
    hideLinkClass : 'hide',
    // Labels
    // Forward and backward links, you can use any HTML here
    forwardsLabel : '<br>
    alt="next" />',
    backwardsLabel : '<br>
    alt="previous" />',
    // Counter text, # will be replaced by the current image count
    // and % by the number of all pictures
    counterLabel : '# of %',

    init : function() {
        if( !document.getElementById || !document.createTextNode ) {
            return;
        }
        var uls = document.getElementsByTagName( 'ul' );
        for( var i = 0; i < uls.length; i++ ) {
            if( !DOMhelp.cssjs( 'check', uls[i], inlineSlides.slideClass ) ) {
                continue;
            }
            DOMhelp.cssjs( 'swap', uls[i], inlineSlides.slideClass, inlineSlides.dynamicSlideClass );
            uls[i].currentSlide = 0;
            inlineSlides.initSlideShow( uls[i] );
        }
    },
    initSlideShow : function( o ) {
        var p, temp, count;
        p = document.createElement( 'p' );
        DOMhelp.cssjs( 'add', p, inlineSlides.slideCounterClass );
        o.parentNode.insertBefore( p, o.nextSibling );
        o.rew = DOMhelp.createLink( '#', '!' );
        o.rew.innerHTML = inlineSlides.backwardsLabel;
        DOMhelp.addEvent( o.rew, 'click', inlineSlides.showSlide, false );
        DOMhelp.cssjs( 'add', o.rew, inlineSlides.hideLinkClass );
        p.appendChild( o.rew );
        o.count = document.createElement( 'span' );
        temp = inlineSlides.counterLabel;
        replace( '/#/ ', o.currentSlide + 1 );
        temp = temp.replace( '/%', o.getElementsByTagName( 'li' ).length );
    }
}

```

```

o.count.appendChild( document.createTextNode( temp ) );
p.appendChild( o.count );
o.fwd=DOMhelp.createLink( '#', '' );
o.fwd.innerHTML = inlineSlides.forwardsLabel;
DOMhelp.addEvent( o.fwd, 'click', inlineSlides.showSlide, false );
p.appendChild( o.fwd );
temp = o.getElementsByTagName( 'li' )[o.currentSlide];
DOMhelp.cssjs( 'add', temp,inlineSlides.showClass );
o.fwd.onclick = DOMhelp.safariClickFix;
o.rew.onclick = DOMhelp.safariClickFix;
},
showSlide : function( e ) {
    var action;
    var t = DOMhelp.getTarget( e );
    while( t.nodeName.toLowerCase() != 'a'
        && t.nodeName.toLowerCase() != 'body' ) {
        t = t.parentNode;
    }
    if( DOMhelp.cssjs( 'check', t,
        inlineSlides.hideLinkClass ) ){
        return;
    }
    var parentlist = DOMhelp.closestSibling( t.parentNode, -1 );
    var count = parentList.currentSlide;
    var photoCount = parentList.getElementsByTagName( 'li' ).length-1;
    var photo = parentList.getElementsByTagName( 'li' )[count];
    DOMhelp.cssjs( 'remove', photo, inlineSlides.showClass );
    count = ( t == parentList.fwd ) ? count + 1 : count - 1;
    action = ( count > 0 ) ? 'remove' : 'add' ;
    DOMhelp.cssjs( action, parentList.rew,➥
        inlineSlides.hideLinkClass );
    action = ( count < photoCount ) ? 'remove' : 'add';
    DOMhelp.cssjs( action, parentList.fwd,➥
        inlineSlides.hideLinkClass );
    photo = parentList.getElementsByTagName( 'li' )[count];
    var counterText = parentList.count.firstChild
    counterText.nodeValue = counterText.nodeValue.➥
        replace( /\d/, count + 1 );
    parentList.currentSlide = count;
    DOMhelp.cssjs( 'add', photo, inlineSlides.showClass );
    DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', inlineSlides.init, false );

```

把嵌入的图片列表转变为幻灯片显示，是一种会在非JavaScript的用户代理上退化的特效，尽管它并不是真正的图片处理或者是动态的。JavaScript的强大之处在于，可以避免页面重新加载并在同样的文档中显示大图而不只是在浏览器中显示它们。我们来看一些例子。

2. 动态的幻灯片显示

我们来拿另一个HTML列表来看一下动态幻灯片演示的例子。还是从这个HTML开始，这次列表中包含了链接到较大的图片的缩略图：

```
exampleMiniSlides.html
<ul class="minislides">
<li>
    <a href="pictures/thumbs/cat2.jpg">
        
    </a>
</li>
<li>
    <a href="pictures/thumbs/dog63.jpg">
        </a>
    </li>
<li>
    <a href="pictures/thumbs/dog7.jpg">
        
    </a>
</li>
<li>
    <a href="pictures/thumbs/kittenflat.jpg">
        
    </a>
</li>
</ul>
```

如果在启用了JavaScript的浏览器中打开这个例子，你会得到一栏缩略图和一个较大的图片。单击缩略图，这个缩略图指向的大图就会替换以前的大图，如图6-9所示。



图6-9 带有缩略预览图的幻灯片显示（缩略图）

没有JavaScript的访问者只会得到一行链接到大图的图片，如图6-10所示。



图6-10 不使用JavaScript的缩略图幻灯片显示

再一次，我们要设计一下脚本的大纲：定义一个类用来识别哪些列表要变成幻灯片显示、一个类用来识别包含大图的列表项，以及一个可替换的文本用来添加到大图上。

这些方法和上次的是一样的：一个全局的初始化方法、一个初始化每张幻灯片显示的方法，以及一个用来显示当前照片的方法。

miniSlides.js (大纲)

```
minislides = {
    // CSS classes
    triggerClass : 'minislides',
    largeImgClass : 'photo',
    // Text added to the title attribute of the big picture
    alternativeText : ' large view',

    init : function(){ },
    initShow : function( o ){ },
    showPic : function( e ){ }
}
DOMhelp.addEvent( window, 'load', minislides.init, false );
```

幻灯片显示的CSS非常简单：

miniSlides.css (摘录)

```
.minislides, .minislides * {
    margin:0;
    padding:0;
    list-style:none;
    border:none;
}
.minislides{
    clear:both;
    margin:10px 0;
    background:#333;
}
.minislides,.minislides li{
    float:left;
}
.minislides li img{
    display:block;
}
.minislides li{
    padding:1px;
}
.minislides li.photo{
    clear:both;
    padding-top:0;
}
```

首先，对列表中使用了正确类和列表本身的所有东西做了一个全局性的设置。一个全局性的设置意味着设置所有的页面外边距和内边距为 0 以及设置所有的边和列表样式为 none。这样可以预防不得不处理的跨浏览器平台的差别，也使得 CSS 文档更简短一些，因为你不必为每个元素重新设置这些值。

接着把所有的列表以及列表项浮动布局浮动到左边，让它们看上去是在一行的而不是一列。你需要设置这个主要列表布局为浮动以确保它包含其他的。

设置图片为块元素显示以避免他们周围的空隙，并对每个列表项添加 1 像素的内边距以显示背景颜色。

“照片”列表项需要一个浮动清除以看起来在其他的列表项下面。设置它的顶部内边距为 0 以避免在缩略图和大图片间有个双线。

`init()`方法功能类似于上一个幻灯片显示的初始化方法。检查它是否支持 DOM，循环遍历文档中的所有列表，并跳过那些没有对应类的列表。拥有对应类的列表项，作为一个参数发送到 `initShow` 方法中。

`miniSlides.js`（摘录）

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var lists = document.getElementsByTagName( 'ul' );
    for( var i = 0; i < lists.length; i++ ) {
        if( !DOMhelp.cssjs( 'check', lists[i], -->
            minislides.triggerClass ) ) {
            continue;
        }
        minislides.initShow( lists[i] );
    }
},
```

`initShow()`方法开始先创建一个新的列表项、一个新的图片，并对新的列表项指定大图的类。它把图片作为子节点添加到列表项上，把新的列表项作为子节点添加到主列表上。加之前面定义的 CSS 类，它会把新的图片显示到其他的下面。

`miniSlides.js`（摘录）

```
initShow : function( o ) {
    var newli = document.createElement( 'li' );
    var newimg = document.createElement( 'img' );
    newli.appendChild( newimg );
    DOMhelp.cssjs( 'add', newli, minislides.largeImgClass );
    o.appendChild( newli );
```

接着获取列表中的第一张图片并读取保存在 `alt` 属性中的它的可替换文本。把这个文本作为可替换文本添加到这个新图片上，把保存在 `alternativeText` 属性中的文本添加到它上面，并把获

得的字符串作为新的图片的title属性。

miniSlides.js（摘录）

```
var firstPic = o.getElementsByTagName('img')[0];
var alt = firstPic.getAttribute('alt');
newimg.setAttribute('alt', alt);
newimg.setAttribute('title', alt + minislides.alternativeText);
```

接下来，获取列表中的所有链接，并在用户单击每个链接的时候应用一个指向showPic的事件，把新的图片作为列表对象的一个叫做photo的属性保存起来，并设置新创建图片的src属性为第一个链接的目标位置。

miniSlides.js（摘录）

```
var links = o.getElementsByTagName('a');
for(i = 0; i < links.length; i++){
    DOMhelp.addEvent(links[i], 'click', minislides.showPic, false);
    links[i].onclick = function() { return false; } // Safari
}
o.photo = newimg;
newimg.setAttribute('src', o.getElementsByTagName('a')[0].href);
},
```

当单击这些链接指向的图片的时候，显示它们就是小事一桩。在这个showPic()方法中，通过getTarget()获取事件的目标，并通过读出列表的photo属性获取老图片。

这次你知道访问者单击的元素就是一张图片，这就是为什么你不需要循环和检查元素名字的原因。作为替换的方法，你需要向上查找3个父节点（A、LI以及UL）并读取以前保存的图片。接着设置可替换文本、标题以及图片的src并通过cancelClick()阻止了链接的默认动作。最后添加一个处理程序完成迷你幻灯片显示，它在窗口完成加载后触发init()方法。

miniSlides.js（摘录）

```
showPic : function( e ) {
    var t = DOMhelp.getTarget( e );
    var oldimg = t.parentNode.parentNode.parentNode.photo;
    oldimg.setAttribute('alt', t.getAttribute('alt'));
    oldimg.setAttribute('title', t.getAttribute('alt') + minislides.alternativeText);
    oldimg.setAttribute('src', t.parentNode.getAttribute('href'));
    DOMhelp.cancelClick( e );
}
DOMhelp.addEvent( window, 'load', minislides.init, false );
```

6.1.5 图片与JavaScript小结

总结一下有关图片与JavaScript的介绍。希望你没有被大量的特效及其实现方式和例子弄晕

了。要牢记这些幻灯片及其工作原理——在本章的末尾，我们会回到文档内嵌式幻灯片并使它可以自己播放。在第10章中，你还会看到如何开发一个更大的启用了JavaScript的图库。

图片与 JavaScript 记要

- 使用图片对象可以做许多的预载处理，但它不是一个万能的方法。浏览器高速缓存设置、坏掉的图片链接以及代理服务器可能会弄乱你的预载脚本。许多DHTML脚本在显示出主页面前都使用图片预载进度条。如果这些脚本执行失败了，用户所得到的就是一个不动的或一直变化的进度条，这样常会让人感到灰心丧气。
- 虽然你可以直接访问每个图片的属性，但它可能不是最好的方法——把可视的东西都留给CSS且你的脚本不能被不知道代码功能的其他人员修改。
- 自DHTML出现以来，CSS已经存在很长时间了，今天，通过提供钩子（预先定义功能指令）的方法，而不是单纯使用脚本去提供一种视觉效果的途径，CSS帮助CSS设计师找到了一种更好的设计方式——你前面看到的父元素翻转就是一个例子。

6.2 窗口与 JavaScript

产生新的浏览器窗口与改变当前的窗口是JavaScript常见的用法。这些用法也是非常烦人且不安全的，因为你永远无法确认网页的访问者是否可以处理调整了大小的窗口，或者当有一个新的窗口时是否会被其用户代理提示到。考虑一下要靠耳朵听你的网站的屏幕阅读器用户或文本浏览器用户。

窗口以前常被用作没有请求的广告（弹出广告）以及在隐藏的窗口中执行代码来达到获取数据的目的（网络钓鱼），这就是为什么浏览器制造商和第三方软件提供商提供了许多软件和浏览器设置来阻止这些滥用行为的原因。Mozilla Firefox用户可以选择是否想要弹出窗口以及使用JavaScript可以改变哪些窗口属性。如图6-11所示。



图6-11 Mozilla Firefox中的高级JavaScript设置

其他的一些浏览器如IE 7或Opera 8不允许隐藏新窗口的地址栏而且可以强制改变新窗口的大小和位置限制。

注解 这是不同的浏览器制造商为了阻止安全攻击而达成的协议中的一点。打开一个不带明显地址栏的新窗口会允许恶意攻击者在第三方网站上通过跨站脚本（Cross-Site Scripting，简写为XSS）打开一个弹出窗口，使这个窗口看上去属于这个第三方站点并要求用户输入信息。在网站Wikipedia: <http://en.wikipedia.org/wiki/XSS>上可以找到更多有关XSS的资料。

这对网络冲浪者来说是个好消息，因为他们可以阻止不想要的广告，且被欺骗而把他们的数据给不正当的人的可能性也更小。对你来说，就不是多么好的消息了，因为这意味着当你想在JavaScript中到处使用窗口的时候，你不得不检查许多不同的情景。

把这些需要考虑的事项放到一边，你仍然可以使用窗口和JavaScript做许多事情。每一个支持JavaScript的浏览器都提供给你一个叫做window的对象，在下一节中会列出它的属性，同样也会给出一些例子说明如何使用它们。

6.2.1 窗口属性

注解 下面的列表并没有显示所有可用的window属性，只是一些获得众多浏览器平台支持的属性（Mozilla/Firefox、Opera 7以及更高版本、IE 5.5以及更高版本和Safari）。如果给出的属性不被某个浏览器支持，我会在讨论的地方给出标注。

- ❑ closed: 布尔值，窗口关闭了或没有关闭（只读）；
- ❑ defaultStatus: 状态栏中默认的状态信息（在Safari浏览器中不支持）；
- ❑ innerHeight: 窗口的文档部分高度；
- ❑ innerWidth: 窗口的文档部分宽度；
- ❑ outerHeight: 整个窗口的高度；
- ❑ outerWidth: 整个窗口的宽度；
- ❑ pageXOffset: 窗口中文档的当前水平起始位置（只读）；
- ❑ pageYOffset: 窗口中文档的当前垂直起始位置（只读）；
- ❑ status: 状态栏的文本内容；
- ❑ name: 窗口的名字；
- ❑ toolbar: 一个属性，当窗口有一个工具栏的时候返回一个带有visible属性为true的对象（只读）。

举例，如果想获得窗口的内部文档大小，你可以使用一些其中的属性：

exampleWindowProperties.html（摘录）

```
function winProps() {
    var winWidth = window.outerWidth;
    var winHeight = window.outerHeight;
    var docWidth = window.innerWidth;
```

```

var docHeight = window.innerHeight;
var message = 'This window is ';
message += winWidth + ' pixels wide and ';
message += winHeight + ' pixels high.\n';
message += 'The inner dimensions are:';
message += docWidth + ' * ' + docHeight + ' pixels';
alert( message );
}

```

这个函数的一些可能输出如图6-12所示。

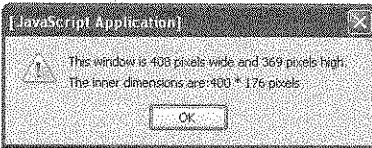


图6-12 读取窗口的属性

有一些其他的属性，它们不被IE以及Opera浏览器支持。它们是scrollbars、locationbar、statusbar、menubar以及personalbar。它们每一个都存储了一个带有只读visible属性的对象，visible属性的值为true或false。为了检查用户是否打开了菜单栏，可以检查这些对象和属性：

```

if ( window.menubar && window.menubar.visible == true ){
  // Code
}

```

6.2.2 窗口方法

window对象也有许多方法，其中的一些已在前面的章节中讨论过了。除了给用户提供反馈信息以外，最常用的函数还有打开新的窗口和定时执行功能。

1. 用户回馈方法

这里列出的用户回馈方法在第4章中已经详细讲解了。

- alert('message'): 显示一个警告信息。
- confirm('message'): 显示一个确认动作的对话框。
- prompt('message', 'preset'): 显示一个输入值的对话框。

2. 打开新窗口

打开和关闭窗口从技术上说是非常容易的；然而，考虑到各种浏览器禁止不同的属性和方法或者恶意编写的禁止弹出窗口的软件甚至会妨碍“友好的”弹出窗口，打开一个窗口就会变成一个噩梦而且需要合适的测试。打开新窗口的特性也非常容易，可以使用4个方法。

- open('url', 'name', 'properties'): 打开一个叫“name”的窗口，加载URL并设置窗口的属性。
- close(): 关闭窗口（如果窗口不是一个弹出窗口，这样会引起一个安全警告）。
- blur(): 把浏览器的焦点从当前窗口移开。

- focus(): 移动浏览器的焦点到当前窗口。

open方法中的properties字符串有一个非常特别的语法：它把窗口的所有属性当作字符串列出来，每一个属性是由一个名字和一个值通过等号连接起来，各个属性间是通过逗号分割的：

```
myWindow = window.open( 'demo.html', 'my', 'p1=v1,p2=v2,p3=v3' );
```

并不是这里所列的所有属性都会在所有的浏览器中得到支持，但是它们可以在大多数浏览器中使用。

- height: 以像素为单位定义了窗口的高度。
- width: 以像素为单位定义了窗口的宽度。
- left: 以像素为单位定义了窗口在屏幕上的水平位置。
- top: 以像素为单位定义了窗口在屏幕上的垂直位置。
- location: 定义了窗口是否含有一个地址栏（yes或no）——记住在未来的浏览器中这个属性会是一个固定的yes状态。
- menubar: 定义窗口是否有一个菜单栏（yes或no）——这个在Opera和Safari中不被支持（作为一个Mac工具，Safari在窗口上没有菜单栏但是在屏幕的顶部有）。
- resizable: 定义是否允许可以在窗口太小或太大的情况下调整窗口的大小。Opera不允许这个属性，因此Opera用户可以调整任意一个窗口。
- scrollbars: 定义窗口是否有滚动条（yes或no）。Opera不允许禁止滚动条。
- status: 定义窗口是否有一个状态栏（yes或no）。Opera不允许关闭状态栏。
- toolbar: 定义窗口是否有一个工具栏（yes或no）——关闭工具栏在Opera中不支持。

注解 其他一些只在Mozilla/Firefox上才有的属性是hotkeys（它会在一个给定的窗口中打开或关闭快捷键）、innerHeight与innerWidth（它定义了显示大小，而不是窗口大小的）以及dependent（它决定了打开它的窗口关闭的时候它是否被关闭）。后者在Linux平台的Konqueror浏览器上也被支持。

要在离屏幕左上角100像素的地方打开一个宽度和高度都为200像素的窗口，接着在它内加载文档grid.html，你必须按如下所示设置适当的属性：

```
var windowprops = "width=200,height=200,top=100,left=100";
```

可以在页面加载的时候试着打开这个窗口：

exampleWindowPopUp.html（摘要）

```
function popup() {
    var windowprops = "width=200,height=200,top=100,left=100";
    var myWin = window.open( "grid.html", "mynewwin", windowprops );
}
window.onload = popup;
```

注意各个浏览器上的执行结果有所不同。IE 6显示的窗口不带任何工具栏，IE 7显示地址栏，而Firefox和Opera会警告你页面正试图打开一个新的窗口，并询问你是否允许它这样做。

这和窗口通过一个链接打开处理上有点不同：

exampleLinkWindowPopUp.html

```
<a href="#" onclick="popup();return false">
  Open grid
</a>
```

现在既不是Opera也不是Firefox对弹出窗口存在抱怨。可以，如果JavaScript被禁用了，就不会是一个新的窗口了，而且链接也不再起作用了。因此，你可能想在href中链接到文档并把URL作为一个参数传递过去。

exampleParameterLinkWindowPopUp.html（摘录）

```
function popup( url ) {
  var windowprops = "width=200,height=200,top=100,left=100";
  var myWin = window.open( url, "mynewwin", windowprops );
}

<a href="grid.html" onclick="popup(this.href);return false">
  Open grid
</a>
```

注意一下window.open()方法的name参数。这个看上去对JavaScript没有意义，因为它什么也没做（例如，没有你可以通过windows.mynewwin用以访问这个窗口的windows集合）。可是，它被用作HTML中的target属性，以使链接在弹出窗口中（而不是主文档中）打开它们的链接文档。

在这个例子中，定义这个窗口的名字为“mynewwin”，并把链接作为目标打开http://www.yahoo.co.uk/：

exampleParameterLinkWindowPopUp.html（摘录）

```
function popup( url ) {
  var windowprops = "width=200,height=200,top=100,left=100";
  var myWin = window.open( url, "mynewwin", windowprops );
}

<a href="http://www.yahoo.co.uk/" target="mynewwin">Open Yahoo</a>
```

除非使用非过渡性的XHTML或严格型HTML（其中target已被废弃），你还可以使用值为blank的target属性来打开一个新的窗口，而不用管JavaScript是否可用。告诉访问者要在一个新的窗口中打开链接是个好习惯，可以避免混淆和可访问性问题：

```
<a href="grid.html" onclick="popup(this.href);return false" target="blank">
  Open grid (opens in a new window)
</a>
```

可是，由于你可能想使用严格型HTML和XHTML，且在使用JavaScript的时候对弹出窗口有许多控制，因此不依赖于target而且当且仅当脚本可用的时候把链接转变为弹出链接可能是一个

较好的解决方案。为此你需要一些东西可以识别这个链接为一个弹出链接。例如你可以使用一个叫popup的类。

exampleAutomaticPopupLinks.html (摘要)

```
<p><a href="grid.html" class="popup">Open grid</a></p>
<p><a href="http://www.yahoo.co.uk/" class="popup">Open Yahoo</a></p>
```

设计脚本并不需要太多：你需要触发弹出窗口的类、附加的文本、以及作为属性的窗口参数，一个init()方法来识别链接和变化、以及一个openPopup()方法来触发弹出窗口。

automaticPopupLinks.js (大纲)

```
poplinks = {
    triggerClass : 'popup',
    popupLabel : '(opens in a new window)',
    windowProps : 'width=200,height=200,top=100,left=100',
    init : function(){ },
    openPopup : function( e ){ },
}
```

这两个方法非常地基础。init()方法检查是否支持DOM并循环遍历文档中所有的链接。如果当前的链接使用了CSS触发类，则通过从标签创建一个新的文本节点把标签添加到链接上，并把它作为一个新的子节点添加到链接上。它在链接被单击的时候添加一个指向openPopup()方法的事件，并应用Safari修正来阻止链接在Safari浏览器中被跟随。

automaticPopupLinks.js (摘要)

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var label;
    var allLinks = document.getElementsByTagName( 'a' );
    for( var i = 0; i < allLinks.length; i++ ) {
        if( !DOMhelp.cssjs( 'check', allLinks[i], -->
            poplinks.triggerClass ) ) {
            continue;
        }
        label = document.createTextNode( poplinks.popupLabel );
        allLinks[i].appendChild( label );
        DOMhelp.addEvent( allLinks[i], 'click', -->
            poplinks.openPopup, false );
        allLinks[i].onclick = DOMhelp.safariClickFix;
    }
},
```

openPopup()方法获取事件的目标，确保它是一个链接，并通过使用事件目标的href属性作为URL、一个空的名字和保存在windowProps中的窗口属性来调用window.open()，从而打开一个新窗口。结束的时候调用cancelClick()方法阻止链接被跟随。

automaticPopupLinks.js (摘录)

```
openPopup : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    var win = window.open( t.getAttribute('href'), '', -->
        poplinks.windowProps );
    DOMhelp.cancelClick( e );
}
```

关于这个主题还有很多，尤其是考虑到可用性和易用性，以及确保弹出窗口只有在它们正在被打开并不被其他软件阻止或者其他打开失败时才使用；可是，更深的对这个问题的研究不在当前讨论的范围之内。只能说在今天的环境中不要轻易相信任何类型的弹出窗口。

3. 窗口交互

使用许多窗口的属性和方法，窗口可以和其他的一些窗口进行交互。首先有focus()和blur()：前者把弹出窗口放到最前面，后者把它放到当前窗口后面。

可以使用close()方法关闭窗口，通过window.opener属性访问打开弹出窗口的窗口。假设你已在一个主窗口打开了2个新的窗口：

```
w1 = window.open( 'document1.html', 'win1' );
w2 = window.open( 'document2.html', 'win2' );
```

可以通过调用blur()方法把第一个窗口隐藏到另一个的后面：

```
w1.blur();
```

注解 在在线广告中，打开一个没有请求的窗口并通过blur()立即把它隐藏起来叫做背后弹出广告 (pop-under)，它没有弹出窗口那么烦人，因为它们没有覆盖当前的页面。当你关闭浏览器的时候，如果你曾发现过几个自己不记得打开过的窗口，那就是背后弹出广告了。

可以通过调用窗口的close()方法关闭它：

```
w1.close();
```

如果想在弹出窗口的任意文档中获取最初的窗口，可以通过下列语句来实现

```
var parentWin = window.opener;
```

如果想从第一个窗口中获取第二个窗口，你同样也需要用到window.opener，因为第二个窗口是从这个窗口中打开的：

```
var parentWin = window.opener;
var otherWin = parentWin.w2;
```

注意你需要使用指定给这个窗口的变量名，而不是这个窗口的名字。

可以以这种方式使用任一窗口的任一窗口方法。举个例子来说，你想在document1.html中关闭w2；那么可以调用

```
var parentWin = window.opener;
var otherWin = parentWin.w2.close();
```

也可以调用主窗口的函数。如果主窗口有一个叫demo()的JavaScript函数，你可以在document1.html中通过下列语句来访问它。

```
var parentWin = window.opener;
parentWin.demo();
```

警告 如果你要通过window.opener.close()关闭原始的窗口，一些浏览器会给出一个安全警告窗口以询问用户他是否允许那样做。这是另外一个安全特性，它可以阻止网站所有者恶意欺骗一个不同的网站。许多用来关闭原始浏览器的设计代理支持一种预定义大小的窗口——这样就不会再出现前面提到的安全警告窗口，而且它是避免这类行为的一个好的办法，除非你想惊吓或骚扰用户。

4. 改变窗口的位置和尺寸

下面列表中的每个方法都有一个x和y参数。X表示水平的位置，y表示垂直的位置，它们的单位是像素，分别从屏幕的左边和上面计算。moveBy()、resizeBy()和scrollBy()方法允许负值，这样会移动窗口或内容到左边和上边或通过像素的数量使窗口变得小一点。

- moveBy(x,y)：移动了窗口x和y像素。
- moveTo(x,y)：移动窗口到坐标为x和y像素的地方。
- resizeBy(x,y)：调整了窗口大小x和y像素。
- resizeTo(x,y)：调整窗口的大小为x和y像素。
- scrollBy(x,y)：滚动了窗口的内容x和y像素。
- scrollTo(x,y)：滚动窗口的内容为x和y像素。

如果看一下示例文档exampleWindowPosition.html，你可以测试一下这些不同的方法，如图6-13所示。注意这个例子是在Firefox中执行的。在Opera 8或IE 7中，小窗口会有一个地址栏。

窗口位置和尺寸

这是一个JavaScript功能性的示范页面，
它需要启用JavaScript！

```
win>window.open()
win.moveBy(10,10)
win.moveTo(150,150)
win.resizeBy(10,10)
win.resizeTo(150,150)
win.scrollBy(10,10)
win.scrollTo(200,200)
```

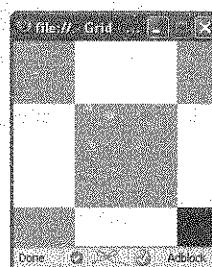


图6-13 改变窗口的位置和尺寸

5. 使用窗口时间间隔和超时设定的动画

可以使用`setInterval()`和`setTimeout()`窗口的方法来定时执行代码。`setTimeout`意味着在执行代码前你需要等待一定的时间（只一次）；`setInterval()`每次在过了给定的时间间隔后就执行一次代码。

- `name = setInterval('someCode', x)`: 每x毫秒执行一遍传递给它的JavaScript代码`someCode`。
- `clearInterval(name)`: 取消叫做`name`的时间间隔的执行（阻止组织代码的再次执行）。
- `name=setTimeout('someCode', x)`: 在等待x毫秒后，执行一次JavaScript代码`someCode`。
- `clearTimeout(name)`: 停止叫做`name`的超时，如果代码还没有被执行。

注解 `setInterval()`和`setTimeout()`方法中的参数`someCode`是一个字符串，可以是任何合法的JavaScript代码。通常，它只是简单地调用一个你在其他地方定义的函数。

使用这些方法的经典例子就是滚动新闻、时钟和动画了。

可是，也可以用它们来使你的网站更人性化。有关的一个例子就是在一定的时间后自动消失的警告消息。演示例子`exampleTimeout.html`说明了如何使用`setTimeout()`来显示一个显而易见的警告信息一段时间，或允许用户直接关闭它。这个HTML文档有一个段落警告用户网页已经过期：

`exampleTimeout.html`（摘录）

```
<p id="warning">This document is outdated  
and kept only for archive purposes.</p>
```

一个基本的样式表用来对非JavaScript用户把这个警告的颜色改变为红色并把它显示为粗体。对于启用了JavaScript的用户，添加一个动态的类使这个警告信息更显而易见。

`timeout.css`

```
#warning{  
    font-weight:bold;  
    color:#c00;  
}  
.warning{  
    width:300px;  
    padding:2em;  
    background:#fcc;  
    border:1px solid #c00;  
    font-size:2em;  
}
```

这2种情况的不同之处如图6-14所示。

用户可以单击“remove warning”链接以除去这个警告或者等待一下，它会在10秒后自动消失。

这个脚本非常简单：检查DOM是否被支持以及对应ID的警

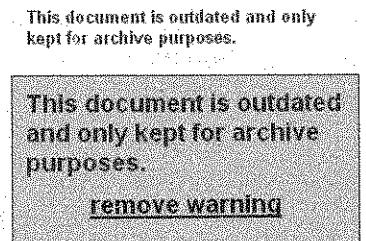


图6-14 没有启用及启用了JavaScript的警告信息

告消息是否存在。接着可以给这个消息添加动态的warning类并创建一个新的链接，这个链接带有一个指向removeWarning()方法的事件处理器。可以把这个链接作为一个新的子节点追加到警告信息上并定义一个超时时间，可以在10秒后自动触发removeWarning()方法。

timeout.js（摘录）

```
warn = {
    init : function() {
        if( !document.getElementById || !document.createTextNode ) {
            return;
        }
        warn.w = document.getElementById( 'warning' );
        if( !warn.w ){ return; }
        DOMhelp.cssjs( 'add', warn.w, 'warning' );
        var temp = DOMhelp.createLink( '#', 'remove warning' );
        DOMhelp.addEvent( temp, 'click', warn.removeWarning, false );
        temp.onclick = DOMhelp.safariClickFix;
        warn.w.appendChild( temp );
        warn.timer = window.setTimeout( 'warn.removeWarning()', 10000 );
    },
}
```

removeWarning()方法所要做的就是从网页文档中移除警告信息、清除超时设置并阻止链接的默认动作。

timeout.js（续）

```
removeWarning : function( e ){
    warn.w.parentNode.removeChild( warn.w );
    window.clearTimeout( warn.timer );
    DOMhelp.cancelClick( e );
}
}

DOMhelp.addEvent( window, 'load', warn.init, false )
```

首先应用这种特效的一个网络应用程序是Basecamp (<http://www.basecamphq.com/>)，它把网页文档最近的改变突出显示为黄色，随着页面的加载逐渐地淡出突出显示。可以在网站37signals上查看这个特效的基本原理 (<http://www.37signals.com/svn/archives/000558.php>)，在网站YourTotalSite (http://www.yourtotalsite.com/archives/javascript/yellowfade_technique_for/) 上可以查看一个JavaScript的演示。

在网站上使用超时设定是非常诱人的，因为它们会给用户一种动态站点的印象，并允许你从一个状态平缓地过渡到另一个状态。

提示 有几个可用的JavaScript特效库可以为你提供实现渐变和动画特效的预作脚本。它们大部分都已经过期了，但存在一些例外，如网站script.aculo.us (<http://script.aculo.us/>) 和FACE (<http://kurafire.net/projects/face>)。

可是，在你的网站中重新考虑使用许多动画和渐变可能是个好主意。记住代码是在用户的电脑上执行的，它取决于电脑有多旧或在处理其他任务有多忙，渐变和动画可能看上很不合适且会变成很讨厌的东西而不是一种富客户体验。

如果网站的功能依赖于动画的话，那它也可能引起可访问性问题，因为它们可能一些残疾人（认知缺陷、癫痫症）无法使用这个站点。508可访问性法律条款 (<http://www.section508.gov/>) 明确规定了软件的开发需要提供一个选项来关闭动画：

(h) 当动画显示的时候，信息应该根据用户的选择至少可以以一个非动画的表现模式显示。

—<http://www.section508.gov/index.cfm?FuseAction=Content&ID=12#Software>

可是，对于网站这个表述不是很明确。另一方面，W3C可访问性指南以2个优先级别明确表述了你应该避免在网页中的任何移动。

除非用户代理允许用户冻结移动内容，否则应该避免在网页里移动。

—<http://www.w3.org/TR/WCAG10-TECHS/#tech-avoid-movement>

我们来试着做一个允许用户启动和停止动画的例子。拿本章前面开发的内嵌式幻灯片显示来说，不再给它的向前和向后的链接，而是添加一个链接使用setInterval()来启动和停止自动的幻灯片演示。

HTML和CSS是保留不变的，但是JavaScript需要改变很多。

如果在浏览器中打开exampleAutoSlideShow.html，你会得到一个带有Play按钮的幻灯片，当单击这个按钮的时候，它就可以显示。在页面加载后你可以很容易地开始这个动画，但是把这个选择留给用户可不是个好主意。尤其是你需要遵守可访问性指南的情况下，因为没有请求的动画可能对身体有缺陷的用户（如癫痫症）会造成问题。一旦单击，这个按钮就变为一个Stop按钮，当它被激活的时候就会停止幻灯片显示。可以在图6-15中看出它在Firefox中的样子。

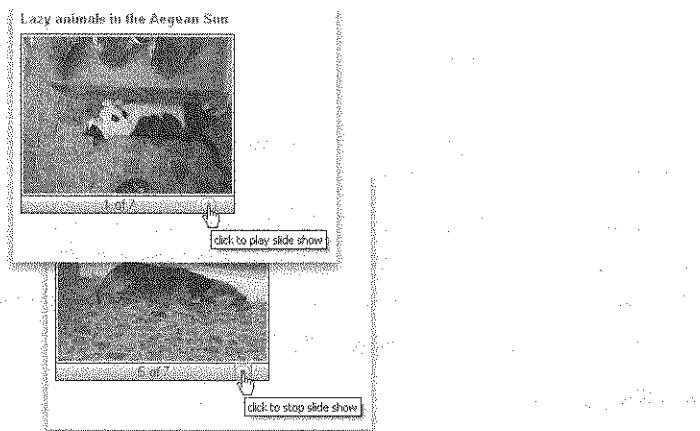


图6-15 一个自动地把播放按钮变换为停止按钮的幻灯片

开始是必需的CSS类，除了hide类，它们和第一个幻灯片显示示例中用到的是一样的。由于你这次不想隐藏任何按钮，所以没有必要用它了。

autoSlides.js (摘录)

```
autoSlides = {
    // CSS classes
    slideClass : 'slides',
    dynamicSlideClass : 'dynamicslides',
    showClass : 'show',
    slideCounterClass : 'slidecounter',
```

其他的属性有些改变；不再需要向前和向后的标签，现在你需要播放和停止标签。表示哪张照片当前正在播放的计数器仍然是一样的。一个新的属性是以毫秒为单位的幻灯片显示延迟时间。

autoSlides.js (续)

```
// Labels
// Play and stop links, you can use any HTML here
playLabel : '',
stopLabel : '',
// Counter text, # will be replaced by the current image count
// and % by the number of all pictures
counterLabel : '# of %',
// Animation delay in milliseconds
delay : 1000,
```

`init()`方法检查DOM是否支持并添加一个叫`slideLists`的新数组，它会存储要变为幻灯片显示的所有列表。它是能够告诉这个函数应该把变化应用到哪个列表上所必需的。

autoSlides.js (续)

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var uls = document.getElementsByTagName( 'ul' );
    autoSlides.slideLists = new Array();
```

首先，循环遍历文档中的所有列表并检查要把它们变为幻灯片显示的类。如果列表中使用了这个类，则初始化`currentSlide`属性为0并把循环计数器保存到一个新的列表属性`showCounter`中。再一次说明一下，这个计数是用来告诉列表要改变的间隔。可以把这个列表作为一个参数来调用`initSlideShow()`方法并把这个列表添加到`slideLists`数组。

autoSlides.js (续)

```
for( var i = 0; i < uls.length; i++ ) {
    if( !DOMhelp.cssjs( 'check', uls[i],
```

```

    autoSlides.slideClass ) ){
        continue;
    }
    DOMhelp.cssjs( 'swap', uls[i], autoSlides.slideClass,
    autoSlides.dynamicSlideClass );
    uls[i].currentSlide = 0;
    uls[i].showIndex = i;
    autoSlides.initSlideShow( uls[i] );
    autoSlides.slideLists.push( uls[i] );
}
},

```

`initSlideShow()`方法与你在`photoListInlineSlidesSafariFix.js`中使用的同名方法有太多的不同；唯一不同的地方就是你创建了一个链接（而不是两个）并把`playLabel`作为内容应用到新链接上：

`autoSlides.js` (续)

```

initSlideShow : function( o ){
    var p, temp ;
    p = document.createElement( 'p' );
    DOMhelp.cssjs( 'add', p, autoSlides.slideCounterClass );
    o.parentNode.insertBefore( p, o.nextSibling );
    o.play = DOMhelp.createLink( '#', '' );
    o.play.innerHTML = autoSlides.playLabel;
    DOMhelp.addEvent( o.play, 'click', autoSlides.playSlide, false );
    o.count = document.createElement( 'span' );
    temp = autoSlides.counterLabel.replace( '/#/ ', o.currentSlide + 1 );
    temp = temp.replace( '%', o.getElementsByTagName( 'li' ).length );
    o.count.appendChild( document.createTextNode( temp ) );
    p.appendChild( o.count );
    p.appendChild( o.play );
    temp = o.getElementsByTagName( 'li' )[o.currentSlide];
    DOMhelp.cssjs('add', temp, autoSlides.showClass );
    o.play.onclick = DOMhelp.safariClickFix;
},

```

`playSlide()`方法是新加的，但它开始和老的`showSlide()`方法是非常相似的。它检查目标和它的节点名，并获取它的父列表。

`autoSlides.js` (续)

```

playSlide : function( e ) {
    var t = DOMhelp.getTarget( e );
    while( t.nodeName.toLowerCase() != 'a'
        && t.nodeName.toLowerCase() != "body" ){
        t = t.parentNode;
    }
    var parentList = DOMhelp.closestSibling( t.parentNode, -1 );

```

检查父列表是否已经有一个叫做loop的属性。这是存储setInterval()实例的属性。你使用列表的一个属性而不是用一个变量以允许在同一个文档中存在不止一个自动播放的幻灯片显示。

定义一个在setInterval()中使用的字符串，它把父列表的showIndex属性作为一个参数来调用showSlide()方法。这是必需的，因为setInterval()是window对象的一个方法，且它不在主要的autoSlides对象的范围之内。

以在autoSlides.delay属性中定义的延迟为参数使用setInterval()方法，并在改变已被激活为Stop按钮的链接的内容前把它存储在loop属性中。

autoSlides.js（续）

```
if( !parentList.loop ) {
    var loopCall = "autoSlides.showSlide( '" + -->
        parentList.showIndex + "' )";
    parentList.loop = window.setInterval( loopCall, -->
        autoSlides.delay );
    t.innerHTML = autoSlides.stopLabel;
```

如果这个列表已经有一个叫loop的属性，这意味着幻灯片显示已经在运行；因此清除它，设置loop属性为null，并把按钮变回Play按钮。接着通过调用cancelClick()方法阻止默认的链接行为。

autoSlides.js（续）

```
} else {
    window.clearInterval( parentList.loop );
    parentList.loop = null;
    t.innerHTML = autoSlides.playLabel;
}
DOMhelp.cancelClick( e );
},
```

showSlide()方法改变得很彻底，但是你会看到其他方法最初中的一些混淆部分（如slideLists数组有什么好处）使这个方法相当容易。

记住你在playSlide()中定义了时间间隔，该时间间隔应该使用列表的showIndex属性作为参数来调用这个showSlide()方法。现在可以使用这个索引，通过从slideLists数组中检索列表，来获取你需要循环遍历的列表。

autoSlides.js（续）

```
showSlide : function( showIndex ) {
    var currentShow = autoSlides.slideLists[showIndex];
```

一旦获得了这个列表，就可以读出当前的幻灯片及其数量。从当前幻灯片上移除showClass类，从而把它隐藏起来。

autoSlides.js (续)

```
var count = currentShow.currentSlide;
var photoCount = currentShow.getElementsByTagName('li').length;
var photo = currentShow.getElementsByTagName('li')[count];
DOMhelp.cssjs( 'remove', photo, autoSlides.showClass );
```

给这个计数器增加1来显示下一张幻灯片。比较这个计数器和所有的幻灯片数量，如果最后一张幻灯片已经显示则把计数器设为0——这样可以在第一张幻灯片重新开始计数。

通过检索这个列表元素并给它添加show类来显示幻灯片。更新计数器并重新设置列表的currentSlide属性为新的列表元素。

autoSlides.js (续)

```
count++;
if( count == photoCount ){ count = 0 };
photo = currentShow.getElementsByTagName('li')[count];
DOMhelp.cssjs( 'add', photo, autoSlides.showClass );
var counterText = currentShow.count.firstChild;
counterText.nodeValue = counterText.nodeValue.replace( /\d/, count + 1 );
currentShow.currentSlide = count;
}
DOMhelp.addEvent( window, 'load', autoSlides.init, false );
```

这里只是略微展示了一下，在开发动画和代码的定时执行时，JavaScript开发人员将面对怎样的复杂性。在JavaScript中创建一个流畅的、稳定的、跨平台的动画是需要技巧的，且需要许多测试浏览器问题的知识。很幸运，已经有许多可用的动画库，它可以帮助你来完成这个任务并且在不同的操作系统和浏览器上经过了许多开发人员的稳定性测试。在第11章的例子中你会接触到其中的一个库。

6. 浏览器窗口的导航方法

下面是导航浏览器窗口的方法列表。

- **back()**: 后退到浏览器历史的上一个页面。(如果你使用了框架，它会返回到不带框架的最后一个文档，而不是框架里最后的改变。)
- **forward()**: 前进到浏览器历史的下一个页面。(如果你使用了框架，它会返回到不带框架的最后一个文档，而不是框架里最后的改变。)
- **home()**: 产生用户单击home按钮的效果(仅用在Firefox和Opera上)。
- **stop()**: 停止窗口中的文档加载(IE不支持)。
- **print()**: 启动浏览器的打印对话框。

使用这些方法来在页面上提供导航功能是相当诱人的，它可以简单地通过下面的方式链接回前一个页面：

```
<a href="javascript:window.back()">Back to previous page</a>
```

考虑到可访问性和现代脚本，这意味着没有JavaScript的用户会得到一个应该有但并不存在的东西。一个更好的解决方法是通过服务端包含（SSI）产生一个真正的“后退到上一个页面”链接或对于对应的文档提供一个真实的HTML链接。如果没有一个是可能的，那么可以使用一个占位符，当JavaScript可用的时候，把它替换为一个产生的链接，正如在下面的例子中那样：

```
exampleBackLink.html (通过exampleForBackLink.html来访问)
HTML:
<p id="back">Please use your browser's back button or
keyboard shortcut to go to the previous page</p>
JavaScript:
function backlink() {
    var p = document.getElementById( 'back' );
    if( p ) {
        var newa = document.createElement( 'a' );
        newa.setAttribute( 'href', '#' );
        newa.appendChild( document.createTextNode(
            'back to previous page' ) );
        newa.onclick = function() { window.back();return false; }
        p.replaceChild( newa, p.firstChild );
    }
}
window.onload = backlink;
```

警告 这些方法的威胁是，你提供的是浏览器已经为用户提供的功能。不同之处在于浏览器做得更好一些，因为它的确支持更多的输入设备。例如，在PC机上的Firefox，可以通过按键Ctrl+P打印文档，通过按键Ctrl+W关闭窗口或制表，通过Alt键和左右箭头键在浏览器历史上向前或向后移动。

更糟糕的是，这些方法提供的功能是依赖于脚本支持的。这就需要你来决定前面讲述的方法——创建调用这些方法的链接，可能是处理这个问题最巧妙的方式了——是否值得应用的特效，或是否应该允许用户来决定如何触发浏览器的功能。

7. 打开新窗口的一种替换：层广告

有时没有办法可以避免弹出窗口，因为站点的设计或功能上需要它们，而且由于前面讲到浏览器的问题和选项使它们不能正常工作。一个专业名词叫层广告（layer ad），它们主要是绝对定位放到主内容顶部的页面元素。

我们来看一个其中的例子。假设你的公司想在页面加载后对最新产品在非常明显的位置作广告。最容易的方式就是把信息添加到文档的结尾，并使用脚本把它转变为一个层广告，这样意味着不用JavaScript访问者也能得到这些广告信息，但是没有给他们机会来关掉页面上的广告。HTML是一个非常简单的带有ID的DIV（为了简化，真正的链接已经被“#”替换了）：

```
exampleLayerAd.html (摘录)
<div id="layerad">
<h2>We've got some special offers!</h2>
```

```

<ul>
  <li><a href="#">TDK DVD-R 8x 50 pack $12</a></li>
  <li><a href="#">Datawrite DVD-R 16x 100 pack $50</a></li>
  <li><a href="#">NEC 3500A DVD-RW 16x $30</a></li>
</ul>
</div>

```

CSS设计师可以设计非JavaScript版本的广告样式，并且脚本会添加一个类来允许广告显示在主内容之上。如果调用这个类dyn，CSS可能类似于下面的代码：

layerAd.css (摘录)

```

#layerad{
  margin:.5em;
  padding:.5em;
}
#layerad.dyn{
  position:absolute;
  top:1em;
  left:1em;
  background:#eef;
  border:1px solid #999;
}
#layerad.dyn a.adclose{
  display:block;
  text-align:right;
}

```

上面的选择器是该脚本要添加到广告上的一个动态链接的样式，允许用户移除它。

脚本本身没有包含任何奇特的东西。首先，定义广告的ID、动态类、作为属性的close链接的类与文本内容。

layerAd.js

```

ad = {
  adID : 'layerad',
  adDynamicClass : 'dyn',
  closeLinkClass : 'adclose',
  closeLinkLabel : 'close',
}

```

init()方法会检查DOM和广告，并在广告上添加动态的类。接着它创建一个新的链接，并给它添加“close”链接的文本和类。然后给这个链接添加一个指向killAd()方法的事件处理程序，并把这个新链接插入到广告的第一个子节点前。

layerAd.js (续)

```

init : function() {
  if( !document.getElementById || !document.createTextNode ) {
    return;
  }
}

```

```

ad.offer = document.getElementById( ad.adID );
if( !ad.offer ) { return; }
DOMhelp.cssjs( 'add', ad.offer, ad.adDynamicClass );
var closeLink = DOMhelp.createLink( '#', ad.closeLinkLabel );
DOMhelp.cssjs( 'add', closeLink, ad.closeLinkClass );
DOMhelp.addEvent( closeLink, 'click', ad.killAd, false );
closeLink.onclick = DOMhelp.safariClickFix;
ad.offer.insertBefore( closeLink, ad.offer.firstChild );
},

```

`killAd()`方法从文档中移除广告，并取消链接的默认行为。

`layerAd.js` (续)

```

killAd : function( e ) {
    ad.offer.parentNode.removeChild( ad.offer );
    DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', ad.init, false );

```

可以通过在浏览器中打开`exampleLayerAd.html`来测试这个特效；如果启用了JavaScript，你会看到广告覆盖了内容，如图6-16所示。可以通过使用`close`链接来关闭它。

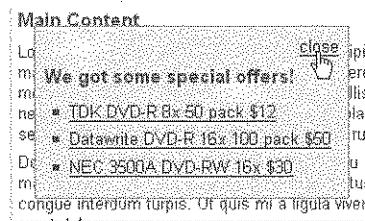


图6-16 层广告示例

弹出窗口的另一个常用的地方是不用离开当前页面来显示另一个文档或文件。经典的例子包括令人厌烦的条款和条件列表或照片列表。特别是在照片的例子中，弹出窗口并不是最佳的解决方案，因为你可以打开一个和照片尺寸大小相同的窗口，但是由于浏览器的内部样式在主体上应用了内边距设置而在图片的周围存在空隙。可以通过使用一个带有样式表的空白HTML文档来解决这个问题，主体的外边距和内边距设置为0，并通过JavaScript把图片添加到窗口的文档中。另一个可选的方案是把照片显示在一个新产生的已定位的元素中，它覆盖了主文档。演示例子`examplePicturePopup.html`就是那样做的；脚本所需要的就是在指向照片的链接上应用一个有确定名字的CSS类。

`examplePicturePopup.html` (摘录)

```
<a class="picturepop" href="pictures/thumbs/dog7.jpg">Sleeping Dog</a>
```

脚本需要做一些前面没有解释的东西，也就是读取元素的位置。通过绝对定位一个元素，你覆盖了主文档。由于你不知道指向照片的链接到底在文档的什么地方，因此需要读出它的位置并把照片显示到那里。

但上面是对于后者来讲的；对于第一个需要预先定义几个属性。你需要一个类来触发脚本，在照片显示的时候要还在链接上应用一个链接类，另一个类需要应用到包含照片的元素上。你还需要定义一个前缀和一个属性，前缀会在照片显示得时候添加到链接上，这个属性用作对新创建的元素的快捷引用。

picturePopup.js (摘录)

```
pop = {
    triggerClass: 'picturepop',
    openPopupLinkClass: 'popuplink',
    popupClass: 'popup',
    displayPrefix: 'Hide ',
    popContainer: null,
```

`init()`方法检查是否支持DOM并循环遍历文档中所有的链接，检查每个链接是否有触发弹出窗口对应的CSS类。对于所有有对应的CSS类的链接，这个方法给它添加一个指向`openPopup()`方法的事件处理程序，接着把链接的`innerHTML`内容存储到`preset`属性中。

picturePopup.js (续)

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var allLinks = document.getElementsByTagName( 'a' );
    for( var i = 0; i < alllinks.length; i++ ) {
        if( !DOMhelp.cssjs( 'check', allLinks[i], -->
            pop.triggerClass ) ) {
            continue;
        }
        DOMhelp.addEvent( allLinks[i], 'click', pop.openPopup, false );
        alllinks[i].onclick = DOMhelp.safariClickFix;
        alllinks[i].preset = allLinks[i].innerHTML;
    }
},
```

`openPopup()`方法获取事件目标并确保它是一个链接。它接着检查是否存在一个`popContainer`，它意味着照片已显示了。否则，这个方法会把前缀和链接的内容连接到一起，并添加应用动态的类以便这个链接看起来有所不同。

picturePopup.js (续)

```
openPopup : function( e ) {
    var t = DOMhelp.getTarget( e );
```

```

if( t.nodeName.toLowerCase() != 'a' ) {
    t = t.parentNode;
}
if( !pop.popContainer ) {
    t.innerHTML = pop.displayPrefix + t.preset;
    DOMhelp.cssjs( 'add', pop.popContainer, pop.popupClass );
}

```

这个方法接着创建一个作为照片容器的新DIV元素，添加应用适当的类，并添加一个新图片作为容器DIV的子节点。它通过把新图片的src属性设置为原始链接的href属性的值来显示图片。这个新创建的照片容器接着被添加到文档中（作为body元素的一个子元素）。最后，openPopup()调用把这个链接对象作为一个参数的positionPopup()方法。

picturePopup.js (续)

```

pop.popContainer = document.createElement( 'div' );
DOMhelp.cssjs( 'add', t, pop.openPopupLinkClass );
var newimg = document.createElement( 'img' );
pop.popContainer.appendChild( newimg );
newimg.setAttribute( 'src', t.getAttribute( 'href' ) );
document.body.appendChild( pop.popContainer );
pop.positionPopup( t );

```

如果popContainer已经存在，这个方法所要做的就是调用killPopup()方法、重新设置该链接为原始的内容以及移除表示照片显示的类。调用cancelClick()方法来防止链接只在浏览器中简单地显示照片。

picturePopup.js (续)

```

} else {
    pop.killPopup();
    t.innerHTML = t.preset;
    DOMhelp.cssjs( 'remove', t, pop.openPopupLinkClass );
}
DOMhelp.cancelClick( e );
},

```

positionPopup()方法定义了两个变量，x和y，并把它们都初始化为0，接着从它的offsetHeight属性中读取这个元素的高度。再接下来读取该元素的水平和垂直位置以及它所有的父元素，并把这些值分别加到x和y上。结果是该元素的位置接近于文档的位置。然后这个方法通过在垂直的位置变量y上添加链接的高度并改变popContainer的样式属性，从而把这个照片容器定位到原始链接的下面。

picturePopup.js (续)

```

positionPopup : function( o ) {
    var x = 0;
    var y = 0;
    var h = o.offsetHeight;
}

```

```

    while ( o != null ) {
        x += o.offsetLeft;
        y += o.offsetTop;
        o = o.offsetParent;
    }
    pop.popContainer.style.left = x + 'px';
    pop.popContainer.style.top = y + h + 'px';
},

```

`killPopup()`方法从文档中移除`popContainer`——通过把这个属性的值设置为`null`来清除它——并调用`cancelClick()`来阻止默认链接的动作发生。

注解 可以通过调用一个节点的父节点的`removeChild()`方法来从文档中移除它，`removeChild()`方法把这个节点作为其父节点的子节点加以移除。然而，由于使用了一个指向这个节点的属性而不是检查这个节点自己，所以你也需要把这个属性设置为`null`。

picturePopup.js (续)

```

killPopup : function( e ) {
    pop.popContainer.parentNode.removeChild( pop.popContainer );
    pop.popContainer = null;
    DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', pop.init, false );

```

这个例子执行的结果是你可以单击任意指向一张带有对应类照片的图片，它就会在它下面显示这张图片。图6-17显示了它的一个示例。

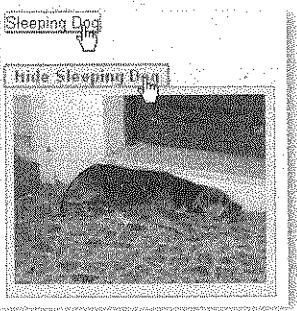


图6-17 动态显示照片示例

这种方法的优点在于它不只限于照片。做一下简单的修改，你就可以在当前文档中显示其他的文档。技巧在于，给`photoContainer`动态地添加一个`IFRAME`元素并设置它的`src`属性为你想嵌入的文档。示例`exampleIframeForPopup.html`就是那样做的，它把一个冗长的条款和条件文档显示到主文档内。

唯一不同的地方（除去不同的属性名字，因为这个方法不显示照片）就在你要添加新的IFRAME的openPopup方法中：

iframeForPopup.js（摘录）

```
var ifr = document.createElement( 'iframe' );
pop.ifrContainer.appendChild( ifr );
ifr.setAttribute( 'src', t.getAttribute( 'href' ) );
```

图6-18显示了它会表现的样子。

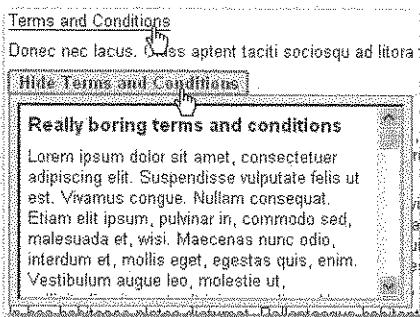


图6-18 动态包含和显示文档示例

通过IFRAME元素来包含其他的文档是个很容易且广泛支持的方法，但是它并不是最易访问的方式。可以使用一种服务器端语言（如PHP）来获取文档的内容并把它包含到当前的文档中且使用与层广告前面使用过的相同的技巧来替换这个方法。对于大多数现代的浏览器，还可以使用Ajax让它“漂浮”起来，第8章将会讲解它。

6.2.3 窗口与JavaScript小结

控制当前的窗口与打开新的窗口在过去的JavaScript开发中占有很大的比重，Web应用程序开发尤其如此。然而，在最近的几年里，由于考虑浏览器安全方面限制的增加以及网络冲浪者安装了阻止软件来防止大量自动弹出的窗口，使用新窗口变的越来越困难了——即使你很正当地使用它们。这些原因和可访问性关注使使用多个浏览器窗口在网络通信中的不可靠性增加。幸好，由于这里讨论的一些可替换的技术（还有Ajax，我把它放在了第8章），对它们几乎不再有任何需要了。

窗口和JavaScript记要

- 在对你打开的窗口进行处理前，测试、测试、再测试它是否存在。
- 始终要记着尽管窗口都是在同样的屏幕上，但它们是完全独立的浏览器实例。如果你想从一个弹出窗口访问另一个弹出窗口，或者从你打开的弹出窗口中访问主脚本中的函数，你需要使用window.opener。

- 尽量避免通过去掉工具栏、在屏幕上四处移动窗口、通过`blur()`和`focus()`来显示和隐藏窗口从而来控制窗口。大多数的这些功能现在还仍然可以使用，但是在未来的浏览器中很有可能被禁止。
- 未来的Web标记语言XHTML不支持不同浏览器窗口实例的概念（这就是为什么废弃`target`属性的原因）。虽然你可以使用脚本解决这个问题，但是使用严格的XHTML脚本并依赖几个窗口仍然是一个很糟糕的主意。
- 可以使用窗口对象的方法模仿许多浏览器的行为或交互式的元素——如关闭和打印窗口或返回浏览器历史中的上一个窗口。可是，最好让用户自己来选择。因为如果你想给用户提供自己的控制功能，就需要JavaScript来创建这些功能。当JavaScript不可用的时候，用户就会得到一个你没有兑现的承诺。
- 如果你使用了弹出窗口，要告诉访问者链接中打开的窗口会是一个新的窗口。这样可以通知使用了不一定支持多窗口用户代理的访问者有一些改变需要处理。多年的主动提供的广告和弹出窗口已经使Web冲浪者习惯于直接地关闭新窗口，甚至不用看它们一眼。
- 通过`window`对象的`setTimeout()`和`setInterval()`来定时执行代码有点像使用化妆品：作为一个女孩你需要学会如何上妆，作为一个妇女你要学会何时卸妆。这两种方法你都可以使用——并且它们可以用来创建各种流行的效果——但是你应该考虑一下用户，要问问自己：“当一个静态的界面可能很快实现同样的效果的时候，是否真的需要动画？”

6.3 小结

很不错！这一章你已学完了，现在应该已经很好地掌握了使用图片和窗口创建自己的JavaScript方案，或者更好一点可以使用窗口交换技术。

如果对其中的一些例子还没有融会贯通，不要沮丧，因为这是JavaScript开发人员经常会体验到的一种感觉。这并不意味着你没有“学会它”。在JavaScript中对于解决给定的问题存在许多方式。虽然有许多方式比这里讲解的要更容易，但是用惯了这种脚本可以使你很容易掌握更多高级的脚本任务，如使用第三方API或Web应用程序开发。

在下一章中，你会对事件和属性处理有更进一步的理解，因为我们会讨论导航和表单。

JavaScript与用户的交互： 导航与表单

在这一章中，我们将讨论JavaScript另外两个常见的用途：导航和表单。二者都与用户交互关系紧密，因此需要仔细地计划和执行。网站的成败的关键在于是它的导航是否容易使用，而且在网上没有比表单难以使用或者无法填写更令人沮丧了。

注解 本章包含了大量的代码示例，你需要在浏览器中打开其中的一些来验证它们的功能，因此，如果你还没有本书例子的源代码，最好现在登录<http://www.beginningjavascript.com>把源代码下载到本地。我坚信对于编码最好的办法就是实践，借助代码可以使你更好地理解功能，请随时准备好你的编辑器以便多加练习。

7.1 导航与JavaScript

自从浏览器支持动态改变网页元素的界面外观以来，不断丰富网站导航已成为DHTML的主要工作了。由DHTML开创的动态导航的时代还没有结束。表面上如果一个网页的导航非常灵活，技术含量很高，那么这个站点一定很了不起。很多时候，用户并不认同，一旦对导航感到厌烦了，他们会使用站点搜索功能，如果提供了搜索选择的话。

在这里我们不会介绍太多炫酷的导航；相反，你可以通过我们的JavaScript例子来使网页和网站导航看起来更加直观而且实用。

7.1.1 重新加载网页的恐惧

许多用JavaScript来增强窗口和导航的功能都是为了防止重新加载页面，或是避免在还没有访问到需要的页面前加载许多无关的页面。这种想法非常好，而且运行得也非常好。可是，我们不要忘了JavaScript只有在整个页面加载完了以后才能访问页面的元素，而且只能处理文档里的元素。（除非你使用Ajax来从外部加载其他的内容，这点将在下一章中具体阐述）。

这意味着你可以建立一个平滑的界面，使用JavaScript只显示整个页面内容的一部分，但这也意味着不使用JavaScript的用户则需要处理文档中的所有数据。在你热衷于增加页面处理能力之前，应随时关闭JavaScript功能，看看页面是否能正确处理好所有的数据。

带有特定信息的一些较小的网页文档的优点在于你可以使用浏览器给你提供的所有功能：向前、向后、书签和打印。而当你将JavaScript用于导航和分页时，这些功能将受限。带来的负面影响是你必须维护更多的页面文档，用户要分别单独地加载它们，这也增加了服务器的负担。

通过JavaScript来增强网页的功能并没有什么坏处，关键是要适度，并且了解用户的想法。

7.1.2 JavaScript 导航基础

JavaScript导航的最基本一点是不能依赖于JavaScript来完成导航的任务。依赖于JavaScript，页面或网站导航就会妨碍一些不能随意启用JavaScript的用户的正常使用，同样也会阻碍搜索引擎对它的搜索收录。

提示 后者是一个非常好的理由，可以用来给非技术的人员讲解为什么使用许多 javascript:navigate('page2')链接不是很好的原因。应该考虑那些网站目标用户中禁用了JavaScript的人清楚可不容易，这一点想对那些甚至不知道如何在浏览器中关闭JavaScript的人解释。给他解释Google不会索引他们的站点，会比较容易些。

使用JavaScript可以使导航更容易使用，而且不用冒着疏远大量用户群的危险。这样的经典例子是使用选择框来导航。选择框非常好，因为它们可以给你提供许多选项而不占用太多的地方。图7-1显示了一个打开的选择框；关闭的时候所有的这些选择只占用一行。

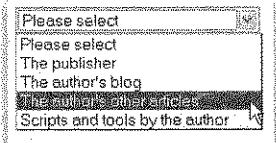


图7-1 使用选择框来导航

打开演示页面exampleSelectNavigation.html，从下拉框中选择一项。如果能连接到Web并且浏览器支持JavaScript，那么很快就会连接到所选择的地址。但是，如果不小心选错了，就没有机会来撤销这个选择了。不用JavaScript的话，也可以选择，但没有任何响应。

exampleSelectNavigation.html

```
<form>
<p>
<select onchange="window.location = this.options[this.selectedIndex].value">
    <option value="#">Please select</option>
    <option value="http://www.apress.com">The publisher</option>
    <option value="http://www.apress.com/~jewett/>The author's blog</option>
    <option value="http://www.apress.com/~jewett/articles">The author's other articles</option>
    <option value="http://www.apress.com/~jewett/tools">Scripts and tools by the author</option>
</select>
</p>
</form>
```

```

<option value="http://wait-till-i.com">The author's blog</option>
<option value="http://icant.co.uk">The author's other articles</option>
<option value="http://onlinetools.org">Scripts and tools by the author</option>
</select>
</p>
</form>

```

通过键盘输入也是一个问题：一般使用Tab键来选中选择框，用up和down键来选中你需要的选项。而在这个例子中，你无法实现这种操作，在按下down键的时候就会自动转到该选项所对应的地址。临时变通的解决办法是同时按下Alt和down这一对组合键来展开所有选项，然后通过up或down键来选择你需要的选项，再敲回车键，但是你知道那样做吗？

这种问题最简单的解决办法是不要使用change、mouseover或focus事件来向服务器端传送数据或者让用户连接到一个不同的网址。这样对许多用户来说是不可访问的，而且会带来麻烦，尤其是在没有任何提示数据会发送或网页会改变的情况下。

相反，提供一个真正的提交按钮，一个处理同样的服务器端重定向的脚本，以及一个submit处理器（它在JavaScript可用的情况下用来重定向）。

exampleSaferSelectNavigation.html (摘录)

```

<form method="post" action="redir.php">
<p>
    <label for="url">Please select your destination:</label>
    <select id="url" name="url">
        <option value="http://www.apress.com">The publisher</option>
        <option value="http://wait-till-i.com">The author's blog</option>
        <option value="http://icant.co.uk">The author's other articles</option>
        <option value="http://onlinetools.org">Scripts and tools by the author</option>
    </select>
    <input type="submit" value="Make it so!" />
</p>
</form>

```

这段脚本非常简单：在第一个表单的submit上应用一个事件处理程序来触发一个方法。这个方法从ID为url的选择列表的selectedIndex中读取用户所做的选择，通过window.location对象把浏览器重定向到那里。在下一节会了解到更多有关window.location的内容，并且在7.2节中会了解到有关selectedIndex和表单对象的所有东西。

exampleSaferSelectNavigation.html (摘录)

```

send = {
    init : function(){
        DOMhelp.addEvent( document.forms[0],
            'submit', send.redirect,
            false );
    },
    redirect : function( e ){
        var t = DOMhelp.getTarget( e );

```

```

var url = t.elements['url'];
window.location.href = url.options[url.selectedIndex].value;
DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', send.init, false );

```

把不支持JavaScript的用户引到其他URI的服务器端脚本在PHP中就是个简单的标题重定向：

```
<?php header( 'Location: ' . $_POST['url'] ); ?>
```

如果用户已启用了JavaScript，那么没有必要来回调用服务；它可以快速地连接到其他的站点。也可以通过设置window.location.href属性来实现。window.location.href属性是浏览器导航内置的一部分。

7.1.3 浏览器导航

浏览器给你提供了几个对象，使你可以自动重定向或在浏览器历史中进行导航。你在上一章中已经接触到了window.back()方法。window对象还提供了window.location和window.history属性。

window.location对象存储当前元素的URI，并且具有下面的属性（在括号中你看到的是URI：http://www.example.com:8080/index.php?s=JavaScript#searchresults所对应的返回值）。

- hash: URI中锚的名字 (#searchresults)。
- host: URI的域名部分 (www.example.com)。
- hostname: URI的域名，包括了子域名和端口号 (www.example.com:8080)。
- href: 完整的URI字符串 (http://www.example.com:8080/index.php?s=JavaScript#searchresults)。
- pathname: URI的路径名 (/index.pah)。
- port: URI的端口号 (8080)。
- protocol: URI的协议 (http:)。
- search: 查找参数 (?s=JavaScript)。

所有上述参数都是可读可写的。例如，如果想通过查找来改变DOM脚本，你可以改变window.location.search='?DOM scripting'。浏览器会自动编码成DOM%20scripting。也可以通过改变window.location.href property来给用户浏览器发送另外的地址。

除了上述属性，window.location还有两种方法。

- reload(): 重新加载当前文档（效果相当于单击Reload按钮，或者按F5或同时按Ctrl和R键）。
- replace (URI)：把用户带到URI上，并用另外的一个替换掉当前的URI。当前URI也不会出现在浏览器的历史当中。

警告 注意这与String对象的replace()方法不同，replace()方法只是替换掉字符串的一部分。

可以使用reload()方法来定时刷新页面以从后台加载新的内容，而不用用户不断单击Reload按钮。这个功能常用于基于JavaScript的聊天系统中。

使用replace()有时会带来很多麻烦，因为这样影响了用户Back按钮的功能。当他不喜欢你传给他的页面时，就不能使用Back按钮来返回当前的页面。

用户在当前页面之前所访问过的页面清单都存储在变量window.history中。这个变量只有一个属性length，它存储着已访问过页面的数字，有3种方法可以调用。

- back(): 转到浏览器历史中的上一个页面。
- forward(): 转到浏览器历史中的下一个页面。
- go(*n*): 根据*n*是正数还是负数，在浏览器的历史中向前或向后走*n*步。也可以通过history.go(0)转到同一个页面。

history对象仅允许导航到其他的页面——而不是读出它们的URI或者改变它们。这个规则的例外情况就是当前页面，当使用replace()的时候会擦去其在浏览器中的历史记录。

在前面几章中讨论过，可以通过JavaScript把用户带到其他页面来模拟浏览器的功能，它可能会是多余的或使用户感到疑惑。

7.1.4 页内导航

可以使用JavaScript来让同一页面的导航更加有趣并且占用较小的屏幕空间。在HTML中，可以通过锚和目标来进行页内导航，两者都是通过标签定义的，对于锚使用href属性或对于目标使用name或id属性。

注解 在XHTML中锚的name属性已被废弃，实际上提供一个ID来链接到锚或目标已经足够了。

然而，为了能与老版本的浏览器兼容，你最好是按下面这个例子来做。

在页内导航的例子中，内容的表中有一个内部链接列表，分别链接到页面底部的其他地方：

```
exampleLinkedAnchors.html (摘录)
<h1>X - a tool that does Y</h1>
<div id="toolinfo">
  <ul id="toolinfotoc">
    <li><a href="#info">Information</a></li>
    <li><a href="#demo">Demo</a></li>
    <li><a href="#installation">Installation</a></li>
    <li><a href="#use">Use</a></li>
    <li><a href="#license">License</a></li>
    <li><a href="#download">Download</a></li>
  </ul>
  <div class="infoblock">
    <h2><a id="info" name="info">Information about X</a></h2>
    [... content ...]
  </div>
  <p class="back">
```

```

<a href="#toolinfotoc">Back to
<acronym title="Table of Contents">TOC</acronym></a>
</p>
</div>
<div class="infoblock">
  <h2><a id="demo" name="demo">Demonstration of what
    X can do</a></h2>
  [... content ...]
  <p class="back">
    <a href="#toolinfotoc">Back to
    <acronym title="Table of Contents">TOC</acronym></a>
  </p>
</div>
[... more sections ...]
</div>

```

你也许会认为类为infoblock的DIV元素对于页内导航的功能来说没什么作用。但那只说对了一部分，因为在处理指定的锚和键盘导航的时候，IE有个非常令人厌烦的bug。

如果在IE中打开演示页面exampleLinkedAnchors.html，那么可以通过敲Tab键在不同的菜单选项之间进行导航，并且通过敲回车键来选择你想要的部分，浏览器会定位到所选择的锚的地方。可是，IE并没有把键盘的焦点转到这个锚上。如果你再敲一下tab键，不是得到了文档中的下一个链接，而是返回到了菜单上。

可以通过其他技巧——把锚嵌套在一个已定义宽度的元素中——来解决这个问题。这正是DIV所要做的。可以通过演示页面exampleLinkedAnchorsFixed.html来测试。实际结果是你可以使用这些元素——在这个例子中是DIV——来进行CSS样式设置。

提示 如果想了解更多关于这个问题及相关的解决方法，可以在<http://juicystudio.com/article/ie-keyboard-navigation.php>上参考可访问性研究专家Gez Lemon有关这个主题的更深层次的文章。

现在我们使用脚本来复制并改进一下这个功能。脚本要做到的是显示菜单，但是要隐藏所有的区域，只显示你所选择的区域以使页面看起来精简而不冗长。逻辑上非常简单：

- 循环遍历菜单上的链接，并添加一个click事件处理程序来显示关联到菜单项的区域。
- 在事件监听方法中，隐藏以前所显示的区域，显示当前的区域。
- 当初始化页面的时候，隐藏所有的区域并显示第一个区域。

然而，这并没有考虑到页内导航的另一方面：这个页面可能是从另一个链接的一个定义好的目标所请求的。试一下在浏览器中给URI添加一个锚，例如，exampleLinkedAnchorsFixed.html#use，就会自动滚动到正在使用的部分。你的脚本应该把这种情况考虑进去。

让我们从定义一个脚本的大纲入手。脚本的主要对象叫做iv，它是用来内部导航的——由于in在JavaScript中是个保留字，而你又想让它的名字更简短一些。需要下面几个属性：

- 一个当菜单是JavaScript增强时要定义的CSS类；
- 一个用来突出显示菜单中当前链接的CSS类；
- 一个用来显示当前部分的CSS类。

提示 不一定非要通过JavaScript来隐藏各个部分，通过使用上一章中提到的CSS parent类技巧也可以实现。

需要定义一些属性，添加应用CSS类的父元素以及用来循环遍历链接的菜单ID。

另外还需要两个属性，一个用来存储当前显示的部分，一个用来保存当前突出显示的链接。

在方法方面，需要一个init()方法、一个用来得到当前区域的事件监听器，一个用来隐藏前一个部分而显示当前部分的方法。

innerNav.js（大纲）

```
iv = {
  // CSS classes
  dynamicClass : 'dyn',
  currentLinkClass : 'current',
  showClass : 'show',

  // IDs
  parentID : 'toolinfo',
  tocID : 'toolinfotoc',

  // Global properties
  current : null,
  currentLink : null,

  init : function(){ },
  getSection : function( e ){ },
  showSection : function( o ){ }
}
DOMhelp.addEvent( window, 'load', iv.init, false );
```

Init()方法开始先检查是否支持DOM，所有需要的元素是否可用。接下来把类添加到父元素上以通过CSS自动隐藏所有的元素。

innerNav.js（摘录）

```
init : function(){
  if( !document.getElementById || !document.createTextNode ) {
    return;
  }
  iv.parent = document.getElementById( iv.parentID );
  iv.toc = document.getElementById( iv.tocID );
  if( !iv.parent || !iv.toc ) { return; }
  DOMhelp.cssjs( 'add', iv.parent, iv.dynamicClass );
```

在变量loc中存储一个可能的URL散列信息，并开始循环遍历菜单中的所有链接。替换散列值中的#以方便后面使用，因为你可以在getElementById()方法中使用名字而不用去掉散列信息。

innerNav.js (续)

```
var loc = window.location.hash.replace( '#', '' );
var toclinks = iv.toc.getElementsByTagName( 'a' );
for( var i = 0; i < toclinks.length; i++ ) {
```

把当前链接的href属性和loc做比较，如果它们相同，则把这个链接保存到currentLink属性中。这里使用的字符串的replace()方法从href属性中删除了除锚名以外的所有信息。这是非常有必要的，因为在一些浏览器如IE中，getAttribute('href')返回的是包括文件路径的整个链接的位置，而不仅仅是HTML的href属性的内容。

innerNav.js (续)

```
if( toclinks[i].getAttribute( 'href' ).replace( /.*#/ , '' ) == loc ){
    iv.currentLink = toclinks[i];
}
```

接下来，添加一个指向getSection()方法的click事件。注意在这个示例脚本中没有必要阻止默认的事件，恰恰相反，允许浏览器跳转到某个部分也会改变地址栏的URI，这样反过来可以使用户为这部分作书签。

innerNav.js (续)

```
DOMhelp.addEvent( toclinks[i], 'click', iv.getSection, false );
}
```

只有当其中的一个链接与URI中散列值相同时才定义current属性。这意味着如果URI没有散列信息或指向一个并不存在的锚，需要把currentLink定义为菜单的第一个锚。Init()方法最后把currentLink作为参数来调用showSection()方法。

innerNav.js (续)

```
if( !iv.currentLink ) {
    iv.currentLink = toclinks[0];
}
iv.showSection( iv.currentLink );
},
```

事件监听方法getSection()不需要做太多事情；它要做的是确定哪一个链接被单击了，并将它作为参数传递给showSection()。如果不是要访问window.location.hash，这两行可以作为showSection()方法的一部分。

innerNav.js (续)

```
getSection : function( e ) {
    var t = DOMhelp.getTarget( e );
    iv.showSection( t );
},
```

`showSection()`方法检索被单击或者以参数`o`定义在`init()`方法中的链接对象，第一个任务是读取这个链接的`href`属性，并通过使用正则表达式删除包括散列符号（含）之前的所有字符以检索锚的名字。然后通过使用锚的ID读取元素，并在结点树上向上走两层节点来重新获取要显示的区域。

innerNav.js (续)

```
showSection : function( o ) {
    var targetName = o.getAttribute( 'href' ).replace( /.*#/,'' );
    var section = document.getElementById(targetName).parentNode.parentNode;
```

为什么要向上走两层结点？如果你还记得它的HTML，是把链接嵌入到标题中，并把标题和区域的其余部分又嵌入在DIV元素中：

exampleLinkedAnchors.html (摘录)

```
<li><a href="#demo">Demo</a></li>
[... code snipped ...]
<div class="infoblock">
    <h2><a id="demo" name="demo">
        Demonstration of what X can do
    </a></h2>
```

由于从`getElementById('demo')`中获得的是链接，上一层节点是H2，再上一层节点是DIV。

接下来需要检查是否有一个旧的区域在显示以及是否有一个链接已突出显示了，并通过移除对应的类来去掉突出显示效果并隐藏显示的区域。然后为当前的链接和当前区域添加类，并设置`current`和`currentLink`这两个属性，确保下一次调用`showSection()`时可以撤销现在所做的操作。

innerNav.js (续)

```
if( iv.current != null ){
    DOMhelp.cssjs( 'remove', iv.current, iv.showClass );
    DOMhelp.cssjs( 'remove', iv.currentLink, iv.currentLinkClass );
}
DOMhelp.cssjs( 'add', section, iv.showClass );
DOMhelp.cssjs( 'add', o, iv.currentLinkClass );
iv.current = section;
iv.currentLink = o;
}
DOMhelp.addEvent( window, 'load', iv.init, false );
```

如果把这段脚本应用到HTML演示页面中，并设置一个适当的样式表，就会得到一个非常短小的页面，当你单击链接的时候会显示不同的区域。可以在浏览器中打开exampleLinkedAnchorsPanel.html来看一下这种效果。在Firefox 1.5、Windows XP环境下，页面就如你所看到的图7-2那样。

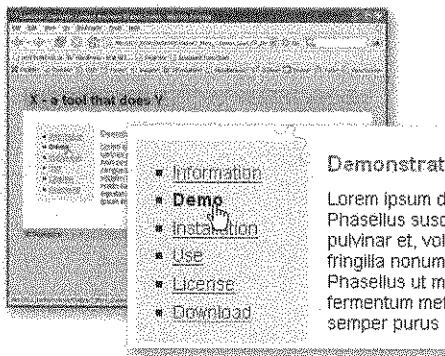


图7-2 从一个锚——目标列表中创建的面板界面

简单地应用一个不同的样式表就可以把页面转变为一个标签界面，正如你在exampleLinkedAnchorsTabs.html和图7-3中看到的那样。

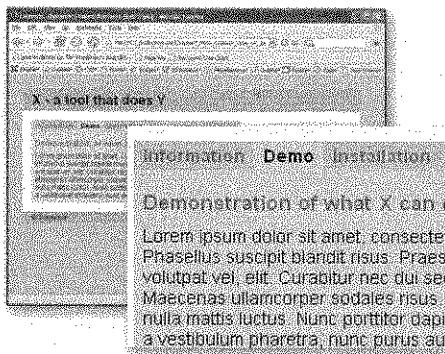


图7-3 从一个锚——目标列表中创建的标签界面

对于比较短的脚本这样做是非常巧妙的；可是，用户每次单击其中的一个选项时都要清除链接的href并重新获取显示的区域就显得重复了。

还有一个可以达到同样效果的不同方法是把链接和区域分别保存在两个相关的数组中，并使用锚的名字提供showSection()来显示及加亮。示例exampleLinkedAnchorsTabsNamed.html就使用了这种技术，并展示了如何应用一个mouseover处理器来同时达到同样的效果。

innerNavNamed.js

```

iv = {
    // CSS classes
    dynamicClass : 'dyn',
    currentLinkClass : 'current',
    showClass : 'show',

    // IDs
    parentID : 'toolinfo',
    tocID : 'toolinfotoc',
}

```

第一个变化就是你需要一个current属性和两个新的叫做sections和sectionLinks的数组属性，后两个属性用来在以后保存区域和链接。

innerNavNamed.js (续)

```

// Global properties
current : null,
sections : [],
sectionLinks : [],
init : function() {
    var targetName,targetElement;
    if( !document.getElementById || !document.createTextNode ){
        return;
    }
    var parent = document.getElementById( iv.parentID );
    var toc = document.getElementById( iv.tocID );
    if( !parent || !toc ) { return; }
    DOMhelp.cssjs( 'add', parent, iv.dynamicClass );
    var toclinks = toc.getElementsByTagName( 'a' );
    for( var i = 0; i < toclinks.length; i++ ){

```

除了click以外，再添加一个mouseover事件处理程序，对于菜单的每一个链接，把href属性保存到叫做targetName的一个属性中。

innerNavNamed.js (续)

```

DOMhelp.addEvent( toclinks[i], 'click', iv.getSection, false );
DOMhelp.addEvent( toclinks[i], 'mouseover', iv.getSection, false );
targetName = toclinks[i].getAttribute( 'href' );
replace( /.*/#, '' );
toclinks[i].targetName = targetName;

```

通过把第一个链接保存在变量presetLink中将它定义为当前活动链接，并且判断这个锚是否指向了一个存在的元素。如果是，把该元素存储到sections数组中，把该链接存储到sectionLinks数组中。注意关联数组中的这些结果，它意味着你可以通过section['info']来访问第一个区域。

innerNavNamed.js (续)

```

if( i == 0 ){ var presetLink = targetName; }
targetElement = document.getElementById( targetName );
if( targetElement ) {
    iv.sections[targetName] = targetElement.parentNode.parentNode;
    iv.sectionLinks[targetName] = toclinks[i];
}
}
}

```

接下来你可以从URI散列中获得可能存在的一一个锚名，通过把锚的名字或者存储在presetLink中的变量作为参数来开始调用showSection()。

innerNavNamed.js (续)

```

var loc = window.location.hash.replace( '#', '' );
loc = document.getElementById(loc) ? loc : presetLink;
iv.showSection( loc );
},

```

getSection()事件监听器用链接的targetName属性值作为参数调用showSection()方法。这个属性是在前面的init()方法中设置的。

innerNavNamed.js (续)

```

getSection:function( e ){
    var t = DOMhelp.getTarget( e );
    iv.showSection( t.targetName );
},

```

所有的这些使得showSection()方法很简单，它所要做的就是重置上一个链接和区域，通过数组找到需要的元素并添加或移除CSS类来设置当前的链接和区域。当前的显示部分存储在一个叫current的属性中，而不是区域和链接共有的属性中。

innerNavNamed.js (续)

```

showSection : function( sectionName ){
    if( iv.current != null ){
        DOMhelp.cssjs( 'remove', iv.sections[iv.current], -->
        iv.showClass );
        DOMhelp.cssjs( 'remove', iv.sectionLinks[iv.current], -->
        iv.currentLinkClass );
    }
    DOMhelp.cssjs( 'add', iv.sections[sectionName], iv.showClass );
    DOMhelp.cssjs( 'add', iv.sectionLinks[sectionName], -->
    iv.currentLinkClass );
    iv.current = sectionName;
}
}
DOMhelp.addEvent( window, 'load', iv.init, false );

```

对于页内导航还有许多可选项，例如，可以提供“上一步”和“下一步”链接来替代“后退”链接以遍历到各个选项。如果想看看这样的脚本，并且每个页面内还提供了几个标签导航，你可以打开`http://onlinetools.org/tools/domtabdata/`看一下DOM标签。

7.1.5 网站导航

网站的导航和页面内部的导航是完全不同的。再次强调依赖于JavaScript的导航是不理想的。是的，可以使用JavaScript将用户自动地转到其他的地址，但这不是一种安全的方法，因为如Opera和Mozilla这样的浏览器允许用户阻止这些（恶意网站过去常用重定向将用户指定到垃圾网站）。不仅如此，网站维护者在使用统计软件计算单击率和记录访问者如何在站内访问时，也会削减最后的结果，因为并不是所有的软件包都会统计JavaScript重定向。

基于这些原因，网站导航一般都限于增强菜单的HTML结构的功能，并通过事件处理程序来给它添加功能。用户真正重定向到其他网页仍然需要通过链接或表单提交来实现。

一个逻辑性很强的网站菜单的HTML结构是个嵌套的列表：

`exampleSiteNavigation.html` (摘录)

```
<ul id="nav">
  <li><a href="index.php">Home</a></li>
  <li><a href="products.php">Products</a>
    <ul>
      <li><a href="cms.php">CMS solutions</a>
        <ul>
          <li><a href="minicms.php">Mini CMS</a></li>
          <li><a href="ncc1701d.php">Enterprise CMS</a></li>
        </ul>
      </li>
    </ul>
  </li>
  <li><a href="portal.php">Company Portal</a></li>
  <li><a href="mailserver.php">eMail Solutions</a>
    <ul>
      <li><a href="privatemail.php">Private POP3/SMTP</a></li>
      <li><a href="lists.php">Listservers</a></li>
    </ul>
  </li>
  <ul>
    <li><a href="services.php">Services</a>
      <ul>
        <li><a href="training.php">Employee Training</a></li>
        <li><a href="audits.php">Auditing</a></li>
        <li><a href="bulkmail.php">Bulk sending/email campaigns</a></li>
      </ul>
    </li>
  </ul>
  <li><a href="pricing.php">Pricing</a></li>
  <li><a href="about_us.php">About Us</a>
```

```

<ul>
  <li><a href="our_offices.php">Our offices</a></li>
  <li><a href="our_people.php">Our people</a></li>
  <li><a href="vacancies.php">Jobs</a></li>
  <li><a href="partners.php">Industry Partners</a></li>
</ul>
</li>
<li><a href="contact.php">Contact Us</a>
  <ul>
    <li><a href="snail.php">Postal Addresses</a></li>
    <li><a href="callback.php">Arrange Callback</a></li>
  </ul>
</li>
</ul>

```

这样做的原因是，即使没有任何样式表，导航的结构和层次对于用户也都是清晰可见的。也可以轻松地设置导航的样式，因为所有元素都嵌套在一个更高层次的元素内，这样可以使用情景选择器。

注解 我们在这里将不讨论在导航中提供站点的所有页面是否有意义（因为这通常都是站点地图的任务）。在下一章中，我们将会再次讨论这个问题并让用户自己来做选择。

从基本的网站可用性和常识来说，当前页面不应链接到它本身。为了防止这类事件发生，用一个STRONG元素来替换当前页面链接，这也意味着不支持CSS的用户能知道他们在导航中的哪个位置，而且你也有机会不借助于CSS类来为导航的当前页面设置不同的样式。使用一个STRONG元素而不用SPAN，也意味着不支持CSS的用户也可以得到哪一项是当前的页面的明确指示。

例如，在Mini CMS页面中，导航和下面的差不多：

exampleHighlightedSiteNavigation.html (摘录)

```

<ul id="nav">
  <li><a href="index.php">Home</a></li>
  <li><a href="products.php">Products</a>
    <ul>
      <li><a href="cms.php">CMS solutions</a>
        <ul>
          <li><strong>Mini CMS</strong></li>
          <li><a href="ncc1701d.php">Enterprise CMS</a></li>
        </ul>
      </li>
    </ul>
  </li>
</ul>

```

你需要在服务器端来做这些处理，因为在JavaScript中突出显示当前的页面没有意义（当然，通过把所有链接的href属性和window.location.href相比较实现这个并不难）。

我们期望这个HTML结构可以让你创建一个像浏览器一样的可以展开和折叠的菜单。一个包

含其他项的菜单项在你单击它的时候也应该显示或隐藏它的子菜单项。然而，这段脚本的逻辑可能跟你所期望的有所不同。首先，没有必要循环遍历菜单中的所有链接，可以使用下面的方法来替换它：

- (1) 给隐藏所有嵌套列表的主导航项添加一个CSS类。
- (2) 循环遍历导航中所有的UL项（因为这些是嵌套的子菜单）。
- (3) 给每个UL的父节点添加一个CSS类，标明这个列表项包含其他的列表项。
- (4) 在父节点的内部给第一个链接上添加一个click事件。
- (5) 检查父节点是否包含STRONG元素，如果有的话则添加一个类来显示UL——这样可以阻止当前页面的子菜单被隐藏。使用一个open类来替换parent类以显示这个部分已经被展开了。
- (6) click事件监听方法需要检查父节点第一个嵌套的UL元素是否应用了show类，如果有的话则把它移除。还需要用parent类替换open类。如果没有show类，则做相反的替换操作。

示例exampleDynamicSiteNavigation.html就是这样做的，它使用了定义的Mini CMS页面作为当前页面来显示这种效果。图7-4是在Firefox 1.5、Windows XP环境下显示的效果。



图7-4 使用JavaScript和CSS的树型菜单

这个脚本的大纲相当的短小；把所有需要的CSS类定义为属性，定义导航的ID作为另一个属性，定义init()和changeSection()方法来应用所有的功能，并相应地展开或折叠某一部分。

siteNavigation.js (大纲)

```
sn = {
    dynamicClass : 'dyn',
    showClass : 'show',
    parentClass : 'parent',
    openClass : 'open',
    navID : 'nav',
    init : function() {},
    changeSection : function( e ) {}
}
DOMhelp.addEvent( window, 'load', sn.init, false );
```

Init()方法定义了一个叫triggerLink的变量，并在应用动态类隐藏嵌套元素前检查是否支持DOM以及所需的导航元素是否可用。

siteNavigation.js (摘录)

```
init : function() {
    var triggerLink;
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    var nav = document.getElementById( sn.navID );
    if( !nav ){ return; }
    DOMhelp.cssjs( 'add', nav, sn.dynamicClass );
```

接下来循环遍历所有嵌套的UL元素，并将父节点的第一个链接的引用保存为triggerLink。它应用了一个click事件来调用changeSection()方法，并把parent类添加到父节点上。

siteNavigation.js (续)

```
var nested = nav.getElementsByTagName( 'ul' );
for( var i = 0; i < nested.length; i++ ){
    triggerLink = nested[i].parentNode.getElementsByTagName('a')[0];
    DOMhelp.addEvent( triggerLink, 'click', sn.changeSection, false );
    DOMhelp.cssjs( 'add', triggerLink.parentNode, sn.parentClass );
    triggerLink.onclick = DOMhelp.safariClickFix;
```

下面这段代码检查了父节点是否包括了一个STRONG元素，如果包含的话，则把show类添加到UL上并把open类添加到父节点上。这样就可以防止当前的区域被隐藏了。

siteNavigation.js (续)

```
if(nested[i].parentNode.getElementsByTagName( 'strong' ).length > 0 ){
    DOMhelp.cssjs( 'add', triggerLink.parentNode, sn.openClass );
    DOMhelp.cssjs( 'add', nested[i], sn.showClass );
}
```

事件的监听方法changeSection()所要做的就是得到事件的目标，检查父节点的第一个嵌套的UL元素是否已应用了show类，如果有的话则把那个UL移除掉。此外，它还需要将父节点的open类替换成parent类，反之亦然。

siteNavigation.js (续)

```
changeSection : function( e ){
    var t = DOMhelp.getTarget( e );
    var firstList = t.parentNode.getElementsByTagName('ul')[0];
    if(DOMhelp.cssjs( 'check', firstList, sn.showClass ) ) {
        DOMhelp.cssjs( 'remove', firstList, sn.showClass )
```

```

DOMhelp.cssjs( 'swap', t.parentNode, sn.openClass,➥
sn.parentClass );
} else {
    DOMhelp.cssjs( 'add', firstList,sn.showClass )
    DOMhelp.cssjs( 'swap', t.parentNode, sn.openClass,➥
sn.parentClass );
}
DOMhelp.cancelClick( e );
}
}
DOMhelp.addEvent( window, 'load', sn.init, false );

```

这段脚本应用到对应的HTML，并设置一种合适的样式，它就会为你提供了可以展开和折叠的导航条。与CSS相关的一部分如下所示：

```

siteNavigation.css ( 摘录 )

#nav.dyn li ul{
    display:none;
}
#nav.dyn li ul.show{
    display:block;
}
#nav.dyn li{
    padding-left:15px;
}
#nav.dyn li.parent{
    background:url(plus.gif) 0 5px no-repeat #fff;
}
#nav.dyn li.open{
    background:url(minus.gif) 0 5px no-repeat #fff;
}

```

可以通过分别地把它们的display属性设置为block和none来显示或隐藏嵌套的UL元素。这也把包含的链接拿出了一般的Tab顺序之外：如果键盘用户想要访问处于同一层的下一个元素而不用展开这个区域时，他们没有必要通过Tab键来遍历嵌套列表中的所有链接。如果他们首先敲回车键来展开显示的部分，那么可以通过Tab键在子菜单的链接中进行导航。

所有的LI元素都有一个左边的内边距以允许指示器图片显示有子链接或已打开的区域。类为open或parent的LI元素都有一个背景图片来指示它们的状态。

所有的这些都非常好，但如果你想给嵌套区域的父页面提供一个链接该怎么办呢？解决方法是在每一个父链接前添加一个新的链接图片，来完成它的显示、隐藏以及保留不变的功能。

演示页面exampleIndicatorSiteNavigation.html展示了其外观以及是其工作原理。脚本并不需要太多的修改内容：

```

siteNavigationIndicator.js ( 摘录 )

sn = {
    dynamicClass : 'dyn',

```

```
showClass : 'show',
parentClass : 'parent',
openClass : 'open',
```

第一个改动的地方是需要2个新的属性以提供要添加到嵌套列表的父节点的图片。这些会通过innerHTML来添加，如果需要的话维护人员可以很容易地把它们替换为其他的图片或文本。

siteNavigationIndicator.js (续)

```
parentIndicator : '' +
    '<title>open section</title>',
openIndicator : '' +
    '<title>close section</title>',
navID : 'nav',
init : function() {
    var parentLI, triggerLink;
    if( !document.getElementById || !document.createTextNode ){
        return;
    }
    var nav = document.getElementById( sn.navID );
    if( !nav ){ return; }
    DOMhelp.cssjs( 'add', nav,sn.dynamicClass );
    var nested = nav.getElementsByTagName( 'ul' );
    for( var i = 0; i < nested.length; i++ ) {
```

这里没有把父节点的第一个链接作为触发链接，你需要新建一个链接元素，把它的href属性设置为一个简单的井号以使它可以被单击，并且添加一个前面定义的父指示图片作为它的内容。接下来将链接的图片作为父节点的第一个子节点插入。

siteNavigationIndicator.js (续)

```
parentLI = nested[i].parentNode;
triggerLink = document.createElement( 'a' );
triggerLink.setAttribute( 'href', '#' )
triggerLink.innerHTML = sn.parentIndicator;
parentLI.insertBefore( triggerLink, parentLI.firstChild );
```

init()方法其余的部分还和原来基本相同，唯一不同的是你不仅要在父节点包含STRONG元素时应用类，还需要将“父节点”的指示图片替换成“打开”的图片。

siteNavigationIndicator.js (续)

```
DOMhelp.addEvent( triggerLink, 'click', sn.changeSection, false );
triggerLink.onclick = DOMhelp.safariClickFix;
DOMhelp.cssjs( 'add', parentLI, sn.parentClass );
if( parentLI.getElementsByTagName( 'strong' ).length > 0 ) {
    DOMhelp.cssjs( 'add', parentLI, sn.openClass );
    DOMhelp.cssjs( 'add', nested[i], sn.showClass );
```

```
parentLI.getElementsByTagName( 'a' )[0].innerHTML = sn.openIndicator
    }
},
},
};
```

`changeSection()`方法中不同的地方是，你需要通过把目标的节点名和A进行比较以确保事件目标是一个链接而不是图片。

siteNavigationIndicator.js (续)

```
changeSection : function( e ){
    var t = DOMhelp.getTarget( e );
    while( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
}
```

这个方法的其余部分保留不变，只有一点不同——除了应用不同的类以外，还需要改变链接的内容。

siteNavigationIndicator.js (续)

```
var firstList = t.parentNode.getElementsByTagName( 'ul' )[0];
if( DOMhelp.cssjs( 'check', firstList, sn.showClass ) ) {
    DOMhelp.cssjs( 'remove', firstList, sn.showClass );
    DOMhelp.cssjs( 'swap', t.parentNode, sn.openClass, sn.parentClass );
    t.innerHTML = sn.parentIndicator;
} else {
    DOMhelp.cssjs( 'add', firstList, sn.showClass )
    DOMhelp.cssjs( 'swap', t.parentNode, sn.openClass, sn.parentClass );
    t.innerHTML = sn.openIndicator;
}
DOMhelp.cancelClick( e );
}
DOMhelp.addEvent( window, 'load', sn.init, false );
```

以上所有的这些只是增强的网站导航的一个示例，它可能是最容易使用的一个了。使多层次下拉导航菜单可访问，例如，让鼠标和键盘用户都可以同样地使用它是一个很大的任务，它已不在本书的范围之内，因为它是一个非常高级的DOM脚本。

注解 在网站Mozilla.org上面，关于动态菜单为什么不应该通过tab功能来遍历各个链接而是提供一种基于光标的导航，有一段非常好的解释，在那里有很多优秀的开发人员在继续解决这个问题（打破了被证明是关于以可访问方式创建动态网页的一些HTML规则）：http://developer.mozilla.org/en/docs/key-navigable_custom_DHTML_widgets。有一个叫做Ultimate Drop Down Menu (<http://www.udm4.com>) 的商业脚本也支持这样做。

7.1.6 分页

分页意味着你将一个很大的数据集合分开放到多个页面中。这通常都是在后台完成的，但你可以使用JavaScript使检查那么一长列元素更快捷一些。

关于分页的一个示例是examplePagination.html，它在WindowsXP平台的Firefox 1.5浏览器上的效果如图7-5所示。

The screenshot shows a paginated table with the following data:

ID	Artist	Album	Comment
1	Depeche Mode	Playing the Angel	They are back and finally up to speed again
2	Monty Python	The final Rip-Off	Double CD with all the songs
3	Ms Kittin	I.com	Good electronica
4	Bad Religion	No control	My first Concert ever
5	Skinny Puppy	Digital Brap	Massive Digipack with live songs and videos

Navigation controls at the top: previous 1 to 5 of 22 next

Navigation controls at the bottom: previous 1 to 5 of 22 next

图7-5 把大量的数据行分页

要操作的内容是由同一HTML表中的许多行组成的，它使用了paginated类。

examplePagination.html（摘录）

```
<table class="paginated">
<thead>
<tr>
    <th scope="col">ID</th>
    <th scope="col">Artist</th>
    <th scope="col">Album</th>
    <th scope="col">Comment</th>
</tr>
</thead>
<tbody>
<tr>
    <th>1</th>
    <td>Depeche Mode</td>
    <td>Playing the Angel</td>
    <td>They are back and finally up to speed again</td>
</tr>
<tr>
    <th>2</th>
    <td>Monty Python</td>
```

```

<td>The final Rip-Off</td>
<td>Double CD with all the songs</td>
</tr>
[... and so on ...]
</tbody>
</table>

```

我们在上一章幻灯片显示的例子里使用了分页，尽管这个例子每次只显示一项。包含几个数据项的分页逻辑更复杂一些，但这个例子已给你提供了一些可以使用的技巧。

- 通过CSS来隐藏表的各行。
 - 定义每个页面应该显示多少行。
 - 显示第一页的各行并生成分页菜单。
 - 菜单有一个“previous”和“next”链接，还有一个计数器告诉用户显示的是哪个页面以及总共有多少项。
 - 如果当前的幻灯片是第一个，则“previous”链接应该是不可用的；如果是最后一个幻灯片，则“next”链接应该是不可用的。
 - “next”链接增加幻灯片的页数直到定义的总数，而“previous”链接则减少这个起始值。
- 在这个例子中你需要使用几个属性以及五个方法。给属性添加注释是一个很好的习惯，这样可以使以后的维护人员根据需要对其进行修改的时候更容易些。

`pagination.js`（大纲）

```

pn = {
  // CSS classes
  paginationClass : 'paginated',
  dynamicClass : 'dynamic',
  showClass : 'show',
  paginationNavClass : 'paginatedNav',
  // Pagination counter properties
  // Number of elements shown on one page
  increase : 5,
  // Counter: _x_ will become the current start position
  //           _y_ the current end position and
  //           _z_ the number of all data rows
  counter : '&#160;_x_&#160;to&#160;_y_&#160;of&#160;_z_&#160;',
  // "previous" and "next" links, only text is allowed
  nextLabel : 'next',
  previousLabel : 'previous',
}

```

使用了一个方法来初始化脚本，一个用来产生你所需要的额外的链接和元素，一个用来在各个页面间进行导航（也就是隐藏当前的结果集而显示下一个页面），一个用来显示当前页面，还有一个用来改变分页菜单。

```

init : function(){},
createPaginationNav : function( table ){},
navigate : function( e ){},

```

```

showSection : function( table, start ){},
changePaginationNav : function( table, start ){}
}
DOMhelp.addEvent( window, 'load', pn.init, false );

```

可以先喝杯咖啡或吃点饼干什么的，因为这是一个对你来说有点超前的脚本；但是不要着急——它的大部分逻辑都很简单。

`init()`方法会检查是否支持DOM，并循环遍历文档中的所有表元素。然后它检查表是否已应用了对应的类（定义在`pn.paginationClass`属性中）以及是否有比你每页要显示的行数还多的行（定义在`pn.increase`属性中）。如果这些条件有一个不满足，它就跳过其他的方法不添加任何菜单。

`pagination.js`（摘录）

```

init : function() {
  var tablebody;
  if( !document.getElementById || !document.createTextNode ){
    return;
  }
  var ts = document.getElementsByTagName( 'table' );
  for(var i = 0;i < ts.length; i++ ){
    if( !DOMhelp.cssjs( 'check', ts[i], pn.paginationClass ) ){
      continue;
    }
    if( ts[i].getElementsByTagName( 'tr' ).length <=
      pn.increase+1 ){
      continue;
    }
  }
}

```

由于你需要隐藏的数据行不包括标题行，只是包含表主体中的数据行，因此你需要将这一点告知其他方法。

最简单的选择是只把相关的行存储到表的一个属性中。获取表中的第一个TBODY，并把它所有的行存储在`datarows`属性中。还需要把所有行的数量保存在`datarowsize`中，并初始化`current`属性为`null`。

这个属性会存储你想显示的起始页面。通过把这些行的信息和行的数量保存为属性，可以使其他方法很容易从表获得信息而无需再一次从DOM中读取信息。

`pagination.js`（摘录）

```

tablebody = ts[i].getElementsByTagName( 'tbody' )[0];
ts[i].datarows = tablebody.getElementsByTagName( 'tr' );
ts[i].datarowsize = ts[i].datarows.length;
ts[i].current = null;

```

给表应用`dynamic`类以隐藏所有的表行。把当前表的一个引用作为参数调用`createPaginationNav()`方法来添加“previous”和“next”链接，再把这个表的引用和0作为参数调用`showSection()`以显

示第一个结果集。

`pagination.js (摘录)`

```
DOMhelp.cssjs( 'add', ts[i], pn.dynamicClass );
pn.createPaginationNav( ts[i] );
pn.showSection( ts[i], 0 );
}
},
```

`createPaginationNav()`方法并没有什么特别的地方；它所做的就是创建链接和计数器并添加指向`navigate()`方法的事件处理程序。开始先创建一个新的段落元素并给它添加一个分页菜单的类。

`pagination.js (摘录)`

```
createPaginationNav : function( table ){
  var navBefore, navAfter;
  navBefore = document.createElement( 'p' );
  DOMhelp.cssjs('add', navBefore, pn.paginationMenuClass );
```

以`previousLabel`属性值作为文本内容给段落添加一个新的链接，并给它再添加一个`SPAN`元素以在其中显示“previous”和“next”链接之间当前结果集的数目。预先把计数器设置为1作为起始值，把定义在`pn.increase`中的每个页面要显示的元素数目作为结束值，把所有数据集的行数作为总数。添加到新段落的最后一个元素是“next”链接。可以通过`parentNode`和`insertBefore()`把新的段落快速添加到表前。

`pagination.js (摘录)`

```
navBefore.appendChild( DOMhelp.createLink( '#', pn.previousLabel ) );
navBefore.appendChild( document.createElement( 'span' ) );
counter=pn.counter.replace( '_x_', 1 );
counter=counter.replace( '_y_', pn.increase );
counter=counter.replace( '_z_', table.datarowsize-1 );
navBefore.getElementsByTagName('span')[0].innerHTML = counter;
navBefore.appendChild( DOMhelp.createLink( '#', pn.nextLabel ) );
table.parentNode.insertBefore( navBefore, table );
```

在后面的表中显示相同的菜单是非常好的。不用再重新创建所有的这些元素，通过`parentNode`、`insertBefore()`和`nextSibling`把这个段落克隆一下并把它插入到表后面就可以了。然后，把每个段落的“previous”和“next”链接存储为它们自己的表属性，以便在其他方法中修改它们更容易些。

`pagination.js (摘录)`

```
navAfter = navBefore.cloneNode( true );
table.parentNode.insertBefore( navAfter, table.nextSibling );
table.topPrev = navBefore.getElementsByTagName('a')[0];
```

```
table.topNext = navBefore.getElementsByTagName('a')[1];
table.bottomPrev = navAfter.getElementsByTagName('a')[0];
table.bottomNext = navAfter.getElementsByTagName('a')[1];
```

不能在前面应用事件处理器，因为`cloneNode()`并没有克隆任何事件处理器。现在你可以为这些链接应用所有的事件处理器以及Safari浏览器的补丁。这个方法最后一个修改的地方是把计数器存储到属性中以使其他方法可以更容易地更新它们。

`pagination.js` (摘录)

```
DOMhelp.addEvent( table.topPrev, 'click', pn.navigate, false);
DOMhelp.addEvent( table.bottomPrev, 'click', pn.navigate, false);
DOMhelp.addEvent( table.topNext, 'click', pn.navigate, false);
DOMhelp.addEvent( table.bottomNext, 'click', pn.navigate, false);
table.bottomNext.onclick = DOMhelp.safariClickFix;
table.topPrev.onclick = DOMhelp.safariClickFix;
table.bottomPrev.onclick = DOMhelp.safariClickFix;
table.topNext.onclick = DOMhelp.safariClickFix;
table.topCounter = navBefore.getElementsByTagName('span')[0];
table.bottomCounter = navAfter.getElementsByTagName('span')[0];
},
```

事件监听方法`navigate()`需要检查是哪个链接调用了它。第一步是通过`getTarget()`获取事件的目标并通过把它的节点名和A相比较确保它是一个链接（记住，Safari会把链接中的文本节点作为事件的目标来传送）。

`pagination.js` (摘录)

```
navigate : function( e ){
  var start, table;
  var t = DOMhelp.getTarget( e );
  while( t.nodeName.toLowerCase() != 'a' ){
    t = t.parentNode;
  }
}
```

接下来它需要通过测试链接是否有一个`href`属性来检查它是否是活动的（稍后你会通过移除`href`属性来关闭“next”或“previous”链接）。如果没有的话，就什么也不做。下一个任务是从激活的链接中查找表。由于在表的上面和下面有导航，因此需要检查`previousSibling`或`nextSibling`是否有节点名`table`，并相应地定义变量`table`。

`pagination.js` (摘录)

```
if( t.getAttribute( 'href' ) == null || t.getAttribute( 'href' ) == '' ){ return; }
if( t.parentNode.previousSibling && t.parentNode.previousSibling.nodeName.toLowerCase() == 'table' ){
  table = t.parentNode.previousSibling;
} else {
  table = t.parentNode.nextSibling;
}
```

然后判断激活的链接是否是“next”链接中的一个或“previous”链接中的一个，而且相应地定义start为表的current属性加上或减去定义的增量。以获取的table和start的值作为参数调用showSection()。

`pagination.js (摘录)`

```
if( t == table.topNext || t == table.bottomNext ){
    start = table.current + pn.increase;
} else if ( t == table.topPrev || t == table.bottomPrev ){
    start = table.current - pn.increase;
}
pn.showSection( table, start );
},
```

showSection()方法调用了changePaginationNav()方法来更新链接和计数器，并且检查在表中是否已经有一个current参数。如果有的话，这意味着数据集的各行中存在需要移除的类。通过循环遍历存储在datarows属性的数据集的各行，并移除定义在showClass()的CSS类来除去它们。

`pagination.js (摘录)`

```
showSection : function( table, start ){
    var i;
    pn.changePaginationNav( table, start );
    if( table.current != null ){
        for( i=table.current; i < table.current+pn.increase; i++ ){
            if( table.datarows[i] ) {
                DOMhelp.cssjs( 'remove', table.datarows[i], pn.showClass );
            }
        }
    }
}
```

接下来从start开始循环到start加上预定义增量，并且添加CSS类来在表中显示这些行。注意你需要测试这些行是否存在；否则，你可能在最后的页面访问并不存在的行（假设一个有22个元素的列表；在第16个～第20个的页面上单击“next”链接就会显示第21个～第25个的元素）。要确保这个方法在下一次被调用时对应的幻灯片能够显示，剩下所要做的就是定义current属性为start值。

`pagination.js (摘录)`

```
for( i = start; i < start + pn.increase; i++ ){
    if( table.datarows[i] ) {
        DOMhelp.cssjs( 'add', table.datarows[i], pn.showClass );
    }
}
table.current = start;
},
```

如前所述，这个changePaginationNav()方法会在第一页把“previous”链接显示为不可用，

而在最后一页会把“next”链接显示为不可用。使链接可以看见但不可单击的技巧就是去掉[href](#)属性。

在第一页，start的值减去预定义的增量会导致产生一个负数，这点很容易检查。当这个数字比0大的时候，就再添加一次[href](#)属性。

`pagination.js` (摘录)

```
changePaginationNav : function( table, start ){
    if(start - pn.increase < 0 ) {
        table.bottomPrev.removeAttribute('href');
        table.topPrev.removeAttribute('href');
    } else {
        table.bottomPrev.setAttribute('href', '#');
        table.topPrev.setAttribute('href', '#');
    }
}
```

如果start加上一个增量比总行数还大，则需要移除“next”链接；否则的话，你应该让它可以使用。

`pagination.js` (摘录)

```
if(start + pn.increase > table.rowsize - 2 ) {
    table.bottomNext.removeAttribute('href');
    table.topNext.removeAttribute('href');
} else{
    table.bottomNext.setAttribute('href', '#');
    table.topNext.setAttribute('href', '#');
}
```

使用适当的值来更新计数器（start需要加一个1来让人更容易理解，而且你需要测试最后的值不能大于既有行的数量），而且你已从一个标准的数据表中创建了一个分页界面。

`pagination.js` (摘录)

```
var counter = pn.counter.replace('_x_', start+1 );
var last = start + pn.increase;
if( last > table.datarowsize ){ last = table.datarowsize;}
counter = counter.replace('_y_', last )
counter = counter.replace('_z_', table.datarowsize )
table.topCounter.innerHTML = counter;
table.bottomCounter.innerHTML = counter;
}
DOMhelp.addEvent( window, 'load', pn.init, false );
```

分页的逻辑还是一样的，即使你决定要显示或隐藏各列表项或其他HTML结构。还可以通过在“previous”和“next”链接之间显示各页面号（而不是计数器）使它更复杂一些，但是这里我把它留给你做进一步研究。

7.1.7 使用JavaScript进行导航小结

使用JavaScript来做一个强大的网站导航是很诱人的，因为很方便而且界面确实很华丽。但需要记住的是你应该经常关闭一下JavaScript功能，看看界面是否仍然可以正常工作。也可以不使用鼠标而用键盘来尝试同样的工作。

可以使用JavaScript让像深层嵌套导航菜单这样大量的数据更容易理解。可是，不要忘了一些用户要获得所有的导航信息而不用你的脚本把它分割成许多小服务，因此最好让使用整个站点地图作为数据源的菜单界面是一个选项，而不是给定的。我们会在下一章中看一下如何来实现它。

JavaScript导航记要

- 在用户没有单击或激活任何界面元素的情况下，不要给他转到其他的地址或提交表单数据。它会让用户感到很迷惑而不是有所帮助，甚至会被认为是一种安全隐患（如果你这样做的话，任何人都可以把用户发送到一个站点）
- 把数据隐藏但不要让它消失；当用一个灵活的界面能够使大量数据更容易理解的时候，一些用户仍然需要在一个服务中获得所有的数据，因此所有用户（包括那些网络连接速度很慢的用户）都不得不下载所有的这些数据。
- 在一个现有的网站导航模型上去做要比重新开发一个新的导航更安全。例如，很容易就可以把一个可用的使用了链接和锚的页内导航转变为使用标签的界面。通过JavaScript来创建所有必需的标签还有很多争议。

7.2 表单与JavaScript

下面，你会学到如何访问、读取和修改表单及它们的元素。在这里我们不会涉及表单验证的细节，我在第9章会对数据验证这个主题作专门的探讨。

我们仍然会谈到一些最基本的表单可用性，以及一些不好的习惯和它们应该避免的原因。首先，我们来看一下在本书的这一章和后面章节的一些示例中会用到的表单：

`exampleForm.html (摘录)`

```
<form method="post" action="send.php">
<fieldset>
  <legend>About You</legend>
  <p><label for="Name">Your Name</label></p>
  <p><input type="text" id="Name" name="Name" /></p>
  <p><label for="Surname">Your Surname</label></p>
  <p><input type="text" id="Surname" name="Surname" /></p>
  <p><label for="email">Your email</label></p>
  <p><input type="text" id="email" value="you@example.com" name="email" /></p>
</fieldset>
<fieldset>
```

```

<legend>Your message</legend>
<p><label for="subject">Subject</label>
<select id="subject" name="subject">
  <option value="generalEnquiry" selected="selected">General question</option>
  <option value="Webdesign">Webdesign</option>
  <option value="Hosting">Hosting</option>
  <option value="Training">Training</option>
  <option value="Partnership">Partnership</option>
  <option value="other">Other</option>
</select></p>
<p><label for="otherSubject">specify other subject</label>
<input type="text" id="otherSubject" name="otherSubject" /></p>
<p><label for="Message">Your Message</label></p>
<p><textarea id="Message" name="Message" cols="20" rows="5"></textarea></p>
</fieldset>
<fieldset>
  <legend>Email options</legend>
  <p><input type="checkbox" name="copyMeIn" id="copyMeIn" />
  <label for="copyMeIn">Send me a copy of this email to
  the above address</label></p>
  <p><input type="checkbox" name="newsletter" value="yes" checked="checked" id="newsletter" />
  <label for="newsletter">Sign me up for the newsletter</label></p>
  <p>Newsletter format:
  <input type="radio" name="newsletterFormat" id="newsHTML" checked="checked" value="html" />
  <label for="newsHTML">HTML</label>
  <input type="radio" name="newsletterFormat" id="newsPlain" checked="checked" value="plain" />
  <label for="newsPlain">Text</label></p>
  <p class="submit"><input type="submit" value="Send Form" /></p>
</fieldset>
</form>

```

注解 可以看到，这是一个严格型HTML 4文档的合法表单，所有的元素都是按照XHTML兼容的方式关闭的。还需要注意的是，在与XML兼容的HTML中，你需要把单个属性如selected和checked对应地写成selected="selected"与checked="checked"。

表单把分组元素的表单域集按其特性划分为逻辑单元，并通过特定的表单元素给连接的解释文本做上标签。这对表单的可访问性非常有用，因为它提供了逻辑组织，可以避免混乱。

7.2.1 JavaScript 表单基础

使用JavaScript来访问和修改表单可以通过几种不同的方式来实现。通常，有一种简单的访问表单的方式是DOM脚本，而且可以通过getElementsByName()和getElementById()来访问它们的元素，但是还有一个叫做forms的对象，它包括了当前document中的所有表单。

这个对象允许你通过3种不同的方式来访问document中的表单：

- 通过它们的index编号来访问，这里index编号是个整数，例如，`document.forms[2]`表示第3个表单。
- 通过在name属性中定义的名字为一个对象来访问，例如`document.forms.myForm`。
- 通过在name属性中定义的名字为一个字符串来访问，例如`document.forms['myForm']`。

1. 表单属性

`forms`对象本身只有一个属性`length`，它保存着在文档中的表单数量。

可是，对于每一个表单都有许多属性可以使用，这些属性都可以读取和修改的。

- `action`: 当表单被提交的时候，表单数据要发送到的脚本。
- `encoding`: 定义在FORM元素的`enctype`属性中的表单的编码。
- `method`: 表单的提交方法——POST或者GET。
- `name`: 定义在name属性（不是定义在id中！）中的表单名称。
- `target`: 表单数据应该被发送到的目标（只有在使用框架或多个窗口的时候才重要）。

2. 表单方法

`Form`对象只有两种方法。

- `reset()`: 重新设置表单为它的初始状态，这意味着用户所有的输入和选择都被撤销，并且表单显示定义在`value`、`selected`或每个元素选择的`attributes`中的初始值。注意一下他们的不同，`reset()`方法并没有清除表单，而是把它恢复到其初始状态。这与用户单击了表单上的重置按钮效果是一样的。
- `submit()`: 提交表单。

这两种方法都模拟了浏览器的功能，它与激活一个重置或提交按钮是一样的，因此你要确保不要使用它们去掉用户必要的交互。要在用户单击提交按钮或在键盘上敲回车键的时候进行表单提交有一个很好的理由，就是它是最容易理解的方式，而且如果你乱用这个功能并在用户处理其他元素时提交表单，那么就会强迫他们过早地提交表单。

7.2.2 表单元素

`forms`集合里的每一个表单都有一个对象属性叫做`elements`，它其实是这个表单中所有表单元素的一个数组（与所有的HTML元素是相反的）。可以通过与最初访问表单相同的方式来访问这些元素：索引编号、对象名称或在方括号中字符串形式的名称。

- `var elm = document.forms[0].elements[2];`
- `var elm = document.forms.myForm.elements.myElement;`
- `var elm = document.forms.myForm.elements['myElement'];`

在上一个例子中你可以看到，可以把这些符号混合匹配起来使用。也可以使用变量来代替索引号或方括号中的字符串。

`elements`集合本身有一个叫做`length`的只读属性。例如，可以使用这个属性来循环遍历表单中的所有属性，并读取它们的类型：

```

var myForm = document.forms[0];
var formElements = myForm.elements;
var all = formElements.length;
for( var i = 0; i < all; i++ ) {
    alert( formElements[i].type );
}

```

集合内的每一个元素又有几个属性，支持哪一个属性取决于元素的类型。下面我会列出所有的这些属性，并在括号内列出支持该属性的元素。

- checked: 布尔值，表示该元素是否被选中（按钮、复选框和单选按钮）。
- defaultChecked: 布尔值，表示该元素在初始化时是否被选中（按钮、复选框和单选按钮）。
- value: 在value属性中定义的元素的值（除选择框以外的所有元素）。
- defaultValue: 元素的初始值（文本框和文本区域）。
- form: 元素所在的表单（只读——所有元素）。
- name: 元素的名称（所有元素）。
- type: 元素的类型（只读——所有元素）。

一个特殊的元素类型是选择框，它自身有一个集合，属性名叫options——后面会有更多介绍。每个元素有一系列的方法，也都取决于元素的类型。这些方法都没有参数。

- blur(): 从该元素中移除用户代理的焦点（所有元素）。
- focus(): 把用户代理的焦点放到该元素上（所有元素）。
- click(): 模拟用户单击该元素（按钮、复选框、文件上传域、重置以及提交按钮）。
- select(): 选择并突出显示该元素的文本内容（密码框、文本框以及文本区域）。

注解 click()看起来意义不太，但当你处理Web应用程序的时候非常有用，开发环境中的中间层会对表单的提交过程作处理，如Java Spring和.NET。可是这不是一个JS初学者的环境，因此超出了本书的范围。

1. 不包含在元素集合中的HTML属性

除了elements集合的属性，你还可以读取和设置（当然是在浏览器允许的情况下）这些元素的属性，只要通过forms和elements集合能访问到它。例如，可以通过修改它的cols和rows属性来改变文本框的大小：

```

var myTextBox = document.forms[0].elements[2];
if ( myTextBox.type == 'text' ){
    myTextBox.rows = 10;
    myTextBox.cols = 30;
}

```

提示 注意在设置元素的属性之前，最好检查一下要操作的元素的类型，以防它们对这个元素不可用，例如，一个select元素就没有cols或rows属性。

2. 全部支持的属性

所有的表单元素开始都就支持type、name、form和value属性。一个新近的变化是文件上传域不再支持设置value了，因为当用户在一个受感染的电脑上上传东西的时候，会允许恶意的脚本注入到要上传到服务器上的文件。

当通过DOM方法直接访问一个元素的时候，使用form可以很容易地访问到父表单。例如，通过DOM在这个例子的表单中访问email域：

```
var mail = document.getElementById('email');
```

可以通过mail.form或mail.parentNode.parentNode.parentNode来访问表单以改变它的一个属性或提交表单。

exampleForm.html（摘录）

```
<form method="post" action="send.php">
<fieldset>
  [... code snipped ...]
  <p><input type="text" id="email" value="you@example.com" name="email" /></p>
</fieldset>
  [... code snipped ...]
</form>
```

不管元素嵌套在表单中有多深，使用form可以免去计算节点的麻烦，并且使得脚本更加容易维护，因为你是不依赖于HTML的。如果想使用节点进行排他的遍历，也可以使用递归的循环来检查父节点的nodeName以实现HTML标记同样的独立：

```
var mail = document.getElementById('email');
parentForm = mail.parentNode;
while( parentForm.nodeName.toLowerCase() != 'form' ) {
  parentForm = parentForm.parentNode;
}
```

虽然这是独立于用户代理的，但是对大多数网站来说可能有点过分，因为你可以假设大多数的浏览器确实提供了forms和elements集合（当然你可以对它们进行测试以确认）。使用forms和elements集合也意味着支持非DOM-2的浏览器，如Netscape 4或IE 4。

3. 使用blur()和focus()

可以使用blur()从元素中去掉用户代理的焦点，或者使用focus()来设置焦点。这样的危险是blur()并没有指定它应该设置到哪一个目标，这意味着用户代理会把焦点移到下一个元素、浏览器的地址栏或其他的地方。对于使用鼠标的看得见的用户来说，这并不是一个问题，但是对于依赖于辅助技术的盲人用户或键盘用户来说，就会有一个要在网页中再次找到他们处理的地方的问题。

当你看一些网站的代码的时候，可能会遇到类似下面这样的东西：

```
<a href="#" onclick="dothings();" onfocus="this.blur()">Home</a>
```

开发人员过去常常通过上面的方式阻止浏览器在当前链接上显示一个蓝框（Mac上的IE）或星点边线（PC上的IE/Mozilla）。这是一个非常不好的做法，因为键盘用户在敲回车时不知道他当前能访问到哪一个元素。

虽然有一些合理的理由使用`focus()`，但在大多数的情况下，不主张改动自动的表单输入顺序。并不是每一个用户都能看到表单，而且即使看得见的用户也可能没注意到它。

尤其是在一些比较长的表单中，有很多不同的数据要输入，你会发现用户并没有看屏幕，而是在盲打，就像他们要从打印输出或他们的信用卡、护照以及其他的东西中读取数据。在它们之间检查表单，却发现没有输入正确的域或仍然停留在前面弹出错误的地方，是十分令人沮丧的。

4. 文本域、文本区域、隐藏以及密码域

文本域、文本区域、隐藏以及密码域可能是你必须处理的最常用的域了，因为它们是用户输入内容的地方。

除了都支持的表单元素属性，它们还支持元素属性`value`和`defaultValue`。这二者的区别是如果用户改变了元素中的内容，它确实改变`value`属性但没有改变`defaultValue`的属性。这也意味着当用户修改该域的`value`的时候，这个变化是可见的，而当你改变`defaultValue`的时候，它是不可见的。如果想改变元素的默认值并且使它可见，你需要紧接着调用`reset()`方法。在这个例子的文档中，`email`域中有一个默认值：

```
<p><input type="text" id="email" value="you@example.com"➥
name="email" /></p>
```

可以按照以下的方式来读取它的值和默认的值：

```
var mail= document.getElementById( 'email' );
alert( mail.defaultValue );
alert( mail.value );
```

当用户在这个域上还没有做什么修改的时候，这两个值都是`you@example.com`。可是，如果用户已经在这个域中输入了`me@otherexample.com`，这两个值就会变得不同。

为`defaultValue`找一个示例是需要技巧的，这不意味着必须使用JavaScript来处理应该是后台的任务。有一个例子可以测试当前的域是否德语并且把Email域中的默认值修改为一个德语的邮件地址。下面的代码应该在文档加载后执行：

```
var mail = document.getElementById( 'email' );
if( window.location.host.indexOf( '.de' ) != -1 ) {
    mail.defaultValue = 'email@adresse.de';
    mail.form.reset();
}
```

注意你需要调用`reset()`方法来使这个修改可见。可以在`exampleFormGermanPreset.html`中看到发生的变化（我们在那里通过不让主机来进行测试而让它可以被看见）。

注解 对于TEXTAREA元素，可以像其他表单文本元素一样读写`value`和`defaultValue`的值。但是，HTML标签没有`value`属性——初始的和修改的值是包括在开始标签和结束标签之间的文本。

文本元素有一个方法叫select()，它把文本元素内部的所有文本都突出显示以方便复制和粘贴。这通常都被认为是网络杂志和在线文档系统的特征。

5. 复选框

复选框提供了一个明确的“是”或“否”的选择，它们可以在服务端很容易读取出来（在选中的情况下，表单会发送复选框的名字和值，而在用户还没有选择它的时候则不发送名字），因此要比一组单选按钮或有一个“是”和“否”选项的下拉选择框更容易使用。

除了前面描述的都支持的元素属性以外，复选框还有checked和defaultChecked两个属性，两个都是布尔值，代表这个对象是否被选中。可以对这两个属性进行读写，但你需要重置表单以使对defaultChecked的修改可见。

与复选框有关的常用JavaScript是，为用户提供选择所有复选框的机会或取消在许多复选框中的选择。其中的一个示例就是基于网站的邮件系统，如图7-6所示。这些功能的逻辑非常简单：循环遍历所有的元素，检查每一个元素的type，再对应地修改checked属性。可以在exampleFormCheckboxes.html中看到这样的示例。



图7-6 通过JavaScript成批地修改复选框

这个例子中有3个按钮在用户单击它们的时候调用了同样的函数：changeBoxes()。每个按钮提供了一个不同的数字作为函数的唯一参数：1表示全选，-1表示取消所有选择，0表示什么也不选择。

exampleFormCheckboxes.html（摘录）

```
<input type="button" onclick="changeBoxes(1)" value="select all" />
<input type="button" onclick="changeBoxes(-1)" value="invert selection" />
<input type="button" onclick="changeBoxes(0)" value="select none" />
```

这个可以使你用这个函数改变复选框的时候能会简单些。循环遍历这个页面内的找到的第一个表单的所有元素。如果元素的type不是checkbox，则不执行其余部分继续循环。

如果这个元素是个复选框，判断action的值是否比0小，并且当它为true的时候通过把checked修改为false来反转复选框的状态，反之亦然。如果action的值为0或比0大，把action的值设置为

`checked`就可以了。

exampleFormCheckboxes.html (摘录)

```
function changeBoxes( action ) {
    var f = document.forms[0];
    var elms = f.elements;
    for( var i = 0; i < elms.length; i++ ){
        if( elms[i].type != 'checkbox' ){ continue; }
        if( action < 0 ){
            elms[i].checked = elms[i].checked ? 0 : 1;
        } else {
            elms[i].checked = action;
        }
    }
}
```

如果你还有点疑惑，那么记住`checked`属性就是个布尔值。这意味着当它是`true`或`1`的时候，它就被选中了；当它为`false`或`0`的时候，它就未被选中。如果你只用到`true`或`false`两个关键字，那么在`else`条件中必须添加另一种情况（在这种情况下使用三元运算符）：

```
function changeBoxes(action) {
    var f = document.forms[0];
    var elms = f.elements;
    for( var i = 0; i < elms.length; i++ ){
        if( elms[i].type != 'checkbox' ){ continue; }
        if( action < 0 ){
            elms[i].checked = elms[i].checked ? false : true;
        } else {
            elms[i].checked = action == 1 ? true : false;
        }
    }
}
```

使用三元操作符，还甚至可以把这段脚本的整个复选框逻辑部分减少到一行：

```
function changeBoxes( action ) {
    var f = document.forms[0];
    var elms = f.elements;
    for( var i = 0; i < elms.length; i++ ){
        if( elms[i].type != 'checkbox' ){ continue; }
        elms[i].checked = action < 0 ? (elms[i].checked ? 0 : 1) : action;
    }
}
```

由于许多复杂的表单代码未必是由面向客户端的开发人员创建的，你可能遇到类似这种结构的几率很高，这就是为什么我要在这里给你特别说明一下的原因。

6. 单选按钮

单选按钮的英文是radio button，这是因为它们看上去就像老式收音机上的调谐度盘。它们和

复选框行为类似，不同之处在于单选按钮属于有同一name的一个组，而且用户只能排他地选择一个。单选按钮对于鼠标用户和键盘用户都很容易使用，而且万一使用选择框出现问题，它们可以很好地替代短小的选择框。

它们和复选框一样有两个布尔类型的属性checked和defaultChecked，但是当你选择了其中一个的时候，会自动将同组中其他的单选按钮的checked属性设置成false。再说一下，可以读取和修改checked和defaultChecked，而且需要重新设置表单以便对defaultChecked的修改可以看得到。由于上面的HTML例子只有一个拥有两个选项的单选按钮组，因此来选择一个不同的例子：

```
exampleFormRadioGroup.html ( 摘录 )
<form method="post" action="send.php">
<fieldset>
<legend>Step 1 of 3 - Your favourite Character </legend>
<p>
  <input type="radio" name="character" id="charC"
  value="Calvin" checked="checked" />
  <label for="charC">Calvin</label>
</p>
<p>
  <input type="radio" name="character"
  id="charH" value="Hobbes" />
  <label for="charH">Hobbes</label>
</p>
<p>
  <input type="radio" name="character"
  id="charSd" value="Susie Derkins" />
  <label for="charSd">Susie Derkins</label>
</p>
<p>
  <input type="radio" name="character"
  id="charS" value="Spaceman Spiff" />
  <label for="charS">Spaceman Spiff</label>
</p>
<p>
  <input type="radio" name="character"
  id="charSm" value="Stupendous Man" />
  <label for="charSm">Stupendous Man</label>
</p>
</p>
</fieldset>
<p class="submit"><input type="submit" value="Next Step" /></p>
</form>
```

注解 这也是一个好机会，可以展示name和id之间的不同之处。虽然一组单选按钮都共有同一个name（在这个例子中是character），但每一个都必须有一个唯一的id以便标签连接到它们。

标签不仅对于屏幕阅读这样的辅助技术很方便，而且使得表单更简单易用；因为用户可以单击复选框旁边的名字来选择它们。

这个HTML例子包含了一些按钮，用来展示JavaScript的输出；可以通过在浏览器中打开它来测试一下输出。该脚本说明了如何来处理单选按钮：

```
formRadioGroup.js

function setChoice( n ) {
    var f = document.forms[0];
    f.character[n].checked = true;
}
function getChoice() {
    var f = document.forms[0];
    var choices = f.elements.character;
    for(var i = 0; i < choices.length; i++){
        if( choices[i].checked ){ break; }
    }
    alert( 'Favourite Character is: ' + choices[i].value );
}
```

可以使用一个共有的名字把单选按钮组作为一个数组来访问（在这个例子中是character）。设置单选按钮组的选项非常简单明了：setChoice()函数使用一个数字作为参数（n），读取第一个表单（forms[0]），并且再把第n个字符项的checked属性设置为true。

formRadioGroup.js（摘录）

```
function setChoice( n ) {
    var f = document.forms[0];
    f.character[n].checked = true;
}
```

如果在这个例子中单击“set choice to Hobbes”按钮，你会看到单选按钮的显示发生变化，如图7-7所示。

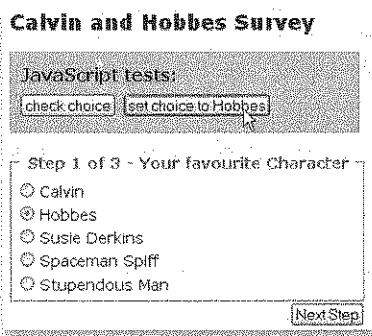


图7-7 在单选按钮组中改变所选择的选项

读取当前选择的选项也非常容易：选择第一个表单，把字符列表保存到一个叫choices的新变量中，并循环遍历它。接下来，检查数组中每一个元素的checked属性，并且当你找到一个返回值为true的checked属性时就跳出循环。这就是当前所选择的单选按钮：在共有同一个名字的

单选按钮组中只能有一个被选中。

formRadioGroup.js (续)

```
function getChoice() {
    var f = document.forms[0];
    var choices = f.elements.character;
    for( var i = 0; i < choices.length; i++ ) {
        if( choices[i].checked ){ break; }
    }
    alert( 'Favourite Character is: ' + choices[i].value );
}
```

7. 按钮

在HTML中有3种按钮：两个不用脚本可以工作；一个包含在规格说明书中，只有关联到脚本的时候才可以工作。

作者可能创建下面3种类型的按钮。

提交按钮：当被激活的时候，提交按钮会提交表单。一个表单可以包含不止一个提交按钮。

重置按钮：当被激活的时候，重置按钮会把所有的控件设置为其初始值。

普通按钮：普通按钮没有默认的行为。每个普通按钮都具有与元素的事件属性相关联的客户端脚本。当一个事件发生时（例如，用户按下按钮，松开它等），关联的脚本就会被触发。

——<http://www.w3.org/TR/REC-html40/interact/forms.html#buttons>

这使得“普通按钮”——输入类型的按钮或按钮元素——成为只使用JavaScript功能最好的触发元素。

另一方面，重置和提交按钮是表单最重要的部分，而且除非你有很好的修改理由，否则不要随意篡改它们。一个重复的请求是当表单被提交的时候修改提交按钮的值或状态以防止缺乏耐心的用户多次单击按钮。你可以通过一个click处理程序来实现这个功能；然而，更好的选择是在表单上使用一个submit处理程序，因为这样表单通过回车键被提交时也会引起改变。图7-8显示了这种效果。

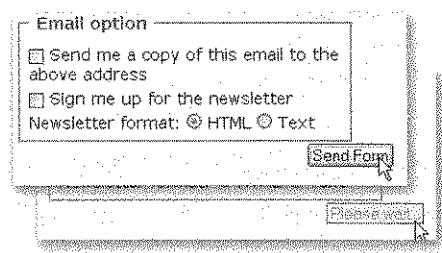


图7-8 表单提交时改变提交按钮的样式和文本内容

实现这个功能你所要做的就是，在表单提交的时候给这个窗口指派一个调用init()函数的事件处理程序，以及调用change()函数的另一个事件处理程序。

这个函数会循环所有的表单元素（在通过getTarget()获得表单后），并且检查这个元素是否是一张图片或一个提交按钮。如果是这种情况的话，它会通过disabled属性禁用这个按钮，并且将按钮的值修改为Please wait：

```
exampleChangeSubmitButton.html (摘录)

submitChange = {
  init : function(){
    DOMhelp.addEvent( document.forms[0], 'submit',➥
      submitChange.change,false);
  },
  change : function( e ){
    var t = DOMhelp.getTarget( e );
    for( var i = 0; i < t.elements.length; i++ ){
      if( !/submit|image/.test( t.elements[i].type ) ) { continue; }
      t.elements[i].disabled = true;
      t.elements[i].value = 'Please wait...';
    }
  }
}
DOMhelp.addEvent( window, 'load', submitChange.init ,false );
```

通过定义的图片按钮表现与提交按钮相同，唯一的不同之处在于它不会提交自己的名称到后台，而是两组名称/值对，包括最初的名称后面跟一个.x和.y以及用户单击的坐标值。这样你可以根据按钮单击的位置来执行不同的动作。这个信息通过JavaScript是无法读取的，只能显示在后台。

从JavaScript的角度出发，使用一个图片输入仅止于此，除非你可以给它提供一个翻转状态。

8. 选择框

选择框可能是表单元素中最复杂而且是最通用的了。设计师都喜爱它们，因为它可以存储许多选项，让用户在一个很小的屏幕空间上从中作出选择（另一方面，设计师也讨厌他们，因为它们不能使用CSS来完全地设置它们的样式）。

每一个选择框都有一个叫options的列表对象，它有下面几个属性。

- length: 在选择框中所有选项的数目。
- selected: 布尔值，表示选项是否被用户选择了。
- selectedIndex: 所选择元素的索引值。如果没有元素被选中，它会返回-1（顺便在这里提一下，它实际上是SELECT元素的一个属性）。
- text: 选项的文本内容。
- value: 选项的值。

注解 注意text和value是包含在选择框中的每一个选项的属性；通过读取选择框对象本身的value属性无法得到所选择的值——因为根本就没这样的东西。

有两种选择框：单选选择框和多选选择框。前者只允许选择所有选项中的某一个选项，后者允许用户通过按住Ctrl键并将需要的选项突出显示来选择多个选项。

注解 多选选择框对于需要辅助工具的用户和键盘用户来说，使用起来就是一个恶梦，这就是为什么你想用复选框列表来代替它的原因。这样也可以使在服务器端读出选择项更容易。

读取单选选择框的信息相当容易。例如，拿示例表单中的选择框来说：

exampleSelectChoice.html（摘录）

```
<p>
  <label for="subject">Subject</label>
  <select id="subject" name="subject">
    <option value="generalEnquiry" selected="selected">General question</option>
    <option value="Webdesign">Webdesign</option>
    <option value="Hosting">Hosting</option>
    <option value="Training">Training</option>
    <option value="Partnership">Partnership</option>
    <option value="other">Other</option>
  </select>
</p>
```

获得选择框最快的方式是使用元素的name而不是index。原因是这个选择框的type依据是否设置了multiple属性可以是select-one或者select-multiple。一旦你得到了确切的对象，就可以使用它的selectedIndex属性来读出所选择的选项，并通过使用selectedIndex作为列表的计数器来显示选项的value或text：

exampleSelectChoice.html（摘录）

```
function checkSingle() {
  var f = document.forms[0];
  var selectBox = f.elements['subject'];
  var choice = selectBox.selectedIndex;
  alert( 'You chose ' + selectBox.options[choice].text )
}
```

而在多选选择框的情况下，这是不够的，因为用户可能已选择了多个选项（selectedIndex只会返回第一个选择）。

不用selectedIndex，你将不得不循环遍历所有的选项，并测试每一个选项的selected属性：

exampleSelectChoice.html (摘录)

```

function checkMultiple() {
    var f = document.forms[0];
    var selectBox = f.elements['multisubject'];
    var choices=[];
    for( var i = 0; i < selectBox.options.length; i++ ) {
        if( selectBox.options[i].selected == 1) {
            choices.push( selectBox.options[i].text );
        }
    }
    alert( choices.join(',') );
}

```

可以在elements集合中通过选择框的name获得它，并且新建一个叫做choices的数组（[]是newArray()的缩写符号）。循环遍历选择框中的每一个options，并且检查它的selected属性是否为true。若是，则将选项的text值作为一个新的数组项放到choices中。然后使用数组的join()方法把这个数组转换成字符串形式并显示它。

这样读取值的方法也同样适用于单选选择框；可是，根据可用选项的数量，它可能有点太强大了。通过依赖于元素类型而读出选项，你可以把两个方法一起放到一个更通用的函数中：

exampleSelectChoice.html (摘录)

```

function getSelectValue( fieldName ) {
    var f = document.forms[0];
    var selectBox = f.elements[fieldName];
    if( selectBox.type == 'select-one' ) {
        var choice = selectBox.selectedIndex;
        alert( 'You chose ' + selectBox.options[choice].text );
    } else {
        var choices = [];
        for( var i = 0;i < selectBox.options.length; i++ ){
            if( selectBox.options[i].selected == 1 ) {
                choices.push( selectBox.options[i].text );
            }
        }
        choices.join( ',' );
        alert( choices );
    }
}

```

9. 添加、替换和移除选项

选择框不同于表单元素，它可以在一定范围内通过程序来添加和删除选项。可以通过使用Option构造函数来添加一个新选项，并将它添加到选项列表中：

```
extraOption = new Option(value, text, defaultSelected, selected);
```

例如，如果需要的话，可以将“DOM scripting”作为一个项目添加到列表中，可以通过如

下来的方式来实现：

exampleSelectChoice.html (摘录)

```
function addOption(fieldName) {
    var f = document.forms[0];
    var selectBox = f.elements[fieldName];
    var extraOption = new Option('DOM scripting', 'domscripting', 0, 0);
    selectBox.options[selectBox.options.length] = extraOption;
}
```

可以通过设置其为null来移除一个选项：

exampleSelectChoice.html (摘录)

```
function removeOption(fieldName,i) {
    var f = document.forms[0];
    var selectBox = f.elements[fieldName];
    selectBox.options[i] = null;
}
```

替换选项也很容易；只要简单地将旧的选项设置成新的选项就可以了：

exampleSelectChoice.html (摘录)

```
function replaceOption( fieldName, i ) {
    var f = document.forms[0];
    var selectBox = f.elements[fieldName];
    var extraOption = new Option( 'DOM scripting', 'domscripting', 0, 0 );
    selectBox.options[i] = extraOption;
}
```

在一个选项前插入另一个选项稍微麻烦一点，因为你需要在重写选项集合前复制所有的选项。函数insertBeforeOption()有2个参数：表单元素的名字和你想要在其前插入新选项的选项索引号。在查找选择框并且创建新选项前，先得定义两个循环计数器i和j，以及一个叫opts的空数组。

exampleSelectChoice.html (摘录)

```
function insertBeforeOption( fieldName, n ) {
    var i = 0, j = 0, opts = [],
        var f = document.forms[0];
    var selectBox = f.elements[fieldName];
    var extraOption = new Option('DOM scripting', 'domscripting', 0, 0);
```

然后将选择框的选项存储到一个叫old的变量中并对它们进行循环遍历，为它们每一个创建一个新选项，并把它们的属性赋给新选项。

exampleSelectChoice.html (续)

```

var old = selectBox.options;
for( i = 0; i < old.length; i++ ) {
    opts[i] = new Option(old[i].text, old[i].value,➥
        old[i].defaultSelected, old[i].selected );
}

```

新列表会多出一个元素，这就是为什么在循环遍历新列表前增加length属性的原因。检查循环计数器是否与传递给函数的参数一样，如果是则插入一个新选项。

exampleSelectChoice.html (续)

```

old.length++;
for( i = 0; i < old.length; i++ ) {
    if( i == n ) {
        old[i] = extraOption;
    }
}

```

否则，把这个选项设置为旧的选项，并增加计数器变量j。注意在这里你还需要另一个计数器，因为在循环内无法修改变量i。由于新的选项列表会多一个项，你需要使用j来获得存储在opts数组中的值。

exampleSelectChoice.html (续)

```

} else {
    old[i] = opts[j];
    j++;
}
}
}

```

依赖于选择框中选项的数量，这可能会变成一个速度慢而且要求高的脚本。可以通过DOM来达到同样的效果并且速度快代码少。

exampleSelectChoice.html (摘录)

```

function insertBeforeOptionDOM( fieldName, i ) {
    var selectBox = document.getElementById( fieldName );
    if( !selectBox ){ return false; }
    var opt = selectBox.getElementsByTagName( 'option' );
    var extraOption = document.createElement( 'option' );
    extraOption.setAttribute( 'value', 'domscripting' );
    extraOption.appendChild( document.createTextNode(➥
        'DOM Scripting' ) );
    selectBox.insertBefore( extraOption, selectBox.options[i] );
}

```

选择框是Web应用程序开发的一大部分，而且常被用作通过来回地移动元素而对两个列表进行排序的界面。

7.2.3 交互式表单：隐藏或显示独立元素

关于JavaScript和表单很酷的一点是，你可以使表单比通常所见的更迷人、更动态。让所有的元素都可以交互，并且可以立即提交表单而无需用户单击提交按钮或按回车键是很有诱惑力的。

这样做的危险是不只牺牲了对用户代理的支持和视觉效果，而且很可能使用户过早地发送数据。

当只是简单地修改界面或者显示在表单中的数量的时候，使用change处理程序是很安全的。我们来使用演示的表单作为一个例子。你可能已注意到有些区域有逻辑上的联系：“other subject”文本域只有在“Other”选项被选中时才有意义，而且以HTML或普通文本方式接受时事通信的选项也只有在用户已经选择订阅了时事通信的时候才有用。

exampleDynamicForm.html（摘录）

```
<form method="post" action="send.php">
[... code snipped ...]
<p><label for="subject">Subject</label>
<select id="subject" name="subject">
<option value="generalEnquiry" selected="selected">General question</option>
<option value="Webdesign">Webdesign</option>
<option value="Hosting">Hosting</option>
<option value="Training">Training</option>
<option value="Partnership">Partnership</option>
<option value="other">Other</option>
</select></p>
<p><label for="otherSubject">specify other subject</label>
<input type="text" id="otherSubject" name="otherSubject" /></p>
[... code snipped ...]
<p><input type="checkbox" name="newsletter" value="yes" id="newsletter" />
<label for="newsletter">Sign me up for the newsletter</label></p>
<p>Newsletter format:<br/>
<input type="radio" name="newsletterFormat" id="newsHTML" value="html" checked="checked" />
<label for="newsHTML">HTML</label>
<input type="radio" name="newsletterFormat" id="newsPlain" value="plain" />
<label for="newsPlain">Text</label></p>
```

通过脚本可以隐藏这些选项，只有用户在选择了适当的选项的时候才让它们显示出来。图7-9显示了它在浏览器中的效果。

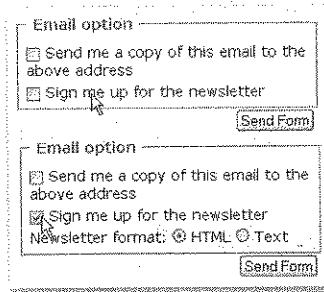


图7-9 根据用户的选择来显示或隐藏表单元素

定义一个你要应用到想隐藏的元素的类以及两个动态元素的ID，把它们作为主要对象df的属性。

dynamicForm.js

```
df = {
  hideClass : 'hide',
  letterOption : 'newsletter',
  subjectOption : 'subject',
```

init()方法会检查是否支持DOM以及必需的元素是否可用。

dynamicForm.js (续)

```
init : function(){
  if( !document.getElementById || !document.createTextNode ){
    return;
  }
  df.news = document.getElementById( df.letterOption );
  df.subject = document.getElementById( df.subjectOption );
  if( !df.subject || !df.news ){ return; }
```

下一步需要找到元素来隐藏。通过使用DOMhelp方法closestSibling()，可以确保你不会试着隐藏换行，而是你实际上想要访问的元素。把这些属性保存到主对象的属性中以使事件处理程序的其他方法可以方便地访问它们。

可以通过给它们添加隐藏类来隐藏这些元素，并且给复选框指派一个指向letterChange()方法的click事件处理程序以及给选择框指派一个指向subjectChange()的change处理程序。

dynamicForm.js (续)

```
df.newsOpt = DOMhelp.closestSibling( df.news.parentNode, 1 );
df.subjectOpt = DOMhelp.closestSibling( df.subject.parentNode, 1 );
DOMhelp.cssjs( 'add', df.newsOpt, df.hideClass );
DOMhelp.cssjs( 'add', df.subjectOpt, df.hideClass );
DOMhelp.addEvent( df.news, 'click', df.letterChange, false );
DOMhelp.addEvent( df.subject, 'change', df.subjectChange, false );
},
```

在测试getTarget()的checked属性前，在letterChange()方法中通过它来获取复选框。如果这个属性是checked，那么就移除隐藏类；否则的话，就添加隐藏类。

dynamicForm.js (续)

```
letterChange : function( e ){
  var t = DOMhelp.getTarget( e );
  var action = t.checked ? 'remove' : 'add';
  DOMhelp.cssjs( action, df.newsOpt, df.hideClass );
},
```

`subjectChange()`方法的工作原理也与上相同：获得`target`并检查第5个选项是否被选中（就是`selectedIndex`是否等于4）。如果是，则从选项元素中移除隐藏类；否则的话就添加隐藏类。另外，该方法将浏览器的焦点设置为新显示的元素以使用户可以立即开始输入。

`dynamicForm.js`（续）

```
subjectChange : function( e ) {
    var t = DOMhelp.getTarget( e );
    var action = t.selectedIndex == 5 ? 'remove' : 'add';
    DOMhelp.cssjs( action, df.subjectOpt, df.hideClass );
    if(action == 'remove') {
        df.subjectOpt.getElementsByTagName( 'input' )[0].focus();
    }
}
}

DOMhelp.addEvent(window, 'load', df.init, false );
```

显示与隐藏关联的元素是让表单的一部分关联到其他选项的一种方法。另一种不同的方法是让保持它们可见，但是添加了一个`disabled`属性。这使用户无法修改它们，而且浏览器显示它们的时候也会变灰。

这比隐藏元素通用性要少，因为`disabled`属性只适用于`input`、`textarea`、`select`、`option`、`optgroup`和`button`。图7-10展示了在Windows平台的Firefox上带有禁用元素的表单效果。

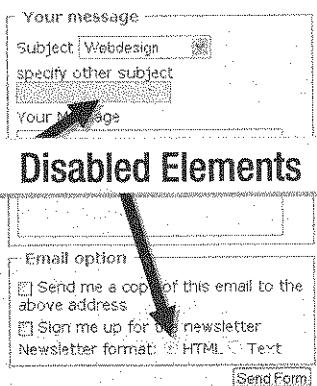


图7-10 禁用元素而不是隐藏它们

在这个脚本中主要的不同之处在于你需要分别定位想要单独禁用的`input`元素。在单选按钮的情况下，这意味着你必须遍历一下循环。脚本中的变化已经用粗体突出显示了而且都是显而易见的。

`dynamicFormDisable.js`

```
df = {
    hideClass : 'hide',
    letterOption : 'newsletter',
```

```

subjectOption : 'subject',
init : function() {
    if( !document.getElementById || !document.createTextNode ){
        return;
    }
    df.news = document.getElementById( df.letterOption );
    df.subject = document.getElementById( df.subjectOption );
    if(!df.subject || !df.news){ return; }
    df.newsOpt = DOMhelp.closestSibling( df.news.parentNode, 1 );
    df.newsOpt = df.newsOpt.getElementsByTagName( 'input' );
    for( var i = 0; i < df.newsOpt.length; i++ ){
        df.newsOpt[i].disabled = 1;
    }
    df.subjectOpt = DOMhelp.closestSibling( df.subject.parentNode, 1 );
    df.subjectOpt = df.subjectOpt.getElementsByTagName( 'input' )[0];
    df.subjectOpt.disabled = 1;
    DOMhelp.addEvent( df.news, 'click', df.letterChange, false );
    DOMhelp.addEvent( df.subject, 'change', df.subjectChange, false );
},
letterChange : function( e ){
    var i;
    var t = DOMhelp.getTarget( e );
    var disable = t.checked ? null: 1;
    for( i = 0; i < df.newsOpt.length; i++ ){
        df.newsOpt[i].disabled = disable;
    }
},
subjectChange : function( e ){
    var t = DOMhelp.getTarget( e );
    if( t.selectedIndex == 5 ){
        df.subjectOpt.disabled = null;
        df.subjectOpt.focus();
    } else {
        df.subjectOpt.disabled = 1;
    }
}
}
DOMhelp.addEvent( window, 'load', df.init, false );

```

使用disabled的实际结果是这些元素再也不能通过Tab键切换到了——而隐藏的元素仍然是可以的（除非你如前面站点导航部分那样通过把display为none来隐藏它们）。

7.2.4 定制表单元素

假设有足够的技巧和测试时间，你可以通过JavaScript使用自己的控件来扩展浏览器为用户提供的常用表单控件，甚至可以用键盘来访问它们。尤其是在Web应用程序开发中，这可能是必需的。

在这里我不想讨论如何开发你自己定制的元素，但在第11章还会再次提到这个主题。现在你

可以看看开发团队在雅虎 (<http://developer.yahoo.com/yui/slider/examples/slider.html>) 以及在Mozilla (<http://www.mozilla.org/access/dhtml/#examples>) 上提出和免费给出的内容。

7.2.5 表单与JavaScript小结

我希望本节能使你领悟到通过表单与JavaScript可以做什么。你已经学到了表单自己不同的属性和方法，以及其中的每一个元素可能包含它们自己的属性和方法。你还看到了如何详细地处理选择框，以及如何通过隐藏依赖于其他元素的元素，并通过只在其他元素被激活或者有一个对应的值时才显示它们，从而使表单更加动态。

表单与JavaScript记要

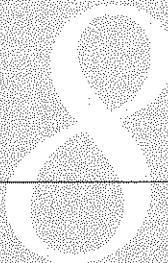
- 试着不要过分追求使用表单你可以做什么。看看使用键盘表单是否依然可用。特别地，填写比较长的表单时，通过Tab键在各个区域切换比单击不同的元素再编辑它们要方便一些。
- 不要通过事件处理程序来自动地提交表单——表单可以通过单击提交按钮或通过按回车键来提交。不要去掉用户的这些选择。
- 虽然旧的表单集合forms和elements不是最新的DOM编程技术（因为它们依赖于HTML，而其他的DOM方法也可以应用于XML字符串），但在普通的表单或你无法控制元素的ID或数目的生成的表单时候，它们可能是更容易的选择。循环遍历一个elements列表，比循环遍历表单所有的子元素并把它们和可能的元素名进行比较或者分别地循环遍历input、textarea和select元素集合，要容易许多。

7.3 小结

现在你应该已经可以处理JavaScript最常见的一些使用了。当你想复习一下如何处理图片、窗口、导航和表单的时候可以回到这一章和前一章再来看一下。

在下一章中，我们会离开浏览器和客户端脚本的领域，并且关注如何可以使JavaScript和后台以及服务器端脚本进行通话。这也会让你看到脚本块中的新事物——Ajax。

与Ajax后端交互



本章将讨论与JavaScript相关且迷人的新技术——Ajax。好消息是，你可以使用Ajax创建漂亮的平滑界面，而且可以把JavaScript的范围扩展到浏览器和当前显示的文档以外。

不太好的消息是Ajax依赖于XMLHttpRequest，简称为XHR对象（或它的微软等价对象），它的全部内容都是使用“HTTP”来写的。这意味着没有服务器你就不能使用任何的Ajax示例，而且你必须有一些服务器端编程的基础知识来使用Ajax（除非你使用一个现成的包——有关这个更多的内容会在8.4节提到）。

这也意味着使用Ajax会强行去掉JavaScript的一个强项：可以创建在计算机的文件系统甚至CD或记忆棒上脱机工作的界面。但是，Ajax所带来的好处补偿了这一点。

如果你不想在本地计算机和远程计算机之间来回传送文件以测试代码，最好的选择是安装一个本地服务器。这并没有表面上看去的那么难，因为现在存在许多可以利用的预先打好包的服务器软件（在20世纪90年代时这是令人非常痛苦的一件事情，尤其是在Windows上）。

我个人比较喜欢的是XAMPP，可以在网站<http://www.apachefriends.org/>下载它。你会得到一个安装程序，遵照说明运行几分钟就可以拥有自己的服务器了。

XAMPP会安装Apache 2、MySQL、PHP以及所有你会用到的插件，而且在许多平台上都是可用的。它还携带了一个FTP和邮件服务器、一个统计包，以及许多可选项，而且Apache Friends (<http://www.apachefriends.org>) 的维护人员会经常更新维护它。当然，它是免费的。

提示 再次说明，要很好地理解本章其余部分的内容，你应该自己执行一下本章中的许多代码示例以明白我在讲什么。与其他章节不同的地方是，本章中的代码示例不会在本地计算机的文件系统上运行；它们需要一台服务器，因为Ajax需要HTTP协议来工作。如果你不想安装一台服务器但是可以联网，那么可以连到本书的主页<http://www.beginning-javascript.com/>，在网站上可以看到所有的代码示例在运行。

安装XAMPP之后，就可以把本章的示例解压到一个目录里，例如jsbook。这个目录是在服务器安装的htdocs目录里，它可能是c:\xampp\htdocs\。要看到这些示例，可以打开一个浏览器，在地址栏里键入<http://localhost/jsbook/>。

提示 除了<http://www.apachefriends.org/en/faq-xampp.html>上的官方帮助FAQ以外，在<http://www.tamba2.org.uk/wordpress/xampp/>上还有一个精彩的手把手的教程教你如何在运行Windows XP的计算机上安装XAMPP（和WordPress）。

8.1 Ajax到底是什么

Ajax代表异步JavaScript和XML（Asynchronous JavaScript and XML），它是Jesse James Garrett于2005年2月在Adaptive Path上首次提出的一个术语（<http://www.adaptivepath.com/publications/essays/archives/000385.php>）。它描述了一种不同于传统Web应用程序的开发方式。

根据这篇文章，传统的Web应用程序和站点是同步工作的——每次打开链接或当提交表单时，浏览器会把数据发送到服务器，（理想情况下）服务器做出响应，然后整个网页被重新刷新。

注解 这并不是必需的，因为老的Web应用程序界面（如微软的Outlook）中使用的框架，它只重新加载整个界面的一小部分，但也不要太过挑剔；让框架成为过去Web连接速度低于56K标准调制解调器的时代产物吧！

Ajax应用程序是异步地（asynchronously）工作的，这意味着可以在用户代理和服务器之间传送数据而无需重新加载整个页面，只替换改变的那部分网页。也可以发出几次请求，然后在后台加载其他部分的时候继续滚动屏幕并使用网页。

一个非常好的比喻就是，Ajax相对于传统的网页就如即时消息相对于邮件：快速反馈而不用长时间等待，并且拥有更多通信选项。图8-1展示了Ajax应用程序流与传统的网站和Web应用程序的对比。

乍看，这好像是在整体上又另外增加了一层复杂度。可是，有关它真正比较酷的事情是，在Ajax引擎和浏览器之间的通信是通过JavaScript而不是通过页面重新加载产生的。

实际上，这意味着终端用户可以少花点时间来等待页面加载和呈现，而且页面更易交互，因为可以一边请求数据一边阅读文本或浏览网页上的其他内容。

这样可以使界面更平滑流畅，例如，不用改变整个站点而同时能够在服务器或数据库中检查输入是否正确，就可以在登录表单上给出反馈信息。

我们来看一个简单的示例。演示文件exampleXHR.

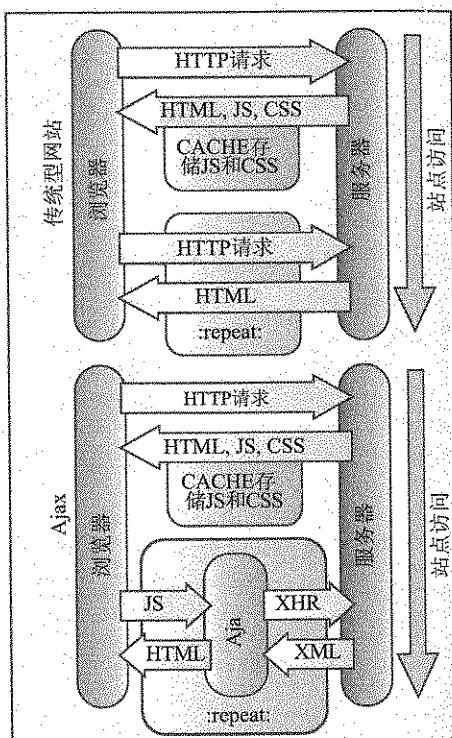


图8-1 Ajax与传统的请求

html在用户单击链接的时候使用Ajax（准确的说，没有X，因为没有涉及XML）来从服务器上加载和显示文件，如图8-2所示。

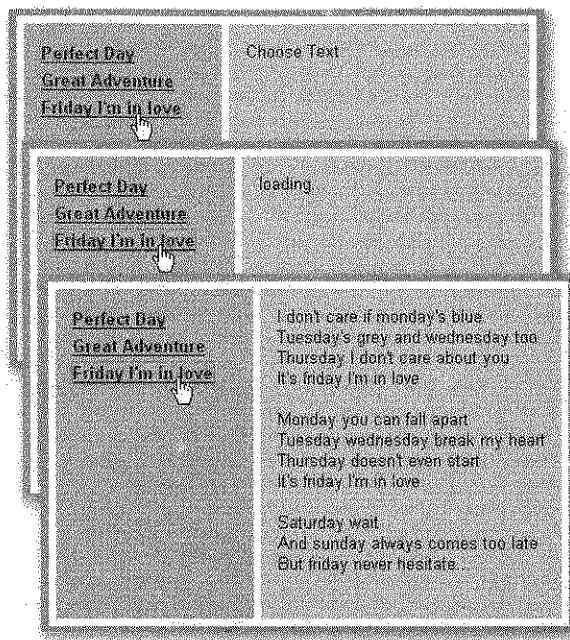


图8-2 通过Ajax加载外部文件

隐藏在这些所有的东西之后的魔法棒是一个叫做XMLHttpRequest的对象，或简称为XHR。它是一个非标准的对象因为它不是W3C站点官方标准的一部分（它现在是一个工作草案：<http://www.w3.org/TR/XMLHttpRequest/>），但是所有的现代浏览器（Safari、Mozilla/Firefox、Opera）都支持它。IE不支持XHR，但是它使用了一个ActiveX控件。幸好这个ActiveX控件的工作原理是一样的。

警告 这样的问题是，当用户在IE中启用了JavaScript但是禁用了ActiveX的时候，他不会体验到你的Ajax成果。在你创建Ajax解决方案和获取用户bug报告时要记住这一点。

我们来逐步分析一下这个例子，让你明白这些不同的部分是什么的。这个HTML文件包括几个指向文本文件的链接，它调用simplexhr.doxhr方法需要两个参数：一个是文本要发送到的HTML元素的ID，另一个是这个文本的URL：

exampleXHR.html（摘录）

```
<li>
    <a href="perfect_day.txt">
```

```

    onclick="simplexhr.doxhr( 'txtcontainer1', this.href );➥
    return false;">
    Perfect Day
  </a>
</li>
<li>
  <a href="great_adventure.txt"
    onclick="simplexhr.doxhr( 'txtcontainer1', this.href );➥
    return false;">
    Great Adventure
  </a>
</li>

```

注解 注意这些链接并没有完全分离和符合本书中其余代码例子的标准，但是至少没有JavaScript它们还可以工作——在脚本不可用的时候浏览器会简单地显示文本文件。这是非常吸引人的，尤其是在利用现成的Ajax库创建不依赖于脚本的链接的时候。不管这种技术有多酷，这样做都不是一个好的主意。

simpleXHR.js

```

simplexhr = {
  doxhr : function( container, url ) {
    if( !document.getElementById || !document.createTextNode ) {
      return;
    }
    simplexhr.outputContainer = document.getElementById( container );
    if( !simplexhr.outputContainer ){ return; }
  }
}

```

这个脚本开始先检查是否支持DOM以及你要写入内容的元素是否可用。如果它是，则把它存到一个叫outputContainer的属性里，让它在这个脚本的所有其他方法中可以使用。

simpleXHR.js (续)

```

var request;
try{
  request = new XMLHttpRequest();
} catch ( error ) {
  try {
    request = new ActiveXObject( "Microsoft.XMLHTTP" );
  } catch ( error ) {
    return true;
  }
}

```

定义一个叫request的变量，并使用try和catch结构来看支持哪个版本的XHR。对于Mozilla和Safari浏览器，试着赋予一个新的XMLHttpRequest()对象；如果那样做不支持的话，就会产生一个错误，它会触发catch语句（可以在本书的附录中学到更多有关try和catch()的内容）。catch语句

试着赋微软一个ActiveX控件对象。如果它还是不支持，方法会返回true，这意味着浏览器会只打开链接并在浏览器中显示文本。

如果这个赋值成功的话，你就拥有了一个可以任意支配的新的XMLHttpRequest对象。

注解 对于XMLHttpRequest对象的方法、处理程序和属性的完整列表，可以在XULPlanet (<http://www.xulplanet.com/references/objref/XMLHttpRequest.html>)或微软的 (<http://msdn.microsoft.com/library/en-us/xmlsdk/html/xmobjpmxmlhttprequest.asp>) 分别地参考相应的文档。

第一步是调用open()方法来建立和服务器之间的连接并接受或发送数据。open()方法有5个参数，其中3个是可选的：

```
request = open(requestMethod, url[, sync, [name, [password]]]);
```

- 参数requestMethod（其他的可选项已超出了本章的范围）可以是GET或POST，对应于一个Form元素的method属性。GET方法非常容易，但是也缺乏安全，因为它是由表单和服务器端数据处理使用的（通过GET发送数据在URL中是可见的，很容易被其他人操纵）。
- 参数url是文件在你的服务器上的地址。

注解 XMLHttpRequest不允许你从其他的服务器上加载内容，因为那会引发一个很大的安全问题。想象一下嵌在邮件或网站上的任意JavaScript都可以从你的计算机里发送数据或从服务器上接受更多代码是多么的恐怖。但是也有一种方式可以加载第三方内容，就是通过使用服务器上的代理脚本——在后面的XML示例中会有更多相关内容。

- 参数sync是个可选项，它是一个布尔值，定义了请求应该被异步地还是同步地发送。它默认为true，意味着请求会被异步地发送。同步的请求会把服务器锁上。
- 参数name和password都是可选的，它们只有在你要调用的文件需要用户认证的情况下才需要。

在这个例子中，你只能从服务器上获取文件，要完成这件事情，需要使用GET作为请求的方法，文件的地址作为url参数，忽略可选的几个参数。

simpleXHR.js (续)

```
request.open( 'get', url );
```

request对象的readyState属性包括一个数字，它描述了连接发生了什么。它在整个试图建立连接的过程中是不断增加的。

readyState几种不同的可能值和它们对应的请求状态如下所示。

- 0：不存在连接——它还没被初始化。
- 1：连接正在加载。

- 2: 数据已经加载完成。
- 3: 连接正在交互。
- 4: 连接已完成——数据已发送和接收。

每次状态改变，XHR都会触发一个readystatechange事件。可以使用相应的onreadystatechange事件处理程序来调用一个方法，在它里面可以再次检查readyState可能的值并采取适当的行为动作。

simpleXHR.js (续)

```
request.onreadystatechange = function() {
  if( request.readyState == 1 ) {
    simplexhr.outputContainer.innerHTML = 'loading...';
  }
}
```

一旦请求初始化了(readyState等于1)，最好给用户一些反馈信息告诉它们后台发生了什么。在这个例子中，脚本在HTML输出元素中显示一个“loading...”消息，如图8-3所示。

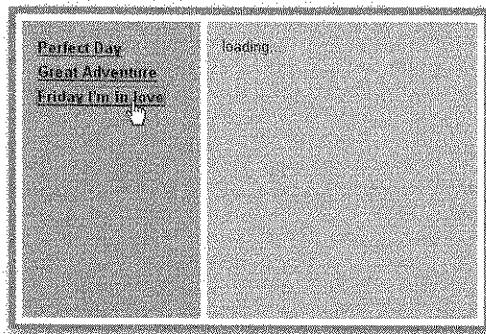


图8-3 通知用户请求已经发送且正在处理中

其他的几个状态不能跨平台浏览器安全读取，这也是我们跳过数字2和3并通过比较readyState是否等于4来检查请求是否已完成的原因。

simpleXHR.js (续)

```
if( request.readyState == 4 ) {
  if( /200|304/.test( request.status ) ) {
    simplexhr.retrieved(request);
  } else {
    simplexhr.failed(request);
  }
}
```

当请求完成的时候，检查另一个叫做status的属性，它保存了这次请求的状态。这个状态是请求的标准HTTP响应代码。它在连接不能建立的时候是0，在文件找不到的时候是404。

注解 对于标准HTTP响应代码的完整列表，可以参考<http://www.w3.org/Protocols/rfc2616/rfc2616-sect10.html>。

如果状态是200（所有的事情都正常）或304（没有更新信息），那么文件已经接收到了，你可以拿它做一些处理。在这个演示脚本的例子中，调用了retrieved()方法。如果状态是其他的任何值，那么调用failed()。

simpleXHR.js (续)

```

    }
    request.send( null );
    return false;
},

```

send()方法把你的请求发送到服务器，它可以使用请求参数来发送到被调用的服务器端脚本。如果你没有任何参数要发送，那么把它设置为null是最安全的。（IE接受不带任何参数的send()方法，但是这样做在Mozilla浏览器上会引起问题。）最后，设置方法的返回值为false以阻止链接被打开。

simpleXHR.js (续)

```

failed : function( requester ) {
    alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
    return true;
},

```

如果请求没有成功，failed()方法会显示一个alert()对话框，告诉用户发生的问题。（这样并不巧妙或完美，但是现在应该这样做。）在用户单击对话框的OK按钮以后返回true值会使链接被打开。可以通过在本地的浏览器（不用http://协议）中打开文件exampleXHR.html并单击链接来检查它是否那样。由于没有HTTP传送，任一请求都会失败返回代码0，如图8-4所示。

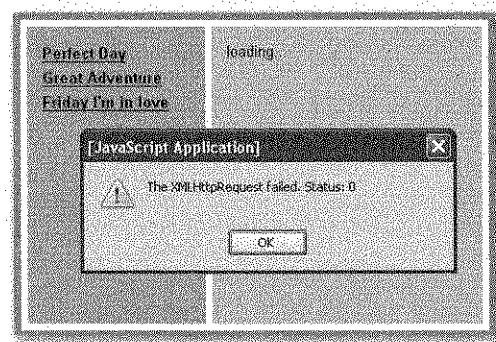


图8-4 通知用户XMLHttpRequest已失败

可是，如果这个请求所有的都正常，就会采用retrieved()方法：

simpleXHR.js (续)

```
retrieved : function( requester ) {
    var data = requester.responseText;
    data = data.replace( /\n/g, '<br />' );
    simplexhr.outputContainer.innerHTML = data;
    return false;
}
}
```

这个方法可以使你获取并使用从XMLHttpRequest发送回来的数据。这种数据可以以2种不同的格式读取出来：responseText和responseXML，它们的不同之处在于输出的类型——responseText返回一个字符串，而responseXML会返回一个XML对象。对responseText你可以使用所有常用的字符串属性和方法（length、indexOf()、replace()等），对responseXML则可以使用所有的DOM方法（getElementsByName()、getAttribute()等）。

在这个例子中，仅仅是获取文本并使用String.replace()方法把所有的换行符\n转换为BR元素。接下来可以把改变的字符串作为innerHTML输出到outputContainer中，并返回false来阻止正常的链接行为。

在大多数情况下，使用responseText并通过innerHTML写出数据已经足够了。对用户的浏览器和CPU来说，它也会快许多而且工作量也少，而使用XML和DOM需要把对象转换为HTML。

注解 Ajax的这个首字母缩写词在这些示例中还没有发挥作用，因为整个过程缺少XML组件。

为此，这种方法也叫AHAH，它被定义为一种微格式（microformat），代码示例在<http://microformats.org/wiki/rest/ahah>。

8.2 高速缓存竟带来了麻烦

通常的情况下浏览器的高速缓存是我们的助手。浏览器会在其中存储下载的文件，这意味着用户不必一遍又一遍地下载我们的脚本。然而，在Ajax的情况下，高速缓存会造成其他问题。

Safari是其中的罪魁祸首，因为它高速缓存了响应的status，不会再次触发修改了（记住状态返回的是HTTP代码200、304或者404）。可是，要避免高速缓存带来的问题却非常简单：在调用send()方法前，给请求添加另外一个头信息。这个头会告诉浏览器来检查数据从某个特定的日期开始是否已经修改了。你设置的那个日期并不重要，只要它是过去的日期就可以，例如，编写这些代码的时间：

```
request.setRequestHeader( 'If-Modified-Since', 'Thu, 06 Apr 2006
00:00:00 GMT' );
request.send( null );
```

8.3 把 X 放回到 Ajax 里面

如果使用responseXML，可以使用DOM方法把接收的XML转换为HTML。演示例子exampleXMLxhr.html就是这样做的。作为一个数据源，看一下最后一章中分页例子中的相册集合的XML格式：

albums.xml（摘录）

```
<?xml version="1.0" encoding="utf-8"?>
<albums>
  <album>
    <id>1</id>
    <artist>Depeche Mode</artist>
    <title>Playing the Angel</title>
    <comment>They are back and finally up to speed again</comment>
  </album>
  <album>
    <id>2</id>
    <artist>Monty Python</artist>
    <title>The final Rip-Off</title>
    <comment>Double CD with all the songs</comment>
  </album>
  [... more albums snipped ...]
</albums>
```

需要通过XHR获取这些数据，并在网页中把它显示为一个表。图8-5显示了请求的不同状态。

ID	Artist	Title	Comment
1	Depeche Mode	Playing the Angel	They are back and finally up to speed again
2	Monty Python	The final Rip-Off	Double CD with all the songs
3	Me Famen	I.com	Good electronica
4	Bad Religion	No control	My first concert ever

图8-5 获得并显示XML数据为一个表

脚本的主要部分不必修改：

simpleXMLxhr.js

```
simplexhr = {
  doxhr : function( container, url ) {
```

```

if( !document.getElementById || !document.createTextNode ) {
    return;
}
simplexhr.outputContainer = document.getElementById( container );
if( !simplexhr.outputContainer ) { return; }
var request;
try {
    request = new XMLHttpRequest();
} catch( error ) {
    try {
        request = new ActiveXObject( "Microsoft.XMLHTTP" );
    } catch ( error ) {
        return true;
    }
}
request.open( 'get', url,true );
request.onreadystatechange = function() {
    if(request.readyState == 1) {
        simplexhr.outputContainer.innerHTML = 'loading...';
    }
    if(request.readyState == 4) {
        if( request.status && /200|304/.test( request.status ) ) {
            simplexhr.retrieved( request );
        } else {
            simplexhr.failed( request );
        }
    }
}
request.setRequestHeader( 'If-Modified-Since', 'Wed, 05 Apr 2006 00:00:00 GMT' );
request.send( null );
return false;
},

```

不同的地方是在retrieved()方法中，它使用responseXML来读取数据并使用XML作为内容的数据源来输出一个数据表。去掉加载的消息并使用DOM的createElement()和createTextNode()方法来创建这个主要的表：

```

simpleXMLxhr.js (续)

retrieved : function( requester ) {
    var data = requester.responseXML;
    simplexhr.outputContainer.removeChild( simplexhr.outputContainer.firstChild );
    var i, albumId, artist, albumTitle, comment, td, tr, th;
    var table = document.createElement( 'table' );
    var tablehead = document.createElement( 'thead' );
    table.appendChild( tablehead );
    tr = document.createElement( 'tr' );

```

```

th = document.createElement( 'th' );
th.appendChild( document.createTextNode( 'ID' ) );
tr.appendChild( th );
th=document.createElement( 'th' );
th.appendChild( document.createTextNode( 'Artist' ) );
tr.appendChild( th );
th = document.createElement( 'th' );
th.appendChild( document.createTextNode( 'Title' ) );
tr.appendChild( th );
th=document.createElement( 'th' );
th.appendChild( document.createTextNode( 'Comment' ) );
tr.appendChild( th );
tablehead.appendChild( tr );
var tablebody = document.createElement( 'tbody' );
table.appendChild( tablebody );

```

注意当你创建表的时候，除非把行和单元格嵌套到一个TBODY元素中，否则IE不会显示它。Firefox是不介意这些的。

下一步，循环获取数据的所有album元素。

simpleXMLxhr.js (续)

```

var albums = data.getElementsByTagName( 'album' );

for( i = 0 ; i < albums.length; i++ ){

```

对于每一个相册，通过它们的标签名读取XML节点的内容并通过firstChild.nodeValue获取它们的文本内容。

simpleXMLxhr.js (续)

```

tr = document.createElement( 'tr' );
albumId = data.getElementsByTagName( 'id' )[i].➥
firstChild.nodeValue;
artist = data.getElementsByTagName('artist')[i].➥
firstChild.nodeValue;
albumTitle = data.getElementsByTagName('title')[i].➥
firstChild.nodeValue;
comment = data.getElementsByTagName('comment')[i].➥
firstChild.nodeValue;

```

通过createElement()、createTextNode()和appendChild()你可以使用这些信息来添加数据单元格到表中。

simpleXMLxhr.js (续)

```

td = document.createElement( 'th' );
td.appendChild( document.createTextNode( albumId ) );
tr.appendChild( td );
td = document.createElement( 'td' );

```

```

        td.appendChild( document.createTextNode( artist ) );
        tr.appendChild( td );
        td = document.createElement( 'td' );
        td.appendChild( document.createTextNode( albumTitle ) );
        tr.appendChild( td );
        td = document.createElement( 'td' );
        td.appendChild( document.createTextNode( comment ) );
        tr.appendChild( td );
        tablebody.appendChild( tr );
    }
}

```

把这个结果表作为一个新的子元素添加到输出容器中，并返回false以阻止链接把XML作为一个新的文档加载。failed()方法仍然是一样的。

simpleXMLxhr.js (续)

```

simplexhr.outputContainer.appendChild( table );
return false;
},
failed : function( requester ) {
    alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
    return true;
}
}
}

```

可以看到依据DOM脚本来做“理所当然的事情”，脚本会变得非常费解。可以通过使用工具方法创建表的行来消减代码的数量，但是那意味着更多的处理，因为这些方法必须在一个循环中调用。

如果知道XML的结构，如这个例子中所做的那样，那么使用innerHTML和字符串的方法来转换这些数据可能会更快更容易些。演示例子exampleXHRxmlCheat.html就是那样做的。大部分脚本仍然是一样的，但是retrieved()方法简短了不少：

simpleXMLxhrCheat.js (摘录)

```

retrieved : function( requester ){
    var data = requester.responseText;
    simplexhr.outputContainer.removeChild( simplexhr.outputContainer.firstChild );
    var headrow = '<tr><th>ID</th><th>Artist</th><th>Title</th><th>Comment</th></tr>';
    data = data.replace( /<\?.*\?\>/g, '' );
    data = data.replace( /<(\/*)id>/g, '<$1th>' );
    data = data.replace( /<(\/*)(artist|title|comment)>/g, '<$1td>' );
    data = data.replace( /<(\/*)albums>/g, '<$1table>' );
    data = data.replace( /<(\/*)album>/g, '<$1tr>' );
    data = data.replace( /<table>/g, '<table>' + headrow );
    simplexhr.outputContainer.innerHTML = data;
    return false;
}
}

```

通过`responseText`获取数据，去掉“loading...”消息，然后创建一个字符串的表标题行并把它存放到变量`headrow`中。因为`responseText`是一个字符串，所以可以使用`String.replace()`方法来修改XML元素。

开始先通过删除所有以`<`开头并以`>`结尾的元素来去掉XML序言（prologue）。

注解 这个示例使用了正则表达式，你可能对它还不太了解，但是我们会在第9章对它作详细的讨论。知道正则表达式是以斜杠分割的而且匹配文本的一种特定模式。如果斜杠里面存在括号，这些字符串会保存在以\$打头的变量中；它们可以被用在替换字符串中来代表匹配模型的子字符串。例如，正则表达式模型`<(\/*)id>/g`匹配一个以`<`打头，紧跟一个可选的`/`（如果它被找到的话是以`$1`保存的），后面跟一个字符串`id`并以`>`字符结束的所有字符串。第二个参数`<$1th>`，输出`<th>`或`</th>`，取决于原始的`id`标签是开始标签还是结束标签。与其使用正则表达式，你不如使用简单的字符串替换：

```
data = data.replace('<id>', '<th>');
data = data.replace('</id>', '</th>');
```

按照这种方案来替换其他的元素：每个`albums`元素变成了一个`table`，每个`album`变为一个`tr`，每个`id`变为一个`th`，`artist`、`title`和`comment`也都变为一个`td`。把`headrow`字符串追加到`<table>`上并使用`innerHTML`把最终的结果保存在`outputContainer`元素中。

8.3.1 使用 JSON 代替 XML

虽然XML是数据传送格式毋庸置疑的首选——它是基于文本的，而且你可以通过DTD、XML Schemata或RELAXNG来保证其相互通信的有效性和规律性，但Ajax开发人员越来越意识到把XML转换为JavaScript对象非常费力。

用XML来读取XML文件并通过DOM来解析它，或者以文本的方式读取它并使用正则表达式，上述两种方式都不如使用一种JavaScript可以直接使用的格式更轻松、更省事。这种格式叫做JSON (<http://json.org/>)，它主要是以对象文字符号表示的数据集。演示例子exampleJSONxhr.html把前面例子中的XML变为JSON：

```
<albums>
  <album>
    <id>1</id>
    <artist>Depeche Mode</artist>
    <title>Playing the Angel</title>
    <comment>They are back and finally up to speed again</comment>
  </album>
  <album>
    <id>2</id>
    <artist>Monty Python</artist>
    <title>The final Rip-Off</title>
    <comment>Double CD with all the songs</comment>
```

```
</album>
<album>
<id>3</id>
<artist>Ms Kittin</artist>
<title>I.com</title>
<comment>Good electronica</comment>
</album>
</albums>
```

转换为JSON后，上面的代码将如下所示：

```
albums.json
{
  'album': [
    [
      {
        'id' : '1',
        'artist' : 'Depeche Mode',
        'title' : 'Playing the Angel',
        'comment' : 'They are back and finally up to speed again'
      },
      {
        'id' : '2',
        'artist' : 'Monty Python',
        'title' : 'The final Rip-Off',
        'comment' : 'Double CD wiid all the songs'
      },
      {
        'id' : '3',
        'artist' : 'Ms Kittin',
        'title' : 'I.com',
        'comment' : 'Good electronica'
      }
    ]
  ]
}
```

好处是这些数据已经变成JavaScript可以理解的一种格式，你所要做的就是对这个字符串使用eval方法并把它转换为对象来显示：

exampleJSONxhr.js（摘录）

```
retrieved : function( requester ) {
  simplexhr.outputContainer.removeChild(⇒
  simplexhr.outputContainer.firstChild);
  var content = '<table><thead>';
  content += '<tr><th>ID</th><th>Artist</th>';
  content += '<th>Title</th><th>Comment</th>';
  content += '</tr></thead><tbody>';
  var data = eval( '(' + requester.responseText + ')' );
```

这样会以对象的方式给出你所有的内容，可以通过属性符号或关联数组符号来访问它（后者在id的例子中展示了，前者在其他的例子中用到）：

exampleJSONxhr.js (摘录)

```
var albums = data.album;
for( var i = 0; i < albums.length; i++ ) {
    content += '<tr><td>' + albums[i]['id'] + '</td>';
    content += '<td>' + albums[i].artist + '</td>';
    content += '<td>' + albums[i].title + '</td>';
    content += '<td>' + albums[i].comment + '</td></tr>';
}
Content += '</tbody></table>';
simplexhr.outputContainer.innerHTML = content;
return false;
},
```

对于你自己服务器上的文件，使用JSON来代替XML会快许多（经过测试证明至少快10倍）；可是，如果你是在第三方服务器上使用JSON，那么使用eval()可能会有危险，因为它可以执行任意的JavaScript代码而不只是JSON数据。

可以使用一个解析器来避免这个危险，确保只有数据才会转换成对象，恶意的代码不会被执行。在网站<http://www.json.org/js.html>上有一个开源版本，第11章会再回到JSON这个主题上。

8.3.2 使用服务器端脚本来访问第三方内容

如前所述，由于安全原因使用XHR从其他服务器上加载内容是不可能的。例如，如果你想从其他服务器获取RSS提要，需要使用一种服务器端脚本来加载它们。

注解 下面是关于Ajax的一种错误观念：它没有替换服务器端代码，但它是以服务器端代码为支持并为它提供了一种平滑流畅的界面。XHR本来只能从同样的服务器获取数据或把信息发送到服务器端脚本。举例来说，你不能使用JavaScript访问一个数据库——除非该数据库供应商提供了一种JavaScript输出，并且你把它包含在它自己的脚本标签中。

服务器端的组件是一个传递或代理脚本，它使用一个URL加载文档的内容，并把结果返回给XHR。这个脚本需要设置正确的头信息来告诉XHR它返回的数据是XML。如果文件没有找到，这个脚本会返回一个XML错误字符串。下面的例子使用的是PHP，但是任何服务器端语言都可以完成同样的任务：

loadrss.php

```
<?php
// Set the XML header
header('Content-type: text/xml');
// Define an error message in case the feed cannot be found
```

```

$error='<?xml version="1.0"?><error>Cannot find feed</error>';
// Clear the contents
$contents = '';
// Read the url variable from the GET request
$rssurl = $_GET['url'];
// Test if the url starts with http to prevent surfers
// from calling and displaying local files
if( preg_match( '/^http:/', $rssurl ) ){
    // Open the remove file, and store it contents
    $handle = @fopen( $rssurl, "rb" );
    if( $handle == true ){
        while ( !feof($handle) ) {
            $contents .= fread( $handle, 8192 );
        }
        fclose( $handle );
    }
}
// If the file has no channel element, delete contents
if( !preg_match( '/<channel/>', $contents ) ){ $contents = '';}
// Return either the contents or the error
echo $contents == '' ? $error : $contents;
?>

```

演示例子exampleExternalRSS.html使用这个脚本以RSS格式从Yahoo网站获取最新的头条新闻。

HTML中的这个相关部分是一个调用了doxhr()方法的链接，该方法把要在其中输出该新闻的元素和RSS URI作为参数：

exampleExternalRSS.html（摘录）

```

<p>
    <a href="http://rss.news.yahoo.com/rss/topstories"
        onclick="return readrss.dohxr('newsContainer',this.href)">
        Get Yahoo news
    </a>
</p>
<div id="newsContainer"></div>

```

注解 RSS是Really Simple Syndication（真正简单聚合）的缩写，它本质上是一种XML，包含了你想与大家分享的内容——典型的就是新闻标题。RSS规范可以在网站http://blogs.law.harvard.edu/tech/rss上获取，还可以在Wikipedia网站中这一网页http://en.wikipedia.org/wiki/RSS_%28file_format%29上看到更多它的好处。

在这个例子中重要的细节是，RSS是一种标准的格式，因此你一定知道XML结构——即使你是从一个第三方网站获得它的。每个有效的RSS文档包含——在很多其他的东西之间——一个items元素嵌套多个item元素。其中每一个item元素都包含至少一个title描述和一个指向完整信

息的link。使用这些可以显示一个可单击新闻标题的列表，单击这些新闻可以把用户带到Yahoo的网站上阅读完整的新闻文章，如图8-6所示。

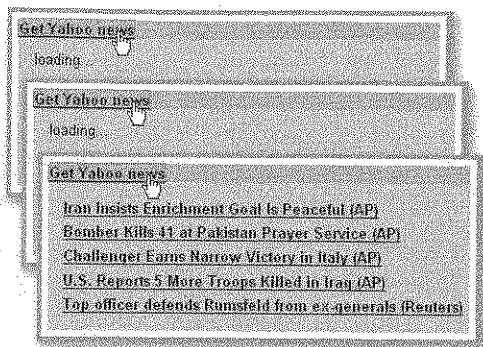


图8-6 获取和显示RSS的数据

这个脚本又一次用到了一个简单的XHR。不同的是它不直接地链接到URL，而是把它作为一个GET参数传给PHP脚本：

```
externalRSS.js
readrss = {
  doxhr:function( container, url ) {
    [... code snipped as it is the same as in the last example ...]
    request.open('get', 'loadrss.php?url=' + encodeURI( url ) );
    request.setRequestHeader( 'If-Modified-Since',➥
      'Wed, 05 Apr 2006 00:00:00 GMT' );
    request.send( null );
    return false;
  },
  retrieved() {
```

retrieved()方法需要修改一下。首先，它从输出容器中删除“loading...”消息并使用responseXML获取XML格式的数据。因为PHP脚本会返回一个XML格式的错误信息，所以你需要检查返回的XML是否包含一个error元素。如果的确是这种情况，那么读取第一个error元素的第一个子节点的值并通过一个段落标签把它写到outputContainer上。

```
externalRSS.js (续)
retrieved : function( requester ) {
  readrss.outputContainer.innerHTML = '';
  var data = requester.responseXML;
  if( data.getElementsByName( 'error' ).length > 0 ) {
    var error = data.getElementsByName('error')[0].➥
      firstChild.nodeValue;
    readrss.outputContainer.innerHTML = '<p>' + error + '</p>';
```

如果没有error元素，那么获取包含在返回的XML中的所有的item元素并检查结果列表的长度。如果少于一个item，那么返回方法，并允许链接在浏览器中加载XML文档。这是确保返回的

RSS是合法的必需一步——因为你没有在服务器端脚本中对这个进行检查。

externalRSS.js (续)

```

} else {
var items = data.getElementsByTagName('item');
var end = items.length;
if( end < 1 ){ return; }

```

如果有多个项要显示，你需要定义几个必要的变量并对它们进行循环遍历。由于一些RSS提要有许多条目，因此需要限制你要显示的数量；在这个例子中，我们选择5。读取link和每个item的title并使用该信息把添加的嵌入链接的新列表项相应地作为它的href属性和文本内容。注意这个例子简单地连接生成了一个HTML字符串；当然，可以使用更巧妙的方式来创建元素以及应用文本节点。

externalRSS.js (续)

```

var item, feedlink, name, description, content = '';
for( var i = 0; i < 5; i++ ) {
    feedlink = items[i].getElementsByTagName('link').item(0).➥
        firstChild.nodeValue;
    name = items[i].getElementsByTagName('title').item(0).➥
        firstChild.nodeValue;
    item = '<li><a href="' + feedlink + '">' + name + '</a></li>';
    content += item;
}

```

把最终的内容字符串插入到一个UL标签中，一起放到outputContainer中，这样你就拥有了最新的Yahoo新闻的可单击新闻标题了。

externalRSS.js (续)

```

readrss.outputContainer.innerHTML = '<ul>' + content + '</ul>';
return false;
}

```

其余的代码保留不要修改；当XHR没有成功的时候，failed()方法会只显示警告信息。

externalRSS.js (续)

```

},
failed : function( requester ) {
    alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
    return true;
}
}

```

8.3.3 关于缓慢链接的XHR问题

可能会发生的一个问题是XHR的连接可能会用很长一段时间，用户看到正在加载的信息而却

根本看不到什么结果。可以通过使用window.setTimeout()在一段时间后停止它的执行来避免这个问题。演示例子exampleXHRtimeout.html就展示了这种技术的使用。

请求的默认设置是10毫秒，这样会造成图8-7所示的超时。可以使用这个例子中的第二个链接来设置超时为10秒钟，然后再试一下，只要连接过程不是缓慢得要死或Yahoo方面的问题，你就会获得这个头条新闻。

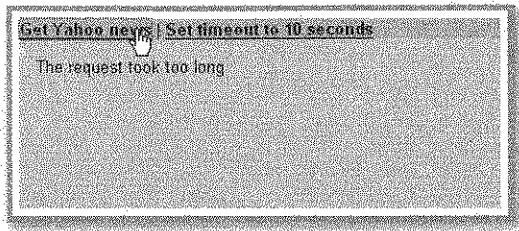


图8-7 让XHR连接超时

这个脚本中的不同之处在于，你需要一个属性用来定义在超时被触发前需要等待多长时间、一个用来存储window.setTimeout以及一个布尔值用于定义是否存在一个超时。后者必须是在doxhr()方法内，因为它每次在调用doxhr()方法的时候都需要初始化。

XHRtimeout.js

```
readrss = {
    timeOutDuration : 10,
    toolong : false,
    doxhr : function( container, url ) {
        readrss.timedout = false;
        if( !document.getElementById || !document.createTextNode ){
            return;
        }
        readrss.outputContainer = document.getElementById( container );
        if( !readrss.outputContainer ){ return; }
        var request;
        try {
            request = new XMLHttpRequest();
        } catch( error ) {
            try {
                request = new ActiveXObject( "Microsoft.XMLHTTP" );
            } catch( error ) {
                return true;
            }
        }
    }
}
```

在onreadystatechange事件监听器里面，添加超时设定，并把它赋给主要对象的toolong属性。在超时设定内面，定义一个匿名函数来检查readyState，并把它与1相比较。如果所规定的时间已过去，请求仍然在第一个阶段而不是第四个和最后一个，那么就调用该请求的abort()方法，

把timedout属性设置为true，并输出消息来显示请求花费了太长时间的元素。

XHRtimeout.js (续)

```
request.onreadystatechange = function() {
    if( request.readyState == 1 ) {
        readrss.toolong = window.setTimeout( function(){
            if( request.readyState == 1 ) {
                readrss.timedout = true;
                request.abort(); // Stop, Hammer Time!
                readrss.outputContainer.innerHTML = 'The request took too long';
            }
        },
        readrss.timeOutDuration
    );
    readrss.outputContainer.innerHTML = 'loading...';
}
}
```

当请求成功结束并且没有任何超时的时候（它存储在timedout属性中），清除timeout。

XHRtimeout.js (续)

```
if( request.readyState == 4 && !readrss.timedout ) {
    window.clearTimeout( readrss.toolong );
    if( /200|304/.test( request.status ) ) {
        readrss.retrieved( request );
    } else {
        readrss.failed( request );
    }
}
}
request.open( 'get', 'loadrss.php?url='+encodeURI( url ) );
request.setRequestHeader( 'If-Modified-Since', 'Wed, 05 Apr 2006 00:00:00 GMT' );
request.send( null );
return false;
},
```

脚本的其余部分保留不变。

8.3.4 一个更大的 Ajax 示例：关联选择框

我们来看一个更大的Ajax示例——虽然我这样说，但这里不会用到XML。关联选择框是一个典型的JavaScript功能示例，它可以使你的界面更快捷。它们的一个常见应用就是航空售票的网站，你可以在选择框中选择一个机场，页面会立即会在第二个选择框中给你显示从这个机场可以到达的所有目的机场。通常，这是通过把所有的航空路线数据保存在JavaScript数组并对select元素的options数组进行操作来实现的。改变第一个机场选择框会自动地把第二个改变为可用的目的地。

当你有鼠标并且JavaScript可用的时候，这是非常好的；可是，如果这两个条件有一个不满足，

就会非常令人失望，甚至二者都不可用你会觉得难以接受。这个例子会为你展示如何创建相互依赖的选择框，它们没有鼠标和JavaScript也可以工作，而且当JavaScript可用的时候也不会重新加载整个页面。

这个技巧是让主要功能运行在服务器端，并添加JavaScript和XHR技巧来阻止整个页面的重新加载。由于你不知道用户是否能够处理这些，因此你可以让它成为可选的（而不是给定的）就可以了。

注解 Ajax的这种处理方法比原始的方法更具可访问性。原因是你不想要再犯DHTML最大的错误——使用一种技术而不考虑不能处理它的用户。Jeremy Keith 在他的*DOM Scripting*^①一书中把这种方法称作HIJAX，但是公众对它的关注没有像对术语Ajax那样多。Ajax是很热的，即使通过指出不能依赖于它对所有的用户都可用，也很难削弱人们对它的期望。

第一步就是创建一个服务器端脚本来完成所有的功能。由于本书不是关于PHP方面的，因此我们这里不会涉及太多有关它的细节。知道主要文档exampleSelectBoxes.php包含一个小一些的PHP脚本selectBoxes.php就足够了。后者包括数组形式的所有机场数据（但是可以很轻松地从数据库中去获取它们）并且根据用户做出的选择和发送的表单输出不同的界面状态，如图8-8所示。

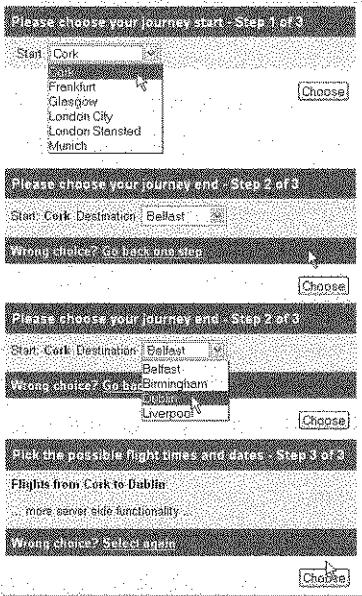


图8-8 关联选择框

^① 中文版《JavaScript DOM编程艺术》。另外同一作者的新作《Bulletproof Ajax中文版》更深入地阐述了HIJAX的理念。——编者注

主要页面的内容是一个DIV元素，可以用其id来进行XHR输出：

exampleSelectBoxes.php（摘录）

```
<form action="exampleSelectBoxes.php" method="post">
<div id="formOutput">
<?php include('selectBoxes.php');?>
</div>
<p class="submit"><input type="submit" name="select"
id="select" value="Choose" /></p>
</form>
```

注解 注意这个示例使用了POST作为发送数据的方法。这样会使你的XHR代码更难一些，但是试着用它是非常好的，因为大多数Web应用程序都会使用POST而不是GET。

这个PHP脚本会返回一个HTML界面，可以进一步了解到这个处理过程的每个阶段：

- 如果还没有任何的表单数据被发送，它会显示一个ID为airport的选择框，其中列出数据集合中的所有机场。
- 如果有一个机场被选择了并且发送到了服务器上，脚本会把强元素内所选择的机场并作为一个隐藏的表单域显示。它还会把对应这个选择的所有目的机场显示为一个ID为destination的选择框中。此外，它还创建了一个指回到主文档的链接来开始一个新的ID为back的选择。
- 如果用户选择了一个机场和一个目的机场并把它们发送到服务器上，那么脚本会提示可以有更多的其他功能，因为这个例子中不需要更多的功能了。可是，它会提供一个链接以返回到初始页面。

如果JavaScript可用，脚本应该完成下列功能：

- 在表单中创建一个新的复选框，允许用户打开Ajax功能——在这里只需要重新加载表单中由selectBoxes.php创建的部分。
- 如果复选框已被选中了，脚本会用一个由事件处理器调用的函数来忽略表单的标准提交过程。作为加载指示器，它应该改变提交按钮的文本为“loading”。
- 它还应该为返回第一阶段的链接添加一个搜索参数以确保用户当单击链接时，他不用再次选择复选框。

先看一下脚本的大纲结构。你需要一个标签用于复选框，一个类用于包含它的段落（并不是必需的，但是它考虑了样式布局），以及表单元素容器和返回到流程初始页面链接的几个ID。

关于方法，你需要一个init()方法、带有获取和失败处理程序的主XHR方法，以及事件处理的cancelClick()和addEvent()方法。

selectBoxes.js（大纲）

```
dynSelect = {
  // ...
  AJAXlabel : 'Reload only the results, not the whole page',
  // ...
}
```

```

AJAXofferClass : 'ajax',
containerID : 'formOutput',
backlinkID : 'back',
init : function(){},
doxhr : function( e ){},
retrieved : function( requester, e ){},
failed : function( requester ){},
cancelClick : function( e ){},
addEvent : function(elm, evType, fn, useCapture){}
}
dynSelect.addEvent( window, 'load', dynSelect.init, false );

```

现在开始填充一下这个大纲：

selectBoxes.js

```

dynSelect = {
  AJAXlabel : 'Only reload the results, not the whole page',
  AJAXofferClass : 'ajax',
  containerID : 'formOutput',
  backlinkID : 'back',
}

```

`init()`方法会检查W3C的DOM是否被支持，获取第一个表单，并把ID为`select`的提交按钮存储到一个属性中——这是在最后一步移除按钮所必需的。接着它创建一个新的段落，并为前面定义的Ajax触发器应用样式类。

selectBoxes.js (续)

```

init : function(){
  if( !document.getElementById || !document.createTextNode ){
    return;
  }
  var f = document.getElementsByTagName( 'form' )[0];
  dynSelect.selectButton = document.getElementById( 'select' );
  var p = document.createElement( 'p' );
  p.className = dynSelect.AJAXofferClass;
}

```

接下来该考虑的是提供了用于打开Ajax的选项的复选框。设置复选框的名称和ID为`xhr`，并判断当前的URI是否已包含?ajax搜索字符串。如果包含，则提前设置复选框为已选择状态（这是确保返回到第一步的链接不阻止Ajax增强效果工作所必需的）。

selectBoxes.js (续)

```

dynSelect.cb = document.createElement( 'input' );
dynSelect.cb.setAttribute( 'type', 'checkbox' );
dynSelect.cb.setAttribute( 'name', 'xhr' );
dynSelect.cb.setAttribute( 'id', 'xhr' );
if( window.location.search != '' ){
  dynSelect.cb.setAttribute( 'defaultChecked', 'checked' );
  dynSelect.cb.setAttribute( 'checked', 'checked' );
}

```

把复选框添加到新的段落上，并在它后面追加一个带有适当文本的标签。新的段落变成了表单的第一个子节点，并且应用一个事件处理程序用来在表单被提交的时候触发dohxhr()方法。

selectBoxes.js (续)

```
p.appendChild( dynSelect.cb );
var lbl = document.createElement( 'label' );
lbl.htmlFor = 'xhr';
lbl.appendChild( document.createTextNode( dynSelect.AJAXlabel ) );
p.appendChild( lbl );
f.insertBefore( p, f.firstChild );
dynSelect.addEvent(f, 'submit', dynSelect.dohxhr, false );
},
```

dohxhr()方法检查复选框是否已被勾选，并在它没有被选择的时候返回。如果它被选择了，则为当前的机场和当前的目的地定义2个变量，并且把输出元素存储在一个属性中。然后测试输出容器是否存在，如果不存在则返回。

selectBoxes.js (续)

```
dohxhr : function( e ) {
if( !dynSelect.cb.checked ){ return; }
var airportValue, destinationValue;
dynSelect.outputContainer = document.getElementById(➥
dynSelect.containerID );
if( !dynSelect.outputContainer ){ return; }
```

下面是XHR代码，定义了正确的对象并设置onreadystatechange事件监听器。

selectBoxes.js (续)

```
var request;
try {
request = new XMLHttpRequest();
} catch( error ) {
Try {
request = new ActiveXObject( "Microsoft.XMLHTTP" );
} catch( error ) {
return true;
}
}
request.onreadystatechange = function() {
if( request.readyState == 1 ) {
dynSelect.selectButton.value = 'loading...';
}
if( request.readyState == 4 ) {
if( request.status && /200|304/.test( request.status ) ) {
dynSelect.retrieved( request );
} else{
dynSelect.failed( request );
}
}
```

```

    }
}

```

判断文档是否包括机场和目的选择框；如果是，则把当前的状态存储到变量`airportValue`和`destinationValue`中。注意你需要在航程选择的第二个阶段检查机场域的类型，因为它是一个隐藏的域。

selectBoxes.js (续)

```

var airport = document.getElementById( 'airport' );
if( airport != undefined ) {
    if( airport.nodeName.toLowerCase() == 'select' ) {
        airportValue = airport.options[airport.selectedIndex].value;
    } else {
        airportValue = airport.value;
    }
}
var destination = document.getElementById( 'destination' );
if( destination ) {
    destinationValue = destination.options[destination.selectedIndex].value;
}

```

因为表单是使用POST，而不是GET发送的，所以你需要定义的请求应该有所不同。首先，需要把请求参数连接成一个字符串（这是发送方法为GET的时候URI上变量的尾部，例如：`http://www.example.com/index.php?search=DOM&values=20&start=10`）。

selectBoxes.js (续)

```

var parameters = 'airport=' + airportValue;
if( destinationValue != undefined ) {
    parameters += '&destination=' + destinationValue;
}

```

接下来，打开该请求。除了修改请求的头部信息以阻止高速缓存，你还需要告诉服务器内容的类型是`application/x-www-form-urlencoded`；然后把所有请求参数的长度作为`Content-length`的值和其他内容一起传送。你还需要告诉服务器在完成获取所有数据后关闭连接。和GET请求不同，在你使用POST的时候`send()`需要一个参数，它是URI编码的参数。

selectBoxes.js (续)

```

request.open( 'POST', 'selectBoxes.php' );
request.setRequestHeader( 'If-Modified-Since', 'Wed, 05 Apr 2006 00:00:00 GMT' );
request.setRequestHeader( 'Content-type', 'application/x-www-form-urlencoded' );
request.setRequestHeader( 'Content-length', parameters.length );
request.setRequestHeader( 'Connection', 'close' );
request.send( encodeURI( parameters ) );

```

注解 如果对这里所有的东西不太了解，不要痛责自己；毕竟它是关于服务器和HTTP编码的，而你只是刚开始学习JavaScript。不需要明白它所有的意思，只要按照这种方式使用它就可以了。

如果你在倒数第2个页面上，而且起始机场和目的机场都可用，那么要移除提交按钮以防止发生错误。

注解 对这个例子来说，这一步是不太实用的。一个真正的应用程序还应该做完下面的几步，但现在没有必要做那么多。

最后，调用cancelClick()阻止通常的表单提交。

selectBoxes.js (续)

```
if( airport && destination ) {
    var sendButton = document.getElementById( 'select' );
    sendButton.parentNode.removeChild( sendButton );
}
dynSelect.cancelClick( e );
},
```

retrieved()方法和其他的例子没有多少差别。在获取请求的responseText和使用新的内容替换以前的表单元素前，通过把提交按钮的值修改为Select以撤销上一步的操作。把?ajax添加到指向第一步的链接的href上，确保激活这个链接不会关闭前面所选择的功能（到现在你应该知道用户需要Ajax界面）。

selectBoxes.js (续)

```
retrieved : function( requester, e ) {
    dynSelect.selectButton.value = 'Select';
    var content = requester.responseText;
    dynSelect.outputContainer.innerHTML = content;
    var backlink = document.getElementById( dynSelect.backlinkID );
    if( backlink ){
        var url = backlink.getAttribute( 'href' );
        backlink.setAttribute( 'href', url+'?ajax' );
    }
    dynSelect.cancelClick( e );
},
```

脚本的其余部分包括我们熟悉的failed()、cancelClick()和addEvent()实用方法。

selectBoxes.js (续)

```
failed : function( requester ){
    alert('The XMLHttpRequest failed. Status: ' + requester.status);
    return true;
```

```

},
cancelClick : function( e ){
    [... code snipped ...]
},
addEvent: function( elm, evType, fn, useCapture ){
    [... code snipped ...]
}
}
dynSelect.addEvent( window, 'load', dynSelect.init, false );

```

这个例子展示了Ajax是非常依赖于服务器端代码的。如果你知道会返回的内容，那么就很容易创建一个既实用也吸引人的界面。

还可以以另一种分离方式使用Ajax方法，优化界面的风格使以前的特效更引人注目，并且更针对于需要它们的用户群。在上一章中，我们开发了一个动态的导航，它可以折叠和展开嵌套的链接，当时我说过我们还会回到那个例子中，下一节就会再来看一下它。

8.3.5 可选的动态 Ajax 菜单

动态导航中的一个问题是，你可能会给用户提供太多的选择。当你看到菜单折叠的时候，都会认为它很容易使用，你可以折叠或展开菜单项来导航，但是当你关闭JavaScript和CSS的时候，就会得到不同的结果，你会一下子看到所有的链接。现在也应该考虑一下那些聆听你网站的盲人访问者，或者需要放大屏幕的某一部分进行页面导航的人。

给访问者选择权去启用增强的动态菜单，并给那些选择其他的用户只提供必需的最小菜单结构，难道不是更有用吗？

回忆一下在前一章中的菜单结构：

```

navigation.php
<ul id="nav">
    <li><a href="index.php">Home</a></li>
    <li><a href="products.php">Products</a>
        <ul>
            <li><a href="cms.php">CMS solutions</a>
                <ul>
                    <li><a href="minicms.php">Mini CMS</a></li>
                    <li><a href="ncc1701d.php">Enterprise CMS</a></li>
                </ul>
            </li>
            <li><a href="portal.php">Company Portal</a></li>
            <li><a href="mailserver.php">eMail Solutions</a>
                <ul>
                    <li><a href="privatemail.php">Private POP3/SMTP</a></li>
                    <li><a href="lists.php">Listservers</a></li>
                </ul>
            </li>
        </ul>
    </li>
</ul>

```

```

</li>
<li><a href="services.php">Services</a>
  <ul>
    <li><a href="training.php">Employee Training</a></li>
    <li><a href="audits.php">Auditing</a></li>
    <li><a href="bulkmail.php">Bulk sending/email campaigns</a></li>
  </ul>
</li>
<li><a href="pricing.php">Pricing</a></li>
<li><a href="about_us.php">About Us</a>
  <ul>
    <li><a href="our_offices.php">Our offices</a></li>
    <li><a href="our_people.php">Our people</a></li>
    <li><a href="vacancies.php">Jobs</a></li>
    <li><a href="partners.php">Industry Partners</a></li>
  </ul>
</li>
<li><a href="contact.php">Contact Us</a>
  <ul>
    <li><a href="snail.php">Postal Addresses</a></li>
    <li><a href="callback.php">Arrange Callback</a></li>
  </ul>
</li>
</ul>

```

我曾说过把对应于当前页面的菜单项显示为STRONG元素是个非常好的主意。我们来进一步扩展一下这种思想，去掉多余菜单只保留实际上所必需的。这意味着你要从菜单项中移除所有其他嵌套的元素而不是当前的一个（或它的父元素）。例如，在公司的入口页面上菜单会是下面的样子：

```

<ul id="nav">
  <li><a href="index.php">Home</a></li>
  <li><a href="products.php">Products</a>
    <ul>
      <li><a href="cms.php">CMS solutions</a></li>
      <li><strong>Company Portal</strong></li>
      <li><a href="mailserver.php">eMail Solutions</a></li>
    </ul>
  </li>
  <li><a href="services.php">Services</a></li>
  <li><a href="pricing.php">Pricing</a></li>
  <li><a href="about_us.php">About Us</a></li>
  <li><a href="contact.php">Contact Us</a></li>
</ul>

```

对于像那样对应于产品页面的顶级菜单项，代码会更少一些：

```

<ul id="nav">
  <li><a href="index.php">Home</a></li>
  <li><strong>Products</strong></li>
  <li><a href="services.php">Services</a></li>
  <li><a href="pricing.php">Pricing</a></li>

```

```
<li><a href="about_us.php">About Us</a></li>
<li><a href="contact.php">Contact Us</a></li>
</ul>
```

现在可以使用JavaScript很容易地实现这个功能；可是，这和我们做练习的目的是相抵触的，因为没有JavaScript用户仍然会得到全部的菜单。

作为替代，可以使用服务器端脚本PHP来为你实现这些功能。在每一个页面这个脚本都会：

- 加载导航模板。
- 检查一遍所有的链接，并把匹配当前文件名的元素替换为一个STRONG元素（通过正则表达式）。
- 通过GET检查是否有一个传送的?ajax参数，如果没有的话，则删除不在LI项中的所有嵌套列表，其中包括一个STRONG元素。

这里我们就不深入脚本的细节了，因为这不是一本关于PHP的书籍；如果你想看一下的话，这个叫做globals.php的脚本在本章代码下载的导航文件夹中。

使用这个脚本，可以为用户提供完整的功能菜单且不包含任何多余的链接，如图8-9所示。

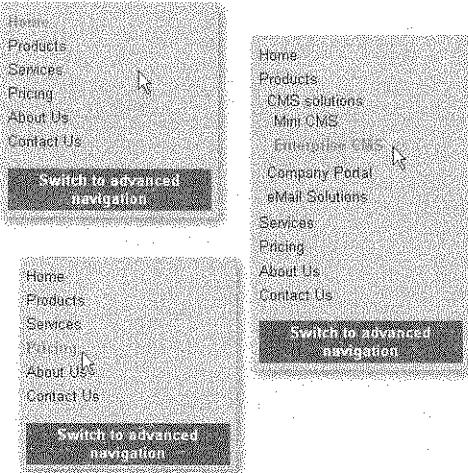


图8-9 不用JavaScript的基本导航

可以使用JavaScript为用户提供更高级的导航菜单，用它来显示完整的站点地图，并且通过XHR加载页面内容而不用重新加载整个页面。要实现这个功能，可以复用前一章中开发的动态导航脚本，并且添加一行代码，当且仅当在当前页面的URI中有一个叫做ajax的搜索参数时触发折叠和展开功能：

siteNavigationIndicator.js (修改后)

```
sn={
  init : function() {
    if( window.location.search.indexOf( 'ajax' ) == -1 ){ return; }
    [... code snipped ...]
  },
}
```

主脚本必须实现下面几个功能：

- 使用链接给导航添加一个新的列表项，允许用户打开和关闭Ajax功能（选择一种基本的或增强效果的导航）。
- 如果在URI中没有ajax参数，则立即停止并让后台停止创建和修改导航。
- 否则的话，在导航上把事件处理程序绑定到所有的链接上，通过XHR在输出容器中加载链接的文档而不是重新加载整个页面。
- 当另一个链接被激活的时候，使用一个指向相应文档和XHR处理程序的链接替换旧的STRONG元素。
- 使用一个STRONG元素替换另一个链接，并且展开和突出显示这个链接所在的菜单选择区。

注解 这意味着你需要知道替换STRONG元素的这些链接应该指向哪个文档。为此，可以把文件名（没有.php扩展名，因为类名中不能包含句点）存储在STRONG的类属性中。这并不是最巧妙的选择，但是相对于使用带有虚构属性的非法HTML，这是唯一的方式了。PHP脚本会为你提供突出显示菜单项的功能。

先定义一些属性——菜单的ID、作为空属性的触发器以及它的ID和两种状态的标签。触发器会变成菜单中的一个列表项用来打开和关闭Ajax功能与增强的导航。根据状态，链接的文本会使用不同的标签。再者，需要定义输出元素的ID和在XHR加载内容的时候显示的消息。

XHRSiteNav.js

```
Xhrsitenav = {
    navID : 'nav',
    trigger : null,
    triggerID : 'AJAXtrigger',
    triggerLabel : 'Switch to advanced navigation',
    downLabel : 'Switch to basic navigation',
    output : 'content',
    loadingMessage : ''
```

init()方法需要测试所有必需的元素，调用createTrigger()来给菜单添加链接，并且当在当前的URL中没有ajax搜索参数的时候就不用做其他处理了。

XHRSiteNav.js (续)

```
init : function() {
    if( !document.getElementById || !document.createTextNode ) {
        return;
    }
    xhrsitenav.nav = document.getElementById( sn.navID );
    if( !xhrsitenav.nav ) { return; }
```

```
xhrsitenav.outputContainer = document.getElementById('xhrsitenav');
xhrsitenav.output );
if( !xhrsitenav.outputContainer ){ return; }
xhrsitenav.createTrigger();
if( window.location.search.indexOf( 'ajax' ) == -1 ){ return; }
```

循环处理除去最后一个和只有一个#符号作为href属性的所有链接；对每一个链接指派一个指向xhr方法的click事件处理程序。因为最后的链接是个触发器，所以需要跳过最后一个；只有#符号作为href属性值的链接是由动态菜单脚本添加的。

XHRSiteNav.js (续)

```
var navlinks = xhrsitenav.nav.getElementsByTagName( 'a' );
for( var i = 0; i < navlinks.length - 1; i++ ){
  if( navlinks[i].href == '#' ){ continue; }
  DOMhelp.addEvent( navlinks[i], 'click', xhrsitenav.xhr, false );
  navlinks[i].onclick = DOMhelp.safariClickFix;
}
```

接下来取出触发器中的链接，并移除问号和它后面的所有字符。这意味着如果用户单击了链接，就会关闭Ajax和动态菜单功能。还需要把链接的文本改为基本导航的标签文本。

XHRSiteNav.js (续)

```
var triggerlink = xhrsitenav.trigger.getElementsByTagName( 'a' )[0];
triggerlink.href = triggerlink.href.replace( /\?.*/ , '' );
triggerlink.innerHTML = xhrsitenav.downLabel;
},
```

xhr()方法需要通过getTarget()获取当前的链接，并通过与正确的nodeName进行比较来确保事件目标是一个链接。

XHRSiteNav.js (续)

```
xhr : function( e ){
  var t = DOMhelp.getTarget( e );
  while( t.nodeName.toLowerCase() != 'a' ) {
    t = t.parentNode;
  }
```

它使用了parentNode来获取链接所在的列表项，并且检查这个列表项是否包括一个STRONG元素。如果存在一个STRONG元素，那么xhr()方法会通过在动态菜单脚本中定义的参数来突出显示当前的区域。它调用了removeOldHighlight()来撤销其他的突出显示，并且获得url来加载内容容器元素。

XHRSiteNav.js (续)

```
var parentLI = t.parentNode;
if( t.parentNode.getElementsByTagName('ul').length > 0 ) {
  var firstList = t.parentNode.getElementsByTagName( 'ul' )[0];
  DOMhelp.cssjs( 'add', firstList, sn.showClass )
  DOMhelp.cssjs( 'swap', parentLI, sn.openClass, sn.parentClass );
```

```

parentLI.getElementsByTagName( 'a' )[0].innerHTML ==>
sn.openIndicator;
}
var url = xhrsitename.removeOldHighlight( t );

```

xhr()的其余部分和它的辅助方法retrieved()和failed()和本章的其他例子一样，没有发生变化。

XHRSiteNav.js (续)

```

var request;
try {
    request = new XMLHttpRequest();
} catch(error) {
    try {
        request = new ActiveXObject('Microsoft.XMLHTTP');
    } catch(error) {
        return true;
    }
}
request.open( 'get', url, true );
request.onreadystatechange = function() {
    if( request.readyState == 1 ){
        xhrsitename.outputContainer.innerHTML ==>
        xhrsitename.loadingMessage;
    }
    if( request.readyState == 4 ) {
        if( request.status && /200|304/.test( request.status ) ) {
            xhrsitename.retrieved( request );
        } else {
            xhrsitename.failed( request );
        }
    }
}
request.setRequestHeader( 'If-Modified-Since', =>
'Wed, 05 Apr 2006 00:00:00 GMT');
request.send( null );
DOMhelp.cancelClick( e );
return false;
},
retrieved : function( requester, e ) {
    var data = requester.responseText;
    xhrsitename.outputContainer.innerHTML = data;
    DOMhelp.cancelClick( e );
    return false;
},
failed : function( requester ) {
    alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
    return true;
},

```

`createTrigger()`方法会检查是否已定义了一个trigger，如果需要的话则创建一个新的。`trigger`是包含链接的列表项，它允许用户在不同的导航状态下切换。在这个方法中没有任何会令你惊讶的东西，因为它毕竟是`init()`方法的一部分。把类似这样的功能放到自己的方法里可能比较好，其他的脚本需要多添加一次触发器。

XHRSiteNav.js (续)

```
createTrigger : function() {
  if( !xhrsitenav.trigger ) {
    xhrsitenav.trigger = document.createElement( 'li' );
    xhrsitenav.trigger.id = xhrsitenav.triggerID;
    var loc = xhrsitenav.shorturl( window.location.href );
    var newlink = DOMhelp.createLink( loc + '?ajax=1', -->
      xhrsitenav.triggerLabel );
    xhrsitenav.trigger.appendChild( newlink );
    xhrsitenav.nav.appendChild( xhrsitenav.trigger );
  }
},
```

`removeOldHighlight()`方法是被`xhr()`方法调用的。它把导航中的当前STRONG元素替换为一个链接或作相反替换。对于被链接替换的STRONG元素，这个方法会设置`href`属性为STRONG元素的类名后面加个“.php”，并且绑定事件处理程序来在用户单击链接的时候调用`xhr()`。

XHRSiteNav.js (续)

```
removeOldHighlight : function( o ) {
  var highlight = xhrsitenav.nav.getElementsByTagName('strong')[0];
  var current = highlight.className + '.php';
  var newlink = document.createElement( 'a' );
  newlink.appendChild( document.createTextNode( -->
    ( highlight.innerHTML ) ) );
  newlink.setAttribute( 'href', current );
  DOMhelp.addEvent( newlink, 'click', xhrsitenav.xhr, false );
  newlink.onclick = DOMhelp.safariClickFix;
  highlight.parentNode.replaceChild( newlink, highlight );
```

对于作为参数`o`获得的链接，这个方法调用了一个名为`shorturl()`帮助方法，它从链接的`href`属性中清除多余的数据（你这里只需要文件名，而不是`href`返回的全部路径）。它接着创建一个新的STRONG元素，移除.php文件扩展名，并且把返回的字符串存储把STRONG元素的`className`属性中。最后，在url前面加上存储内容页面的文件夹名称，并把完整的url返回到`xhr()`以加载正确的内容。

XHRSiteNav.js (续)

```
var shorturl = xhrsitenav.shorturl( o.getAttribute( 'href' ) );
var st = document.createElement( 'strong' );
st.className = shorturl.replace( '.php', '' );
st.innerHTML = o.innerHTML;
```

```

o.parentNode.replaceChild( st, o );
var url = 'content/' + shorturl;
return url;
},

```

`shorturl()`帮助方法会从获得的作为参数的字符串中去掉最后一个反斜杠之前的所有字符，并返回最终的结果。

XHRSiteNav.js (续)

```

shorturl : function( url ) {
    return url.replace( /.*/g, '' );
}
}

DOMhelp.addEvent(window, 'load', xhrsitenav.init, false );

```

如果用户选择增强的导航，他会得到一个树状的菜单，可以在页面中直接加载所需的内容而不用重新加载整个文档。如图8-10所示。

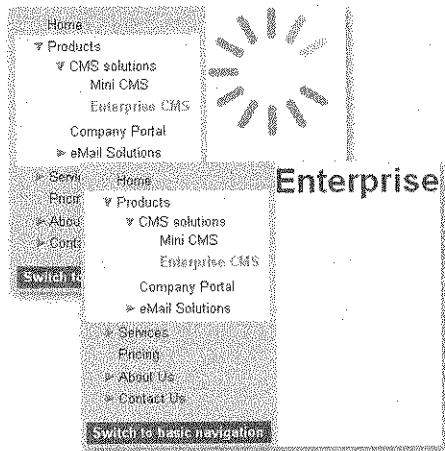


图8-10 增强的XHR导航

这个示例教你如何使高端的JavaScript功能成为可选的，而不用期望访问者都可以处理它。它不是一个非常通用的方法，因为它给界面又添加了一层复杂度；但是依赖你的产品的用户，可能会留下比较好的印象，而且还会告诉访问者如果他们选择了高级的导航，就不能期望可以获得和静态导航同样的体验——例如，返回按钮的功能和书签。

8.4 小结

希望这一章会使你领悟到使用JavaScript和XMLHttpRequest能够做什么，从而来创建后台和浏览器之间的动态连接而不用重新加载页面。

Ajax的确很酷，但有几点必须牢记：

- 首先，Ajax是作为开发Web应用程序而不是网站的一种方法提出的，而不是网站。对于小的表单和菜单都使用Ajax就有点小题大做了。
- Ajax是客户端脚本和后台之间的连接器；它的能力受后台脚本或信息的限制。
- Ajax被限制在和使用它的脚本处于相同服务器上的脚本和数据源中，除非你在服务器上使用一种传递脚本或者你有一个可以提供JSON格式数据的第三方服务，正如<http://del.icio.us>那样。
- 创建一个看起来能给人留下深刻印象的Ajax应用程序非常诱人和容易，但是也非常令人讨厌，它依赖于鼠标和JavaScript的可用。创建有亲和力的界面是一件非常困难的任务。

Ajax现在很“热”，许多非常天才的开发人员致力于框架和函数库的开发，帮助你不用了解它所有的细节就可以快速地创建Ajax应用程序——使用它们甚至可以阻止你重复犯这些开发人员自己过去所犯的错误。可用的函数库数量惊人，很难告诉你哪一个对于你手头的任务是适合的。

Leland Scott已经做了一个令人惊异的工作，他在网站上调查了大量不同的Ajax函数库，并且根据跨浏览器和跨平台兼容性对它们进行了比较：<http://www.musingsfrommars.org/2006/03/ajax-dhtml-library-scorecard.html>。

这里是最知名的也是最流行的几种函数库的性能和目标的信息：

- DOJO (<http://www.dojotoolkit.org/>) 是其中目前可用的最成熟和最流行的DHTML/Ajax工具包之一。仅有的问题是它没有大量的示例。
- prototype (<http://prototype.conio.net/>) 是一个非常强大且值得鉴赏的框架；可是，它既没有文档也没有例子，这也是为什么以下好几种原型的改进版本你都可以使用的原因。
 - script.aculo.us (<http://script.aculo.us/>) 可能是可用的最好文档化函数库了。大量的例子和解释会帮助你很快地掌握库函数。
 - moofx (<http://moofx.mad4milk.net/>) 是专门开发的小型函数库。整个函数库只有3KB大小，并且提供了许多可视的技巧和特效，还有一个基本的Ajax引擎。
 - Rico (<http://openrico.org/rico/home.page>) 是专门为Web应用程序开发的，它的特点是有—个面板工具条、一个数据网格和一些特效，还有它自己的一个Ajax引擎。
 - S@rdalya (<http://www.sarmal.com/sardalya/Default.aspx>) 是一个比较新的而且模块化很强的函数库。更新速度很快。S@rdalya的开发人员在evolt的thelist邮件列表 (<http://lists.evolt.org/>) 上也非常活跃。
- YUI (<http://developer.yahoo.com/yui/index.html>) 是由雅虎创建和维护的，提供了许多文档和例子，可以使你对Ajax很容易上手。在网站<http://groups.yahoo.com/group/ydn-javascript/>上可以找到一个支持的邮件列表。
- Sarissa (<http://sarissa.sourceforge.net/doc/>) 在你需要处理XML时非常有用；它允许你使用XSLT很容易地把XML转换为字符串，把一种XML文档转换为另一种文档。

在第9章中，我们会进一步看一下正则表达式以及如何使用它们来验证数据。你会从示例应用程序中学到如何创建一个联系表单，而且可以复用一些这里实现的XHR功能来使它比你通常的联系表单更平滑流畅。

数据验证技术

本章将介绍如何使用JavaScript对用户输入的或从其他系统传过来的数据进行验证。第2章中我们已经了解了一些数据处理方面的知识，这里会用到其中的一部分并对其进行扩展。

首先，将介绍使用JavaScript进行数据验证的优点和缺点，然后讲解几种不同的技术，首先是字符串和数学验证技术，然后逐渐讲到开发人员的开发利器：使用正则表达式进行模式匹配。

还将介绍几种对表单中哪些数据进行验证、如何使表单验证维护起来更加容易以及如何显示验证结果的不同方法。最后介绍一个可以提供建议值的Ajax示例。

以往针对JavaScript初学者的书会介绍所有客户端验证的输入和输出，在数据传输到服务器端之前进行数据验证能够达到什么样的效果。我特意减少了这方面内容，原因是单纯地依赖JavaScript验证并不安全（下面你将看到），当你真正应该依赖于网站服务器端进行验证时，可以花上几个小时编写一些验证函数，并且使用JavaScript验证只是检查用户数据输入的第一步。你将学习到在JavaScript中使用服务器端组件验证规则的技术。

9.1 客户端 JavaScript 验证的优点和缺点

出于下面几点原因，客户端使用JavaScript验证用户输入是非常不错的：

- 当输入错误数据的时候页面不需要重新加载；可以保存输入的变量的状态，所以不需要再一次输入所有变量，只输入错误数据即可。
- 减轻了服务器端通信压力，因为发生错误的时候并没有数据往返到服务器后端。
- 使得用户界面响应更加迅速，因为可以立刻为用户提供反馈信息。

另一方面，使用JavaScript验证也存在几个问题：

- JavaScript不能作为唯一的验证方法（JavaScript可能不可用，甚至可能被故意关闭以阻止进行验证）。
- 这可能会产生这样一个错觉，验证输入数据是非常简单的过程——并不是这样，而且它对你产品的安全性和可用性都是非常关键的。一些网站的表单使用起来并不那么顺手，因为验证的可用性方面没有得到重视。输入数据已经很麻烦了，所以不要为用户增加不必要的难度了，如要求数据遵循一种难以理解的格式，或者使用错误报告的方法阻碍用户的操作（如，警告消息和客户端验证在不使用JavaScript情况下都不可能发送表单），而

不是阻止用户犯错误。

- 可能出现这种情况，当页面发生了一些动态变化的时候，用户代理不会通知用户——这种情况对于那些使用旧一些屏幕读取器的视觉受到损伤的用户就属于这种情况。
- 除非你在客户端和服务器端使用了相同的验证规则（本章中，你会看到这样的例子），否则就意味着当验证规则发生一处变化的时候，需要进行两次维护。
- 如果不想让你的验证规则是可视的——如阻止一些垃圾信息或出于确认用途等——在 JavaScript 中这是不可能实现的。

9.2 使用 JavaScript 保护文件内容

数据验证与通过密码或通过加密混淆保护文件内容是两码事。如果你上网看一下，可以找到大量的通过 JavaScript 对网页提供密码保护的例子，那些脚本通常是这样的：

```
examplePassword.html
var pw = prompt( 'Enter Password', '' );
if( pw != 'password123' ) {
    alert( 'Wrong password' );
    window.location = 'boo.html' ;
} else {
    window.location = 'creditCardNumbers.html';
}
```

破解这种密码保护唯一的方法是查看网页源代码，如果提供密码保护的源代码是在它自己的 JavaScript 文件当中，那么使用浏览器或文本编辑器将其打开就可以了。一些情况下，无法重定向到正确的网页，而只能定向到错误的网页，那么只要简单地关闭 JavaScript 就可以了。

有一些看起来更加巧妙的保护办法，它们把密码作为文件名称的一部分：

```
var pw = prompt( 'Enter Password', '' );
window.location = 'page' + pw + '.html';
```

这些可以通过查看服务器上有哪些可用文件来破解——因为显示的目录可能尚未关闭（令人惊讶的是这种情况很多——比如在 Google 中搜索 “index of /mp3” ——包含引号）或页面可以在计数器统计中或者在浏览器或 Google 的高速缓存中找到。

同样的方法也可以应用到混淆的（通过加密或替换单词使文件很难理解）内容和脚本上。所有通过 JavaScript 进行保护的东西都可以破解——只要有足够的时间和决心。只是不要用它去浪费时间。

注解 JavaScript 是一种大多时候在客户端计算机上执行的语言，这使得那些不怀好意的攻击者可以轻而易举地破解你的保护方法。通过禁止鼠标右击来防止图片或内容被复制的方法，可以阻止一般的访问者，对于真正的攻击者不会起到任何的作用。

使用一些类似Dean Edward的打包工具 (<http://dean.edwards.name/packer/>) 对JavaScript进行打包，以使得比较笨重的脚本可以更简短一些是另外一个问题，尽管有时候这也是个很好的主意，例如，要在一个负载很大的网站上使用很大的JavaScript函数库脚本。

9.3 全能验证的神话

验证用户的输入可以很简单。例如，确保表单的一个文本域有数据输入，下拉框内要有一个选项被选择，或者复选框被选中。可是，对于一些类似日期和时间这样更复杂的表单数据，就没有一个通用的验证方法了。在网上有许多已编写好的脚本，而且每个开发框架或集成开发工具都提供了许多承诺可以帮你解决任何问题的通用表单验证方法。

遗憾的是，它们大多数都是令人费解的或是没有实现它所承诺的效果。好的Web表单与用户界面和可用性问题都是一个需要技术的问题，熟悉你的用户以及他们的环境是非常重要的。这就是在这里我为什么没有给出许多验证的“银弹”示例的原因，而是讲解了该使用哪些工具去编写你自己的验证脚本。

许多客户端验证指南和已有的解决方案所犯的最大错误就是没有考虑到区域之间的差异。

- 我的生日在美国应表示为4/26/1975，在欧洲为26/04/1975，而在其他地方则可能把它写做26.April 1975或Apr.26,1975。
- 数字在英国和美国表示为1,000.95，而德国却写成1.000,95。
- 美国和英国都用1pm或1am，而其他的欧洲人则使用13.00或1.00。
- 英国的邮政编码语法类似N165UN，美国的邮政编码为12345或-1234，而德国在统一之前邮政编码为4位阿拉伯数字，现在则是5位。
- 测试电话号码是否完全由数字构成并不够，因为人们可能提供了国家代码或者分机号码，如+44(0)208 11111-1122。

非常有趣的是，很多的服务器端解决方案和架构如.NET、Spring和Mono已经为你考虑了这些问题，可以在本地化的文件中定义显示和验证的规则。这意味着如果你在一个使用了这些架构的环境里工作，那么最好和后台共用验证规则。在9.7.7节我们会再讨论这个问题。

9.4 使用字符串和数字方法的基本JavaScript验证

验证用户输入最简单的方式就是使用字符串和数字方法。在第2章中讨论数据类型和条件判定的时候已经讲了许多这方面的内容，在这里我会对它们再进行一下归纳。

9.4.1 字符串验证方法

字符串验证中最常用的方法如下。

- `charAt(n)`: 返回字符串中第n个位置的字符，从0开始。
- `charCodeAt(n)`: 返回字符串中第n个位置字母的Latin-1 ASCII值，第1个位置为0。
- `indexOf(search)`: 返回search在主字符串中的位置，如果没有找到则返回-1。

- `lastIndexOf(search)`: 返回search在主字符串中的最后位置，如果没有找到则返回-1。
- `slice(start,end)`: 返回start和end之间的字符串，如果没有提供end值则返回剩余的所有字符串。
- `split(search)`: 将字符串分割成为数组，数组元素为search周围的字符串。search本身不包含在数组之内。
- `substr(start,n)`: 将从start开始的n个字符作为字符串返回。
- `substring(start,end)`: 返回start和end之间的字符串。

其他功能强大的字符串方法，`search`、`replace`和`match`，经常与正则表达式连用，这些我会在后面作讲解。

举个例子，可以使用这些字符串的方法来检查一个值是否为有效数字。

`exampleCheckNumberString.html` (摘录)

```
<form onsubmit="return isNumber()">
<p>
    <label for="total">Total</label>
    <input type="text" name="total" id="total" />
</p>
<p><input type="submit" value="send" /></p>
</form>

function isNumber() {
    var currentCode;
    var total = document.getElementById( 'total' );
    if( !total ) { return false; }
    total = total.value;
    if( total.length == 0 ) {
        alert( 'Field is empty' );
        return false;
    }
    if( total.indexOf( '-' ) != -1 && total.substring( 0,1 ) != '-' || 
        total.lastIndexOf( '-' ) != total.indexOf( '-' ) ) {
        alert( 'A number can only have a minus at the beginning' );
        return false;
    }
    if( total.indexOf( '.' ) != -1 &&
        total.lastIndexOf( '.' ) != total.indexOf( '.' ) ) {
        alert( 'A number can only have one decimal point' );
        return false;
    }
    for( var i = 0; i < total.length; i++ ) {
        currentCode = total.charCodeAt( i );
        if( currentCode != 45 && currentCode != 46 &&
            currentCode < 48 || currentCode > 57 ) {
            alert( 'Only Numbers are allowed' );
            return false;
        }
    }
}
```

```

    }
}

return true;
}
}

```

我们来从头到尾看一下这个脚本，看它究竟做了些什么。

exampleCheckNumberString.html

```

function isNumber () {
    var currentCode;
    var total = document.getElementById( 'total' );
    if( !total ){ return false; }
    total = total.value;
}

```

定义一个名为currentCode的变量，并获取表单中ID为total元素的值。

exampleCheckNumberString.html (续)

```

if( total.length == 0 ) {
    alert( 'Field is empty' );
    return false;
}

```

接下来，通过校验值的length是否为0来测试这个表单域是否输入了值。如果没有输入的话，则在显示错误信息之后，通过返回false来阻止表单的提交。

exampleCheckNumberString.html (续)

```

if( total.indexOf( '-' ) != -1 && total.substring( 0,1 ) != '-' || 
    total.lastIndexOf( '-' ) != total.indexOf( '-' ) ) {
    alert( 'A number can only have a minus at the beginning' );
    return false;
}

```

由于有效数字应该只有一个减号作为第一个非数字字符（理论上，也可以是一个加号或是一个小数点，但有谁会这样输入呢），因此你需要测试值是否包含减号(indexOf()方法不返回-1)，并通过substring()方法查看其是否在第一个位置上。然而这样还不够，因为用户输入的可能是-12-12，这意味着你需要检查lastIndexOf()和indexOf()方法是否有所不同。这似乎是包含多个减号的唯一一种情况。如果包含多个减号或减号没有在开始的位置，那么显示错误信息并且返回false。

exampleCheckNumberString.html (续)

```

if( total.indexOf( '.' ) != -1 &&
    total.lastIndexOf( '.' ) != total.indexOf( '.' ) ) {
    alert( 'A number can only have one decimal point' );
    return false;
}

```

同样的逻辑也可以应用到小数点上；在有效的数字中它只能有一个。

```
exampleCheckNumberString.html (续)

for( var i = 0; i < total.length; i++ ) {
    currentCode = total.charCodeAt( i );
    if( currentCode != 45 && currentCode != 46 &&
        currentCode < 48 || currentCode > 57 ) {
        alert( 'Only Numbers are allowed' );
        return false;
    }
}
```

使用charCodeAt()方法检查输入值中是否包含非法字符。只有减号（45）、小数点（46）以及0~9的数字字符（48~57）是允许的。循环遍历给定字符串中的每个字符，如果有任何字符的ASCII码值在这个范围之外，那么显示一个错误消息并返回false。

```
exampleCheckNumberString.html (续)

    return true;
}
```

如果所有条件都满足，则返回true，并把数据表单提交到服务器上。

一个更复杂的示例是检查Email地址是否有效。Email中必须包含@符号，用户名的开始部分不应该以句点(.)或连字符(-)开头，也不能以句点(.)结尾。@之后的域名开头和结尾部分也不应该是句点(.)或连字符(-)。@前后的两部分都应该只包含字母a~z、数字0~9、句点、下划线和连字符。

```
exampleCheckEmailString.html

function isEmail() {
    var mail = document.getElementById( 'email' );
    var error = '';
    if( !mail ){ return false; }
    var mailstring = mail.value;
```

开始先检查表单中必需的表单域并获取它的值。定义一个叫做error的变量来存储错误信息。在这个例子中，用它只是为了控制输入行的长度，但在本章结尾部分的表单例子当中，就不是这样了。

```
exampleCheckEmailString.html (续)

if( mailstring.length == 0 ) {
    error = 'You didn\'t enter a value';
    alert( error );
    return false;
}
```

通过检查length值是否为0来确定用户是否输入了Email。如果文本域中没有输入则显示警告信息并取消表单的提交。

```
exampleCheckEmailString.html (续)

if( mailstring.indexOf( '@' ) == -1 ) {
    error = 'This email has no @ sign';
    alert( error );
    return false;
} else if( mailstring.lastIndexOf( '@' ) != mailstring.indexOf( '@' ) ) {
    error = 'An email may only contain one @ sign.';
    alert( error );
    return false;
}
```

使用indexOf()方法检查Email中是否包含@符号，然后通过比较indexOf()和lastIndexOf()的返回值来判断Email中是否只包含一个@符号。

```
exampleCheckEmailString.html (续)

var chunks = mailstring.split( '@' );
var n = chunks[0];
```

在@符号处将值分成用户名和域名两部分，分别对它们进行检查。定义n为第一“块”的值，即用户名。

```
exampleCheckEmailString.html (续)

if( n.substring( 0,1 ) == '.' || n.substring( 0,1 ) == '-' ||
    n.substr( n.length-1, 1 ) == '.' ) {
    error = 'The user name may not start with a period or hyphen';
    error += ' and may not end with a period';
    alert(error);
    return false;
}
```

用户名不应该以句点或连字符开头，也不能以句点结尾，可以通过substring()和substr()方法对这些条件进行测试。

注解 这个例子显示了在检查字符串最后一个字符时使用substr()是个更简短的方法。检查最后一个字符，使用string.substr(string.length-1,1)，而不是string.substring(string.length-1, string.length)。

如果邮件地址无效，则显示一个错误消息并阻止表单的提交。

```
exampleCheckEmailString.html (续)

if( !checkValidCharacters( n ) ) {
    error = 'The email name contains invalid characters';
    alert( error );
    return false;
}
```

名称中应该只包含数字、字母、破折号、下划线和句点。实用函数checkValidCharacters()就是用来验证这些的，并相应地返回true或false。如果在用户名中包含非法字符，应该提示给用户，并取消表单的提交。

exampleCheckEmailString.html (续)

```
n = chunks[1];
if( n.substring( 0, 1 ) == '.' || n.substring( 0, 1 ) == '-' ||
    n.substr( n.length-1, 1 ) == '-' || 
    n.substr( n.length-1, 1 ) == '.' ) {
    error = 'The domain name may not start or end with a hyphen or period';
    alert( error );
    return false;
}
```

接下来看一下第二“块”，即Email的域名部分，确保其开头和结尾都不是连字符或句点。否则的话，提示用户并取消表单提交。

exampleCheckEmailString.html (续)

```
if( !checkValidCharacters( n ) ) {
    error = 'The domain name contains invalid characters';
    alert( error );
    return false;
}
return true;
```

检查域名是否包含非法字符，如果一切都是没有问题的话，则返回true，这样就将表单提交到了服务器端了。

exampleCheckEmailString.html (续)

```
function checkValidCharacters( n ) {
    for( var i = 0; i < n.length; i++ ) {
        currentCode = n.charCodeAt( i );
        if( currentCode == 45 || currentCode == 46 ||
            currentCode == 95 ||
            ( currentCode > 96 && currentCode < 123 ) ||
            ( currentCode > 47 && currentCode < 58 ) ) {
                continue;
            } else {
                return false;
            }
    }
    return true;
}
```

在实用函数checkValidCharacters()中，循环遍历整个字符串（作为参数n被传递过来的）的

长度，并确保字符串中的每个字符的charCode没有落在允许的字符范围之外。当字符既不是连字符(45)、句点(46)，也不是下划线(96)、a~z的字符(97~122)或0~9的字符(48~57)时，返回false；否则的话，返回true。

这个数字检验示例看起来有些费解，而且的确也是这样的。对于数字的验证和比较，使用数字的验证方法会更好一些。

9.4.2 数字验证方法

- Number(): 将括号内的变量的值转换为数字。
- isNaN(n): 检验n是否为数字(浮点或整形)，如果不是则返回true。
- parseFloat(n): 将n转换为浮点数。它从左至右依次解析字符串中的每个字符，直至在数字中无法使用的字母，然后停止，将字符串转换为数字。如果第一个字符在数字中就无法使用，结果就是NaN，表示非数字(Not A Number)。
- parseInt(n): 通过直接把小数部分移除而不考虑四舍五入，将n转化为整数。任何传到这个函数当中的非数字字符都会被丢弃掉。如果第一个字符不是+、-或数字，结果就是NaN。

只要验证是关于数学的，比如测试一个实数，那么数学方法更容易使用些。前面提到的isNumber()函数如果使用isNaN()函数进行重写就会简单很多，isNaN()函数会在它的参数不是数字时返回true。

exampleCheckNumberMath.html (摘要)

```
function isNumber() {
    var total = document.getElementById('total');
    if( !total ) { return false; }
    total = total.value;
    if( total.length == 0 ) {
        alert('Field is empty');
        return false;
    }
    if( isNaN( total ) ) {
        alert('Please enter a number');
        return false;
    }
    return true;
}
```

一个更复杂的例子是检查一个日期的格式是否正确。日期的最大问题是它们不遵从一种直接的算法；如果每个月有10天，每年有10个月那就简单得多了。但事实上，你需要检查很多细节：

- 月份是否在1至12当中？
- 某天在月份中是否存在——例如，31.04.2000？(不存在)
- 是否为闰年，2月29日是否可以存在？

让我们一步一步地分析下面的例子：

exampleCheckDateMath.html (摘要)

```
function isValidDate() {
    var leap;
    var datestring = document.getElementById( 'date' );
    if( !datestring ){ return false; }
    datestring = datestring.value;
    if( datestring.length == 0 ) {
        alert( 'Field is empty' );
        return false;
    }
}
```

首先为闰年定义一个变量，读取日期字段的值，并检查该字段中是否有输入内容。

exampleCheckDateMath.html (续)

```
if( datestring.length != 10 && datestring.indexOf( '/' )== -1 || 
    datestring.indexOf( '/' ) != 2 || 
    datestring.lastIndexOf( '/' ) != 5){
    alert( 'The date is not in the format dd/mm/yyyy' );
    return false;
}
```

通过字符串的长度检查日期是否为正确的格式，并且确保斜杠 (/) 在正确的位置。

exampleCheckDateMath.html (续)

```
var chunks = datestring.split( '/' );
var day = chunks[0];
var month = chunks[1];
var year = chunks[2];
```

然后将日期分开为日、月、年，并进行单独的验证。

exampleCheckDateMath.html (续)

```
if( ( month < 1 ) || ( month > 12 ) ) {
    alert( 'The month must be in between 01 and 12' );
    return false;
}
```

月份非常容易，所要做的就是检查其是否小于1或大于12。

exampleCheckDateMath.html (续)

```
if( day < 1 ) {
    alert('The day cannot be negative');
    return false;
}
```

日期的检查首先要排除负数，在前面的格式检验中被跳过了。

exampleCheckDateMath.html (续)

```

if( (year % 4 == 0) || (year % 100 == 0) || (year % 400 == 0) ) {
    leap = 1;
}
if( (month == 2) && (leap == 1) && (day > 29) ) {
    alert( 'This month does not have that many days' );
    return false;
}
if( (month == 2) && (leap != 1) && (day > 28) ) {
    alert( 'This month does not have that many days' );
    return false;
}

```

对于2月份，需要检查日期是否为闰年。闰年是那些可以被4或100或400整除的年份。因此，可以通过求余计算（%）并将结果与0进行比较来判断。如果是闰年，2月份可以有29天；否则的话它只能有28天。

exampleCheckDateMath.html (续)

```

if( ( day > 31 ) && ( ( month == "01" ) || ( month == "03" ) ||
    ( month == "05" ) || ( month == "07" ) || ( month == "08" ) ||
    ( month == "10" ) || ( month == "12" ) ) ) {
    alert( 'This month does not have that many days' );
    return false;
}
if( ( day > 30 ) && ( ( month == "04" ) || ( month == "06" ) ||
    (month == "09" ) || ( month == "11" ) ) ) {
    alert( 'This month does not have that many days' );
    return false;
}
return true;
}

```

对于其他的月份，需要判断输入的天数是否超过了30或31，以及那个月是否有那么多天。由于JavaScript没有提供简便方法，你需要对这些比较进行硬编码。如果一切顺利，则返回true以将表单提交给服务器。

必须对这些比较进行硬编码看起来有些笨拙，而且确实也是这样。如果你还记得JavaScript有个Date对象，它包含了所有的这些逻辑。那么，使用它可以使表单验证更简短些：

exampleCheckDateObj.html (摘录)

```

function isValidDate() {
    var datestring = document.getElementById( 'date' );
    if( !datestring ){ return false; }
    datestring = datestring.value;
    if( datestring.length == 0 ) {
        alert( 'Field is empty' );
    }
}

```

```

    return false;
}
if( datestring.length != 10 && datestring.indexOf( '/' ) == -1 ||
datestring.indexOf( '/' ) != 2 || datestring.lastIndexOf( '/' ) != 5 ) {
    alert( 'The date is not in the format dd/mm/yyyy' );
    return false;
}

```

开始的时候是一样的，要测试文本字段是否存在、是否有数据输入以及输入是否格式正确。

exampleCheckDateObj.html (续)

```

var chunks = datestring.split( '/' );
var testDate = new Date( chunks[2], chunks[1]-1, chunks[0] );
if( testDate.getDate() == chunks[0] ) {
    if( testDate.getMonth()+1 == chunks[1] ) {
        if( testDate.getFullYear() == chunks[2] ) {
            return true;
        } else {
            alert( "This is not a valid year." );
        }
    } else {
        alert( "This is not a valid month." );
    }
} else {
    alert( "This is not a valid date." );
}
return true;
}

```

将日期分解为它的几个组成部分，并使用这些设置创建一个新的Date对象（注意需要将月份减1）。接下来把Date对象的获取方法返回的值和日、月、年进行比较，而且如果不匹配的话，则警告用户发生了错误——Date方法会为你做所有的这些测试！不过，需要将月份加1以调整人类和计算机读取日期的差别。

提示 你应该熟练掌握JavaScript的Math和Date对象。很多时候不需要自己编写代码，直接使用已经为你提供好的就可以了。

9.5 正则表达式^①

正则表达式可以帮你查找匹配一种字符模式的字符串，对于用户输入的验证或修改文档的内容都是非常有用的。它并不局限于JavaScript，还出现在类似Perl、PHP以及UNIX服务器脚本这样的语言当中。它的功能非常强大，如果你与Perl或PHP的狂热者和服务器的管理员进行交谈，你会非常惊讶他们会经常使用一个正则表达式替换你用JavaScript写的50行switch/case或if/else

^① 关于正则表达式，推荐进一步阅读《正则表达式必知必会》（人民邮电出版社，2007）。——编者注

结构。许多编辑环境也允许使用正则表达式的“查找”和“搜索与替换”功能作为其特性。

只要你对它比较熟悉了，就会发现正则表达式非常有用；可是，第一眼看到类似这样的结构：
`/^[\w]+(\.[\w]+)*@[[\w]+\.\.]+[a-z]{2,7}\$/i`（检查一个字符串是否为一个有效的Email），你可能会有一种恐惧感。好消息是它并不像看上去那样恐怖。

9.5.1 语法和属性

假设你想在文本中搜索字符串“cat”，可以以两种不同的格式来定义正则表达式：

```
// String notation; notice that you must not use quotation marks!
var searchTerm = /cat/;
// Object constructor
var searchTerm = new RegExp( 'cat' );
```

如果你通过match()、search()、exec()或test()方法在某个字符串上使用这个表达式，它会返回所有包含cat的字符串，不管在字符串中它的位置在那里，如catalog、concatenation或scat。

如果你希望只匹配单词“cat”的字符串，旁边没有任何其他的字母，那么需要使用一个^表示开始以及\$表示结尾：

```
var searchTerm = /^cat$/;
var searchTerm = new RegExp( '^cat' );
```

也可以省略开始标识符^或结尾标识符\$，那样这些下面的代码就会匹配cat、catalog或catastrophe：

```
var searchTerm = /cat/;
var searchTerm = new RegExp( 'cat' );
```

而下面的会找到polecat或wildcat：

```
var searchTerm = /cat$/;
var searchTerm = new RegExp('cat$');
```

如果想要查找“cat”，而不考虑大小写（例如，要匹配“cat”、“Catherine”或“CAT”）那么你需要在第二个斜杠（/）后面使用i属性，这样就可以忽略大小写了：

```
var searchTerm=/cat/i;
var searchTerm=new RegExp('cat','i');
```

如果有多个字符串，在它内面包含了单词“cat”多次，而你想要以数组的形式获得所有的匹配，那么你需要添加参数“g”表示“全局”：

```
var searchTerm = /cat/g;
var searchTerm = new RegExp('cat','g');
```

默认情况下，正则表达式只在单行字符串中匹配模式。如果想要在多行字符串中匹配一个模式，那么需要使用参数m表示“多行”。也可以把这些参数混合起来使用，而且顺序并不重要。

```
var searchTerm = /cat/gim;
var searchTerm = new RegExp( 'cat', 'mig' );
```

9.5.2 通配符搜索、约束范围以及其替换

句点字符(.)在正则表达式中扮演着非常重要的角色，它表示“任意字符”（这会让人感到迷惑，因为在高级网页搜索或在DOS与UNIX的命令行中使用的是星号：*）。

```
var searchTerm = /c.t/gim;
var searchTerm = new RegExp('c.t', 'mig');
```

这个表达式匹配“cat”、“cot”、“CRT”，甚至无意义的字符串如“c#t”和“c!T”，或者像“cT”或“c t t”这样包含空格（\t表示制表符）。

这样的表达式对于你的需求来讲可能弹性太大了，这就是为什么你需要使用方括号([])来把选择的范围限制到你想提供的范围之内：

```
var searchTerm = /c[aou]t/gim;
var searchTerm = new RegExp('c[aou]t', 'mig');
```

使用这个正则表达式，可以匹配所有大写或小些的“cat”、“cot”或“cut”。也可以在方括号内提供一些区间，如a-z匹配全部小写字母，A-Z匹配所有大写字母，而0-9则匹配数字。

警告 注意正则表达式匹配的是数字的字符，而不是它们的值。使用[0-9]的正则表达式会返回0200作为一个有效的4位数。

举例来说，如果想要查找所有小写字母后紧跟大写字母的单词，可以使用：

```
var searchTerm = /[a-z][A-Z]/g;
var searchTerm = new RegExp('[a-z][A-Z]', 'g');
```

可以在方括号内使用字符^在搜索条件中排除一个选项。例如，如果希望结果中避免出现“cut”，可以使用：

```
var searchTerm = /c[^u]t/g;
var searchTerm = new RegExp('c[^u]t', 'g');
```

方括号内每次只能匹配一个字符，这也是为什么使用这个表达式不会匹配类似“cost”、“coast”或“cast”这样的字符串了。如果想匹配几个选项，可以在圆括号内使用管道符号(|)，它的功能像一个逻辑的OR：

```
var searchTerm = /c(^u|a|o|os|oas|as)t/g;
var searchTerm = new RegExp('c(^u|a|o|os|oas|as)t', 'g');
```

现在这个表达式匹配“cat”、“cot”、“cost”、“coast”和“cast”，但不包括“cut”（因为有^u）。

9.5.3 使用量词约束字符的数量

在许多情况下，你希望允许一定范围里的字符，如a-z，但需要限制它们的数量。要达到这个目的，可以在正则表达式中使用量词（quantifier），如表9-1所示：

表9-1 正则表达式中的量词符号

符 号	可能的次数
*	0或1次
+	1次以上
?	0或1次
{n}	n次
{n,m}	n~m次

注解 在每个表达式后面添加“?”意味着正则表达式应该匹配字符串，但次数越少越好。

举个例子，如果你想匹配一串数字的语法，它们由两组4个字符组成，每个之间都由破折号（-）分开，你可以使用：

```
var searchTerm = /[a-z|0-9]{4}\-[a-z|0-9]{4}/gim;
var searchTerm = new RegExp( '[a-z|0-9]{4}\-[a-z|0-9]{4}', 'igm' );
```

注解 注意你需要对按照字面意思直接使用的字符转义，它们没有在正则表达式模式中可能有的特殊含义，如这个例子中的破折号。可以通过在这个字符前面添加反斜杠符号(\)来达到这个目的。需要转义的字符有-、+、/、(、)、[、]、*、{、}和?。例如，/c.t/匹配“cat”或“cot”或“c4t”，而/c\ .t/只匹配“c.t”。

9.5.4 词界、空白字符以及其他快捷符号

所有的这些不同选项都可以产生非常费解的正则表达式，这也是为什么提供有一些快捷符号的原因。你可能还记得第2章中表示空白字符的一些特殊字符符号，如\n表示换行、\t表示制表符。正则表达式也提供了这样的一些符号，如表9-2所示：

表9-2 正则表达式的快捷符号

符 号	等价符号	含 义
\d	[0-9]	整数
\D	[^0-9]	整数以外的所有字符
\w	[a-zA-Z0-9_]	所有字母数字字符、下划线
\W	[^a-zA-Z0-9_]	所有非字母数字字符
\b	__	词界
\B	__	非词界
\s	\t\n\r\f\v	所有空白字符
\S	[^\t\n\r\f\v]	所有非空白字符

举个例子，如果想验证美国的社会保障号[它由9位数字组成，在第3位和第5位数字后面有破折号（如456-33-1234）]，那么可以使用下面的正则表达式，带有可选的破折号（使用?数量词），

因为用户可能不输入它们：

```
var searchTerm = /[0-9]{3}\-?[0-9]{2}\-?[0-9]{4}/;
var searchTerm = new RegExp(' [0-9]{3}\-?[0-9]{2}\-?[0-9]{4}', '' );
```

作为选择，对于阿拉伯数字也可以使用快捷符号：

```
var searchTerm = /\d{3}\-?\d{2}\-?\d{4}/;
var searchTerm = new RegExp(' \\\d{3}\-?\\\d{2}\-?\\\d{4}', '' );
```

注意，如果在引号内或构造器符号中使用快捷符号，需要在其前面添加两个反斜杠，而不是一个，因为需要对它们进行转义！有了这些知识，你完全可以编写自己的正则表达式了。我们可以回过头来看一下本节前面的例子：

```
var validEmail = /^[\\w]+(\\.\\[\\w]+)*@[\\w]+\\.[a-z]{2,7}$/
```

邮件地址可以很简单，如me@example.com，也可以很复杂，如chris.heilmann.webdev@example.museum。正则表达式将两者都视为有效的邮件地址。

首先验证在符号@之前的字符串是否以一个或多个单字字符开始的，`^[\w]+`，在句点符号后面@符号之前紧跟着0个或多个单字字符，`(\\.\\[\\w]+)*`。在符号@之后，字符串可能有一组或以上的一个或多个单字字符，后面紧跟一个句点，`([\\w]+\\.)+`，而且它应该以一个2~7个字符组成的字符串结束。最后的字符串是域名，它可以像de这样短，也可以比较长，如name或museum。注意，由于允许多个单词后跟句点的形式，因此还要确保类似user@open.ac.uk这样的Email可以被识别。

9.5.5 使用正则表达式的方法

有几个方法将正则表达式作为参数使用。表达式本身（斜杠或RegExp构造器里的东西）被称作模式（pattern），它会匹配你需要获取或检查的信息。

- `pattern.test(string)`：检查字符串是否匹配模式，并返回true或false。
- `pattern.exec(string)`：对字符串和格式进行一次匹配，并返回匹配结果的数组或null。
- `string.match(pattern)`：对字符串和模式进行匹配，并将匹配结果作为字符串数组返回，或返回null。
- `string.search(pattern)`：对字符串和模式进行匹配，并返回有效匹配的位置。如果字符串不匹配任何一模式，搜索结果则会返回-1。
- `string.replace(pattern, replaceString)`：对字符串和模式进行匹配，使用replaceString替换每一个有效匹配。
- `string.split(pattern, limit)`：对字符串和模式进行匹配，并将字符串分割成为数组，数组元素为模式匹配旁边的子字符串。可选参数limit会限制数组元素的数量。

9.5.6 圆括号分组的功能

你可能还记得要对表达式进行分组，你会使用圆括号()。这不仅仅是对模式进行分组，还可以将组内匹配结果储存在特定的变量之中以便以后使用。当你把它与replace()方法连接使用的

时候尤其方便。结果可以存储在名为\$1~\$9的变量当中，这意味着在每个正则表达式中最多可以使用9组圆括号。如果想从中排除一组，可以在它前面加上?:。

举个例子，如果有一个格式为*Surname,Name*的名字列表，而且你想要将列表中的每一项都转换为*Name Surname*的格式，那么可以这样做：

exampleNameOrder.html

```
names=[  
    'Reznor, Trent',  
    'Eldritch, Andrew',  
    'Clark, Anne',  
    'Almond, Marc'  
];  
for( i = 0; i < names.length; i++ ) {  
    alert(names[i].replace( /(\w+),\s?(\w+)/g, '$2 $1' ));  
}
```

模式可以匹配逗号之前的任何单词（后面跟一个可选的空格字符）以及跟在它后面的单词，并将两者都储存在变量当中，替换字符串通过使用\$变量来反转两个变量的顺序。

一个更复杂的例子是把链接之后的内容区域的每个外部链接的URL打印出来：

exampleShowURL.html

```
showURLs = function(){  
    var ct = document.getElementById( 'content' );  
    var searchTerm = '<a href="((?:http|https|ftp):\//';  
    searchTerm += '(?:[\w]+\.)+[a-z]{2,7})">';  
    searchTerm += '(?:\w|\s|\.)+</a>';  
    var pattern = new RegExp( searchTerm, 'mgi' );  
    ct.innerHTML = ct.innerHTML.replace( pattern, '$1 ($2)' );  
}
```

模式以一组圆括号开始，以将整个结构存储在变量\$1中，并匹配链接的开头部分[关闭圆括号，并将链接的href属性中的所有东西存储在变量\\$2中，匹配链接元素中的所有字符串（它可能是一个或多个单词、空白字符或一个句点），但并不在变量中保存。接着在元素结束符之后关闭主要的圆括号，并且使用replace方法替换模式所匹配的所有链接，它会很好地把example变成example \(http://www.example.com\)。](”。紧接着还是一组圆括号，它包含了href属性内部的东西，即以http、https或ftp开头的URL，这部分不会在变量中存储（因为这一组圆括号后面跟着?:），下面是冒号和两个斜杠（需要被转义），后面跟一个以域名结束的URL（这部分与在Email验证例子中使用的模式是相同的）。</p>
</div>
<div data-bbox=)

9.5.7 正则表达式资源

和其他的程序语言一样，有许多方式可以达到同样的目标，我并不想只是提供些例子在需要的时候供你复制和粘贴，希望你在正则表达式方面走得更远。

有许多在线的资源，根据它们的任务列举了一些模式：

- The Regular Expression Library (<http://regexlib.com/>) 包含一些数据库搜索方面的模式。
- 在Regular-Expressions.info (<http://www.regular-expressions.info/>) 上，可以找到大量关于正则表达式的教程。
- RegEx Advice (<http://regexadvice.com/>) 有一个很好的关于正则表达式方面的论坛和博客。相关的书有Tony Stubblebine的*Regular Expressions Pocket Reference* (O'Reilly, 2003) 和Jeffrey Friedl的*Mastering Regular Expressions* (O'Reilly, 2002)。

对于很多使用UNIX的用户，有一本Nathan A. Good编写的*Regular Expression Recipes: A Problem-Solution Approach* (Apress, 2004)。

9.6 验证方法小结

在真正编写脚本的时候，永远不要只使用上面的某一种方法，应该混合使用以尽快地达到目的。并没有固定的规则规定什么时候使用什么，但有几点最好记住：

- 正则表达式只是匹配字符；不能使用它们进行任何计算(至少在JavaScript中不允许；PHP提供了e转换，它可以判断是否匹配PHP代码)。
- 正则表达式是不依赖于语言的，可以在客户端和服务器端语言上使用相同的规则（在后面的9.7.7节会看到一个这样的例子）。字符串和数学方法都是只适用于JavaScript的，在其他语言中可能是不一样的。
- 在正则表达式中匹配很大范围的选项是非常简单的，包含非常复杂的字符串，除非这个范围是非常简单的，如A~Z或0~9。
- 如果一定要验证数字的话，大多时候是不需要使用字符串或正则表达式方法进行验证的，只要使用数学的方法测试值就行了。字符串包含的东西太多了，因为你无法使用它们进行值的比较和计算，唯一的情况就是要确定字符串长度或检查一些特殊字符。
- 使用别人开发的现成模式和方法并没有什么不好意思的，它们中的大多数都被大量的开发人员在不同的环境中测试过。

9.7 表单验证技术

现在我们会讨论一些可以在表单中使用的技术，确定哪些域需要验证以及如何告诉用户有地方出错了。这里的例子现在还不完善，新的技术、更好的浏览器以及用户的行为模式在不断地变化，而且现在看来可能非常理想的方法在一年时间里就会变得非常一般了，甚至对某些用户来说可能已经太一般了。选择的时候最好的办法就是拜访你的用户，并获取他们的反馈信息。不要做太多的假设，已经有很多几乎无法使用的表单了。

9.7.1 指定强制字段

可以使用几种不同的方法指定表单的元素为强制的(mandatory)并需要验证。在继续学习

如何为用户提供验证反馈信息的不同例子之前，我们会讨论一些最常用的方法以及它们每一个所存在的问题。

注解 Chris Campbell以前在Particletree上发表过这样的一篇文章：*A Guide to Unobtrusive JavaScript Validation* (<http://particletree.com/features/a-guide-to-unobtrusive-javascript-validation/>)。如果想要了解更多，这篇文章及其注解都是非常不错的。

9.7.2 隐藏字段方法

识别必需的字段的传统方法是使用一个叫做mandatory的隐藏字段或一些类似的东西，它按名称列出了所有强制字段的名称。这使你可以在客户端和服务器端处理的时候使用相同的信息。一个非常常用的表单邮件脚本，Matt Wright的formmail.pl (<http://www.scriptarchive.com/formmail.html>)，就使用了这种方法。

使用JavaScript在客户端验证这样的表单只需要简单地取得这个隐藏字段的值，使用逗号或其他分隔符将其分割到数组中，并且在将其ID传递到验证方法之前，需要检查每个字段是否存在：

exampleHiddenFieldForm.html (摘录)

```
<p class="submit">
  <input type="submit" name="send" value="Send Form" />
  <input type="hidden" name="mandatory" id="mandatory"
    value="email,Message,subject,Name" />
</p>
```

hiddenFieldForm.js (摘录)

```
init:function() {
  [... code snipped ...]
  var mandatory = document.getElementById('mandatory');
  if( !mandatory ){ return; }
  hfv.mandatory = mandatory.value.split( ',' );
  [... code snipped ...]
},
send:function(e) {
  [... code snipped ...]
  for( var i = 0; i < hfv.mandatory.length; i++ ) {
    if( !document.getElementById( hfv.mandatory[i] ) ) { continue; }
    hfv.checkValue( hfv.mandatory[i] );
  }
  [... code snipped ...]
},
```

send()方法会循环遍历数组中的所有项，这个数组是通过在逗号处分割名为mandatory的隐藏字段的值而生成的，并且检查ID为当前项的值的元素是否存在，然后把它传给叫做checkValue()的工具方法。

9.7.3 指示元素方法

可用性要求强制字段对用户来说应该是非常清晰的，这就是为什么在HTML中要求使用指示器来标记强制字段的原因，通常的形式是在标签的文本之后加上一个星号。

可以使用这种方法来表示出哪些字段需要强制验证。循环遍历表单内的所有SPAN元素，并检查它们的类是否为mandatory。如果是的话，则从它的LABEL的for属性（注意属性的名不是for而是htmlFor!）获取元素的ID进行验证。

exampleAsteriskForm.html（摘录）

```
<p>
  <label for="Name">Your Name
    <span class="mandatory">*</span></label>
</p>
<p>
  <label for="Message">Your Message
    <span class="mandatory">*</span></label>
</p>
```

asteriskForm.js（摘录）

```
init:function() {
  hfv.mandatory = [];
  [... code snipped ...]
  hfv.f = document.getElementsByTagName( 'form' )[0];
  var msgs = hfv.f.getElementsByTagName( 'span' );
  for( var i = 0; i < msgs.length; i++ ) {
    if( DOMhelp.cssjs( 'check', msgs[i], 'mandatory' ) ) {
      hfv.mandatory.push( msgs[i].parentNode.htmlFor );
    }
  }
  [... code snipped ...]
},
```

9.7.4 CSS类方法

另一种方法是给强制字段应用适当的CSS类，并对其进行检查：

exampleClassesForm.html（摘录）

```
<p>
  <input type="text" id="Name" name="Name" class="mandatory" />
</p>
<p>
  <input type="text" id="email" name="email" class="mandatory" />
</p>
```

在这个脚本中，必须循环遍历所有的表单域，并检查它们是否有对应的类。也可以使用DOM-2方法，但那样需要几次循环，以分别检查INPUT、SELECT和TEXTAREA元素。

classesForm.js (摘录)

```
hfv.f = document.getElementsByTagName( 'form' )[0];
var msgs = hfv.f.elements;
for( var i = 0; i < msgs.length; i++ ) {
  if( DOMhelp.cssjs( 'check', msgs[i], 'mandatory' ) ) {
    hfv.mandatory.push( msgs[i].id );
  }
}
```

9.7.5 自定义属性方法

实用的自定义属性方法是由Peter-Paul Koch (<http://www.quirksmode.org>) 于2005年2月份在A List Apart (<http://www.alistapart.com/articles/scripttriggers/>) 上发表的文章*JavaScript Triggers*中提出来的。我们这里不会再去分析例子了，但它意味着你需要使用元素非标准的属性，如：

```
<input type="text" name="name" id="name" required="true" />
```

在脚本中可以通过getAttribute()方法检查这些属性并做出相应的处理。甚至可以将这种思想扩展到在HTML中考虑验证规则的可维护性：

```
<input type="text" name="name" id="name" required="true"
error="Please enter a Name at least 5 characters long"
pattern="/.{5}/" />
```

这种方法的主要问题是彻底改变了HTML，为此你必须为它自定义一个DTD。David Eisenberg 在*Validating a Custom DTD* (<http://www.alistapart.com/articles/customdtd/>) 中阐述了如何实现它。

9.7.6 这些方法的缺点

所有的这些方法都有一个共同的问题：把验证规则分别放到了HTML和JavaScript中以及后台的脚本中，这意味着需要双倍的维护。而且，除了隐藏字段的其他所有方法，都使得HTML必须配合JavaScript使用，因为在后台脚本中是无法通过类或SPAN来确定哪些字段是需要强制检查的。

9.7.7 共用验证规则

一种更巧妙的方法是把验证规则和反馈信息放在同一个地方。可以使用XML保存这些信息，并使用XHR来获取它，但这很容易产生错误。一种更简单的解决方案是使用JSON：

```
validationRules : {
  'Name' : {
    'error' : 'Please enter a name',
    'pattern' : /.{10}/i
  },
  'subject' : {
    'error' : 'Please choose a subject',
    'pattern' : /.{5}/i
  },
  'Message' : {
```

```

    'error' : 'Please enter a message at least 20 characters long',
    'pattern' : '/.{20}/i
},
'email' : {
    'error' : 'Please enter a valid email',
    'pattern' : '/^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/i
}
}
}

```

可以通过循环遍历JSON对象来获取所有强制元素：

```

for( i in validationRules ) {
    // i will be Name, subject, Message, email
    if( !document.getElementById( i ) ) { continue; }
    toplist.checkValue( i );
}

```

获取模式和错误信息也很容易；`alert(validationRules['email']['error'])`就会得到“Please enter a valid Email”这样的警告陈述。

在后台也可以生成JSON字符串。例如，在PHP语言中：

`validationRules.php`

```

<?php
$validationRules = array (
    'Name' => array (
        'error' => 'Please enter a name',
        'pattern' => '/.{10}/i'
    ),
    'subject' => array (
        'error' => 'Please choose a subject',
        'pattern' => '/.{10}/i'
    ),
    'Message' => array (
        'error' => 'Please enter a message at least 20 characters long',
        'pattern' => '/.{20}/i'
    ),
    'email' => array (
        'error' => 'Please enter a valid email',
        'pattern' => '/^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]{2,7}$/i'
    )
);
if( isset( $_GET['json'] ) ) {
    header( 'Content-type:text/javascript' );
    echo 'validationRules = {';
    foreach( array_keys( $validationRules ) as $a ) {
        echo '\'' . $a . '\'' : {' . "\n";
        foreach( array_keys( $validationRules[$a] ) as $b ) {
            if( $b == 'pattern' ) {
                echo '\'' . $b . '\'' : ' . $validationRules[$a][$b].';
            }
            ',' . "\n";
        }
    }
}

```

```

        } else {
            echo '\'' . $b . '\'' : '\'' . $validationRules[$a][$b].➥
                '\'' . "\n";
        }
    }
echo '},' . "\n";
}
echo '}';
?

```

如果对它的语法不太熟悉的话，也不要着急；重要的是要知道如果一个不同的PHP脚本包含了这段脚本，它就会得到PHP数组形式的验证规则；但是如果使用一个JavaScript参数json包含这段PHP脚本，你会得到一个可以用来做其他处理的validationRules对象。这就是粗体部分通过把一个text/javascript头部发送到浏览器上并以对应的格式输出数组数据所做的事情。如果没有text/javascript头部，浏览器将无法识别出从PHP作为javascript返回的字符串，因此你可以看到，它是使用服务器端语言输出JavaScript的一个非常重要的方面。

使用这个功能非常容易，在文档的头部将PHP脚本作为JavaScript包含进来就可以了：

```
<script type="text/javascript" src="validationRules.php?json=1"></script>
```

在关于如何给用户确认反馈信息的第一个例子中，我们很快就会用到它。

9.8 为用户反馈验证信息

在第4章和第7章中我们已经讲过了一些为用户提供反馈信息的方法，现在我们来详细看一个表单验证示例，并用其他一些选项列出对该脚本所做的必要修改。

我们不会再重复最简单、也可能是最古老的方法：JavaScript警告，因为前面已经讲过了，而且现在你应该已经知道如何使用它了。下面，我们会看一些更为灵巧的例子。

9.8.1 显示错误字段的列表

通过这个方法，可以给用户显示出那些包含错误的字段的列表。这是后台脚本所常用的反馈方法，它会在网页完成加载的时候在表单上显示此列表。由于通过JavaScript验证不需要重新加载页面，因此最好在用户所在的地方——提交按钮的上方、表单的下方显示此列表。图9-1展示了它的显示效果，也可以在本地机器上打开exampleTopList.html来看一下这个示例，本地机器必须是运行着PHP的服务器，如XAMPP或Apache，和我们在第8章中讲过的一样。

开始先将验证规则嵌入到HTML的标题当中：

exampleTopList.html（摘录）

```
<script type="text/javascript" src="validationRules.php?json=1"></script>
```

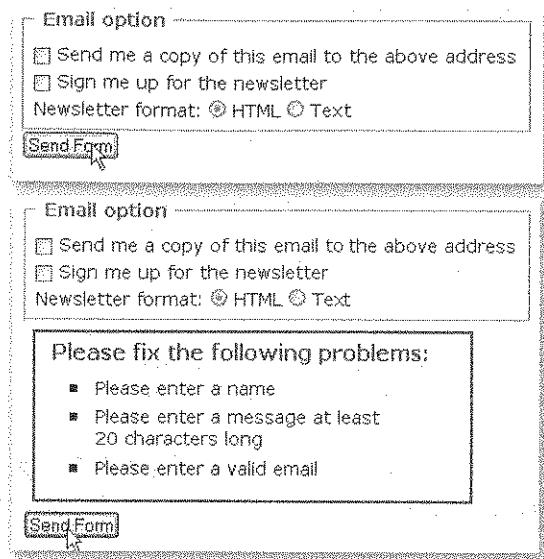


图9-1 在提交按钮上方显示错误字段的列表

设计脚本的时候，应该先从属性和方法的大纲开始：

`topList.js (大纲)`

```
topList = {
  error:[],
  errorMessage : null,
  errorClass : 'error',
  errorTitle : 'Please fix the following problems:',
  sendButtonID : 'send',
  init:function(){},
  send:function( e ){},
  flushErrors:function(){},
  checkValue:function( o ){ }
}
```

需要一个名为`error`的数组，保存表单中发生的所有错误。预先定义`errorMessage`为`null`，这将是显示在提交按钮上方的消息。`errorTitle`会显示在错误列表的上方，`errorClass`用于定义界面风格。`sendButtonID`是查找提交按钮所必需的。

表单提交脚本的方法都是非常标准的：一个用来在提交表单的时候应用事件处理程序的`init()`方法、一个事件监听器的`send()`方法、一个用于移除旧的错误信息的方法（用户不会重新加载页面，而可能是输入了好几次数据，因为在发生错误的时候，脚本会阻止页面的重新加载）以及一个用于检查每个强制表单域的值。

下面我会一步一步地分析一下这段脚本，并且解释一下这些不同方法的作用。

```
toplist.js

toplism = {
    error:[],
    errorMessage : null,
    errorClass : 'error',
    errorTitle : 'Please fix the following problems:',
    sendButtonID : 'send',
    init:function() {
        toplist.sendButton = document.getElementById( toplist.sendButtonID );
        if( !toplism.sendButton ) { return; }
        toplist.f = document.getElementsByName( 'form' )[0];
        DOMhelp.addEvent( toplist.f, 'submit', toplist.send, false );
    },
},
```

在init()方法中没有什么出奇的地方：检查提交按钮是否存在，在文档中获取第一个表单，并且应用一个指向send的提交处理器。

```
toplism.js (续)

send:function( e ) {
    toplist.flushErrors();
    for( var i in validationRules ) {
        if( !document.getElementById( i ) ) { continue; }
        toplist.checkValue( i );
    }
}
```

调用flushErrors()方法来移除所有可能已经显示了的错误信息，并且循环遍历JSON对象validationRules。判断每一个强制元素是否存在，并且把元素的ID作为参数调用checkValue()方法。

toplism.js (续)

```
if( toplist.error.length > 0 ) {
    toplist.errorMessage = document.createElement( 'div' );
    toplist.errorMessage.className = toplist.errorClass;
    var errorTitle = document.createElement( 'h2' );
    errorTitle.appendChild( document.createTextNode( toplist.errorMessage ) );
    toplist.errorMessage.appendChild( errorTitle );
    entry = document.createElement( 'ul' );
    toplist.errorMessage.appendChild( entry );
    toplist.errorList = entry;
```

如果其中的一些字段存在问题的话，实用方法checkValue()会把新的元素添加到错误数组中，这就是为什么在统计错误信息之前检查它的length是否大于0的原因。

接下来，创建一个DIV元素，把它保存在errorMessage属性当中，然后给它应用一个error类使它可以设置样式。创建一个H2，并将errorTitle中储存的文本作为标题的内容。将标题添加到errorMessage层中、创建一个新的UL元素，并把这个UL添加到errorMessage当中，然后把新创建

的UL元素保存到errorList属性当中。

toplisp.js (续)

```
for( i = 0; i < toplist.error.length; i++ ) {
    entry = document.createElement( 'li' );
    entry.appendChild( document.createTextNode( toplist.error[i] ) );
    toplist.errorList.appendChild( entry );
}
```

循环遍历error数组，创建一个包含每个错误文本的新列表项，然后将其作为子节点添加到errorList UL中。

toplisp.js (续)

```
var sendPara = toplist.sendButton.parentNode;
sendPara.parentNode.insertBefore( toplist.errorMessage, sendPara );
DOMhelp.cancelClick( e );
},
},
```

现在，获取表单中发送 (send) 按钮的父元素，然后将错误信息作为新的兄弟元素在其前面插入。如果出现错误，则通过调用cancelClick()取消表单提交。

toplisp.js (续)

```
flushErrors : function() {
    toplist.error = [];
    if( toplist.errorMessage ) {
        toplist.errorMessage.parentNode.removeChild( toplist.errorMessage );
        toplist.errorMessage = null;
    }
},
```

flushErrors()方法重新定义error数组为一个空数组，并且检查是否已经存在一个errorMessage。如果有的话，则移除文档的节点并将这个对象设置为null。

toplisp.js (续)

```
checkValue : function( o ) {
    var elm = document.getElementById( o );
    switch( elm.type ) {
        case 'text':
            if( !validationRules[o][ 'pattern' ].test( elm.value ) ) {
                toplist.error.push( validationRules[o][ 'error' ] );
            }
        break;
        case 'textarea':
            if( !validationRules[o][ 'pattern' ].test( elm.value ) ) {
```

```

        toplist.error.push( validationRules[o][ 'error' ] );
    }
    break;
}

```

checkValue()方法会完成真正的验证任务，首先根据作为参数发送的ID获取需要验证的元素。这里不需要检查这个元素是否存在，因为在send方法中已经检查过了。重新获取元素的type，并使用一个switch语句，用来对不同的表单元素做相应的测试。

在type是text和textarea的情况下，获取该元素的value属性，并再次检验它是否与存储在validationRules对象中的模式匹配。如果这个模式不匹配的话，则把一个新项添加到error数组中，其中的错误信息与作为值的该元素ID相关联。

注解 参数o是这个元素的ID，这意味着如果你使用validationRules['Name']['pattern']，可能会得到这样的结果/.{10}/i，它是一个测试名字是否至少为10个字符的正则表达式模式。

如果使用validationRules['Name']['error']的话，会得到这样的结果'Please enter a name'，它就是需要添加到error数组新项中的错误信息。

toplist.js（续）

```

case 'select-one' :
    var curelm = elm.options[ elm.selectedIndex ].value;
    if( elm.selectedIndex == 5 ) {
        curelm = document.getElementById( 'otherSubject' ).value;
    }
    if( !validationRules[o][ 'pattern' ].test( curelm ) ) {
        toplist.error.push( validationRules[o][ 'error' ] );
    }
    break;
}
}
DOMhelp.addEvent( window, 'load', toplist.init, false );

```

在类型为选择框的情况下——本例中是一个单选的选择框，所以类型为select-one，需要通过options数组和selectedIndex属性获取所选择的选项的值。由于表单提供了一个“其他”选项，因此需要检查其是否被选中，如果是的话，则获取otherSubject文本字段的值。

选中的值和其他Subject文本字段的值都需要与存储在validationRules对象中的模式进行匹配，如果不匹配的话，则添加一条错误记录。为window对象添加一个事件处理程序，以在窗体完成加载的时候执行init()方法。到这里，你已经创建了一个通过JavaScript进行验证的表单，并且会显示错误的消息列表，直到用户在所有的强制字段中都输入了正确的值。

9.8.2 使用可单击的错误消息代替主表单

另一种方法是当验证出现错误的时候隐藏表单，并且显示一个能够链接回表单的警告信息，如图9-2所示的那样。可以在本机上打开exampleHideForm.html自己看一下这个例子。

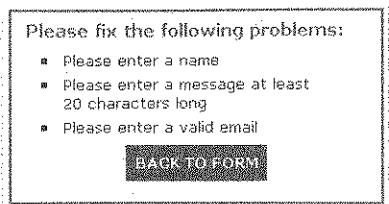


图9-2 隐藏表单并提供返回链接

它们的区别并不多：

在exampleHideForm.html中使用的hideform.js

```
toplist = {
  error:[],
  errorMessage : null,
  errorClass : 'error',
  errorTitle : 'Please fix the following problems:',
  errorLink : 'Back to form',
  errorLinkClass : 'errorlink',
  init:function() {
    [... code snipped ...]
  },
}
```

需要两个新属性：一个用来存储链接文本；另一个用来定义一个CSS类，它用来为关闭这个消息的链接应用样式。

hideform.js（续）

```
send:function( e ) {
  toplist.flushErrors();
  for( var i in toplist.validationRules ) {
    [... code snipped ...]
  }
  if( toplist.error.length > 0 ) {
    [... code snipped ...]
    for( i = 0; i < toplist.error.length; i++ ) {
      [... code snipped ...]
    }
    entry = document.createElement( 'li' );
    var closeLink = DOMhelp.createLink( '#', toplist.errorLink );
    DOMhelp.addEvent( closeLink, 'click', toplist.flushErrors, false );
    closeLink.onclick = DOMhelp.safariClickFix;
    entry.appendChild( closeLink );
    entry.className = toplist.errorLinkClass;
    toplist.errorList.appendChild( entry );
    toplist.f.style.display = 'none';
  }
}
```

```

        toplist.f.parentNode.insertBefore( toplist.errorMessage, toplist.f );
        DOMhelp.cancelClick( e );
    }
},

```

在send()方法中，需要再添加一个额外的列表项并创建一个把errorLink属性作为文本的链接。应用一个指向flushErrors()方法的单击事件处理程序、添加CSS类，并把这个新创建的列表项添加到errorList上。通过设置样式显示属性为none，可以隐藏主表单（也可以添加一个动态的CSS类来隐藏表单）。

hideform.js (续)

```

flushErrors : function() {
    toplist.error = [];
    if( toplist.errorMessage ) {
        toplist.errorMessage.parentNode.➡
        removeChild( toplist.errorMessage );
        toplist.errorMessage = null;
    }
    toplist.f.style.display = 'block';
},
[... code snipped ...]
DOMhelp.addEvent( window, 'load', toplist.init, false );

```

现在flushErrors()方法需要将表单的display属性设置回block，这就是我们所要进行的更改，其他部分代码不变。

9.8.3 单独地突出显示错误的字段

尤其是对于那些比较大的表单，在出现错误字段的旁边直接显示出错误信息是非常有意义的。图9-3展示了显示的效果，也可以在本机上打开exampleErrorFields.html自己看一下。

The screenshot shows a web form with several input fields and error messages displayed in red boxes:

- Email Input:** "Please enter a valid email"
- Message Subject Input:** "Your message
Subject*: General question" (with a red box around the subject field)
- Message Content Input:** "specify other subject" (with a red box around the input field)
- Message Body Input:** "Your Message*
Please enter a message at least 20 characters long" (with a red box around the input field)
- Newsletter Options:** "Send me a copy of this email" and "Sign me up for the newsletter" (checkboxes)
- Newsletter Format:** "Newsletter format: HTML Text"
- Final Error Message:** "Please fix the marked issues"
- Send Form Button:** "Send Form"

图9-3 单独地突出显示错误的字段

这个脚本就略有不同了，send、flushErrors()和checkValue()方法都必须修改，而且还需要一个新的方法addErrorMessage()，它会在发生错误的地方添加错误信息。

errorFields.js

```
ef = {
  error:[],
  errorMessage : null,
  errorClass : 'error',
  errorTitle : 'Please fix the marked issues',
  init : function() {
    ef.sendButton = document.getElementById( 'send' );
    if( !ef.sendButton ) { return; }
    ef.f = document.getElementsByTagName( 'form' )[0];
    DOMhelp.addEvent( ef.f, 'submit', ef.send, false );
    ef.f.onsubmit = function() { return false; }
  },
};
```

属性和init()方法与toplits示例中的仍然是一样的。

errorFields.js (续)

```
send:function( e ) {
  ef.flushErrors();
  for( var i in validationRules ) {
    if( !document.getElementById( i ) ) { continue; }
    ef.checkValue( i );
  }
  if( ef.error.length > 0 ) {
    ef.errorMessage = document.createElement( 'div' );
    ef.errorMessage.className = ef.errorClass;
    var errorTitle = document.createElement( 'h2' );
    errorTitle.appendChild( document.createTextNode( ef.errorTitle ) );
    ef.errorMessage.appendChild( errorTitle );
    var sendPara = ef.sendButton.parentNode;
    sendPara.parentNode.insertBefore( ef.errorMessage, sendPara );
    DOMhelp.cancelClick( e );
  }
},
```

发送方法简单了许多，因为不需要创建错误列表了，只在errorMessage主元素中创建标题就可以了。

errorFields.js (续)

```
checkValue : function( o ) {
  var elm = document.getElementById( o );
  switch( elm.type ) {
    case 'text' :
      if( !validationRules[o][ 'pattern' ].test( elm.value ) ) {
        ef.error.push( validationRules[o][ 'error' ] );
```

```

        ef.addErrorMsg( elm, validationRules[o][ 'error' ] );
    }
    break;
    case 'textarea' :
        if( !validationRules[o][ 'pattern' ].test( elm.value ) ) {
            ef.error.push( validationRules[o][ 'error' ] );
            ef.addErrorMsg( elm, validationRules[o][ 'error' ] );
        }
    break;
    case 'select-one' :
        var curelm = elm.options[elm.selectedIndex].value;
        if( elm.selectedIndex == 5 ) {
            curelm = document.getElementById( 'otherSubject' ).value;
        }
        if( !validationRules[o]['pattern'].test( curelm ) ) {
            ef.error.push( validationRules[o][ 'error' ] );
            ef.addErrorMsg( elm, validationRules[o][ 'error' ] );
        }
    break;
},
),

```

checkValue()方法所做的更改是，除了将错误添加到error数组当中外，还把出现错误的元素和错误文本作为参数，调用了addErrorMsg()方法。

errorFields.js (续)

```

addErrorMsg : function( o, msg ) {
    var errorMsg = document.createElement( 'span' );
    errorMsg.className = ef.errorClass;
    errorMsg.appendChild( document.createTextNode( msg ) );
    o.parentNode.insertBefore( errorMsg, o );
},

```

addErrorMsg方法创建了一个新的SPAN元素，添加错误类以使它可以应用样式，并将错误文本作为文本节点添加到其中。接下来在出现错误的元素前插入新创建的SPAN。

errorFields.js (续)

```

flushErrors:function() {
    var elm;
    ef.error = [];
    if( ef.errorMessage ) {
        ef.errorMessage.parentNode.removeChild( ef.errorMessage );
        ef.errorMessage = null;
    }
    for( var i in validationRules ) {
        elm = document.getElementById( i );
        if( !elm ) { continue; }
        if( elm.previousSibling &&

```

```

        elm.previousSibling.nodeName.toLowerCase() == 'span' &&
        elm.previousSibling.className == ef.errorClass ) {
    elm.parentNode.removeChild( elm.previousSibling );
}
}
}
}

DOMhelp.addEvent(window, 'load', ef.init, false );

```

这些更改意味着不是使用单一的新元素来显示所有错误信息，而是在每个错误的字段都有。所以flushErrors()方法需要循环遍历所有的强制域，检查其前一个兄弟节点是否为SPAN并且应用了对应的类。如果所有的这些条件都为真，则移除这个节点。

9.8.4 即时验证反馈

前面所有的例子都是在表单提交时进行验证的，正如前面章节中所提到的，它是验证表单可访问性最好的方式。然而，在不改变提交表单时进行验证的同时，使得表单交互性更强并没有什么坏处。如果用户能够即时发现并改正错误，那么在提交的时候就不会引起错误了。示例exampleDynamicFields.html会在更改之后立即验证字段，你可以在本机上测试一下。

这个脚本需要一个新的方法sendField()，以及init()方法里的一些细微修改：

dynamicFields.js

```

df = {
  error : [],
  errorMessage : null,
  errorClass : 'error',
  errorTitle : 'Please fix the marked issues',
  init : function() {
    var elm;
    df.sendButton = document.getElementById( 'send' );
    if( !df.sendButton ) { return; }
    df.f = document.getElementsByTagName( 'form' )[0];
    DOMhelp.addEvent( df.f, 'submit', df.send, false );
    df.f.onsubmit = function() { return false; }
    for( var i in validationRules ) {
      elm = document.getElementById( i );
      if( !elm ){ continue; }
      DOMhelp.addEvent( elm, 'blur', df.sendField, false );
    }
  },
}

```

除了表单的submit处理器，还需要循环遍历所有的强制字段，把blur处理器作为事件监听器添加到指向sendField()的字段。如果你想要在某个字段发生改变时而不是用户改完之后提供反馈，那么也可以使用change事件处理程序。

```
dynamicFields.js (续)

sendField : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.previousSibling &&
        t.previousSibling.nodeName.toLowerCase() == 'span' &&
        t.previousSibling.className == df.errorClass ) {
        t.parentNode.removeChild( t.previousSibling );
    }
    df.checkValue( t.id );
},
}
```

`sendField()`方法需要通过`getTarget()`获取当前的元素。接着像`flushErrors()`所做的那样移除所有可能存在的错误信息，并且把这个元素ID作为参数调用`checkValue()`。这就是它所有的东西了。

```
dynamicFields.js (续)

send : function( e ) {
    [... code snipped ...]
},
flushErrors : function() {
    [... code snipped ...]
},
checkValue : function( o ) {
    [... code snipped ...]
},
addErrorMsg : function( o, msg ) {
    [... code snipped ...]
}
}
DOMhelp.addEvent( window, 'load', df.init, false );
```

代码的其他部分仍然保留不变。

9.9 其他的动态验证方法

当用户进行修改的时候，立即对每个字段进行验证是非常不错的，而且通过Ajax和一些适当的后台数据集或函数可以很好地完成这项任务。一个非常好的例子就是在输入表单的时候，建议一下哪些数据是有效的。

Google Suggest (<http://www.google.com/webhp?complete=1>) 就是Web表单中最早这样做的一个示例。它为你提供其他用户所做的搜索以及搜索结果的数量，如图9-4所示。

这种提供建议的方法并不难做，而且你可以使用前面章节的知识来创建一个提供建议的表单元素。示例`exampleContactSuggest.html`就是这些做的。它使用Ajax读取一个包含联系人名称的XML文件，当用户键入的时候与用户的输入进行对比。图9-5展示了一种可能的结果。

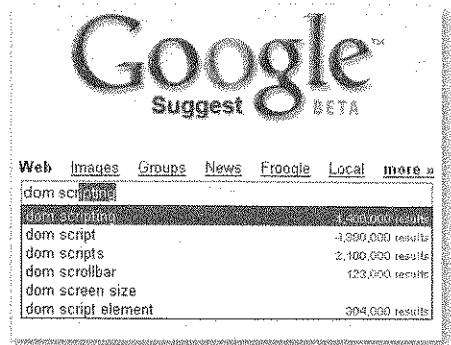


图9-4 Google Suggest在输入的时候显示可能的结果

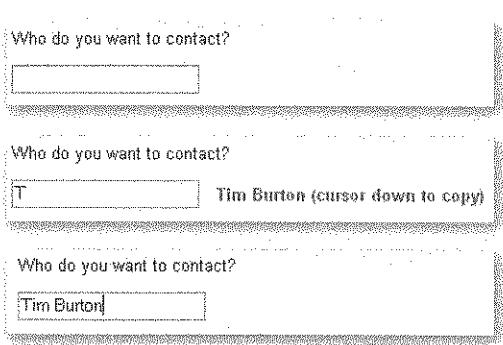


图9-5 从XML数据集中动态地提供数据

这个XML数据是非常简单的：

contacts.xml

```
<?xml version="1.0" encoding="utf-8"?>
<contacts>
    <name>Bill Gates</name>
    <name>Linus Torvalds</name>
    <name>Douglas Coupland</name>
    <name>Ridley Scott</name>
    <name>George Lucas</name>
    <name>Dan Akroyd</name>
    <name>Sigourney Weaver</name>
    <name>Tim Burton</name>
    <name>Katie Jane Garside</name>
    <name>Winona Ryder</name>
</contacts>
```

在真实的示例当中，你可能希望在每个联系人名称周围添加一个contact元素，以便添加一些其他信息，如Email、电话和单位。

这个脚本使用了上一章中所见过的XHR以及正则表达式：

contactSuggest.js

```
cs = {
    init : function() {
        if( !document.getElementById || !document.createTextNode ) {
            return;
        }
        cs.f = document.getElementById( 'contact' );
        if( !cs.f ) { return; }
        cs.output = document.createElement( 'span' );
        cs.f.parentNode.insertBefore( cs.output, cs.f.nextSibling )
        DOMhelp.addEvent( cs.f, 'keyup', cs.check, false );
    },
};
```

首先检查是否支持DOM以及是否有一个ID为contact的字段存在。如果两者都通过，创建一个新的SPAN元素，并将其插入在ID为contact的元素后面。在keyup上添加事件处理程序，以触发事件监听方法check()。

```
contactSuggest.js (续)

check : function( e ) {
    if( window.event ) {
        var key = window.event.keyCode;
    } else if( e ) {
        var key = e.keyCode;
    }
    if( key == 8 ) { return; }
    if( key == 40 ) {
        cs.f.value = cs.output.innerHTML.replace( /\s\(.*\)\$/ , '' );
        cs.output.innerHTML = '';
        return;
    }
    cs.doxhr( 'contacts.xml' );
},
```

这个方法会查找按的是哪个键，并且在按键为Backspace(8)的时候会返回。如果用户按下的 是下箭头键(40)，那么从SPAN的内容当中移除所有以空格开头并以圆括号结尾的东西，并将其复 制到表单字段当中。接下来，它会删除SPAN的内容并返回。如果用户按下的 是其他键，那么check() 方法会把contacts.xml作为要加载的URI来调用doxhr()方法。

```
contactSuggest.js (续)

doxhr : function( url ) {
    var request;
    try{
        request = new XMLHttpRequest();
    } catch ( error ){
        try{
            request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch ( error ) {
            return true;
        }
    }
    request.open( 'get', url, true );
    request.onreadystatechange = function() {
        if( request.readyState == 4 ) {
            if (request.status && /200|304/.test( request.status ) ) {
                cs.retrieved( request );
            } else {
                cs.failed( request );
            }
        }
    }
}
```

```

request.setRequestHeader( 'If-Modified-Since', =>
    'Wed, 05 Apr 2006 00:00:00 GMT' );
request.send( null );
return false;
},

```

xhr()方法没有什么变化；创建一个XMLHttpRequest，并加载作为参数传过来的URI。

contactSuggest.js (续)

```

retrieved : function( requester ) {
    var v;
    var pattern = new RegExp( '^'+cs.f.value, 'i' );
    var data = requester.responseXML;
    var names = data.getElementsByTagName( 'name' );
    for( var i = 0; i < names.length; i++ ) {
        v = names[i].firstChild.nodeValue;
        if( pattern.test( v ) ) {
            cs.output.innerHTML = v + ' (cursor down to copy)';
            break;
        }
    }
},

```

retrieved方法创建了一个正则表达式，它会检查是否有以contact字段的值为开头的东西。从responseXML中读取contacts.xml的内容，使用getElementsByTagName()方法获取所有的名字，并且循环遍历它们。如果遇到匹配模式的名称，那么在SPAN中写出该名称的文本内容，后面紧跟消息“(cursor down to copy)”，并停止循环。

contactSuggest.js (续)

```

failed : function( requester ) {
    alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
    return true;
}
}
DOMhelp.addEvent( window, 'load', cs.init, false );

```

如果出现了错误，用户会通过警告得到通知。当窗口完成加载文档后，cs.init()方法就会被调用。

对于少量的联系人，使用XHR可能就没有必要了，而且最好使用JSON格式获取数据，这样也可以减少每次用户击键时来回服务器端的开销。不过，这个例子也可以使用数据库来获取相应的联系人。

9.10 小结

希望在学完这章之后，你可以游刃有余地编写表单验证脚本以及正则表达式。这并不难，但

正如前面所说的那样，因为它太容易了所以不能很容易地进行验证或假设太多的条件。

表单验证既是一个实用性的问题，也是一个技术问题。必须确保所做的表单验证在你所处理的技术环境中是有意义的，并且解决方案符合产品的规程。目前许多在线的表单都必须遵守可用性原则或是一些预定的规则（例如，在地方政府网站上工作的时候）。如果你的验证机制需要用户注意表单或是使用鼠标，那么你就不能满足这些原则。

目前人们喜欢最流行的方法，并且使得网站上的所有东西都是动态的。要注意你所面对用户的一些限制和使用方式，确保在不使用JavaScript的情况下一切都能正常运行，这样会使你和客户都很愉快。

第10章将介绍一个更大的项目，创建一个有后台支持的动态图库，并通过CSS、JavaScript和Ajax使其变得丰富多彩。

现代的JavaScript案例研究： 动态图库

在本章中，你会学到如何使用PHP脚本来开发一个JavaScript效果增强的缩略图图库。作为开始，我们会先学习一下静态图库的技术和如何使用PHP和Ajax增强它们的效果，并在服务器上把这些图片动态地放到图库中。

注解 可以下载本章的示例代码或者在网站<http://www.beginningjavascript.com>上在线地看一下结果。由于本章包含了图片库，因此下载会占用比较多的时间，但是它可以使你在本地服务器上看到所有的代码，包括服务器端的PHP脚本。

10.1 缩略图图库基础

我们首先看一下基本原理，并规划设计一下我们的缩略图图库。我考虑了很长一段时间是否在本书中应该包含这样的一个实例，因为对于JavaScript和CSS书籍，使用库图作为示例几乎已经变得很老旧了。然而，我编写这一章是为了给你这样一个示例，教你如何给非常普通的解决方案，如缩略图图库，融入现代的编程和CSS，并且仍然不依赖于它们两个。许多示例，尤其是只使用CSS的图库，在现代的浏览器里看上去效果都非常好；然而，它们无法向下兼容，而且无法真正表达出缩略图图库应该表达的内容。

10.2 缩略图图库是什么以及它应该做什么

缩略图图库的思想要追溯到浏览器刚开始支持图片的时候，当时网络的连接速度可以用千字节衡量。这样的一个图库的任务过去是和现在仍然是，通过为图库中的每张图片提供一个小一些的预览图片，来给出提供的照片的一个概括。“小一些”意思是尺寸大小上更小一些，还有很重要的一点是文件的大小也更小一些。这意味着，一个只对你的图库中的某一张图片感兴趣的用户不需要下载所有的图片，只需要下载他感兴趣的一张就可以了，这样既节省了他的时间，还减轻了你的服务器的通信量。许多只使用CSS或JavaScript/HTML的缩略图图库都没有达到这样的目

的，而且假设每个用户都需要下载许多东西来看一张图片。可以提供下载所有的图片，但是它应该是一个可选的，而不是必需的。最糟糕的缩略图图库通过HTML属性或CSS把照片大小调整为缩略图，从而强迫访问者下载所有的大图片以把它们看作低质量的缩略图。通过使用JavaScript改变它们在CSS中的尺寸或者通过使用HTML属性调整图片的大小并不会生成高质量的缩略图；这是一种比较偷懒的错误做法。

如果你想提供和原始图效果一样的缩略图图库，那么需要为你显示的大图片生成小一点的缩略图图片。可以通过在上传图库前做一个批处理或者在服务器上使用脚本来快速实现。

提示 有许多可以使用的缩略图生成和批量生成的工具。比较好的（最重要的是免费的）是Google的Picasa（提供在<http://picasa.google.com/>）和IrfanView（提供在<http://www.irfanview.com/>）。在服务器上生成缩略图可以很容易地通过PHP和GD库函数来实现。我已经写了一篇有关如何来做的文章，放在<http://icant.co.uk/articles/phpthumbnails/>上了，而且有一个主要的预先做好的PHP类叫做phpThumb()，可以在网站<http://phpthumb.sourceforge.net/>上看到。由于这是一本有关JavaScript的书籍，因此我不会深入讲解通过PHP来进行图片生成的细节，尽管它对于在线图库是非常方便的。

10.3 静态缩略图图库

传统的缩略图图库是在一个表或列表中提供小缩略图图片的。每一个缩略图链接到一个包含大图片的页面，这个页面再链接到缩略图图库或提供前一张和后一张图片的链接。

如果存在许多图片，那么缩略图页面可能就需要分页，每次显示一定数量的缩略图并且在整个集合中提供向前和向后的导航。使用纯静态的图库，意味着你必须生成所有的缩略图页面，而且每张图片一个，开始的时候需要大量的工作，而且在图库每次更新的时候都需要把许多文件传到服务器上。

10.4 使用JavaScript模拟动态图库

可以使用JavaScript通过在所有的缩略图上应用事件处理器，把一个静态的缩略图图库转变为一个表面上看去动态的图库。当缩略图被单击的时候，用一个包含大图的新元素覆盖缩略图。通过把缩略图链接到大图并且在浏览器中简单地显示，可以使非JavaScript用户也能访问。

exampleFakeDynamic.html（摘要）

```
<ul id="thumbs">
<li>
  <a href="galleries/animals/dog2.jpg">
    
  </a>
</li>
```

```

<li>
  <a href="galleries/animals/dog3.jpg">
    
  </a>
</li>
<li>
  <a href="galleries/animals/dog4.jpg">
    
  </a>
</li>
[... more thumbnails ...]
</ul>

```

提示 最好还是为缩略图图库使用表或定义列表，因为表可以更好地分解布局，即使在非CSS的浏览器中，它们仍然可以保留多列的结构；定义列表语义上也是正确的。对于本章中的这些例子，我使用了一个简单的列表来使其比较容易理解，而且允许缩略图在屏幕上占据尽可能多的可利用空间。

可以通过打开示例exampleFakeDynamic.html测试一下效果。我们来逐步地看一下这个功能性，开始是脚本的大纲：

fakeDynamic.js (大纲)

```

fakegal = {
  // IDs
  thumbsListID : 'thumbs',
  largeContainerID : 'photo',
  // CSS classes
  closeClass : 'close',
  nextClass : 'next',
  prevClass : 'prev',
  hideClass : 'hide',
  showClass : 'show',
  // Labels
  closeLabel : 'close',
  prevContent : '',
  nextContent : '',

  init : function(){ },
  createContainer : function(){},
  showPic : function(e){ },
  setPic : function(pic){ },
  navPic : function(e){ }
}
DOMhelp.addEvent( window, 'load', fakegal.init, false );

```

你需要：

- 包含所有缩略图的元素的ID；

- 赋给大图片容器的ID；
- 链接的CSS类以移除大图片；
- 在大图片之间进行导航的链接；
- 显示和隐藏元素的类；
- 一个标签，以告诉用户这个链接会隐藏大图片；
- 下一个和上一个图片链接的标签。

在方法方面，你需要：

- 一个方法用来初始化函数；
- 一个实用方法在最初的时候建立图片容器；
- 一个方法用来显示图片；
- 一个方法用来设置要被显示的图片；
- 一个方法用来导航到下一个或上一个图片。

设置要显示的图片的方法setPic()是必要的，因为显示的方法showPic()和导航的方法navPic()都会改变容器中的图片。

fakeDynamic.js

```
fakegal = {
  // IDs
  thumbsListID : 'thumbs',
  largeContainerID : 'photo',
  // CSS classes
  closeClass : 'close',
  nextClass : 'next',
  prevClass : 'prev',
  hideClass : 'hide',
  showClass : 'show',
  // Labels
  closeLabel : 'close',
  prevContent : '',
  nextContent : '',
  init:function() {
    if( !document.getElementById || !document.createTextNode ) {
      return;
    }
    fakegal.tlist = document.getElementById( fakegal.thumbsListID );
    if( !fakegal.tlist ){ return; }
    var thumbsLinks = fakegal.tlist.getElementsByTagName( 'a' );
    fakegal.all = thumbsLinks.length;
    for(var i = 0 ; i < thumbsLinks.length; i++ ) {
      DOMhelp.addEvent( thumbsLinks[i], 'click', fakegal.showPic, false );
      thumbsLinks[i].onclick = DOMhelp.safariClickFix;
      thumbsLinks[i].i = i;
    }
  }
}
```

```
    fakegal.createContainer();
},
```

`init()`方法会检查是否支持DOM，并且获取包含缩略图的元素。它在把所有链接的数目存储到一个叫`all`的属性中（这对于稍候在最后一张图片上避免下一个链接是非常必要的）以后，接着循环遍历所有的链接。它接下来对缩略图列表中的每个链接应用一个指向`showPic()`的事件处理程序，并且在调用`createContainer()`以给文档添加必要的图片容器元素前，把它的索引数保存到一个叫`i`的新属性中。

`fakeDynamic.js` (续)

```
createContainer : function() {
    fakegal.c = document.createElement( 'div' );
    fakegal.c.id = fakegal.largeContainerID;
```

`createContainer()`方法开始先创建一个新的DIV元素，把它存储到一个叫`c`的属性中，并且把大图片容器的ID赋给它。

`fakeDynamic.js` (续)

```
var p = document.createElement( 'p' );
var cl = DOMhelp.createLink( '#', fakegal.closeLabel );
cl.className = fakegal.closeClass;
p.appendChild( cl );
DOMhelp.addEvent( cl, 'click', fakegal.setPic, false );
cl.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild(p);
```

创建一个新的段落，并在其中插入一个使用`closeLabel`作为文本内容的链接。为这个链接指派一个指向`setPic()`的事件处理程序，应用Safari补丁，并且把这个段落添加到容器元素中。

`fakeDynamic.js` (续)

```
var il = DOMhelp.createLink( '#', '' );
DOMhelp.addEvent( il, 'click', fakegal.setPic, false );
il.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild( il );
```

现在给这个容器添加另一个空链接，它有一个调用了`setPic()`的事件处理程序。这个链接在后面会包围大图片，使它可以单击，这样可能使键盘用户可以处理它。

`fakeDynamic.js` (续)

```
fakegal.next = DOMhelp.createLink( '#', '' );
fakegal.next.innerHTML = fakegal.nextContent;
fakegal.next.className = fakegal.nextClass;
DOMhelp.addEvent( fakegal.next, 'click', fakegal.navPic, false );
fakegal.next.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild( fakegal.next );
```

```

fakegal.prev = DOMhelp.createLink( '#', '' );
fakegal.prev.innerHTML = fakegal.prevContent;
fakegal.prev.className = fakegal.prevClass;
DOMhelp.addEvent( fakegal.prev, 'click', fakegal.navPic, false );
fakegal.prev.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild( fakegal.prev );

```

还需要添加两个链接以分别地显示上一个和下一个图片，它们两个都拥有指向navPic()的事件处理程序。

fakeDynamic.js (续)

```

fakegal.tlist.parentNode.appendChild( fakegal.c );
}

```

把这个新的容器添加到缩略图列表的父节点上，接下来就可以开始显示了。

fakeDynamic.js (续)

```

showPic : function( e ) {
  var t = DOMhelp.getTarget(e);
  if( t.nodeName.toLowerCase() != 'a' ) {
    t = t.parentNode;
  }
  fakegal.current = t.i;
  var largePic = t.getAttribute( 'href' );
  fakegal.setPic( largePic );
  DOMhelp.cancelClick( e );
},

```

在事件监听方法showPic()中，首先获取目标并且通过测试nodeName判断它是否为一个链接。接着把init()方法中赋给每个缩略图链接的i属性存储为一个新的属性current的值，它在主对象中告诉所有的其他方法当前显示的是哪一张图片。获取这个链接的href属性，并且在通过cancelClick()阻止浏览器打开链接前，使用这个href作为一个参数来调用setPic()方法。

fakeDynamic.js (续)

```

setPic : function( pic ) {
  var a;
  var picLink = fakegal.c.getElementsByTagName('a')[1];
  picLink.innerHTML = '';

```

setPic()方法获取图片容器中的第二个链接（它是结束链接后的链接），并且通过设置它的innerHTML属性为一个空的字符串，以移除这个链接可能有的任何内容。这对于避免一次显示多个图片是必需的。

fakeDynamic.js (续)

```

if( typeof pic == 'string' ) {
  fakegal.c.className = fakegal.showClass;
}

```

```

var i = document.createElement( 'img' );
i.setAttribute( 'src' , pic );
picLink.appendChild( i );

```

把参数pic的类型和string进行比较，因为这个方法可能使用（或不使用）一个URL作为参数而被调用。如果有一个参数是合法的字符串，则给这个容器添加应用show类来把它显示给用户，并且使用pic参数作为源来给它添加一张新图片。

fakeDynamic.js (续)

```

} else {
    fakegal.c.className = '';
}

```

如果没有一个string类型的参数，则移除这个图片容器的所有类，这样来隐藏它。

fakeDynamic.js (续)

```

a = fakegal.current == 0 ? 'add' : 'remove';
DOMhelp.cssjs( a, fakegal.prev, fakegal.hideClass );
a = fakegal.current == fakegal.all-1 ? 'add' : 'remove';
DOMhelp.cssjs( a, fakegal.next, fakegal.hideClass );
},

```

测试一下看看主对象的current属性是否等于0，如果是，则隐藏上一个图片的链接。对下一个图片链接做同样的处理，并且把current和所有缩略图的数量（保存在all中）进行比较。通过添加或移除隐藏类隐藏或显示每一个链接。

fakeDynamic.js (续)

```

navPic : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    var c = fakegal.current;
    if( t == fakegal.prev ) {
        c -= 1;
    } else {
        c += 1;
    }
    fakegal.current = c;
    var pic = fakegal.tlist.getElementsByTagName('a')[c];
    fakegal.setPic( pic.getAttribute( 'href' ) );
    DOMhelp.cancelclick( e );
}
}

DOMhelp.addEvent( window, 'load', fakegal.init, false );

```

获取被单击链接的引用（通过DOMhelp的getTarget()获取事件的目标并且确保nodeName为

A)，并且通过把这个节点和保存在prev属性中的节点进行比较，以判断这个链接是否为上一个链接。根据哪一个链接被激活了来对current加1或减1。接着使用新的当前链接的href属性作为参数调用setPic()，并且通过调用cancelClick()阻止浏览器打开被激活的链接。

所有剩下的就是添加一个样式表；结果如图10-1所示。

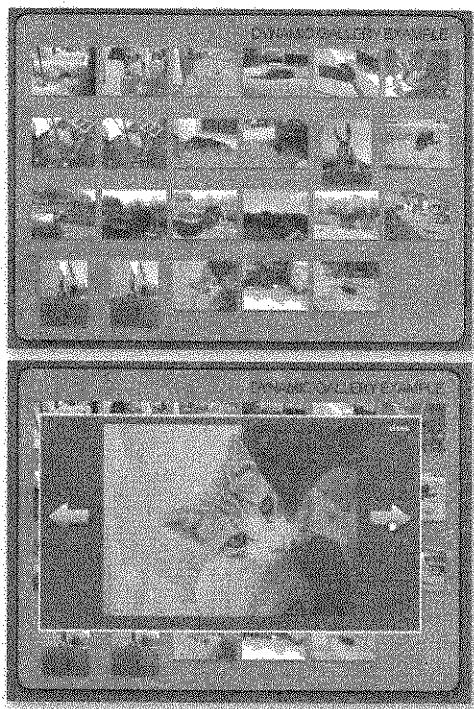


图10-1 使用JavaScript来模拟一个服务器控制的动态图库

10.5 显示标题

缩略图库都是可以看得见的结构，尽管如此，最好还是考虑一下可替换文本和图片标题。不仅要使你的图库对于盲人用户可以访问，而且它们考虑到在搜索引擎中搜索缩略图数据并对它进行索引。

许多的工具，像Google的Picasa一样，都考虑了动态的标题和可替换文本的添加。可以使用XHR来创建类似的东西，但由于这是一本关于JavaScript方面的书籍，而且如何把输入的数据存储到服务器上还需要一些解释，所以它不是一个相关的示例。作为替换的方案，我们来修改一下“模拟的”图库以让它可以显示标题。

使用图片的title属性作为这个图片的标题；这意味着静态的HTML需要合适的可替换文本和标题数据。

```
exampleFakeDynamicAlt.html (摘要)
<ul id="thumbs">
  <li>
    <a href="galleries/animals/dog2.jpg">
      
    </a>
  </li>
  <li>
    <a href="galleries/animals/dog3.jpg">
      
    </a>
  </li>
  [... More thumbnails ...]
</ul>
```

这段脚本本身并不需要修改太多；基本上所需要做的全部改变就是所生成的图片容器中的一个额外的段落以及把标题和可替换文本数据发送到大图片容器的方法改变。

fakeDynamicAlt.js

```
fakegal = {
  // IDs
  thumbsListID : 'thumbs',
  largeContainerID : 'photo',
  // CSS classes
  closeClass : 'close',
  nextClass : 'next',
  prevClass : 'prev',
  hideClass : 'hide',
  closeLabel : 'close',
  captionClass : 'caption',
  // Labels
  showClass : 'show',
  prevContent : '',
  nextContent : '',
```

第一个要修改的是修饰用的：添加一个新的会被应用到标题上的CSS类。

fakeDynamicAlt.js (续)

```
init : function() {
  if( !document.getElementById || !document.createTextNode ) {
    return;
  }
  fakegal.tlist = document.getElementById( fakegal.thumbsListID );
  if( !fakegal.tlist ) { return; }
  var thumbsLinks = fakegal.tlist.getElementsByTagName( 'a' );
```

```

fakegal.all = thumbsLinks.length;
for( var i = 0; i < thumbsLinks.length; i++ ) {
    DOMhelp.addEvent( thumbsLinks[i], 'click', -->
        fakegal.showPic, false );
    thumbsLinks[i].onclick = DOMhelp.safariClickFix;
    thumbsLinks[i].i = i;
}
fakegal.createContainer();
},
showPic : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    fakegal.current = t.i;
    var largePic = t.getAttribute( 'href' );
    var img = t.getElementsByTagName( 'img' )[0];
    var alternative = img.getAttribute( 'alt' );
    var caption = img.getAttribute('title');
    fakegal.setPic( largePic, caption, alternative );
    DOMhelp.cancelClick( e );
},

```

init()方法保持不变，但是showPic()方法除了读取链接的href属性外，还需要读取可替换文本和图片的title属性，并且把它们3个作为参数传递给setPic()。

fakeDynamicAlt.js（续）

```

setPic : function( pic, caption, alternative ) {
    var a;
    var picLink = fakegal.c.getElementsByTagName( 'a' )[1];
    pictLink.innerHTML = '';
    fakegal.caption.innerHTML = '';
    if( typeof pic == 'string' ) {
        fakegal.c.className = fakegal.showClass;
        var i = document.createElement( 'img' );
        i.setAttribute( 'src', pic );
        i.setAttribute( 'alt', alternative );
        picLink.appendChild( i );
    } else {
        fakegal.c.className = '';
    }
    a = fakegal.current == 0 ? 'add' : 'remove';
    DOMhelp.cssjs( a, fakegal.prev, fakegal.hideClass );
    a = fakegal.current == fakegal.all-1 ? 'add' : 'remove';
    DOMhelp.cssjs( a, fakegal.next, fakegal.hideClass );
    if( caption != '' ) {
        var ctext = document.createTextNode( caption );
        fakegal.caption.appendChild( ctext );
    }
},

```

`setPic()`方法现在有3个参数（而不是一个）——大图片的源、标题，以及可替换文本。这个方法需要删除所有可能已经是可见的标题，设置大图片的可替换文本属性，并且显示新的标题。

fakeDynamicAlt.js (续)

```
navPic : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    var c = fakegal.current;
    if( t == fakegal.prev ) {
        c -= 1;
    } else {
        c += 1;
    }
    fakegal.current = c;
    var pic = fakegal.tlist.getElementsByTagName( 'a' )[c];
    var img = pic.getElementsByTagName( 'img' )[0];
    var caption = img.getAttribute( 'title' );
    var alternative = img.getAttribute( 'alt' );
    fakegal.setPic( pic.getAttribute('href'), caption, alternative );
    DOMhelp.cancelClick( e );
},
```

`navPic()`方法和`init()`方法类似，需要获取大图片的可替换文本、标题以及源，并且把它们传递给`setPic()`。

fakeDynamicAlt.js (续)

```
createContainer : function() {
    fakegal.c = document.createElement( 'div' );
    fakegal.c.id = fakegal.largeContainerID;

    var p = document.createElement( 'p' );
    var cl = DOMhelp.createLink( '#', fakegal.closeLabel );
    cl.className = fakegal.closeClass;
    p.appendChild( cl );
    DOMhelp.addEvent( cl, 'click', fakegal.setPic, false );
    cl.onclick = DOMhelp.safariClickFix;
    fakegal.c.appendChild( p );

    var il = DOMhelp.createLink( '#', '' );
    DOMhelp.addEvent( il, 'click', fakegal.setPic, false );
    il.onclick = DOMhelp.safariClickFix;
    fakegal.c.appendChild( il );
    fakegal.next = DOMhelp.createLink( '#', '' );
    fakegal.next.innerHTML = fakegal.nextContent;
    fakegal.next.className = fakegal.nextClass;
    DOMhelp.addEvent( fakegal.next, 'click', fakegal.navPic, false );
```

```

fakegal.next.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild( fakegal.next );

fakegal.prev = DOMhelp.createLink( '#', '' );
fakegal.prev.innerHTML = fakegal.prevContent;
fakegal.prev.className = fakegal.prevClass;
DOMhelp.addEvent( fakegal.prev, 'click', fakegal.navPic, false );
fakegal.prev.onclick = DOMhelp.safariClickFix;
fakegal.c.appendChild( fakegal.prev );

fakegal.caption = document.createElement( 'p' );
fakegal.caption.className = fakegal.captionClass;
fakegal.c.appendChild( fakegal.caption );

fakegal.tlist.parentNode.appendChild( fakegal.c );
}
}

DOMhelp.addEvent( window, 'load', fakegal.init, false );

```

createContainer()方法只需要一个很小的修改，就是在容器中创建一个新的段落来存放标题。

正如你所看到的，使用JavaScript创建动态图库意味着生成许多HTML元素以及属性的读写。这样做是否值得由你来决定。当你需要提供缩略图的分页的时候这甚至会变得更糟糕。

可以在后台（例如，使用PHP或ASP.NET）来做，而不是使用JavaScript来完成所有的这些事情，只需要通过XHR和JavaScript改进一下它，就可以给所有的用户提供一个功能齐全的图库。

10.6 动态的缩略图库

真正的动态缩略图库使用URL参数，而不是许多静态的页面，并且会根据这些参数来创建分页以及显示缩略图或大图。

示例examplePHPgallery.php就是以那种方式工作的，图10-2展示了这个可能显示的效果。

这个图库是功能齐全的，而且不用JavaScript也能访问，但你可能不希望每次用户单击缩略图的时候重新加载整个页面。使用XHR，这两个功能就都可以提供了。在gallerytools.php的这个案例中，不用使用原始的PHP文档，可以使用只生成你所需要的内容的一个裁减版本。我不想深入这个PHP脚本的细节；知道它可以为你完成下面的事情就可以了。

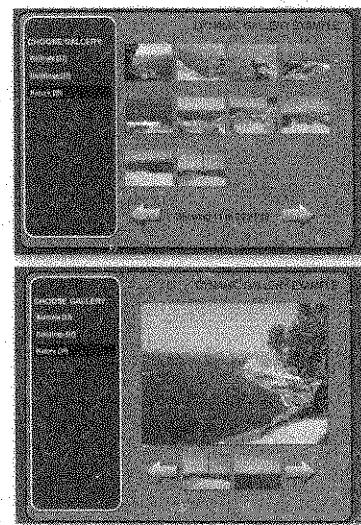


图10-2 在大图页面上，包含缩略图分页和上一个与下一个图片预览的动态的PHP驱动的缩略图库示例

- 它读取主菜单的链接指向的文件夹的内容，检查图片，并且每次返回10个图片作为一个HTML列表，其中每个缩略图都链接到对应的大图。
- 它添加了一个分页菜单来显示总共有多少图片以及其中的哪10个在显示，并且提供上一个和下一个的链接。
- 如果缩略图中的任何一个被单击了，那么它会返回大图的HTML，以及一个菜单来显示下一个和上一个缩略图。

使用这个输出来重写原始PHP脚本的HTML输出，如在示例examplePHPXHRgallery.php中所示。没有JavaScript的话，它就和examplePHPgallery.php所作的效果一样；可是，当JavaScript可用的时候，它不会重新加载整个文档，而只是刷新图库本身。可以通过把有链接的内容区域中的链接替换为gallerytools.php和XHR调用，而不是重新加载整个页面来实现这种效果。

dyngal_xhr.js

```
dyngal = {
    contentID : 'content',
    originalPHP : 'examplePHPXHRgallery.php',
    dynamicPHP : 'gallerytools.php',
    init : function() {
        if( !document.getElementById || !document.createTextNode ) {
            return;
        }
        dyngal.assignHandlers( dyngal.contentID );
    },
};
```

开始先定义你需要的属性：

- 包含应被gallerytools.php中返回的HTML替换的内容的元素ID；
- 原始脚本的文件名；
- 通过XHR返回你调用数据的脚本的文件名。

init()方法检查是否支持文档对象模型，并且把这个内容元素的ID作为参数调用assignHandlers()方法。

注解 在这个例子中，你只替换了一个内容元素；然而，由于可能存在你需要替换页面中许多不同区域的情况，因此最好为类似这样的任务创建独立的方法。

dyngal_xhr.js (续)

```
assignHandlers : function( o ) {
    if( !document.getElementById( o ) ){ return; }
    o = document.getElementById( o );
    var gLinks = o.getElementsByTagName('a');
    for(var i = 0; i < gLinks.length; i++ ) {
        DOMhelp.addEvent( gLinks[i], 'click', dyngal.load, false );
```

```

gLinks[i].onclick = DOMhelp.safariClickFix;
}
},

```

`assignHandlers()`方法检查ID为被传递过来作为参数的元素是否存在，并且循环遍历这个元素中所有的链接。接下来，它添加一个指向这个`load`方法的事件处理程序，并且应用Safari补丁以阻止浏览器打开原始的链接（记住——在`cancelClick()`中使用的`preventDefault()`方法是被Safari支持的，但是由于Safari的一个bug，它并没有阻止住链接被打开）。

```

dyngal_xhr.js (续)

load : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    var h = t.getAttribute( 'href' );
    h = h.replace( dyngal.originalPHP, dyngal.dynamicPHP );
    dyngal.doxhr( h, dyngal.contentID );
    DOMhelp.cancelClick( e );
},

```

在`load`方法中，你首先要获取事件的目标并且确保它是一个链接。接着读取这个链接的`href`属性，并且使用只返回你需要的内容的动态脚本名来替换原始的PHP脚本名。把`href`的值和内容元素的ID作为参数调用`doxhr()`方法，并且通过调用`cancelClick()`来阻止链接继续传播。

```

dyngal_xhr.js (续)

doxhr : function( url, container ) {
    var request;
    try{
        request = new XMLHttpRequest();
    } catch( error ) {
        try {
            request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch ( error ) {
            return true;
        }
    }
    request.open( 'get', url, true );
    request.onreadystatechange = function() {
        if( request.readyState == 1 ) {
            container.innerHTML = 'Loading...';
        }
        if( request.readyState == 4 ) {
            if( request.status && /200|304/.test( request.status ) ) {
                dyngal.retrieved( request, container );
            } else {

```

```

        dyngal.failed( request );
    }
}
request.setRequestHeader( 'If-Modified-Since', =>
'Wed, 05 Apr 2006 00:00:00 GMT');
request.send( null );
return false;
},
retrieved : function( request, container ) {
var data = request.responseText;
document.getElementById(container).innerHTML = data;
dyngal.assignHandlers( container );
},
failed : function( request ) {
alert( 'The XMLHttpRequest failed. Status: ' + requester.status );
return true;
}
}
DOMhelp.addEvent( window, 'load', dyngal.init, false );

```

XHR方法和你在第8章中使用的那些XHR方法是一样的。唯一不同的地方是你要再次调用assignHandlers()方法，因为你已经替换了原始的内容，结果丢失了链接上的事件处理程序。

注解 PHP和JavaScript是一个强强组合。掌握了JavaScript以后，最好深入学习一下PHP，因为没有一种服务器端语言的知识，Ajax就会失去不少乐趣。PHP可以做一些JavaScript不能做的事情——访问服务器上的文件并且读取它们的名称和属性，甚至获取第三方服务器上的内容。PHP语法和JavaScript语法在某些方面是非常相似的，而且这种语言的本性使它很容易入门而不用涉及编译器或命令行接口来构建你的脚本。

10.7 从文件夹中创建图片徽章

在下一章学习已经做好的第三方代码和在线服务之前，让我们先来看一个使用PHP和JavaScript/XHR做的另一个缩略图库示例。

如果使用过Flickr (<http://www.flickr.com>) 或者阅读过许多网络日志，那么你很可能碰到过Flickr徽章，它是显示了网站的维护者上传到系统中的最新的图片的缩略图库。以图10-3所展示的我的博客为例。

通过让用户使用上一个和下一个链接在缩略图之间进行导航以及通过单击缩略图来显示大图，我们来给徽章再增加些趣味。示例exampleBadge.html就是这样做的，图10-4展示了在Windows XP平台的Firefox上包含2个徽章图库的样子。

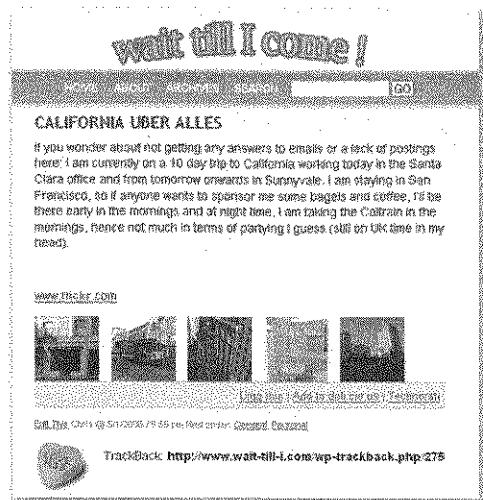


图10-3 在Flickr网站上作为我的博客徽章的我的最新照片

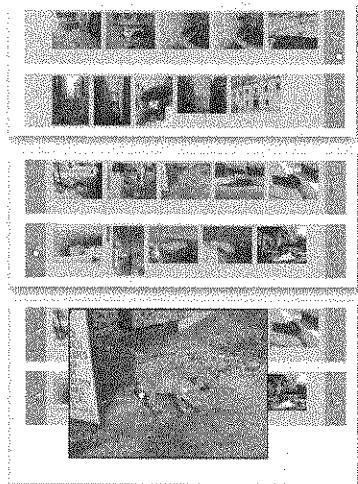


图10-4 作为徽章图库的2个图片文件夹

当创建类似这样的脚本的时候，最好使HTML保持尽可能的简单。你对维护者的期望越少，大家越可能使用你的脚本。在这个例子中，所有的维护者为了往HTML文档中添加一个徽章图库所需要做的就是，添加一个类为badge的元素以及一个指向包含图片的文件夹的链接：

exampleBadge.html（摘要）

```
<p class="badge"><a href="galleries/animals/">Animals</a></p>
<p class="badge"><a href="galleries/buildings/">Buildings</a></p>
```

由于JavaScript不能检查服务器上的文件夹里的文件，因此你需要一个PHP脚本来为你做这件事情。文件badge.php就是做这件事情的，并且以列表项的方式返回缩略图。

注解 下面是这个PHP脚本的一个简单解释；虽然这不是JavaScript，但是我希望你对将要学习的徽章脚本所使用的工具的工作方式能多了解一点。

badge.php

```
<?php
$c = preg_match( '/\d+/' , $_GET['c'] ) ? $_GET['c'] : 5;
$s = preg_match( '/\d+/' , $_GET['s'] ) ? $_GET['s'] : 0;
$cd = is_dir( $_GET['cd'] ) ? $_GET['cd'] : '';
```

你需要定义三个变量：\$c，它存储着要显示的缩略图的数目；\$s，它是在所有缩略图的列表中的当前第一个缩略图的索引；而\$cd则是服务器上文件夹的URL。PHP的\$_GET数组存储着这个URL的所有参数，这意味着如果这个URL是badge.php?c=3&s=0&cd=animals，那么\$_GET['c']就是3，\$_GET['s']就是0，而\$_GET['cd']就是animals。可以使用正则表达式来确保\$c和\$s都是整数并且预先分别地设置为5和0，而PHP的这个is_dir()函数则确保\$cd的确是个可以使用的文件夹。

badge.php (续)

```
if( $cd != '' ) {
    $handle = opendir($cd);
    if( preg_match( '/^tn_.*(jpg|jpe|jpeg)$/i', $file ) ) {
        $images[] = $file;
    }
}
closedir( $handle );
```

如果这个文件夹可用，那么开始先使用opendir()方法读出这个文件夹中的每个文件，并且通过匹配它的名字和模式`^tn_.*(jpg|jpe|jpeg)$`（以tn_开头并且以jpg、jpe或jpeg结尾）来检查文件是否为一个缩略图。如果这个文件是一个缩略图，那么把它添加到图片数组中。当文件夹中再没有剩下的文件时，通过调用closedir()方法关闭这个文件夹。

badge.php (续)

```
$imgs = array_slice( $images, $s, $c );
if( $s > 0 ) {
    echo '<li class="badgeprev">';
    echo '<a href="badge.php?c=' . $c;
    echo '&s=' . ( $s - $c ) . '&cd=' . $cd . '>';
    echo 'previous</a></li>';
} else {
    echo '<li class="badgeprev"><span>previous</span></li>';
}
```

可以使用PHP的array_slice()方法来把这个数组缩减为选择的图片（从\$s开始的\$c个图片），并且检查\$s是否大于0。如果它大于0的话，则输出一个类为badgeprev的列表元素，它在链接的href属性中有对应的参数。如果它不大于0的话，则在列表项中输出一个SPAN元素来代替链接。

badge.php (续)

```
for( $i=0; $i<sizeof($imgs); $i++ ) {
    echo '<li><a href="'.str_replace('tn_','', $cd.$imgs[$i]).'">';
    echo '</a></li>';
}
```

循环遍历这些图片，并且对于每个数组元素，在指向大图的链接中显示一个IMG元素。可以通过使用str_replace()去掉数组元素值中的tn_字符串。

badge.php (续)

```
if( ( $c+$s ) <= sizeof( $images ) ) {
    echo '<li class="badgenext">';
    echo '<a href="badge.php?c=' . $c . '&s=' . ( $s + $c );
    echo '&cd=' . $cd . '>next</a></li>';
} else {
```

```

echo '<li class="badgenext"><span>next</span></li>';
}
?

```

检查\$c和\$s加到一起是否小于等于文件夹中所有图片的数量，如果是则显示一个链接，否则显示一个SPAN。

正如你所看到的，JavaScript和PHP编程的语法和逻辑非常的相似，这也是PHP成功的原因之一。我们现在来创建一下JavaScript脚本，它使用这个PHP脚本把这些链接转换为图片徽章。

badge.js

```

badge = {
  badgeClass : 'badge',
  containerID : 'badgecontainer',

```

首先定义用来指定徽章链接的CSS类以及显示大图的图片容器的ID，把它们作为主对象badge的属性。

badge.js（续）

```

init : function() {
  var newUL, parent, dir, loc;
  if( !document.getElementById || !document.createTextNode ) {
    return;
  }
  var links = document.getElementsByTagName( 'a' );
  for( var i = 0; i < links.length; i++ ) {
    parent = links[i].parentNode;
    if( !DOMhelp.cssjs( 'check', parent, badge.badgeClass ) ) {
      continue;
    }
  }

```

接下来检查是否支持DOM并且循环遍历文档中的所有链接，检查特定链接的父节点是否已经指派了badge类。如果不是的话，则跳过这个链接。

badge.js（续）

```

newUL = document.createElement( 'ul' );
newUL.className = badge.badgeClass;
dir=links[i].getAttribute( 'href' );
loc = window.location.toString().match( /(.*\/)/g );
dir = dir.replace( loc, '' );
badge.doxhr( 'badge.php?cd=' + dir, newUL );
parent.parentNode.insertBefore( newUL, parent );
parent.parentNode.removeChild( parent );
i--;
}

```

创建一个新的列表元素并把badge类添加到它上面。接着获取链接的href属性，读取窗口的

location，并且从这个href属性值中移除window.location中最后一个/之前的所有字符。这是使脚本可以在微软的IE中运行所必需的。

注解 你可能记得这个链接是[Animals](galleries/animals/)，而且如果你的本地主机上有这个脚本，例如，在`http://localhost/book/mybook/Chapter10_A_Dynamic_gallery/exampleBadge.html`上，Mozilla返回的href属性值为`galleries/animals/`。可是，IE返回的是`http://localhost/book/mybook/Chapter10_A_Dynamic_gallery/galleries/animals/`，而且你需要移除多余的数据，因为`badge.php`期望URL中只包含文件夹的名字。

使用正确的URL和新创建的列表作为参数来调用`doxhr()`方法，并且在当前链接的父元素之前添加这个列表。接着使用DOM方法`removeChild()`移除这个链接的父元素，并且把循环计数器减少1（循环遍历这个文档的所有链接，意味着当你移除它们其中一个的时候，计数器需要减1以防止循环跳过后面的链接）。

badge.js (续)

```
badge.container = document.createElement( 'div' );
badge.container.id = badge.containerID;
document.body.appendChild( badge.container );
},
```

创建一个新的DIV作为大图的容器，设置它的ID，并且把它添加到文档的主体中。

badge.js (续)

```
doxhr : function( url, container ) {
    var request;
    try {
        request = new XMLHttpRequest();
    } catch ( error ) {
        try {
            request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch ( error ) {
            return true;
        }
    }
    request.open( 'get', url, true );
    request.onreadystatechange = function() {
        if( request.readyState == 1 ) {
        }
        if( request.readyState == 4 ) {
            if( request.status && /200|304/.test( request.status ) ) {
                badge.retrieved( request, container );
            } else{
                badge.failed( request );
            }
        }
    }
}
```

```

        }
    }
}

request.setRequestHeader( 'If-Modified-Since', =>
'Wed, 05 Apr 2006 00:00:00 GMT');
request.send( null );
return false;
},
retrieved : function( request, container ) {
    var data = request.responseText;
    container.innerHTML = data;
    badge.assignHandlers( container );
},
failed : function( requester ) {
    alert('The XMLHttpRequest failed. Status: ' + requester.status);
    return true;
},

```

Ajax/XHR方法仍然没有太大的变化，唯一不同的地方是当数据成功地接受后，要使用这个列表项作为参数来调用assignHandlers()方法。

badge.js (续)

```

assignHandlers : function( o ) {
    var links = o.getElementsByTagName('a');
    for( var i = 0; i < links.length; i++ ) {
        links[i].parent = o;
        if( /badgепrev|badgenext/.test( links[i].parentNode.className ) ) {
            DOMhelp.addEvent( links[i], 'click', badge.load, false );
        } else {
            DOMhelp.addEvent( links[i], 'click', badge.show, false );
        }
    }
},

```

assignHandlers()方法循环遍历以参数o传递过来的元素中的所有链接。它把这个元素作为每个链接的一个新的属性parent保存起来，并且检查这个链接是否应用了类badgепrev或badgenext，这个你可能还记得，是通过badge.php添加到上一个和下一个链接上的。如果存在这个CSS类，那么assignHandlers()会添加一个指向load方法的事件处理程序；否则，添加一个指向show方法的事件处理程序，因为一些链接需要在这些缩略图之间进行导航，其他的链接需要显示大图。

badge.js (续)

```

load : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
}

```

```

var dir = t.getAttribute('href');
var loc = window.location.toString().match( /(.*\/)/g );
dir = dir.replace( loc, '' );
badge.doxhr( 'badge.php?cd=' + dir, t.parent );
DOMhelp.cancelClick( e );
},

```

load方法会获取事件的目标，确保它是一个链接。接着它获取这个事件目标的href属性的值，并且在使用保存在链接的parent属性中的元素作为输出容器调用doxhr之前对它进行处理。最后通过调用DOMhelp的cancelClick()来阻止这个链接被打开。

badge.js (续)

```

show : function( e ) {
    var t = DOMhelp.getTarget( e );
    if( t.nodeName.toLowerCase() != 'a' ) {
        t = t.parentNode;
    }
    var y = 0;
    if( self.pageYOffset ) {
        y = self.pageYOffset;
    } else if( document.documentElement &&
               document.documentElement.scrollTop ) {
        y = document.documentElement.scrollTop;
    } else if( document.body ) {
        y = document.body.scrollTop;
    }
    badge.container.style.top = y + 'px';
    badge.container.style.left = 0 + 'px';
}

```

在show方法中，再一次获取事件目标，并且测试它是否为一个链接。接着在屏幕上定位大图容器。由于你不知道这个徽章会在文档的什么地方，因此要显示图片最安全的方法就是读出文档的滚动位置。要做到这一点，你需要对不同的浏览器做一些对象的侦测。

注解 Firefox、Safari和Opera把当前垂直的滚动位置保存在window对象的一个属性中，叫做pageYOffset，没有严格的HTML DOCTYPE的MSIE把它保存在document对象的body.scrollTop中，而具有严格的HTML DOCTYPE的IE和Firefox则把它保存在document对象的documentElement.scrollTop属性中。

对所有这些可能发生的情况进行测试，并且通过设置它的样式属性集合的left和top属性相应地定位图片容器。通过这种方式，可以确保大图在用户的浏览器窗口中一直是可见的。

badge.js (续)

```

var source = t.getAttribute('href');
var newImg = document.createElement('img');

```

```

badge.deletePic();
newImg.setAttribute( 'src', source );
badge.container.appendChild( newImg );
DOMhelp.addEvent( badge.container, 'click', badge.deletePic, false );
DOMhelp.cancelClick( e );
},

```

读取这个链接的href属性并且创建一个新的IMG元素。通过调用deletePic()方法来去掉可能已经显示过的其他图片，并且把这个新图片的src属性设置为这个链接的href。接着把这个新图片作为一个子节点添加到图片容器中，应用一个事件处理程序来在用户单击这个图片的时候调用deletePic()，并且通过调用cancelClick()来阻止这个链接被打开。

badge.js (续)

```

deletePic : function() {
    badge.container.innerHTML = '';
}
DOMhelp.addEvent( window, 'load', badge.init, false );

```

deletePic方法所需要做的就是设置这个容器元素的innerHTML属性设置为一个空字符串，从而移除大图。

10.8 小结

在这一章中，我们学习了如何使用JavaScript来增强已有的HTML结构或缩略图库动态的服务器端脚本，以及在用户选择另一个图片或缩略图子集的时候，如何通过不加载整个文档来使它变成动态的或者看起来更加动态。

图库创建起来总是很有趣，而且有许多新的和时髦的解决办法，我希望通过学习在这一章中展示的一些技巧，你能够有信心熟练地处理它们，并能提出自己的图库概念。

使用第三方JavaScript

现在你已可以融会贯通了，当创建一个JavaScript应用程序的时候，你不需要重新做起，也不需要每次都从头开始对所有的功能进行编码——JavaScript有许许多多的函数可以供你使用，而且你还可以创建自己可复用的函数和对象。不止于此，你还可以使用第三方的代码库和API，目前网络上有很多这样的东西。但是要知道什么地方有、为什么要用它们以及如何使用它们是关键所在，而这正是编排这一章的原因。

在这一章中，我们会先看一些目前在网络上已经有的内容和代码。我们会快速讨论一下在网络上你可以发现的不同种类的资源：RSS格式的内容、REST服务、API、代码生成器以及函数库，它们会使得你的JavaScript开发更加容易。我们会看一个代码库并且举一些例子，教你如何使用Google Maps API在几分钟里给你的网站添加一个交互式的地图，并进一步看一下Yahoo的用户界面函数库。

11.1 网络为你提供了什么

作为开发人员，我们现在生活在非常令人兴奋的时代。在过去的几年里，各公司在分享内容和技术方面的观念发生了彻底的变化。在过去，每个公司都把它的内容和代码看作是白金做的，要获取一个系统关于工作原理方面或者如何与它进行通信的信息是一个令人痛苦且漫长的过程，包括价格谈判、非工作示范、幻灯片展示、代码预览以及其他间接的市场交易。

许多这样闭源的思想依然很盛行，甚至个人网站的维护者想知道如何确保其他人不能保存他们的图片或者复制他们的代码和内容。我们在前几章中已经讨论过这个话题以及对这种方法的一些误解，这里就不详细讨论它了。相反，我们会看一下思想超前的公司的现代观念，他们正将走上一条方向完全相反的道路。例如，许多内容提供商（如媒介代理商）已经认识到，当他们以轻量级、易于聚合的格式（如RSS）提供新闻的时候，人们会使用它们来在自己的站点上显示标题新闻，这样会给他们的站点带来流量和用户。

此外，许多电子商务公司（如eBay、PayPal以及Amazon）都认识到，如果允许其他程序访问他们的服务的话，将会在线卖出更多的东西，因为他们可以在网站上提供特定内容的产品，而不会把用户赶到第三方网站上去。还有一条，这些公司开始认识到，如果给大家分享自己使用的

技术的话，就会得到许多好的反馈，使得自己的产品在更加多样的环境下进行测试（因为每个开发人员的计算机和网络连接都是不同的），而且会吸引到以后可能会雇用的人才。拥有一个开发人员网络，你可以立即看到一个人能够干什么，而不必依赖于他的个人简历（CV）。

作为开发人员，这意味着如果浏览一下网络，就会发现大量可以免费使用的内容和技术。这个观点的最好证明就是网站Don't Meet Your Heroes (<http://www.dontmeetyourheroes.com>)，它是一个网页设计和网页开发方面的网站，它的内容由于是从许多不同的博客与杂志上的RSS提要聚合而来的，因此会不断地更新。这个网站因此可以给读者提供许多内容，而维护者不需要花费时间来编写或研究这些内容。如果维护人员发现了一些很好的内容，那么可以很容易地添加，而且他们还为用户提供了一种推荐资源的工具。维护人员不需要做太多的事情，列出资源的网页的所有者就会得到更多的读者，而这个站点的用户也会在某一个地方获得为他们整理的最新的内容。从而达到多赢的效果。

这个网站还收集了数不尽的第三方JavaScript函数库，可以下载并把它们插入到自己的应用程序中，这样花费很小的力气就可以得到许多强大的功能。稍后在这一章中，我们会通过看一下jQuery和Yahoo用户界面函数库来了解一些这样的例子。

11.2 代码片段、RSS 提要、各种 API 以及函数库

Web上提供的第三方JavaScript最古老的类型大概就是可以很容易地嵌入到网站上的脚本了。它们可能是：

- 记录有多少人访问过你的站点的计数器，如StatCounter (<http://www.statscounter.com/>)；
- 广告程序，如Google AdSense (<https://www.google.com/adsense/>)；
- 自己的内容预览，它们会被添加到类似Flickr或Yahoo Answers这样的产品中；
- 类似TheFreeDictionary的双击脚本的搜索工具（双击你网站上的任何词以在TheFreeDictionary——<http://www.thefreedictionary.com/lookup.htm#script>中对它进行搜索）。

另一个例子是脚本生成器，它们可以根据你定义的参数为你生成JavaScript脚本。Google AdSense就有这个特性，而且在Web上也有许多这样免费的脚本生成器（在<http://www.scriptsearch.com/JavaScript/Scripts/Generators/>上就列出的一些脚本生成器）。处理这些脚本时尤其要小心，因为许多都已经过时了而且会生成令人厌恶的代码。

預先生成的第三方JavaScript脚本有一个普遍存在的问题。大多数这样的脚本需要嵌入到文档的主体中而不是头中，它们对性能进行优化而不是校验语法或者使其更具适用性。许多计数器脚本还使用浏览器探查法来判断用户代理（读取navigator数据而不是使用对象侦查），因此相当的不准确。

11.2.1 RSS 提要和 REST API

在第8章中，我已经谈到过RSS提要，但是在里很有必要再一次提到它，因为你在网络上发现的信息可以拿来使用是非常令人惊讶的：

- 几乎没有任何新闻代理网站不提供最近发生的新闻的RSS提要。其中比较突出的例子有CNN (<http://www.cnn.com/services/rss>)、Reuters (<http://today.reuters.com/rss/newsrss.aspx>)，以及非常令人讨厌的The Register (<http://www.theregister.co.uk/odds/about/feeds/>)。
- 你还可以从Flickr或像Cute Overload (http://mfrost.typepad.com/cute_overload/rss.xml)这样主要处理图片的博客上获取图片提要，例如San Francisco的图片。
- 如果你希望为那些喜欢听而不是读的人提供一些东西，可以在Podcast Networks (<http://www.podcast.net/>) 和Odeo (<http://odeo.com/>) 上获取Podcast信息。

有几个网站提供的一项真的非常不错的服务是REST API，它可以使你正确地获得所需要的信息。

注解 API (<http://www.webopedia.com/TERM/A/API.html>)就是应用编程接口 (application program interface)，它是构造软件应用程序的工具和集合。基本上，你有一组方法、对象和属性可以使用，它们允许你在其他程序甚至操作系统的功能之上进行编程。在某种意义上讲，你在本书中已经使用了一个API——浏览器的API，它允许你访问window对象以及它的所有方法，而DOM则允许你改变和读取文档。稍后我们在这一章中讨论Google Maps的时候，会来看一个API的例子。

要深入有关REST的细节，可能需要整整一本书，因此有关它的详细讨论超出了这里的范围。如果你感兴趣的话，可以在Wikipedia (http://en.wikipedia.org/wiki/Representational_State_Transfer) 上阅读一下相关文章。REST API所做的就是允许在URL中定义你所需要的信息。这与把不同的数据添加到URL中简直一样简单，例如，要访问不同的Wikipedia条目：

- <http://en.wikipedia.org/wiki/Javascript>;
- http://en.wikipedia.org/wiki/DOM_scripting。

在API允许你定义数据和其他参数的输出格式的地方，它还可以变得相当的复杂，例如在Yahoo搜索中：<http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=webstandards>。这个URL在Yahoo的数据库中搜索术语“webstandards”并且返回了结果的一个XML字符串。默认情况下，它返回10条记录；但是你可以通过添加一个参数来限制它只返回2条：<http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=webstandards&results=2>。

这个API还考虑到了其他参数，如你希望看到的搜索结果部分的起始位置或者输出格式。例如，如果你想看到以JSON格式返回的搜索术语“webstandards”结果的第10条～第15条，可以使用：<http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=webstandards&results=5&start=10&output=json>。

可以通过把回调函数名作为参数传递到这个API中，来直接在JavaScript中使用这个信息：

```
<script type="text/javascript">
  function results(d) {
```

```

        for(i in d.ResultSet.Result) {
            alert( d.ResultSet.Result[i].Title );
        }
    }</script>
<script type="text/javascript" =>
src="http://api.search.yahoo.com/>
WebSearchService/V1/webSearch?appid=YahooDemo&
&query=webstandards&results=5&start=10&
output=json&callback=results"></script>

```

简而言之，REST API允许你从最简单的表单系统中通过组装一个静态URL获取信息；而在很复杂的表单中，可以发送参数来定制数据的输出格式，并且调用不同的方法来获取不同类型的数据。

11.2.2 REST API示例

REST API其中的一个示例就是Yahoo的HotJobs，它允许你以下面的格式组合一个URL来获取适合你的网页大小的工作：<http://hotjobs.yahoo.com/rss/version/country/state/city/category/keywords>。

例如，

- 这个URL搜索在美国的所有JavaScript工作，并且返回一个RSS提要：<http://hotjobs.yahoo.com/rss/0/USA/-/-/javascript>。
- 这个URL返回首都航空公司在洛杉矶的所有工作：<http://hotjobs.yahoo.com/rss/0/USA/CA/Los+Angeles/-/->。
- 而这个例子会返回在圣何塞的新媒体和因特网的工作：<http://hotjobs.yahoo.com/rss/0/USA/CA/San+Jose/NEW/->。

另外一个示例是Upcoming.org，一个共享的事件日历，它列出了赛艇、展览会，以及其他社会事件，并且提供了REST API。例如，可以查找在下面的地方将要发生的所有事件。

- 英国伦敦：<http://upcoming.org.metro/uk/london/london/>。
- 纽约市：<http://upcoming.org.metro/us/ny/nyc/>。

BBC现在在<http://www0.rdhdo.bbc.co.uk/services/api/>上也提供了一个REST API的原型，一旦它变得成熟了，就会作为<http://backstage.bbc.co.uk/>上其他开发人员参考的一部分。这个API允许以RSS提要格式浏览BBC节目表。在<http://www0.rdhdo.bbc.co.uk/services/api/examples/ajax/doc.html>上，BBC还有一个Ajax应用程序示例。

注解 在Christian Gross的两本书*Ajax Patterns and Best Practices* (Apress, 2006) 和*Ajax and REST Recipes* (Apress, 2006) 中，你可以找到更多关于REST (与Ajax相关) 的信息。

11.3 使用简短精练的函数库：jQuery^①

使用代码函数库最主要的一个原因就是，开发人员希望它可以使得其他开发人员更容易完成

^① 关于jQuery，推荐阅读《jQuery基础教程》（人民邮电出版社，2008）。——编者注

日常的编码任务。在这本书的DOMhelp函数库中，你已经通过创建实用方法来解决浏览器的矛盾和不断重复出现的任务，从而做了这样的工作。你还没有做的就是，提供自己的编码语法或非DOM的在页面中访问元素的其他方法。如果你已经做了这样工作，那么就会使得代码更加简短，但是同时也牺牲了“标准的”JavaScript语法的公认效果，并且使得开发依赖于这个函数库的知识。

可是，有一种趋势，就是在JavaScript的开发中会走这样的一条道路，而许多比较大的函数库，如Rico (<http://openrico.org/>) 和prototype (<http://prototype.conio.net/>)，要完成一项任务在学习函数库特定的编码语法和规则方面也变得越来越令人头疼。在这方面做得最好的可能要数jQuery (<http://jquery.com/>) 了，它只包含一个单独的JavaScript文件，大小只有16K，你可以把它添加到文档的头部。它为你提供了许多实用方法来实现特定的网页任务。为了使用jQuery而必须编写的代码常令JavaScript初学者或者还没有接触过类似Ruby、Python或Java的开发人员感到困惑。可是，一旦你掌握了它，它还是非常强大的。

jQuery的思想是提供对文档元素的快速访问，要进行这样的访问你要有一个叫做\$（任意的一个东西）的实用方法，它可以表示下面的其中一种情况。

- DOM结构：例如，`$(document.body)`。
- CSS选择器：例如，`$('p a')`，表示在文档中段落的每个链接。
- XPath表达式：例如，`$(" //a[@rel='nofollow'] ")`，它匹配文档中rel属性值为nofollow的每一个链接。

注解 XPath是一种W3C标准语言，它用来访问XML文档的某一部分，通常被用在与XSLT或者XPOINTER的连接中(<http://www.w3.org/TR/xpath>)。由于现代的HTML要与XML语法规则一致（所有的标签都要有结束符号，所有的元素都要小写，属性值要在引号中间、单个属性应定义为名字/值对），你也可以使用XPath来查找HTML文档的某一部分。和XSLT一起使用，也是把一种XML格式转换为另一种格式的一种非常强大的工具。

jQuery实现非常简短的代码的另一种手段就是一个叫做链方法（chainable method）的概念，从DOM中你应该已经对它有所了解了。可以通过使用一个句点把每个要添加的方法连接到上一个方法上。无需使用：

```
$p = $( 'p' );
$p.addClass( 'test' );
$p.show();
$p.html('example');
```

可以使用下面的语句来代替它：

```
$( 'p' ).addClass( 'test' ).show().html( 'example' );
```

这两个例子做的事情是一样的：获取文档的每个段落，添加一个叫做test的CSS类，显示该段落（在它被隐藏的情况下），并且把这个段落的HTML内容变为“example”。jQuery为你提供

的就是大量的这样名字非常简短的方法，来满足日常的Web应用程序开发任务。在jQuery的网站（<http://jquery.com/docs/>）和Mark Constable的网站（<http://markc.renta.net/jquery/>）上提供有健全的文档和示例。

我们来看一个示例。如果你是一个开发人员且编写过教程指南，那么你会不得不经常在HTML网页中展示一些代码示例。你会把它们包含在PRE和CODE元素中，使得代码中的空格看上去格式是正确的，如下：

exampleJQuery.html（摘要）

```
<h1>Showing and hiding a code example with jQuery</h1>
<p>The code</p>
<pre><code>
[... code example ...]
</code></pre>
<p>The CSS</p>
<pre><code>
[... code example ...]
</code></pre>
```

我们来使用jQuery编写一个脚本，它会生成前面代码示例中的链接，允许展开和折叠示例而不只是简单地把它们显示出来，如图11-1所示。

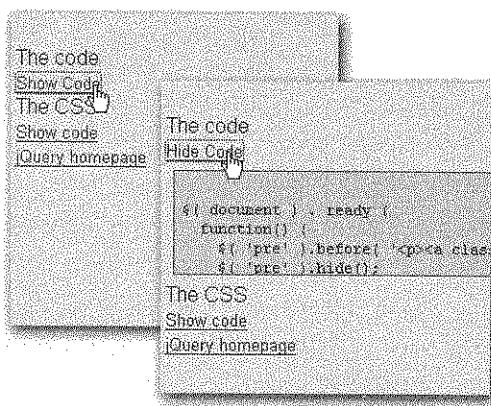


图11-1 使用jQuery显示和隐藏代码示例

jqueryTest.js

```
$( document ).ready (
  function() {
    $('pre').before( '<p><a class="trigger" href="#">Show code</a></p>' );
    $('pre').hide();
    $('a.trigger').toggle (
      function() {
        $(this.parentNode.nextSibling).slideDown('slow');
```

```
        $(this).html( 'Hide Code' );
    },
    function() {
        $(this.parentNode.nextSibling).slideUp('slow');
        $(this).html( 'Show Code' );
    }
)
}
```

正如你所看到的，这段代码非常简短，但在语法方面还是相当复杂的。我们来一点点地看一遍这个例子，让你可以明白代码是在做什么：

jqueryTest.js (摘要)

```
$( document ). ready (
```

这个`$(document).ready()`方法是一个事件处理程序，当文档准备好可以被操作时，它会调用一个以参数形式提供的函数（在这种情况下是一个匿名函数）。这意味着在文档加载完后，在这段代码中紧跟在其后的所有代码（它的意思是只有文档而不是所有类似图片这样嵌入到它里面的东西）都会被执行。

你可能还记得，我们在第5章中讨论过网页内容在可以隐藏之前显示的效果不好看。这个方法可以解决那个问题。

jqueryTest.js (续)

```
$( 'pre' ).before( '<p><a class="trigger" href="#">Show code</a></p>' );
$( 'pre' ).hide();
```

获取文档中的每一个PRE元素，并且使用before()方法在这个元素前的DOM树中添加一个HTML字符串，在这个例子中，段落里有一个类为trigger的嵌入链接。可以使用jQuery的hide()方法来隐藏所有的PRE元素（hide()方法设置CSS的属性display为none）。

jqueryTest.js (续)

```
$(‘a.trigger’).toggle (
```

使用CSS选择器`a.trigger`来匹配类为`trigger`的所有链接（希望它只是脚本通过`before()`方法添加的链接），并且使用`toggle()`方法。这个方法会在用户单击这个元素的时候交替地执行以参数形式提供的这两个函数。第一个参数是一个匿名函数，它会显示前面隐藏的代码示例并且把链接的文本修改为“Hide Code”或者相反。

jqueryTest.js (续)

```
function() {
  $(this.parentNode.nextSibling).slideDown('slow');
  $(this).html( 'Hide Code' );
},

```

可以使用几个jQuery的方法来显示和隐藏元素，最基本的方法是show()和hide()。一个更高级的效果是使用slideDown()和slideUp()产生的，它会以逐行动画的方式显示元素。这两个方法都有一个参数，表示动画的速度，它可以是slow、normal或fast。为了获取要显示或隐藏的PRE元素，你需要使用\$(this)结构，它会返回toggle()的事件目标。这意味着你可以使用this.parentNode.nextSibling像嵌套在段落中的链接一样来获取PRE元素。可以通过\$(this)和html()方法来改变这个链接本身的内容，html()方法使用一个HTML字符串作为唯一的参数并且修改这个元素的innerHTML属性。

jqueryTest.js（续）

```
function() {
    $(this.parentNode.nextSibling).slideUp('slow');
    $(this).html( 'Show Code' );
}
)
}
)
```

在这个示例中toggle()方法的另一种情形是使用slideUp()来缓慢地隐藏代码示例，并且把链接的文本修改回“Show Code”。

jQuery的一个主要扩展（jQuery的实现或扩展叫做插件）是Ajax插件。这个插件通过http://jquery.com/docs/ajax/上所述的load()、\$.get()和\$.post()方法可以很容易地实现Ajax。例如，如果你想在用户单击链接的时候创建PRE元素，并把真正的代码示例加载到它里面，那么用它则很容易实现。打开示例exampleJQueryAjax.html来看一下下面运行的脚本：

```
jqueryTestAjax.js
$( document ).ready (
    function() {
        $('a.codeExample').each (
            function( i ) {
                $( this ).after( '<pre class="codeExample">' +
                    '<code></code></pre>' );
            }
        )
        $('pre.codeExample').hide();
        $('a.codeExample').toggle (
            function() {
                if( !this.old ){
                    this.old = $(this).html();
                }
                $( this ).html('Hide Code');
                parseCode( this );
            },
            function() {
                $( this ).html( this.old );
                $( this.nextSibling ).hide();
            }
        )
    }
)
```

```

        )
        function parseCode(o){
            if(!o.nextSibling.hascode){
                $.get(o.href,
                    function(code){
                        code=code.replace(/&/mg,'&#38;');
                        code=code.replace(/</mg,'&#60;');
                        code=code.replace(/>/mg,'&#62;');
                        code=code.replace(/\"/mg,'&#34;');
                        code=code.replace(/\r?\n/g,'<br>');
                        code=code.replace(/<br><br>/g,'<br>');
                        code=code.replace(/ /g,'&nbsp;');
                        o.nextSibling.innerHTML='<code>' + code + '</code>';
                        o.nextSibling.hascode=true;
                    }
                );
            }
            $(o.nextSibling).show();
        }
    }
)

```

我们来逐步看一下这段脚本：

jqueryTestAjax.js（摘录）

```

$( document ).ready (
    function() {

```

再一次以ready()方法和匿名函数开始（也可以创建一个命名的函数并通过ready()方法来调用它）。

jqueryTestAjax.js（续）

```

$('a.codeExample').each (
    function( i ) {
        $( this ).after( '<pre class="codeExample"><code></code></pre>' );
    }
)
$( 'pre.codeExample' ).hide();

```

使用了jQuery的一个遍历方法each()，来循环遍历应用了CSS类codeExample的所有链接。接下来通过after()方法和\$(this)选择器，在每个链接的后面创建CSS类为codeExample并且其中嵌套了CODE元素的PRE元素，然后使用jQuery的hide()方法隐藏类为codeExample的所有PRE元素。

jqueryTestAjax.js（续）

```

$('a.codeExample').toggle (
    function() {
        if( !this.old ){

```

```

        this.old = $(this).html();
    }
    $( this ).html('Hide Code');
    parseCode( this );
},
function() {
    $( this ).html( this.old );
    $( this.nextSibling ).hide();
}
)

```

使用toggle()来显示和隐藏代码示例；可是，和上一个脚本不一样，它在显示代码的时候，把链接的原始文本保存到一个叫old的属性中，并且把链接文本替换为“Hide Code”。接着在显示代码的时候，把这个链接作为参数来调用函数parseCode()。当隐藏代码的时候，通过把链接的文本设置回保存在old参数中的值来恢复原来的链接文本。接下来使用jQuery的hide()方法隐藏PRE元素。

jqueryTestAjax.js（续）

```

function parseCode(o){
    if(!o.nextSibling.hascode){
        $.get (o.href,
            function(code){

```

这个函数会检查链接后的PRE元素是否有一个属性叫做hascode，在代码被第一次加载的时候你需要设置它。这是避免脚本在用户每次单击链接的时候都去加载代码（而不是只加载一次）所必需的。接下来把这个链接的href属性值和一个匿名函数作为参数来使用\$.get()方法。这样会发送一个XHR请求来加载链接的文档，并且会在文档被加载完后调用这个函数。传递一个叫做code的参数，它会包括通过XHR加载的文档的内容。

jqueryTestAjax.js（续）

```

        code=code.replace( /&/mg, '&#38;');
        code=code.replace( /</mg, '&#60;');
        code=code.replace( />/mg, '&#62;');
        code=code.replace( /\"/mg, '&#34;');
        code=code.replace( /\r?\n/g, '<br>');
        code=code.replace( /<br><br>/g, '<br>');
        code=code.replace( / /g, '&#160;');
        o.nextSibling.innerHTML = '<code>' + code + '</code>';
        o.nextSibling.hascode = true;
    }
)

```

接下来使用正则表达式来把所有的&、标签括号、引号替换为它们已编号的HTML实体，把换行替换为
，并在把这个结果添加到CODE元素中作为PRE元素的innerHTML属性之前，把空格替换为它们的HTML实体。把hascode属性设置为true，确保在下次用户单击这个链接的时候显示代码，跳过\$.get()结构体。

```
jqueryTestAjax.js (续)
```

```

        }
    );
}
$(o.nextSibling).show();
}
)
)
```

剩下的就是使用jQuery的show()方法显示PRE元素。注意你需要在\$.get()结构体之外来做，以确保用户选择显示代码时，代码会被再次显示出来。

使用自己语法的 jQuery 与其他函数库的危险

使用jQuery，你可以轻松快捷地完成许多日常的网络应用程序和网络开发的任务。可是，如果你要把这个文档提交给一个第三方开发人员进行维护，那么他就必须了解jQuery，不然就会感到很迷惑。这是使用函数库的一个危险。不是依赖于JavaScript语法和规则，你为这个过程又添加了额外的一层必需的知识。函数库所带来的益处是否值得那样去做则需要你自己来决定。

另一个问题是许多代码库示例都不是非分离式的，但却需要显示该库的功能。当使用和说明函数库的时候，人们非常期望JavaScript是可用的。确定不要犯这种错误。函数库是为了让开发的过程更加快速和简便，而不是使得我们依赖于它们，或者重复我们过去使用函数库已经犯过的错误——创建的应用程序和网站没有JavaScript就不能工作。

接下来我们会看一下使用Google Maps API来创建地图应用程序。

11.4 使用 API: 用 Google Maps 为你的网站添加地图

Google Maps (<http://maps.google.com>) 很可能是充分使用了Ajax的网络应用。它为用户提供地图可四处移动、缩放以及执行通行证，甚至还有注解。你可以以地图、卫星图片或者二者混合的方式来显示你想看到的位置。

Google允许Web开发人员通过API的方法来在他们自己的站点上使用Google Maps。要使用这个API，你需要在它的主页<http://www.google.com/apis/maps/>上注册获取一个免费的开发人员序列号。在网站上还可以找到相关的文档和许多教你如何使用Google Maps的示例。这个序列号可以使你在单个域名或这个域名的子文件夹中使用地图。这一章的示例中使用了一个在本机上工作的序列号，就是说你需要在本地服务器上通过<http://localhost/>而不是文件系统来运行它们。

获得了开发人员序列号以后，你就可以链接到在文档的头部包含所有地图代码的JavaScript。粗体的“your key”应该被替换为你从Google获取的序列号——如果你是在一个不同的服务器而不是本地服务器上尝试运行这个示例的代码，那么你必须修改这个示例文件中的序列号。

```
<script src="http://maps.google.com/maps?file=api_
&v=2&key=your key" type="text/javascript">
</script>
```

警告 这个URL未来可能会发生变化，因此要检查这个API的主页确保它不会突然不能正常工作。最可能发生变化的是URL中的v参数，它表示API的版本。

你要做的下一个步骤是获取你希望显示的位置的纬度和经度值。如果是在美国，那么你很幸运，因为在<http://geocoder.us/>上有一个免费的服务可以把地址转换为纬度和经度值。例如，如果你想显示加利福尼亚州派拉蒙工作室（Paramount Studio）的位置，它的位置在Melrose大街5555号，那么可以从<http://geocoder.us/demo.cgi?address=5555+Melrose+Ave%2C+Hollywood%2C+CA>上获取这些数据。结果是：

```
Address
5555 Melrose Ave
Los Angeles CA 90038
Latitude
34.083517 °
N 34 ° 5' 0.7"
Longitude
-118.321951 °
W 118 ° 19' 19.0"
```

更简单的办法是使用<http://www.localsearchmaps.com>（说明在<http://emad.fano.us/blog/?p=277>）和<http://www.zeesource.net/maps/geocoding.do>上的REST应用编程接口。前者返回与Google Maps API兼容的代码（遗憾的是现在这些老的API已经过时了，但是到本书出版的时候它可能已经被修改了），后者返回一系列的逗号分割的纬度和经度信息——这两者返回的都是全球的信息，而不是美国的信息。

当你有了这些信息，就可以在网站中添加自己的地图了。举一个世界性地址的例子，我们使用一下在英国伦敦北部我所在的街道。我所在街道的坐标是51.5623°N和0.0934°W。使用这个API为你提供的这些信息和方法，来显示我所在地区的地图非常容易。

开始需要一个包含地图的HTML元素。可以在这个元素中添加JavaScript不可用或者浏览器不支持的时候要显示的内容。这个内容可以是任何的HTML、文本，甚至同一地图的一个静态图片。后者是一个比较好的选择，可以确保向后兼容；你只需要确保不要在文本中的任何地方告诉用户地图是动态的就可以了，因为它可能会不是动态的。

exampleGoogleMaps.html（摘录）

```
<div id="map">
<p>Here you should see an interactive map, but you
either have scripting disabled or your browser
is not supported by Google Maps.</p>
</div>
```

确保在CSS中已经给出了那个元素的尺寸，因为不然的话地图会显得很没有规律。

googleMaps.css

```
#map{
    width:400px;
    height:300px;
    border:1px solid #999;
    margin:1em;
}
```

要把地图添加到你的HTML文档中，需要你像前面说明的那样在添加主要的Google Maps代码之后嵌入下面的脚本（内联的或者放到SCRIPT元素中）。

googleMaps.js

```
function addMap() {
    if ( GBrowserIsCompatible() ) {
        var mapContainer = document.getElementById( 'map' );
        var map = new GMap2( mapContainer );
        var point = new GLatLng( 51.5623, -0.0934 );
        map.setCenter( point, 13 );
    }
}
window.onload = addMap;
window.onunload = GUnload;
```

API的GBrowserIsCompatible()函数会检查浏览器是否支持Google Maps，如果是则返回true。在文档加载完成后，你可以调用自己的函数addMap()；当文档关闭的时候，调用API提供的GUnload()函数（因为关闭窗口会触发unload事件）。后者是必需的，因为这个地图使用了许多的事件处理，这可能会使得IE由于内存溢出的问题变得比较慢。

提示 在<http://javascript.weblogsinc.com/2005/03/07/javascript-memory-leaks/>上，你可以学习到更多有关IE内存溢出以及其解决办法的内容。

通过调用GMap2()方法来添加地图，它会创建一个新的地图对象。这个方法使用要插入地图的元素作为其参数，在这个例子中是ID为map的元素。你可以在一个文档中随意添加多个地图。

使用纬度和经度值作为参数的GLatLng()方法定义一个新的点。接着通过setCenter()方法把地图居中到这一点，这个方法把地图要居中的点和放缩级别（1~17）作为参数。这就是要在网页上创建一个交互式地图所要做的，如图11-2所示。

没有<http://maps.google.com>以及其他动态地图网站上所示的常用控件的话（缩放、移动、修

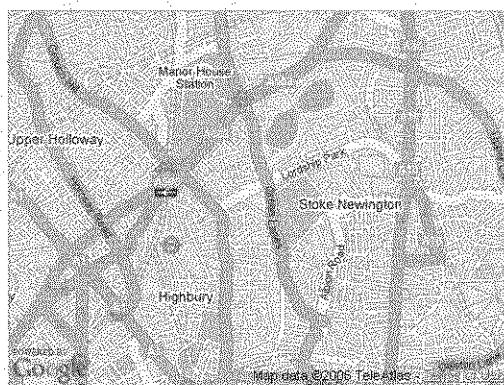


图11-2 使用Google Maps的简单地图

改地图类型)，这张地图就会搞混用户，因为没有东西表明这是一个交互式的地图而不是一个静态图片。因此，使用这个API的addControl()方法来添加控件是非常有意义的。

```
googleMapsControls.js

function addMap() {
    if ( GBrowserIsCompatible() ) {
        var mapcontainer = document.getElementById('map');
        var map = new GMap2( mapcontainer );
        map.addControl( new GSmallMapControl() );
        map.addControl( new GMapTypeControl() );
        map.addControl( new GScaleControl() );
        map.addControl( new GOOverviewMapControl() );
        var point = new GLatLng( 51.5623, -0.0934 );
        map.setCenter( point, 13 );
    }
}
window.onload = addMap;
window.onunload = GUnload;
```

addControl()方法把控件对象的一个实例作为它唯一的参数。这个例子添加了一个GSmallMapControl()，它是由可以4个移动地图的箭头以及用来放大和缩小的+和-按钮组成的。GMapTypeControl()创建了3按钮用来显示地图、卫星照片或者混合体——它的意思是带有路名的卫星照片。GScaleControl()在底部左边靠近Google标示的地方显示了地图的比例，最后GOOverviewMapControl()方法在右下角提供了一个概览的矩形。结果如图11-3所示。

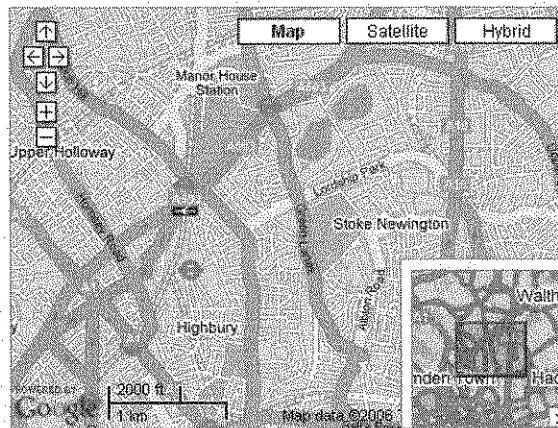


图11-3 带有控件的地图

在画图的时候，被GOOverviewMapControl()调用的概览控件有一个问题：API会把控件放到地图屏幕的左下角之外。为了使得概览图看上去是在这个地图上面，你需要应用下面的补丁：

googleMapsControlsFixed.js

```

function addMap() {
    if ( GBrowserIsCompatible() ) {
        var mapContainer = document.getElementById('map');
        var map = new GMap2( mapContainer );
        map.addControl( new GSmallMapControl() );
        map.addControl( new GMapTypeControl() );
        map.addControl( new GScaleControl() );
        map.addControl( new GOverviewMapControl() );
        var point = new GLatLng( 51.5623, -0.0934 );
        map.setCenter( point, 13 );
        var overview = document.getElementById('map_overview');
        mapContainer.style.position = 'relative';
        overview.style.position = 'absolute';
        mapContainer.appendChild(overview);
    }
}
window.onload = addMap;
window.onunload = GUnload;

```

这个API创建的概览控件是ID带有map_overview的

。它对所有的地图都创建这样的元素；如果地图的ID为myMap，那么概览控件的ID就是myMap_overview。你可以使用这个信息来绝对地定位控件以及相对地定义主要地图DIV，并且把概览控件作为一个子节点追加到mapContainer上。这样就会使控件保持在主要地图里面了。

地图现在显示了我所居住的地区，但是你不知道我具体在什么地方。要在地图上突出显示某些点，需要使用GMarker()方法，它需要一个点作为参数。地图的addOverlay()方法会在这个点的位置添加标记。

googleMapsMarker.js

```

function addMap() {
    if ( GBrowserIsCompatible() ) {
        var mapContainer = document.getElementById('map');
        var map = new GMap2( mapContainer );
        var point = new GLatLng( 51.5623, -0.0934 );
        map.setCenter( point, 13 );
        var marker = new GMarker( point );
        map.addOverlay( marker );
    }
}
window.onload = addMap;
window.onunload = GUnload;

```

这个修改在你定义为点的位置显示了一个红色的标记图标，如图11-4所示。可以根据自己需要在一个地图上设置多个标记。

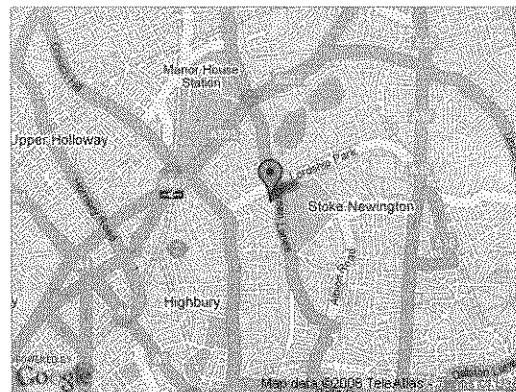


图11-4 带有标记的地图

Google API还允许你为地图以及它的每个元素添加事件处理程序。例如，如果你想在用户单击标记的时候，显示一个消息窗口（因为标记默认情况下是可以单击的），那么可以使用GEvent.addListener()方法。这个方法把要添加事件的元素、事件以及要调用的函数作为参数，非常类似于addEvent()所做的。在添加消息的函数中，创建一个文档的新的文本节点，并把这个点和新的文本节点作为参数使用地图的openInfoWindow()方法来显示消息窗口，如图11-5所示。

exampleGoogleMapsMarkerEvent.js

```
function addMap() {
    if ( GBrowserIsCompatible() ) {
        var mapContainer = document.getElementById('map');
        var map = new GMap2( mapContainer );
        var point = new GLatLng( 51.5623, -0.0934 );
        map.setCenter( point, 13 );
        var marker = new GMarker( point );
        map.addOverlay( marker );
        GEvent.addListener( marker, 'click', addMessage );
    }
}
function addMessage() {
    var message = 'This is where Chris lives';
    var msgNode = document.createTextNode( message );
    map.openInfoWindow( point, msgNode );
}
window.onload = addMap;
window.onunload = GUnload;
```

提示 在消息窗口中你可以使用HTML，只需要用openInfoWindowHtml()方法代替openInfoWindow()方法就可以了。

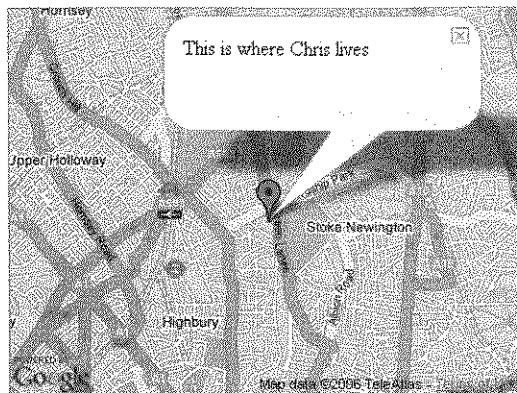


图11-5 带有一个标记和消息窗口的地图

Google Maps另一个很优秀的地方是你可以使用`setZoom()`方法设置缩放的级别，以及使用`panTo()`方法把地图的中心移动到另一个位置。此外，`openInfoWindowHtml()`方法还允许你定义一个在用户关闭消息窗口时要调用的函数，因此你不需要为它使用自己的事件处理程序了（从技术上讲，这当然还是用一个事件处理程序完成的，只是Google的API帮你把这件事做了）。这个函数的语法有点古怪：添加一个JSON对象作为`openInfoWindowHtml()`方法的最后一个参数，其中有一个属性叫做`onCloseFn`，它的值是用户关闭窗口时应当调用的函数名。

我们来使用`setZoom()`、`panTo()`以及包含3个参数的`openInfoWindowHtml()`方法创建一个交互式的地图，它会在用户单击初始化标记的时候打开一个窗口，在用户关闭那个窗口时缩小地图，它还会把地图移动到另一个位置，并且在那里打开另一个窗口。

googleMapsPan.js

```
function addMap() {
    if (GBrowserIsCompatible()) {
        var mapContainer = document.getElementById('map');
        var map = new GMap2(mapContainer);
        var home = new GLatLng(51.5623, -0.0934);
        var work = new GLatLng(51.5138, -0.1284);
        var homeMarker = new GMarker(home);
        var workMarker = new GMarker(work);
        var homeMessage = 'This is where Chris lives';
        var workMessage = 'This is where Chris works';
        map.setCenter(home, 13);
        map.addOverlay(homeMarker);
        map.addOverlay(workMarker);
        function addMessage() {
            map.openInfoWindowHtml(home, homeMessage,
                { onCloseFn : goToWork });
        }
    }
}
```

```

function goToWork() {
    map.setZoom( 12 );
    map.panTo( work )
    map.openInfoWindowHtml( work, workMessage,
        { onCloseFn : backToHome } );
}
function backToHome(){
    map.panTo( home )
    map.openInfoWindowHtml( home, homeMessage,
        { onCloseFn : goToWork } );
    map.setZoom( 13 );
}
GEvent.addListener( homeMarker, 'click', addMessage );
}

window.onload = addMap;
window.onunload = GUnload;

```

结果如图11-6所示。

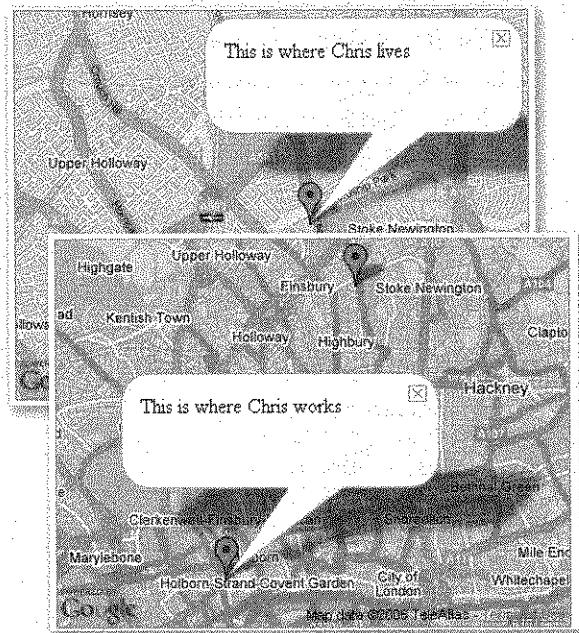


图11-6 把地图从一个位置移动到另一个位置

这个例子需要缩小一个显示级别以平稳地从一个位置移动到其他位置。如果你给予panTo()作为参数的点在当前可以看得见的地图之外，API会以这一点为中心重新绘制一个新的地图。如果起始点和终点在同一个地图上，那么他会激活地图并且平稳地移动到那里。

我们来逐步地看一下这段脚本：

googleMapsPan.js (摘录)

```
function addMap() {  
    if (GBrowserIsCompatible()) {  
        var mapContainer = document.getElementById('map');  
        var map = new GMap2(mapContainer);  
    }  
}
```

首先检查浏览器是否支持Google Maps，并定义mapContainer为包含地图的元素。通过把mapContainer作为参数调用GMap2()构造器来创建一个新的地图。

googleMapsPan.js (续)

```
var home = new GLatLng(51.5623, -0.0934);  
var work = new GLatLng(51.5138, -0.1284);  
var homeMarker = new GMarker(home);  
var workMarker = new GMarker(work);  
var homeMessage = 'This is where Chris lives';  
var workMessage = 'This is where Chris works';
```

使用前面提到的从其中一个服务而获取的纬度和经度信息定义两个新的点，并使用GMarker()定义两个标记。一个是我的家，另一个是我工作所在的公司。然后再定义两个消息告诉网页的访问者这些点是什么。

googleMapsPan.js (续)

```
map.setCenter(home, 13);  
map.addOverlay(homeMarker);  
map.addOverlay(workMarker);
```

通过把我家所在的点和缩放级别13作为参数调用setCenter()来显示地图。接着使用addOverlay()方法在地图上为每个点（家和公司）显示标记。

googleMapsPan.js (续)

```
function addMessage() {  
    map.openInfoWindowHtml(home, homeMessage,  
                           {onCloseFn: goToWork});  
}
```

接下来，需要在添加处理器前定义一些会被该处理器调用的函数，否则的话Google Maps的API不会运行。addMessage()函数使用openInfoWindowHtml()方法来显示消息，它会在用户单击第一个标记的时候告诉他这是我的家。注意与第一个标记示例的不同：这次需要使用{ onCloseFn: goToWork }作为第3个参数，来添加会在用户关闭消息窗口时调用goToWork()函数的处理器。

注解 这个参数的语法非常令人不解，我能想到的唯一原因就是Google的开发人员希望第3个参数尽可能的灵活，不用在说明书中添加其他的参数。就现在的样子来看，它是一个使用反引号的字符串对象，onCloseFn是其中的一个可能存在的属性。要了解更多信息，可以在<http://www.google.com/apis/maps/documentation/>上参考它的API文档。

```
googleMapsPan.js (续)

function goToWork() {
    map.setZoom( 12 );
    map.panTo( work )
    map.openInfoWindowHtml( work, workMessage,
        { onCloseFn : backToHome } );
}

function backToHome(){
    map.panTo( home )
    map.openInfoWindowHtml( home, homeMessage,
        { onCloseFn : goToWork } );
    map.setZoom( 13 );
}
```

goToWork()函数使用setZoom()方法设置地图的缩放级别为12，以确保家和公司的标记在地图的同一可见区域里。这会使得Google API在你把工作地点作为唯一参数调用panTo()的时候可以平稳地滚动到其他区域。API把地图移动到工作地点后，就会显示一个消息窗口，并再次使用openInfoWindowHtml()的第3个参数在用户关闭它的时候调用backToHome()。

backToHome()函数的工作原理是一样的，只是顺序正好反了过来。它告诉Google API首先把地图移回到我家所在的点，在那里打开一个消息窗口，在用户关闭它的时候调用goToWork()，最后设置地图的缩放级别为13。

```
googleMapsPan.js (续)

GEvent.addListener( homeMarker, 'click', addMessage );
}
}
window.onload = addMap;
window.onunload = GUnload;
```

这个示例应该会使你对Google Maps所涉及的东西有所了解。其实Google API还提供了许多选项，如定义效果不同的标记、你自己的修饰以及更多可以用来处理的方法。它的文档在<http://www.google.com/apis/maps/documentation/>上，而且维护得也很好，和API中的修改也会一致更新的。

注解 地图现在非常热门，许多开发人员使用Yahoo地图或Google Maps来创建所谓的mashups(相关的地图应用)。它们这些混合程序混合了几个提供API的在线应用程序来创建一个新的应用程序。相关的示例有显示事件、当前新闻以及某地天气状况的地图。可以在<http://www.programmableweb.com/>上查看不同地图API的mashups列表。

11.5 完整的服务：雅虎开发人员网络以及YUI

雅虎作为最早的因特网内容和服务提供商，通过给网络开发人员提供雅虎开发人员网络

(Yahoo Developer Network)，迈出了非常吸引人的一步。这个网络的主页是<http://developer.yahoo.com>，在那里你可以找到所有雅虎API、RSS提要以及REST服务的一个列表。真正吸引人的地方是REST的结果在很多情况下也提供了JSON格式，这意味着你可以把它们作为数据源嵌入到一个SCRIPT元素中，而不必使用Ajax。也有一个JavaScript开发人员中心，在<http://developer.yahoo.com/javascript/>上，它列出了关于如何使用雅虎和它的公司为你提供的服务、文章以及代码示例。

它的开发人员网络还包含设计模式，提供了处理某种特定网页设计任务及其解决方法的信息，以及一个自底向上开发JavaScript增强的网站和应用程序的函数库——包括为不同的布局和字号问题预先定义的CSS。这个库叫做雅虎用户界面库，简称为YUI，可以在<http://developer.yahoo.com/yui/>下载。在下载的压缩文件中，可以在build文件夹中找到库文件，在docs文件夹中找到文档，以及在examples文件夹中找到示例。这个库由几个组件组成，每个都必须被包含在他们自己的<script>标签中。这些组件要么以可读的JavaScript文件形式提供，如yahoo.js，要么以文件名以-min结尾的文件大小优化的版本提供。如yahoo-min.js。后者没有空格，并且排列紧凑，使得文件更小。

现在，这个库有下面几个组件组成：

- 一个对元素进行动画效果以及淡入淡出的动画组件；
- 一个创建Ajax应用程序的连接管理器；
- 一个访问及修改元素，并且能动态应用CSS类的DOM组件；
- 一个拖放组件；
- 一个用来进行事件处理的事件组件；
- 一个自动完成表单域的控件；
- 一个从表单中获取日期的日历控件；
- 一个可以创建脚本的页面元素的容器控件，该页面元素可以放到当前文档的上面；
- 一个用来创建动态菜单的菜单控件；
- 一个滑动块控件；
- 一个树视图控件。

我们使用其中的一些组件，来复制并增强一下在其他章节已经讨论过的一些解决方案。

11.5.1 使用YUI的弹性标题

我们使用YUI以及它的组件来创建一个更华丽的可单击标题的示例。示例exampleBouncyHeadlines.html使用了DOM、事件以及动画组件，以便在用户单击标题的时候以动画的形式来显示和隐藏标题下面的内容。动画会显示该内容，把其余的部分相应地在页面上往下移，并从白色平滑地淡入到最终的颜色。图11-7展示了它看上去的效果。

乍一看，这段脚本相当复杂，但是只要你理解了它们做了什么，使用库脚本还是很容易的。这里我会只提它的一些特性，而不是你拥有的所有选项，因为这些内容自己就需要一章的篇幅；YUI文档会经常地更新，你可以在那里阅读相关的变更和新的选项。

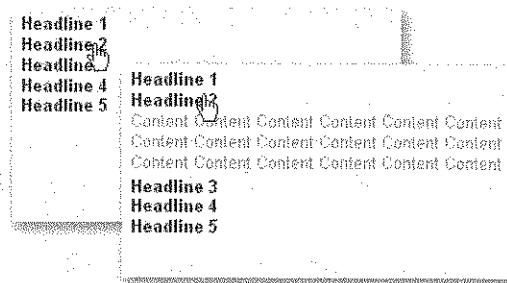


图11-7 使用YUI库的平滑动画以及内容渐变

bouncyHeadlines.js

```

YAHOO.namespace('bh');
bh = {
    triggerClass:'show',
    init : function() {
        var listitems, i, content;
        bh.headings = document.getElementById('headlines');
        if( !bh.headings ) { return; }
        listitems = bh.headings.getElementsByTagName( 'h3' );
        for( i = 0; i < listitems.length; i++ ) {
            content = listitems[i].parentNode.➥
                getElementsByTagName( 'p' )[0];
            content.defaultHeight = content.offsetHeight;
            listitems[i].content=content;
            YAHOO.util.Event.addListener( listitems[i], 'click',➥
                bh.toggle);
        }
    },
    toggle : function() {
        var attributes, anim;
        var c = this.content;
        if( c.show ) {
            attributes = {
                height: { from : c.defaultHeight, to : 0 },
                opacity: { from : 1, to : 0 }
            };
            anim = new YAHOO.util.Anim( c, attributes, .6,➥
                YAHOO.util.Easing.easeBoth );
            anim.animate();
            anim.onComplete.subscribe( bh.toggleCustom );
        } else {
            YAHOO.util.Dom.addClass( c, 'shown' );
            attributes = {
                height: { from:0, to:c.defaultHeight },
                opacity: { from:0, to:1 }
            };
        }
    }
};

```

```

anim = new YAHOO.util.Anim( c, attributes, .6,➥
YAHOO.util.Easing.backOut );
anim.animate();
anim.onComplete.subscribe( bh.toggleCustom );
}
},
toggleCustom:function() {
var c=this.getEl();
c.shown = c.shown ? false : true;
},
hideContents : function() {
YAHOO.util.Dom.addClass( 'headlines', 'dynamic' );
}
}
YAHOO.util.Event.onAvailable( 'headlines', bh.hideContents );
YAHOO.util.Event.addListener( window, 'load', bh.init );
}

```

开始先为主对象定义一个命名空间以及定义对象本身（在这个示例中，我们把弹性的标题叫做bh）。这个命名空间定义是可选的；可是，它额外地提供了一层，以确保你的脚本不与其他脚本相冲突，而且你的脚本是构建在YUI基础上的，以后有可能成为一个比较大的应用程序的一部分。接着定义一个属性叫做triggerClass，它是在用户单击标题的时候用来显示新项的类。

bouncyHeadlines.js (摘录)

```

YAHOO.namespace('bh');
bh = {
triggerClass:'show',

```

初始化方法init()先定义变量listitems、i以及content，并把ID为headlines的元素保存到主对象bh的headings属性中。如果没有ID为headlines的元素，则使用return停止其他脚本的执行；否则，定义listitems为ID为headlines的元素中所有H3元素的数组。

bouncyHeadlines.js (续)

```

init : function() {
var listitems, i, content;
bh.headings = document.getElementById('headlines');
if( !bh.headings ){ return; }
listitems = bh.headings.getElementsByTagName( 'h3' );

```

接下来，循环遍历所有的列表项，并定义content为当前标题的父节点中的第一个段落元素。通过读取offsetHeight属性读出这个段落的高度，并把它存储到这个段落元素的一个叫做defaultHeight的新属性中。这是在后面告诉动画方法这个段落初始有多高以及在动画结束后要把它设置为什么所必需的。把这个段落存储为这个列表项的一个叫做content的新属性，以使得在其他的方法中可以很容易重新获得它，并添加一个事件，在用户单击标题的时候触发监听方法toggle()。

bouncyHeadlines.js (续)

```

for( i = 0; i < listitems.length; i++ ) {
    content = listitems[i].parentNode.➥
    getElementsByTagName( 'p' )[0];
    content.defaultHeight = content.offsetHeight;
    listitems[i].content=content;
    YAHOO.util.Event.addListener( listitems[i], 'click',➥
    bh.toggle);
}
},

```

在toggle()方法中，你会看到YUI的addListener()方法的高明之处。不用必须使用事件对象e，通过getTarget()方法来获取被激活的元素，进而会带来浏览器的问题，要获取触发事件的对象你所要做的就是使用this关键字，因为YUI的addListener()方法已经为你自动设置了它；这是Scott Andrew LePera的addListener()函数做的失败的一个地方。定义变量attributes和anim，并通过this.content重新获得这个段落以应用动画效果——因为this返回的是被单击的标题，而你把段落保存在一个叫做content的属性中。读取这个段落的shown属性（它表示这个段落是否可见）。如果这个属性没有设置的话，则为这个段落添加CSS类shown来使它可见。添加和移除类是通过YUI中的DOM实用程序的addClass()方法实现的，它的参数为要应用类的元素以及要应用的类名。

bouncyHeadlines.js (续)

```

toggle : function() {
    var attributes, anim;
    var c = this.content;
    if( !c.shown ) {
        YAHOO.util.Dom.addClass( c, 'shown' );
    }
},

```

接下来你可以为这个段落应用动画。使用YUI为一个对象应用动画效果是非常容易的：把动画的属性设置为一个JSON对象，这个对象的每个属性都有一个from和to属性或者是一个by属性。前者的动画会把该对象从一个状态过渡到另一个状态，而后者会对该对象应用一定次数的动画（在你只想修改它而不是有一个固定的起点和终点值的情况下）。在这个例子中，我们会把段落的高度从0变为原始的高度，不透明度从0变到1，这样会使得段落一行一行地显示，而且变得越来越模糊。通过读取在前面的init()方法中为这个段落定义的defaultHeight属性，你重新获得动画所需要的最终高度值。

bouncyHeadlines.js (续)

```

attributes = {
    height:{ from : 0, to : c.defaultHeight },
    opacity:{ from:0, to:1 }
};

```

设置完这些属性后，你就可以通过YUI的实用方法Anim()调用一个新的动画对象了。这个方

法拥有4个参数：要应用动画效果的对象、动画的属性（你刚定义的）、动画的持续时间（在这里是600毫秒）以及提供动画所需的值的方法。后者可以是YUI的Easing实用程序的几个属性，它会提供一个确保平稳动画所需要的值。在屏幕上的动画对象不只意味着把一个变量从起始值增加到结束值；如果一个动画先缓慢地开始，在变化的过程中逐渐加快，或者开始时比较快，而在结尾的时候逐渐慢下来，那么该动画看上去会更加平滑自然。计算这些过渡并确保他们在不同的显示器和计算机上都能平稳地运行是件非常需要技巧的工作，而YUI的开发人员已经帮你把这件事给做了。实用程序Easing有几个对象，它们包含下面这些值：

- ❑ easeIn、easeOut和easeBoth，它分别表示开始慢、结尾慢或者开始结尾都慢。
- ❑ backIn、backOut和backBoth，它分别表示从小于起始值的地方开始、或者到大于结尾值的地方结束以及平稳地返回正确的值。

要显示这个段落，可以使用backOut来使它显得平稳些，在它下面显示一些额外的点线，当达到真正的值时快速回到正确的高度。这个动画开始先调用动画对象（在这里是anim）的animate()方法。这个anim对象也有一个onComplete事件，可以通过onComplete.subscribe()来订阅它，并且告诉它当动画达到最终的值时调用toggleCustom()方法。

bouncyHeadlines.js (续)

```
anim = new YAHOO.util.Anim( c, attributes, .6,➥
YAHOO.util.Easing.backOut );
anim.animate();
anim.onComplete.subscribe( bh.toggleCustom );
```

如果这个段落已经是可见的（由已经设置的shown属性所表明），那么你可以给它应用另一种动画效果。开始的不透明度为1，高度为段落的defaultHeight，然后开始减少两者直到它们达到0。这里不使用backOut，我们可以使用easeBoth来使段落的消失跳跃更少。

bouncyHeadlines.js (续)

```
} else {
    attributes = {
        height: { from : c.defaultHeight, to : 0 },
        opacity: { from : 1, to : 0 }
    };
    anim = new YAHOO.util.Anim( c, attributes, .6,➥
YAHOO.util.Easing.easeBoth );
    anim.animate();
    anim.onComplete.subscribe( bh.toggleCustom );
}
},
```

事件监听方法toggleCustom()需要重新获得应用动画的元素（段落）以及切换它的shown属性。因为onComplete.subscribe()方法在创建的动画对象的范围内，是使用YAHOO.util.Anim()以一个新的实例运行的，所以你可以使用this关键字重新获得这个对象以及它的所有属性和方法。这意味着你可以使用getEl()方法获取这个元素，它是使用YUI的Anim()方法创建的每个动画对象的一

部分。

bouncyHeadlines.js (续)

```
toggleCustom:function() {
    var c=this.getEl();
    c.shown = c.shown ? false : true;
},
```

你的脚本所需要的最后一个东西是，一个在初始时隐藏所有的段落并调用init()方法的方法。可以通过使用addClass()为标题元素添加一个新的CSS类来隐藏段落。

bouncyHeadlines.js (续)

```
hideContents : function() {
    YAHOO.util.Dom.addClass( 'headlines', 'dynamic' );
    bh.init();
}
```

无需在window对象上使用通常的load事件来调用hideContents()方法从而在最初时隐藏段落，YUI有另一种机制：onAvailable()方法，它会在窗口完成加载前试着访问带有你提供的作为属性的ID的元素，并且在这个元素可用以后会调用作为第2个参数所提供的函数。它的实际结果是你可以隐藏段落，而不必采用第5章中讨论的像内联CSS这样的hack。

bouncyHeadlines.js (续)

```
YAHOO.util.Event.onAvailable( 'headlines', bh.hideContents );
```

希望通过这个例子，你能对YUI的DOM、事件以及动画组件的强大功能有所认识。在从它的主页上下载的压缩文件中，每个组件都有它自己的示例和完整文档，很值得你试验一下并自己做些修改。

11.5.2 使用YUI的连接管理器和容器组件代替弹出窗口

我们再来看YUI库的另2个组件：处理Ajax调用的连接管理器，以及考虑到覆盖网页的模块化窗口和元素的容器组件。连接管理器是YUI非常强大的一部分，因为它允许你以非常简单的语法创建Ajax请求。

- 定义一个处理器对象，它拥有2个属性：success和failure，它们两个都以属性值的方式把Ajax请求指向所提供的函数。
- 定义用于响应这些处理器的函数，可以如你在第8章中看到的那样，使用XHR调用的所有响应数据。
- 使用YAHOO.util.Connect.asyncRequest()方法处理XHR请求，该方法的参数为请求的方法、URL以及处理器对象。

下面的简短示例会在窗口完成加载后加载文件demotext.html：

exampleXUIAjax.html（摘录）

```
var handlers = {
  success: success,
  failure: failure
}
function success( t ) {
  alert( t.responseText );
}
function failure( t ) {
  alert( 'There was an error: ' + t.statusText );
}
window.onload = function() {
  call = YAHOO.util.Connect.asyncRequest('GET', 'demotext.html', →
    handlers);
}
```

可以通过YAHOO.util.Connect.isCallInProgress()方法监控连接的状态，当调用仍然在处理的时候它会返回true。如果你想在一段时间后中止一个连接，那么可以使用YAHOO.util.Connect.abort()方法和window.timeOut()。isCallInProgress()和abort()两个方法都使用连接的变量名（在前面的例子中为call）作为参数。

容器工具允许创建与当前文档有特定关系的动态元素。可以创建模块（它是可以脚本化的页面元素）、覆盖（它可以覆盖文档）、提示（当你悬停在元素上面的时候会出现）、面板（用户可以像移动一个新的浏览器窗口一样移动它）、对话框（代替窗口的方法alert()、confirm()或prompt()，甚至允许它们自己的表单来覆盖文档）。容器真正令人惊讶的特性是，它解决了你手卷自己的对话框或面板时所遇到的许多问题：

- 可以把模块自动地居中到浏览器当前可见的部分。
- 用户可以通过拖动面板的标题来移动模块，但是你可以阻止他把面板移到当前可见文档区域以外。
- 可以应用特效使得面板平稳地显示和消失。
- 可以为面板应用投影效果。
- 可以应用一个补丁防止文档中的表单元素可见，尽管这个元素已经覆盖了它们。这是IE中覆盖元素的一个非常常见的问题。

我们来使用这2个库组件和一些DOM技巧，来在同样的文档中模拟一个浏览器弹出窗口，通过Ajax加载和显示文档，并且允许用户移动和关闭它。图11-8展示了这个结果。

这是一个相当复杂的练习，需要在这个HTML文档中包含几乎所有的库组件：

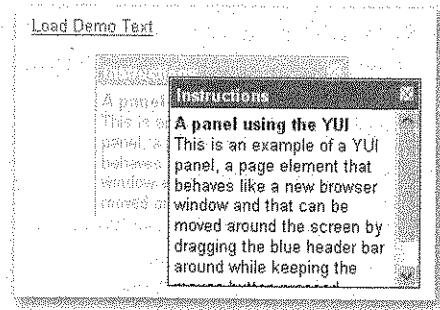


图11-8 使用YUI库组件模拟一个弹出窗口

examplePopUpReplace.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html dir="ltr" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8">
  <title>Example: Using YUI to replace pop-up windows</title>
  <script type="text/javascript" src="yahoo-min.js"></script>
  <script type="text/javascript" src="event-min.js"></script>
  <script type="text/javascript" src="dom-min.js"></script>
  <script type="text/javascript" src="dragdrop-min.js"></script>
  <script type="text/javascript" src="container-min.js"></script>
  <script type="text/javascript" src="connection-min.js"></script>
  <script type="text/javascript" src="animation-min.js"></script>
  <script type="text/javascript" src="popUpReplace.js"></script>
  <style type="text/css">
    @import 'reset-min.css';
    @import 'fonts-min.css';
    @import 'popUpReplace.css';
  </style>
</head>
<body>
  <a href="demotext.html"
    onclick="makeRequest(this);return false">Load Demo Text</a>
  <div id="win">
    <div class="hd"></div>
    <div class="bd"></div>
  </div>
</body>
</html>

```

注意这个示例不是真的非分离式的：它使用了一个内联的onclick处理器，并且创建依赖于脚本的HTML——ID为win的

会变成模拟的弹出窗口。可以通过DOM和Event库组件很容易地创建这些，但是我们现在集中关注一下使用连接管理器和容器。Onlick处理器把这个链接作为参数传递给函数makeRequest()，它读取链接的href属性，并且使用handleSuccess()和handleFailure()作为事件处理程序启用一个新的Ajax请求。

examplePopUpReplace.js

```

function makeRequest( o ) {
  var sUrl = o.href;
  var request = YAHOO.util.Connect.asyncRequest( 'GET', sUrl,
  {
    success : handleSuccess,
    failure : handleFailure
  });
}

```

`handleSuccess()`方法会获取链接的文档的内容，并且检查`responseText`是否不等于`undefined`，然后使用`YAHOO.widget.Panel()`构造器方法的一个新实例来创建模拟的弹出窗口。这个方法拥有两个参数：要转变为面板的元素的ID以及作为JSON对象的面板属性。在这里面板的ID是`win`。

examplePopUpReplace.js (续)

```
function handleSuccess( o ) {
    if( o.responseText !== undefined ) {
        panel = new YAHOO.widget.Panel(
            "win",
            {
```

面板有着数量惊人的许多属性，它们所有的都列在`http://developer.yahoo.com/yui/container/panel/#config`上的文档中。在这个例子中，我们选择一种特效，使面板平稳地淡入和淡出，并且设置特效持续的时间为半秒钟。`Constraintviewport`属性定义了是否允许用户把面板拖曳到当前可见的浏览器区域外面。设置它为`true`可以确保用户拖曳面板到最右边或下面的时候不会有难看的滚动条。当设置`draggable`为`true`的时候，`Panel()`方法会自动地允许用户四处拖放面板；当设置`close`为`true`的时候，可以创建一个链接来隐藏面板。由于你会经常修改面板的内容，因此最好通过设置`visible`属性为`false`来隐藏它。当你定义了所有的参数后，就可以调用新面板的`render()`方法来创建所有必要的HTML元素并且应用所有的事件处理程序。

examplePopUpReplace.js (续)

```
{
    effect : {
        effect : YAHOO.widget.ContainerEffect.FADE,
        duration : 0.5
    },
    constraintviewport : true,
    close : true,
    visible : false,
    draggable : true
};
panel.render();
```

函数`handleSuccess()`从Ajax调用获取的参数`o`包含所有的连接数据，可以通过`o.responseText`获取加载的文档的文本内容。在你可以使用这个数据来组装模拟的弹出面板前，需要先把它清除，因为对于面板的标题栏你只需要标题的信息，对于内容只需要主体的内容。其余的东西，从`DOCTYPE`到样式和脚本块都要去掉。可以使用正则表达式来做，但还有另一种方式。首先，需要使用一个ID为`popupbody`的`<div>`来替换文本中的`BODY`元素。这是必需的，因为设置数据为一个元素的`innerHTML`会移除`BODY`元素——这是由于在浏览器窗口中可能只有一个`BODY`。要使用`<div>`来替换`BODY`元素，你可以使用正则表达式。

examplePopUpReplace.js (续)

```

var content = o.responseText;
content = content.replace( /<body>/, '<
<div id="popupbody">' );
content = content.replace( /<\!>/, '</div>' );

```

接下来获取ID为win的元素中的第二个

（它是面板的主体），并设置它的innerHTML属性为修改后的内容。接着可以很容易地通过getElementsByName()获取标题，通过getElementById()获取主体的内容。

examplePopUpReplace.js (续)

```

var win = document.getElementById('win');
var windowbody = win.getElementsByTagName('div')[1];
windowbody.innerHTML=content;
var title = win.getElementsByTagName('title')[0].innerHTML;
var body = document.getElementById('popupbody').innerHTML;

```

使用setBody()和setHeader()方法设置面板的主体和标题内容（前者会很方便地覆盖内容中你不需要的其余内容），并且通过调用show()方法来显示面板。

examplePopUpReplace.js (续)

```

panel.setBody( body );
panel.setHeader( title );
panel.show();
}
}

```

剩下所要做的就是定义handleFailure()方法，当链接的文档不能被加载时告诉用户。

examplePopUpReplace.js (续)

```

function handleFailure( o ){
if( o.responseText !== undefined ) {
alert( 'Couldn\'t load the content: ' + o.statusText );
}
}

```

11.5.3 YUI 小结

YUI是一个非常吸引人去使用的库，尤其是因为它不只是和其他库一样为你提供了许多方法，从而使你的生活更加轻松，同样地，它给出了许多文档和示例，还有CSS布局和出版的资源，可以很快地拼建起一个JavaScript增强的网站或者Web应用程序，可以在所有现代的浏览器上运行。

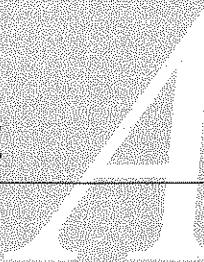
这个库所涉及的许多人以及在邮件列表上的讨论都必定使这个函数库变得越来越好，并使得测试新的组件更加容易。现在来说，这个函数库还不成熟，而且文档有时会让人望而生畏，但是

在邮件列表上提问非常容易，许多开发人员都很乐意帮助你——在更多有疑问的情况下或者当你有一个扩展函数库的请求或想法的时候，你也会寻找一位处理这些函数库的雅虎网络开发人员来回答你——在未来可能正好是我。与其他函数库相比较，尤其是jQuery，要实现一些特效，你需要在YUI中编写更多的代码；可是，YUI所做的是保持语法与JavaScript标准一致，它不会改变你做循环或遍历的方式。乍一看，YUI驱动的代码非常复杂，许多的YAHOO.something.somethingElse语句让人感到迷惑。但是你用的时间越长，你会发现越容易，你会开始欣赏根据它们所做的事情或者属于函数库的哪一部分来对这些方法和属性命名的方式。

11.6 小结

我希望这一章可以让你知道一下目前都有哪些应用，而且我相信这只是分享内容、信息和服务更长体验的一个开始。许多开发人员花费了很多时间来创建完美的代码，结果发现已经有另一个产品和他所做的完全一样，或许做得更好些；可是，这并不是什么问题，正是通过沟通以及试验和错误，我们才做得越来越好。

通过不断地关注以及查看社区提供的服务，你可以学会许多东西并在其中帮助其他人，尤其是从你的角度给出关于一些服务和函数库如何才能更容易使用的反馈。不要认为自己的代码已经非常完美了，有时候不等别人给你指出的时候，你已经意识到了不足。这是两方面的。你不能搪塞自己的缺点，坚持下去，你就会变得越来越好。也不要害羞——在JavaScript社区中要多与人交流。



在这个附录中，我会为你介绍一些调试JavaScript代码的技巧和工具。熟悉调试工具是非常重要的，因为编程有很大一部分时间是在查找特定情况下错误的代码。一些浏览器可以帮助你解决这些问题；而其他的浏览器则隐藏了它们的调试工具使得调试很困难，或者返回了加密的错误信息，让人感到更迷惑而不是有所帮助。我比较偏爱的包括“未定义就是没有定义”，或者IE标准“对象不支持这个属性或方法”。

A.1 常见的 JavaScript 错误

我们先来看一些常见的错误，这些错误可能每个JavaScript开发人员在他的职业生涯中都犯过。记住这些东西，会使你在检查一个错误的脚本时，更快速地发现问题所在。

A.1.1 拼写错误以及区分大小写的问题

最容易发现的错误是JavaScript方法名或属性的拼写错误。一些典型的错误包括使用`getElementsByTagName()`替换了`getElementsByTagName()`、使用`getElementById()`替换了`getElementById()`以及`node.style.colour`（尤其是对英国英语的作者）。很多时候问题也可能是没有区分大小写，例如，把关键字写成了混合大小写的形式而不是小写的形式。

```
If( elm.href ) {  
    var url = elm.href;  
}
```

没有叫做If的关键字，但是有一个叫做if的关键字。同样的区分大小写问题在变量名中也存在：

```
var FamilyGuy = 'Peter';  
var FamilyGuyWife = 'Lois';  
alert( 'The Griffins:\n' +  
      familyGuy + ' and ' +  
      FamilyGuyWife );
```

这样会导致一个错误信息，提示“familyGuy没有定义”，虽然确实存在一个变量叫做`FamilyGuy`，但是没有一个叫`familyGuy`的变量。

A.1.2 试图访问未定义的变量

我们在本书的第一章中讲过，你可以通过使用或不使用var关键字来声明变量从而定义它们（后者对于定义变量的范围是必需的）。

```
Stewie = "Son of Peter and Lois";
var Chris = "Older Son of Peter and Lois";
```

如果试图访问一个还没有定义的变量，那么你会得到一个错误信息。下面的脚本中的alert()会抛出一个错误，因为Meg还没有定义。

```
Peter = "The Family Guy";
Lois = "The Family Guy's Wife";
Brian = "The Dog";
Stewie = "Son of Peter and Lois";
Chris = "Older Son of Peter and Lois";
alert( Meg );
Meg = "The Daughter of Peter and Lois";
```

在像这样非常明显的例子中，错误是很容易发现的，但是要在下面的例子中推测bug的位置会怎么样呢？

exampleFamilies.html

```
function getFamilyData( outputID, isTree, familyName ) {
    var father, mother, child;
    switch( familyName ) {
        case 'Griffin':
            father = "Peter";
            mother = "Lois";
            child = "Chris";
            break;
        case 'Flintstone':
            father = "Fred";
            mother = "Wilma";
            child = "Pebbles";
            break;
    }
    var out = document.getElementById( outputID );
    if( isTree ) {
        var newUL = document.createElement( 'ul' );
        newUL.appendChild( makeLI( father ) );
        newUL.appendChild( makeLI( mother ) );
        newUL.appendChild( makeLI( child ) );
        out.appendChild( newUL );
    } else {
        var str = father + ' ' + mother + ' ' + child;
        out.appendChild( document.createTextNode( str ) );
    }
}
getFamilyData( 'tree', true, 'Griffin' );
```

微软的IE会告诉你在第23行有一个错误——‘outputID’ is undefined，如图A-1所示。



图A-1 IE显示了第23行的一个错误

然而，如果看一下第23行的代码，如图A-2所示，你会发现看起来并没有什么错误。

```

9  function getFamilyData(outptID,isTree,familyName){
10    var father,mother,child;
11    switch(familyName){
12      case 'Griffin':
13        father = "Peter";
14        mother = "Lois";
15        child = "Chris";
16        break;
17      case 'Flintstone':
18        father = "Fred";
19        mother = "Wilma";
20        child = "Pebbles";
21        break;
22    }
23    var out = document.getElementById(outptID);
24    if(isTree){
25      var newUL=document.createElement('ul');
26      newUL.appendChild(makeListElement(father));

```

图A-2 在UltraEdit中对第23行突出显示的代码

问题在于函数参数中的一个拼写错误，就是图A-3中突出显示的部分，这意味着outputID没有定义而是定义了outptID。

```

9  function getFamilyData(outptID,isTree,familyName){
10    var father,mother,child;
11    switch(familyName){
12      case 'Griffin':
13        father = "Peter";
14        mother = "Lois";
15        child = "Chris";
16        break;
17      case 'Flintstone':
18        father = "Fred";
19        mother = "Wilma";
20        child = "Pebbles";
21        break;
22    }
23    var out = document.getElementById(outptID);
24    if(isTree){
25      var newUL=document.createElement('ul');
26      newUL.appendChild(makeListElement(father));

```

图A-3 拼错的函数参数引起了错误

参数的排印错误是个非常容易让人混淆的bug，因为浏览器告诉你错误发生在该变量被使用的行，而不是你真正发生错误的地方。

A.1.3 错误的右大括号和小括号数

另一个非常常见的错误是没有右大括号或在删除一些行的时候在代码中留下一个被遗弃的右大括号。举个例子，假设你不再需要这个isTree判断选择了，把它从这段代码中删除掉：

exampleCurly.html

```
function getFamilyData( outputID, familyName ) {
    var father, mother, child;
    switch( familyName ) {
        case 'Griffin':
            father = "Peter";
            mother = "Lois";
            child = "Chris";
            break;
        case 'Flintstone':
            father = "Fred";
            mother = "Wilma";
            child = "Pebbles";
            break;
    }
    var out = document.getElementById( outputID );
    var newUL = document.createElement( 'ul' );
    newUL.appendChild( makeListElement( father ) );
    newUL.appendChild( makeListElement( mother ) );
    newUL.appendChild( makeListElement( child ) );
    out.appendChild( newUL );
}
getFamilyData( 'tree', true, 'Griffin' );
```

以黑体显示的被遗弃的右大括号会引起一个“在第30行中有语法错误”。当你没有关闭一个程序结构中所有的大括号的时候也会引发同样的问题，尤其是当你没有缩进代码的时候，这样的错误就很容易发生：

exampleMissingCurly.html

```
function testRange( x, start, end ) {
if( x <= end && x >= start ) {
if( x == start ) {
alert( x + ' is the start of the range');
}
if( x == end ) {
alert(x + ' is the end of the range');
}
```

```

if( x != start && x != end ) {
    alert(x + ' is in the range');
} else {
    alert(x + ' is not in the range');
}
}

```

运行这个例子会引起一个“第27行需要一个‘}’”的错误，它就是这段脚本块的最后一行。这意味着在这个条件结构内部的某个地方，我们忘记了添加一个右大括号。什么地方缺少大括号是相当难找的，但是当代码是完全缩进编排的时候就会容易很多。

`exampleMissingCurlyFixed.html`

```

function testRange( x, start, end ) {
    if( x <= end && x >= start ) {
        if( x == start ) {
            alert(x + ' is the start of the range');
        }
        if( x == end ) {
            alert(x + ' is the end of the range');
        }
        if( x != start && x != end ) {
            alert(x + ' is in the range');
        }
    } else {
        alert( x + ' is not in the range' );
    }
}

```

前面丢失的大括号是以黑体显示的（紧跟在`alert()`消息“is in the range”之后）。

没有右小括号或右小括号过多是另一个常见的问题。当你在`if()`条件语句中嵌套函数而稍后又删除一部分的时候常发生这种情况。例如：

```
if (all = parseInt(getTotal())){ doStuff(); }
```

这样会引起一个错误，因为你忘记了关闭条件语句自己的小括号。

```
if (all = parseInt(getTotal())){ ... }
```

当你嵌套太多的方法和返回结构的时候也会发生这种问题：

```
var elm=grab(get(file)).match(/<id>(\w+)</id>/)[1];
```

这一个语句在[1]的后面缺少右小括号：

```
var elm=grab(get(file)).match(/<id>(\w+)</id>/)[1]);
```

一般而言，这种函数的拼接不是很好的代码风格，但是有些情况下你还是会遇到类似这样的例子。技巧就是从左到右数一下左、右小括号——好的编辑器还会自动地突出显示左、右小括号。

你也可以编写一个实用函数来进行检查，该函数本质上是你遇到代码语法问题时对所注意的细节的一个检查：

```
exampleTestingCodeLine.html
function testCodeLine( c ) {
    if( c.match( /\(/g ).length != c.match( /\)/g ).length ) {
        alert( 'closing ) missing' );
    }
}
c = "var elm=grab(get('demo.xml')) +
    ".match( /<id>(\w+)</id>/ )[1] );";
testCodeLine( c );
```

A.1.4 拼接出错

拼接操作常发生在使用JavaScript来输出HTML的时候。要确保拼接到一起的不同部分之间不要忘了+号。

```
father = "Peter";
mother = "Lois";
child = "Chris";
family = father+ " +mother+ " child;
```

前面的代码在child变量之前缺少一个加号：

```
father = "Peter";
mother = "Lois";
child = "Chris";
family = father+ " +mother+ " +child;
```

另一个问题就是，要确保不要拼接错误的数据类型。

```
father = "Peter";
fAge = 40;
mother = "Lois";
mAge = 38;
child = "Chris";
cAge = 12;
family = father + ", " + mother +
         " and " + child + " Total Age: " +
         fAge + mAge + cAge;
alert( family );
```

这样并不会显示所期望得到的结果，而是下面的结果：

```
Peter, Lois and Chris Total Age: 403812
```

这个错误在于你拼接了多个字符串和数字，而+运算符是按从左到右的顺序执行的。你需要在年龄术语外面加上小括号：

```
father = "Peter";
fAge = 40;
mother = "Lois";
mAge = 38;
```

```

child = "Chris";
cAge = 12;
family = father + ", " + mother +
    " and " + child + " Total Age: " +
    (fAge + mAge + cAge);
alert(family);

```

这样就会得到所期望的结果：

```
Peter, Lois and Chris Total Age: 90
```

A.1.5 误用赋值语句替代条件测试语句

当检查一个变量的值时，很容易写成赋值而不是条件判断，就是因为你忘记了一个等号。

```

if(Stewie = "talking") {
    Brian.hear();
}

```

这样会使得Brian一直在听，不只是等Stewie要说话的时候；然而，只要添加一个等号就会使Brian只有在Stewie说话的时候才听：

```

if(Stewie == "talking") {
    Brian.hear();
}

```

A.2 使用 alert() 和“控制台”元素跟踪错误

跟踪错误最容易的方式就是，在你需要检查某个值的地方使用alert()方法。alert()方法会（使用可能仍在后台运行的Ajax调用的异常）停止脚本的执行，并且为你提供与某个特定变量的值相关的信息，从而你可以推断出这个值是否正确或者是否是它引起了这个错误。在一些实例中，使用alert()并不是合适的选择，举个例子，假设你希望在循环遍历数组的时候跟踪几个值的变化。根据数组的大小，这可能会变得很枯燥，因为当你每次希望关掉alert()窗口并且继续下一个数组元素的时候，都需要按一下回车键。对于这个问题的解决办法是使用你自己的调试控制台或者日志元素。大多数JavaScript库函数都有一个日志子库（DOJO有一个dojo.logging，在`http://manual.dojotoolkit.org/index.html#infrastructure`上可以看到；Mochikit有一个Logging，在`http://mochikit.com/doc/html/MochiKit/Logging.html`上可以看到），还有在本书中我们放到一起的DOMhelp库函数也不例外。可以使用initDebug()、setDebug()和stopDebug()方法来模拟一个调试控制台。只要简单地为ID是DOMhelpdebug的元素添加一个样式并使用这些方法来显示这个元素并且把内容写到它里面就可以了。例如：

`exampleDebugTest.html (摘要)`

```
#DOMhelpdebug{
    position: absolute;
    top: 0;
    right: 0;
```

```

width:300px;
height:200px;
overflow:scroll;
background:#000;
color:#0F9;
white-space:pre;
font-family:courier,monospace;
padding:1em;
}
html>body #DOMhelpdebug{
position:fixed;
min-height:200px;
height:200px;
overflow:auto;
}

exampleDebugTest.html (摘要)
<script type="text/javascript"
src="../DOMhelp.js"></script>
<script type="text/javascript">
function DOMDebugTest(){
DOMhelp.initDebug();
for(var i = 0; i < 300; i++ ) {
DOMhelp.setDebug( i + ' : ' + ( i % 3 == 0 ) + '\n' );
}
DOMhelp.addEvent( window, 'load', DOMDebugTest, false );
</script>

```

这个例子会从数字0循环到299，并且会显示这个数字是否可以被3整除。不用敲300次回车键，要查看结果你所需要的就是在使用前面的样式创建的“控制台窗口”中滚动一下。

A.3 使用 try 和 catch() 处理错误

可以使用try...catch结构来测试一下脚本。只要简单地把你想测试的代码放到一个try语句中就可以了，如果有错误的话，catch()里的代码就会被执行。例如：

```

exampleTryCatch.js
try{
    alert( 'this is a code example' );
    alert( myVariable );
} catch( exceptionObject ) {
    // Predefine empty output
    var errorString = '';
    for( i in exceptionObject ) {
        errorString += i + ':' + exceptionObject[i] + '\n';
    }
    alert( errorString );
}

```

当try语句中有错误发生的时候，catch()方法会获得Exception对象作为其参数。可以给这个对象任意一个变量名；在这个例子中，我们把它叫做exceptionObject。根据错误和浏览器的不同，这个对象会有不同的属性，而在不同浏览器上的相同的这些属性会有不同的值。在Windows XP平台的IE 6上，前面的代码会以下面的方式显示这个错误，它在第二个alert()函数中试图显示一个未定义的变量：

```
name:TypeError
message:'myVariable' is undefined
number:-2146823279
description:'myVariable' is undefined
```

在Windows XP平台的Firefox 1.5.0.3上：

```
message:myVariable is not defined
fileName:file:///c:/exampleTryCatch.html
lineNumber:11
stack:@file:///c:/exampleTryCatch.html:11
name:ReferenceError
```

在Windows XP平台的Opera 8.54上：

```
message:Statement on line 4: Reference to undefined variable: myVariable
Backtrace:
Line 4 of inline#1 script in
file://localhost/C:/exampleTryCatch.html
alert(myVariable);
opera#sourceloc:4
```

如果你试验一个不同的错误，那么你仍然会得到属性相同但值不同的结果。通过误拼alert()，我们来试着调用一个并不存在的函数。

```
exampleTryCatchTypo.js
try{
    alert( 'this is a code example' );
}
catch( exceptionObject ) {
    var errorString = '';
    for( i in exceptionObject ) {
        errorString += i + ':' + exceptionObject[i] + '\n';
    }
    alert( errorString );
}
```

在IE 6上的结果如下：

```

name:TypeError
message:Object expected
number:-2146823281
description:Object expected

```

在Firefox 1.5.0.3上：

```

message:allert is not defined
fileName:///c:/exampleTryCatchTypo.html
lineNumber:10
stack:@file:///c:/exampleTryCatchTypo.html:10
name:ReferenceError

```

在Opera 8.54上：

```

message:Statement on line 3: Reference to undefined variable: allert
Backtrace:
Line 3 of inline#1 script in
file://localhost/C:/exampleTryCatchTypo.html
allert("this is a code example");
opera#sourceloc:3

```

在调试的过程中使用try和catch会非常的有用，并且依靠浏览器，你会很容易地发现问题所在。

注解 注意Opera会报告在SCRIPT元素中的代码行数，而Firefox报告的是整个HTML文档的行数。IE则会把问题留给你自己，看看在使用try和catch()的时候什么地方可能发生错误。

然而，由于它里面的东西是JavaScript，因此不能使用它来调试JavaScript语法错误：

exampleTryCatchFailing.js

```

try{
    alert( 'this is a code example' );
} catch( exceptionObject ) {
    var errorString = '';
    for( i in exceptionObject ) {
        errorString += i + ':' + exceptionObject[i] + '\n';
    }
    alert( errorString );
}

```

在第一个alert()函数后多余的右小括号是一个语法错误，它会使浏览器显示内置的错误信息，并停止脚本的执行，而不是执行在catch结构体中的代码。

A.4 连续地不加注释

另一个非常容易跟踪问题的方式是，对所有的脚本添加注释，然后逐个函数去掉注释。如果是单个函数的话，则逐行去掉注释。在每次去掉一行注释后，需要在浏览器中重新加载它来检查脚本，而且很快就会获得引起错误的地方。尽管这种方式需要花费些时间，但很容易大概地知道错误发生的地方，这就是为什么你需要依靠浏览器来给你提供这种信息的原因了。

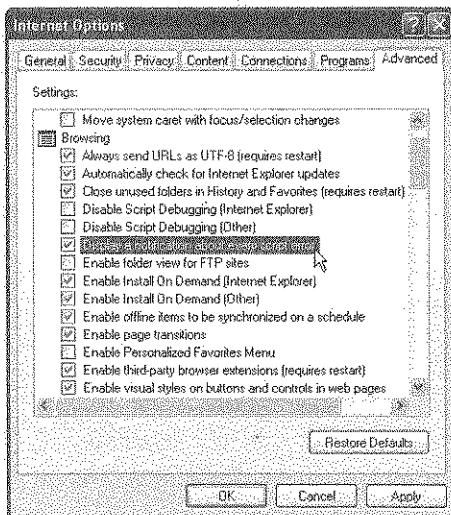
A.5 各种浏览器中的错误报告

各种浏览器可以在不同的程度上帮助你调试代码。下面是不同浏览器以及它们显示错误和定位它们来源的方法的综述。因为我想总结一下你可能使用到的不同浏览器，所以我不会深入到关于每个浏览器不同插件的细节。相反，我会提供一些信息和URL地址，你可以获得这些插件并且自己去读一下它们的文档。这样做的原因并不是我比较懒，而是事实上这些浏览器插件以及JavaScript调试工具的数量在过去几年里增长太快了，尤其是随着Ajax的出现，很多人对JavaScript又产生了兴趣。因此，关于某种特定工具的详细解释可能在本书到达书店的时候就已经过期了。

A.5.1 微软的IE 6

IE 6在很长一段时间里可能仍然会是Windows平台上最常用的浏览器。尽管你是一个开发人员，但它并不能给你多少帮助，大多数时候它会抛出一些其他浏览器不会抛出的错误。

IE老一点的版本（PC机上的版本4和版本5）会在JavaScript每次发生错误的时候显示一个报告窗口。这个特性在新版本中已经给关闭掉了；你可以在Tools→Internet Options（工具→Internet选项）的Advanced（高级）选项卡的Browsing（浏览）部分再把它打开，如图A-4所示。



图A-4 在微软的IE中打开错误报告

由于IE提供的这些错误信息含义可能会比较模糊，如“对象不支持这个方法或属性”，因此如果IE是你开发唯一的选择，你可能希望在<http://www.microsoft.com/downloads/details.aspx?familyid=2F465BE0-94FD-4569-B3C4-DFFDF19CCD99&displaylang=en>下载免费的微软脚本调试器。微软脚本调试器是一个允许你设置断点（它会停止脚本的执行直到你告诉它可以继续为止）的调试工具。你也可以使用它来读取变量的值并且在脚本正在运行的时候改变它们。这些所有的东西都是在浏览器之外工作的，这意味着你不需要借助前面提到的那些技巧，如alert()、调试控制台或者对部分代码按顺序添加注释。关于微软脚本调试器的详细介绍可以在http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdbug/Html/sdbug_1.asp上获得。

A.5.2 Safari

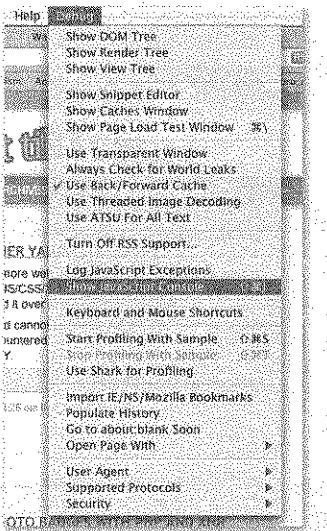
苹果公司的拳头浏览器产品看上去一点也不能为你提供JavaScript调试的帮助。我用“看上去”这个词是因为Safari有一个非常便利的调试菜单；唯一的问题就是默认时它是隐藏的。

启动调试菜单有两种方式：通过命令行或者通过安装Safari增强版。命令行的办法是非常简单的。

- (1) 关闭Safari浏览器。
- (2) 在浏览器的操作系统控制台（应用程序→附件→命令提示符）中输入% defaults write com.apple.Safari IncludeDebugMenu 1，然后敲击回车键。

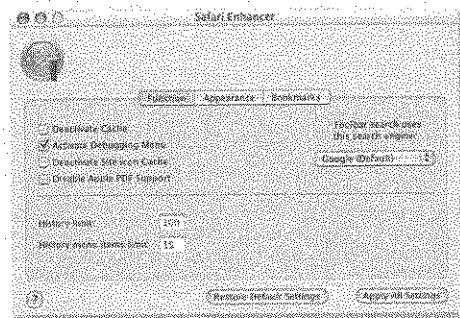
(3) 重新启动Safari浏览器。

你会发现在顶部的菜单栏中有一个新的调试（Debug）菜单选项，如图A-5所示。和许多其他有用的如DOM树查看器以及在网站不支持Safari的时候模仿其他浏览器的功能特性一样，你可以选择把JavaScript错误输出到操作系统的控制台上或在一个JavaScript控制台窗口中显示错误。



图A-5 Safari浏览器的调试菜单

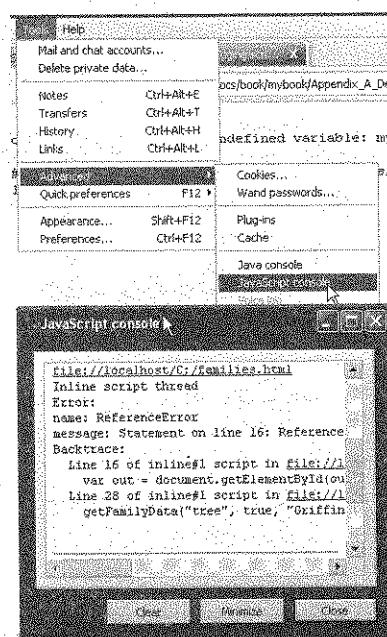
如果你喜欢用图像化的界面来修改Safari的设置，那么可以在<http://www.versiontracker.com/dyn/moreinfo/macosx/17776>下载Safari增强版，并且可以如图A-6所示，选择代表激活调试菜单（Activate Debugging Menu）的选择框，然后单击应用所有的设置（Apply All Settings）按钮。



图A-6 在Safari增强版中启用调试菜单

A.5.3 Opera 8.5

Opera不只有可以通过按F12快速关闭所有浏览器支持（JavaScript、cookies、插件和弹出窗口）的各种选项，可以帮助你很快地检查网页是否依赖于JavaScript，而且它还有一个内置的JavaScript控制台，可以给出非常详细的错误报告。可以通过工具（Tools）→高级（Advanced）→JavaScript控制台（JavaScript console）来打开这个控制台，如图A-7所示。

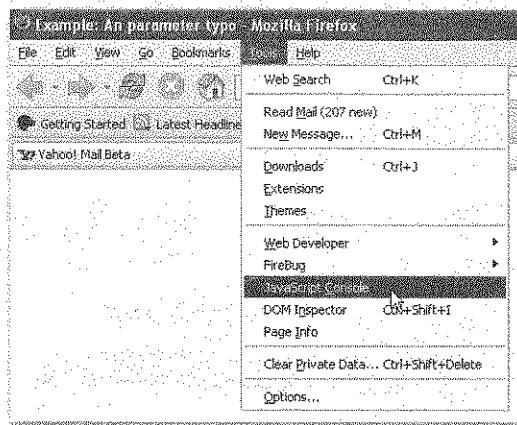


图A-7 在Opera浏览器中显示JavaScript控制台

A.5.4 Firefox 1.5.0.3

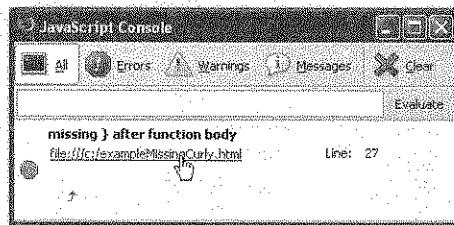
Firefox（火狐）对此保持沉默，它不会通过报告JavaScript错误来打扰用户。可是，它带有一个非常强大的JavaScript控制台，可以通过在主要工具栏里选择工具（Tools）→JavaScript控制台（JavaScript console）来显示它，如图A-8所示。

注解 如果你的firefox浏览器没有与图A-8所示的同样的选项，那么也不要感到困惑；它们中的一些独立于我安装的扩展插件。不过，你应该有JavaScript控制台。



图A-8 在Firefox浏览器中显示JavaScript控制台

一旦被激活了，JavaScript控制台在它的浏览器窗口上是可见的，它会在你的Firefox窗口或选项卡中列出你打开的所有页面中所发生的JavaScript错误。当把控制台打开并在网络上冲浪的时候，可以看到累计有多少个错误，这点是非常令人惊异的。exampleMissingCurly.html中的错误将显示为如图A-9所示。



图A-9 Firefox浏览器的JavaScript控制台显示了一个错误

可以显示错误、警告以及普通消息，或者对列表的每一个进行过滤。单击清除（Clear）按钮会清除这个列表。甚至可以通过把它粘贴到代码框中并单击评估（Evaluate）按钮来对它进行评估。

默认情况下，JavaScript控制台会在一个新窗口中打开，但是有一个技巧，可以用来把它显示到侧栏中。

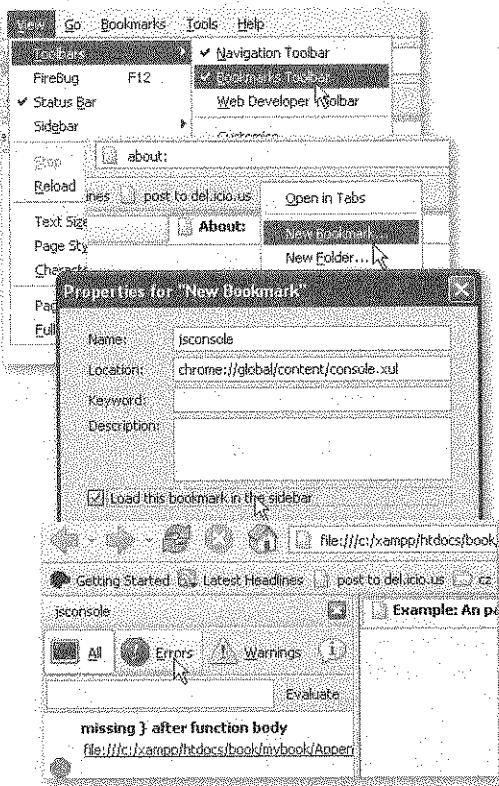
(1) 通过选择视图 (View) → 工具栏 (Toolbars) → 书签工具栏 (Bookmarks Toolbar) 确保书签工具栏可以看得见。

(2) 右键单击这个工具栏并选择新建书签 (New Bookmark)。

(3) 如图A-10所示添加信息。地址必须是chrome://global/content/console.xul，但是你可以随心所欲地给这个书签命名。

(4) 确保在侧栏载入此书签 (Load this bookmark in the sidebar) 选项被选择了。

这样就会在工具栏中添加一个新的书签按钮，当你单击它的时候就会在工具栏中打开JavaScript控制台了。



图A-10 使Firefox在侧栏中显示JavaScript控制台的一个技巧

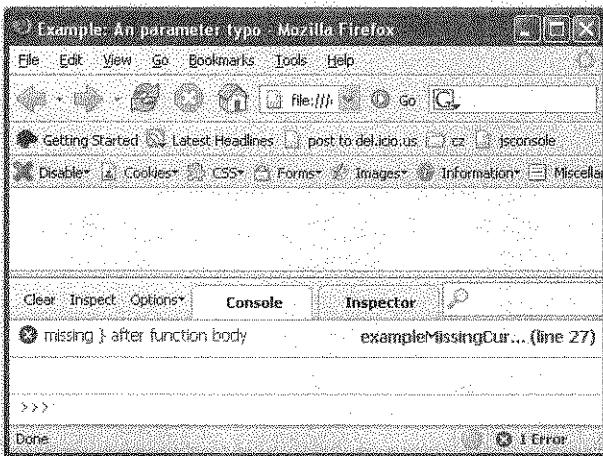
这可以使你在Firefox中很快地发现错误；然而，与这个浏览器给你提供的强大功能相比，它只不过是冰山之一角。Mozilla和Firefox可以很方便地使用JavaScript扩展和一种基于XML名为XUL的语言进行提升。开发人员们不断开发出许多新的扩展，并且把它们放到了网上。其中有2个对于作为JavaScript开发人员的你来说是非常便利的：

1. Web Developer扩展

Web Developer扩展是一个由Chris Pederick开发的工具栏，在网站<http://www.chrispederick.com/work/webdeveloper/>可以下载。这个工具栏允许你快速地切换JavaScript支持[否则，你可以通过工具（Tools）→选项（Options）→内容（Content）→启用JavaScript（Enable JavaScript）]，在源代码查看器中显示通过JavaScript生成的HTML，使用数据自动地填充表单，随意地编辑页面的样式风格，并且可以从许多选项中进行选择。它就是网页开发的一把利器。

2. FireBug

如果你认为Opera、Safari以及Firefox的JavaScript控制台很便利的话，那么可以在网站<http://www.joehewitt.com/software/firebug/>上获得一下由Joe Hewitt（他也是作为默认的Firefox安装的一部分的DOM检查器的开发人员）开发的FireBug扩展。这个FireBug扩展在浏览器窗口的右下角是可以看到的，并且它会在控制台中显示带有许多错误的警告图标或者显示一个绿色图标表明所有的代码都是正确的。如果有错误的话，你可以单击这个按钮（或按F12）在浏览器窗口的下半部分打开控制台，如图A-11所示。



图A-11 打开控制台的FireBug扩展

FireBug的特性太多了，这里就不提了。这个文档中所发生的东西在控制台中你都可以看到。它显示了DOM结构、应用的CSS类、元素的大小和属性、指派到元素上的事件、Ajax请求的返回值以及很多其他的东西。在DOM检查器模式下，你甚至可以随意地改变这个文档，来快速地测试远程文档中的变化。当然，这只是为了检查的目的，你不能把这个变化保存到远程的服务器上。

3. Venkman

Mozilla的一个与微软的脚本调试器（Microsoft Script Debugger）类似的产品叫做Venkman，它可以从<https://addons.mozilla.org/firefox/216/>上获得。Venkman是一个功能齐全的调试工具，可以让你监视变量、设置调试点并真正深入到JavaScript开发的范围。要解释Venkman所有的特性需要单独的一章，而对于日常的JavaScript任务，它可能会有点大材小用。如果你想了解Venkman

所有的特性以及使用方法，那么你可以查看在它的主页<http://www.mozilla.org/projects/venkman/venkman-walkthrough.html>上提供的教程和说明。另一个非常好的入门教程是由Svend Tofte编写的介绍性的文章*Learning Venkman*，可以在http://www.svendtofte.com/code/learning_venkman/上获得。前面所述的FireBug扩展的最新版本也包括一种类似Venkman的调试器，这使得它在某些方面显得有点过时；可是，它还是值得提一下，因为一些公司还在使用它。

A.6 JSLint 和 JSUnit

一些不依赖于浏览器的工具也可以帮助你进行JavaScript的开发。其中一个就是由Douglas Crockford开发的在线JavaScript校验器JSLint，可以在<http://www.jslint.com/lint.html>上看到它。JSLint是一个使用JavaScript编写的工具，可以根据句法验证结果校验脚本的正确性，并且确保良好的编码风格。由于“良好的编码风格”是件主观的事情，因此你可能对JSLint报告有所保留。JavaScript开发大多是在浏览器的环境中，而且有时你需要遵从标准或者走捷径使得脚本运行得更快或者解决一些浏览器的问题。也就是说，如果你不必走捷径的话，那么从代码的整洁上来看，它还是一种用以寻找优化脚本的方式的非常好的工具。JSLint最大的好处就是它可以给你一个脚本完整的分析，包括全局变量以及不同的函数被调用多少次的报告。

如果接触过后台编程环境，那么你可能已经使用过或者至少听说过单元测试了（http://en.wikipedia.org/wiki/Unit_testing）。简要地说，单元测试意味着你为所有的方法和函数编写测试用例，并且一个测试用具（testing harness）可以使你单击一个按钮就可以连续地运行所有的这些测试。通过这种方式，在开发代码前你就可以保证代码会如你定义的它必须满足的测试用例那样运行。对于Java，有JUnit；对于PHP，有PHPUnit；而对于JavaScript则有JSUnit，可以在<http://www.jsunit.net/>上看到更多相关的信息。

A.7 小结

总而言之，调试JavaScript现在比以前容易多了，尤其是在Firefox的世界里，开阔你的视野是非常重要的，因为你可能从中获益的新产品几乎每周都有新版本发布。随着Ajax方法而来的JavaScript的复兴，还使得开发IDE的厂商更加意识到良好的JavaScript调试工具的市场需求。传统上Java或.NET开发所专有的产品也开始增加对JavaScript的支持，不再局限于源代码的着色。

人民邮电出版社图灵公司专注于引进国外经典教材与优秀科技图书，出版国内高校教师编写的专业教材，专业方向包括：计算机、数学、统计学和电子电气等。合作伙伴包括Prentice-Hall、Addison-Wesley、Wiley、McGraw-Hill、剑桥大学出版社、Elsevier、IEEE Press、Wrox、SIAM等多家世界知名出版公司，具有丰富的选题资源和雄厚的出版力量。

修炼之道系列

<p>书名：设计模式解析 [第2版] 原书名：Design Patterns Explained: A New Perspective on Object-Oriented Design 作者：Alan Shalloway, James R. Rothery 译者：徐言如 定价：45.00元</p> <p>书名：敏捷软件开发 原书名：Agile Principles, Patterns, and Practices in C# 作者：Robert C. Martin, Michael Feathers 译者：郭东、孙海 定价：69.00元</p>	<p>书名：重构与模式 原书名：Refactoring to Patterns 作者：Joshua Kerievsky 译者：杨光、刘继诚 定价：45.00元</p> <p>书名：敏捷软件开发：原则、模式和实践 [C#版] [英文注释版] 原书名：Agile Principles, Patterns, and Practices in C# 作者：Robert C. Martin, Michael Feathers 译者：郭东、孙海 定价：69.00元</p>	<p>书名：修改代码的艺术 原书名：Working Effectively with Legacy Code 作者：Michael C. Feathers 译者：刘未第 定价：59.00元</p> <p>书名：敏捷——改善既有代码的设计 [英文注释版] 原书名：Refactoring: Improving the Design of Existing Code 作者：Martin Fowler</p>	<p>书名：程序员修炼之道（英文注释版） 原书名：The Pragmatic Programmer From Journeyman to Master 作者：Andrew Hunt, David Thomas 译者：刘未第 定价：49.00元</p> <p>书名：企业应用架构模式 [英文注释版] 原书名：Patterns of Enterprise Application Architecture 作者：Martin Fowler</p>	<p>书名：数据软件开发：原则、模式与实践 [英文注释版] 原书名：Agile Software Development: Principles, Patterns, and Practices 作者：Robert C. Martin, Michael Feathers 译者：郭东、孙海 定价：69.00元</p>

Java 系列

<p>书名：Java教程(第4版) 原书名：The Java Tutorial: A Short Course on the Basics 作者：Sharon Zukhori 译者：胡丽霞 定价：49.00元</p> <p>书名：Hibernate实战 原书名：Java Persistence with Hibernate 作者：Christian Bauer, Gavin King 定价：89.00元</p>	<p>书名：Java语言设计语言(第4版) 原书名：Java Programming Language 作者：Ken Arnold, James Gosling, David Holmes 译者：陈昊鹏、章程、张恩博、李楠 定价：69.00元</p> <p>书名：深入浅出Struts 原书名：Struts Design and Programming: A Tutorial 作者：Budi Kurniawan 译者：王维维、杨晓云、薛兰等 定价：45.00元</p>	<p>书名：Beginning Java Objects 原书名：Beginning Java Objects: From Concepts to Code 作者：Jacquie Barker 译者：万波 定价：78.00元</p> <p>书名：JSF实战 原书名：JavaServer Faces in Action 作者：Kito Mann 译者：钱伟强、陈晓冬、何墨 定价：69.00元</p>	<p>书名：Struts基础教程 原书名：Beginning Apache Struts From Novice to Professional 作者：Arnold Doray 译者：钱伟强、孙清松 定价：45.00元</p> <p>书名：Struts基础教程 原书名：Beginning Apache Struts From Novice to Professional 作者：Arnold Doray 译者：钱伟强、孙清松 定价：45.00元</p>	<p>书名：Ajax与Java高级程序设计 原书名：Pro Ajax and Java: Advanced Web Design 作者：Nathaniel T. Schutta, Ryan Aleson 译者：杨光 定价：45.00元</p>
<p>书名：Java疑惑 原书名：Java Puzzlers: Traps, Pitfalls, and Corner Cases 作者：Joshua Bloch, Neal Gafter 译者：陈昊鹏 定价：39.00元</p>	<p>书名：深入分析Spring MVC与Web Flow 原书名：Exploring Spring MVC and Web Flow 作者：Aren Dawson, Steven Deverell, Sait Lardil 译者：雷斯、沈艳 定价：39.00元</p>	<p>书名：Java编程风格 原书名：The Bertrand of Java Style 作者：Aldo Vermeulen, Arjan Arnsdorf, Greg Bumgarner, Patrick Thompson 译者：雷鸣 定价：39.00元</p>	<p>书名：Erlang程序设计 原书名：Programming Erlang: Software for a Concurrent World 作者：Joe Armstrong 译者：赵东伟等 定价：39.00元</p>	<p>书名：Groovy实战 原书名：Groovy in Action 作者：Darrin Kinney, Guilherme Lafer, Andrew Glover 译者：黄光华等 定价：39.00元</p>



Web 开发系列



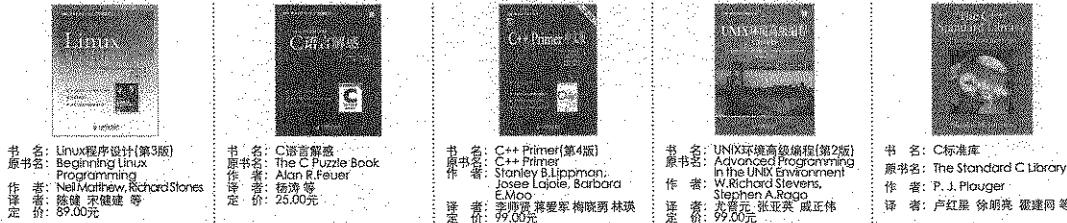
北京图灵文化发展有限公司

户名：北京图灵文化发展有限公司
开户行：工商银行北京翠微路支行金四季分理处
账号：0202096209200003993
电话：010-88593240, 88593802 传真：010-88593803
网址：<http://www.turingbook.com>
地址：北京市海淀区西四环北路140号鼎源商务楼405室
邮编：100097 E-mail：contact@turingbook.com

数据 库 系 列



其 他 经 典 图 书



站在巨人的肩上
Standing on Shoulders of Giants



www.turingbook.com

Beginning JavaScript with DOM Scripting and Ajax



深入浅出 JavaScript

“本书是绝佳的 JavaScript 教程，能够使你掌握最新的业界实践……强烈推荐。”

——JavaRanch.com

“我们惊喜地看到，开发人员所一直期盼的 JavaScript 图书终于出版了……任何想要透析 CSS、HTML 和 JavaScript 最新知识的人，都应该阅读本书。”

——Cody Lindley，资深 Web 程序员

学习 JavaScript 有捷径吗？当然有。如果你已经有一定的经验，本书将是你迅速成为优秀 JavaScript 程序员的捷径。

在这一部与众不同的著作中，世界级的 JavaScript 专家以平实易懂的语言，详述了 JavaScript 的主要语言特性和功能，重点放在现代 JavaScript 开发的理念（Unobtrusive JavaScript）和实践上。书中“纯手工”打造了一个名为 DOMHelp 的程序库，使读者能够直观地学习如何编写优秀的实战代码。通过阅读本书，读者将逐渐培养结构、行为与表现三层分离这一关键的现代 Web 开发理念，巩固并提升 Web 可用性、兼容性和可维护性的意识，最终，步入基于标准的 Web 开发的殿堂。



Christian Heilmann 世界顶尖的 JavaScript 程序员，Web 标准项目（WaSP）DOM 版本编程任务组成员。目前就职于 Yahoo! 英国公司。担任 Flickr 项目的交互架构师。Heilmann 拥有丰富的 Web 开发经验，曾经为宝马、麦当劳、EToys、惠普等世界级大公司开发网站。她的博客 <http://wait-till-i.com> 经常成为业界关注的焦点；此外他还维护着一个信息非常丰富的技术文档网站 <http://ican.co.uk>。

图灵 Web 开发图书阅读路线图



Apress®

本书相关信息请访问：[图灵网站](http://www.turingbook.com) <http://www.turingbook.com>

读者/作者热线：(010)88569802

反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机 / 网络开发 / 程序设计

人民邮电出版社网址 www.ptpress.com.cn

ISBN 978-7-115-17168-9



9 787115 171689

ISBN 978-7-115-17168-9/TP

定价：55.00 元